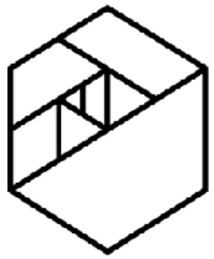
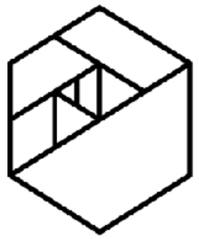


METIS

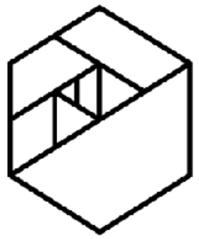
Day 3: Exploratory Data Analysis

John Navarro
john.navarro@thisismetis.com
https://www.linkedin.com/in/johnnavarro/



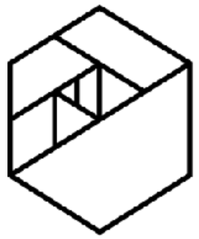
METIS

Dimensions	Name	Description
1	Series	Indexed 1 dimensional data structure
1	Timeseries	Series using timestamps as an index
2	DataFrame	A two dimensional table
3	Panel	A three dimensional mutable data structure



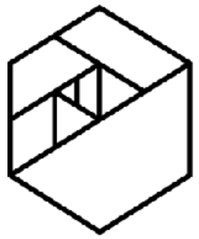
METIS

The Series Object



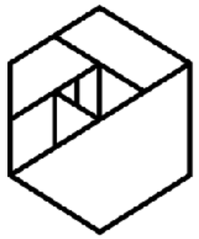
METIS

The **Series** is the **primary building block** for all the other data structures we will discuss



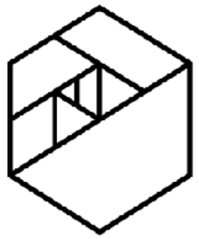
METIS

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



METIS

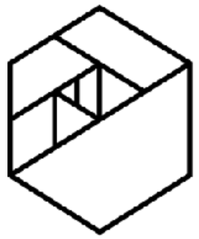
```
myData  = pd.Series( <data>, index=<index> )
```



METIS

Creating a Series

```
series1 = pd.Series( ['a','b','c','d', 'e'])

series2 = pd.Series( {'firstName':'Charles',
'lastName':'Givre'} )
```



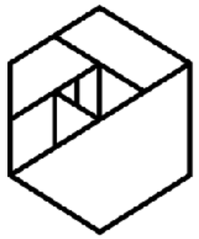
METIS

Accessing a Series

```
series1 = pd.Series( ['a','b','c','d', 'e'])
```

```
print( series1 )
```

```
0    a
1    b
2    c
3    d
4    e
dtype: object
```



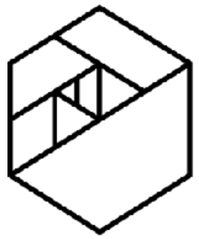
METIS

Accessing a Series

```
series1 = pd.Series( ['a','b','c','d', 'e'])
```

```
print( series1[3] )
```

```
d
```



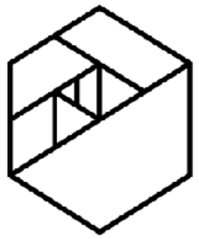
METIS

Accessing a Series

```
series1 = pd.Series( ['a','b','c','d', 'e'])
```

```
print( series1[1:3] )
```

```
1    b
2    c
dtype: object
```



METIS

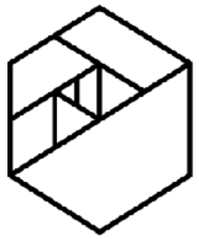
Accessing a Series

```
series1 = pd.Series( ['a','b','c','d', 'e'])
```

```
print( series1[1:3] )
```

```
1    b
2    c
dtype: object
```

```
series2 = series1[1:3]
series2.reset_index( drop=True )
```

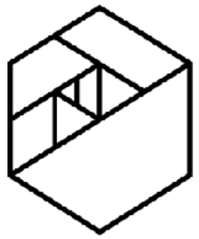


METIS

Accessing a Series

```
record = pd.Series(
    {‘firstname':'Charles',
     'lastname':'Givre',
    'middle':'classified' } )
```

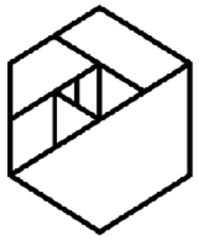
```
record['firstname']                    Charles
```



METIS

Accessing a Series

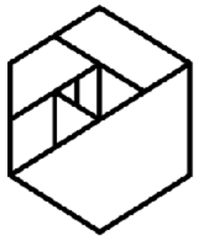
```
record = pd.Series(
    {‘firstname':'Charles',
     'lastname':'Givre',
    'middle':'classified' } )


record[['firstname', ‘lastname']]

firstname    Charles
lastname       Givre
dtype: object
```



METIS

Accessing a Series

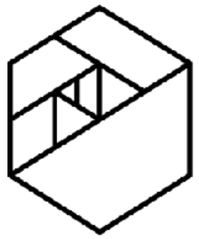
```
record = pd.Series(
   {‘firstname':'Charles',
    'lastname':'Givre',
   'middle':'classified' } )

record[0]

Charles
```



METIS

Accessing a Series

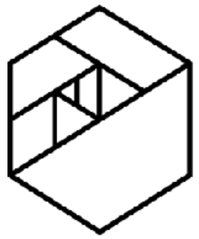
```
randomNumbers = pd.Series(
 np.random.random_integers(1, 100, 50) )

randomNumbers.head()
0     48
1     34
2     84
3     85
4     58
dtype: int64
```



METIS

Accessing a Series

```
randomNumbers = pd.Series(
 np.random.random_integers(1, 100, 50) )

randomNumbers.tail( 7 )
43    66
44    66
45    43
46    55
47    99
48    82
49    19
dtype: int64
```

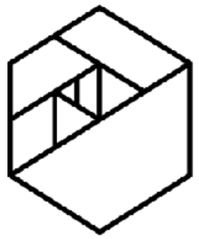


METIS

Filtering Data in a Series

```
randomNumbers [ <boolean condition> ]
randomNumbers [ randomNumbers < 10 ]
```

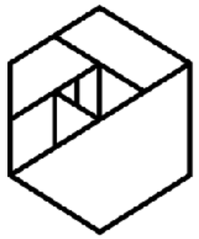
```
12    5
21    2
24    1
27    1
29    1
dtype: int64
```



METIS

Filtering Data in a Series

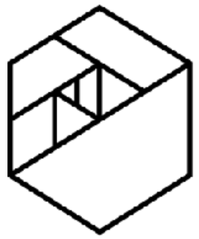
```
record.str.contains('Cha')
firstname      True
lastname      False
middle        False
dtype: bool

record[ record.str.contains('Cha') ]
firstname    Charles
dtype: object
```



METIS

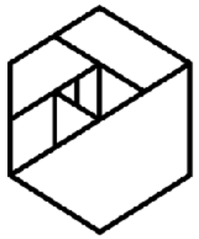
String Functions

Function	Explanation
Series.str.contains(<pattern>)	Returns true/false if text matches a pattern
Series.str.count(<pattern>)	Returns number of occurrences of a pattern in a string
Series.str.extract(<pattern>)	Returns matching groups from a string
Series.str.find(<string>)	Returns index of first occurrences of a substring (Note: not regex)
Series.str.findall(<pattern>)	Returns all occurrences of a regex
Series.str.len()	Returns the length of text
Series.str.replace(<pat>, <replace>)	Replaces matches with a replacement string



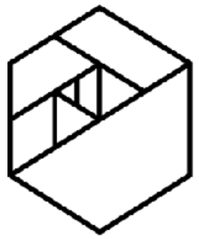
METIS

```
combinedSeries = series1.add( series2 )
```



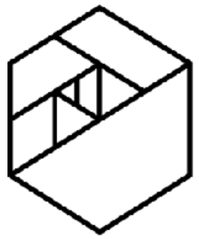
METIS

```
def addTwo( n ):
    return n + 2

odds.apply( addTwo )
```



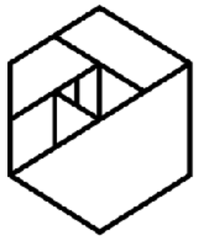
METIS

```
odds.apply( lambda x: x + 1 )
```



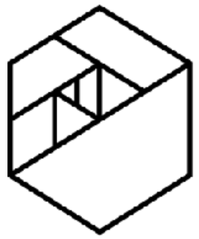
METIS

```
Series.dropna()
Series.fillna(value="<something>")
```



METIS

Questions?



METIS

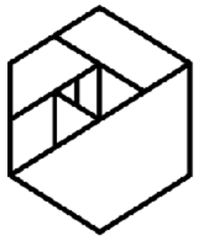
Exercise 1

```
email_series = pd.Series(emails)
filtered_emails =
email_series[email_series.str.contains('.edu')]
```

```
2           mdixon2@cmu.edu
11      jjonesb@arizona.edu
15     eberryf@brandeis.edu
```

```
accounts = filtered_emails.str.split( '@').str[0]
```

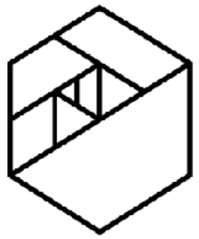
```
2      mdixon2
11     jjonesb
15     eberryf
dtype: object
```



METIS

Exercise 2

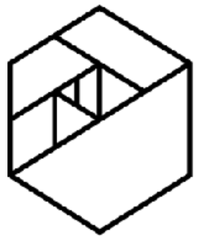
```
pounds = pd.Series( weights )
kilos = pounds.apply( poundsToKilograms )
```



METIS

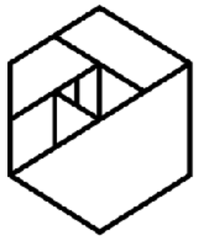
Exercise 3

```
IPData = pd.Series( hosts )
privateIPs = IPData[
   IPData.apply(
    lambda x : ip_address(x).is_private
    ) ]
```



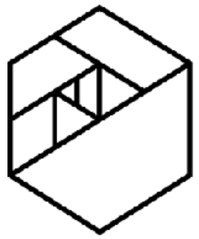
METIS

The Data Frame



METIS

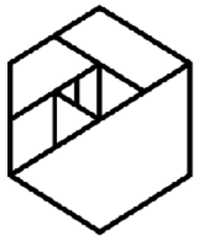
```
data = pd.DataFrame( <data>, <index>, <column_names> )
```



METIS

CSV

```
data = pd.read_csv( <file> )
```

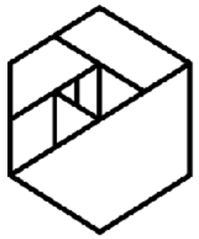


METIS

DataFrames

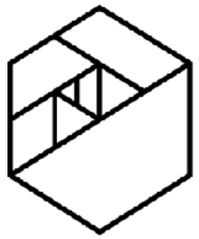
Excel

```
data = pd.read_excel( <file>, sheetname=<sheetname> )
```



METIS


```
[
  {
   “changed":"2015-04-0300:00:00",
   "created":"2014-04-10 00:00:00”,
   “dnssec":"False",
   "expires":"2016-04-10 00:00:00”,
   “isMalicious":0,
   “url":"nuteczki.com"
  },
…
]
```



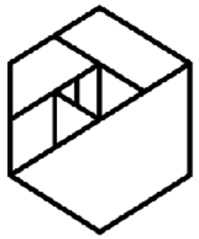
METIS

JSON

```
data = pd.read_json( <file> )
```

or

```
data = pd.read_json( <url> )
```



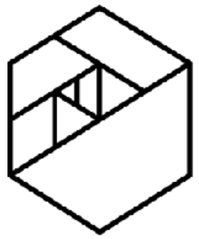
METIS

From a Database

```
data = pd.read_sql( <query>, <connection> )
```

METIS

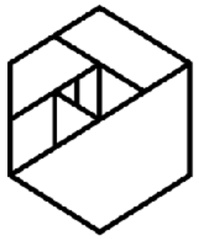
Nested Data?



METIS

Nested Data?

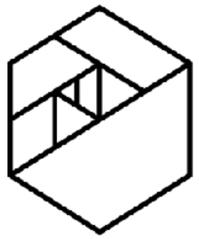

```
{"time": 1084443427.311224,
 "timestamp": "2004-05-13T10:17:07.311224",
"IP": {
   "version": 4,
   "ttl": 128,
   "proto": 6,
   "options": [],
   "len": 48,
   "dst": "65.208.228.223",
   "frag": 0,
   "flags": 2, "src": “145.254.160.237",
   "chksum": 37355
   },
   "Ethernet": {"src": "00:00:01:00:00:00", "type": 2048, "dst": “fe:ff:20:00:01:00"},
   …
```



METIS

Nested Data

```
pd.read_json( ‘nested_data.json’ )
```

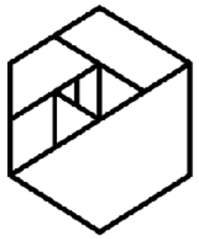


METIS

Nested Data

```
pd.read_json( 'nested_data.json' )
```

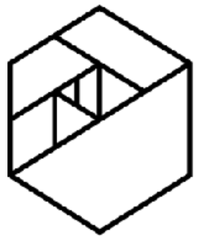
	DNS	Ethernet	IP	TCP	UDP	time	timestamp
0	NaN	{'type': 2048, 'dst': 'fe:ff:20:00:01:00', 'sr...	{'dst': '65.208.228.223', 'len': 48, 'version'...	{'window': 8760, 'chksum': 49932, 'sport': 337...	NaN	1.084443e+09	2004-05-13 10:17:07.311224
1	NaN	{'type': 2048, 'dst': '00:00:01:00:00:00', 'sr...	{'dst': '145.254.160.237', 'len': 48, 'version...	{'window': 5840, 'chksum': 23516, 'sport': 80,...	NaN	1.084443e+09	2004-05-13 10:17:08.222534
2	NaN	{'type': 2048, 'dst': 'fe:ff:20:00:01:00', 'sr...	{'dst': '65.208.228.223', 'len': 40, 'version'...	{'window': 9660, 'chksum': 31076, 'sport': 337...	NaN	1.084443e+09	2004-05-13 10:17:08.222534
3	NaN	{'type': 2048, 'dst': 'fe:ff:20:00:01:00', 'sr...	{'dst': '65.208.228.223', 'len': 519, 'version...	{'window': 9660, 'chksum': 43352, 'sport': 337...	NaN	1.084443e+09	2004-05-13 10:17:08.222534
4	NaN	{'type': 2048, 'dst': '00:00:01:00:00:00', 'sr...	{'dst': '145.254.160.237', 'len': 40, 'version...	{'window': 6432, 'chksum': 33825, 'sport': 80,...	NaN	1.084443e+09	2004-05-13 10:17:08.783340



METIS

Nested Data

```
from pandas.io.json import json_normalize
import json
import pandas as pd

with open('nested.json') as data_file:
    pcap_data = json.load(data_file)

df = pd.DataFrame( json_normalize(pcap_data) )
```

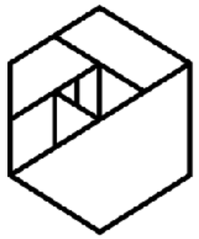


METIS

Nested Data

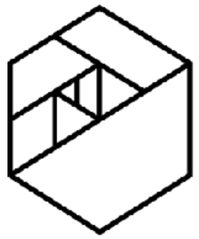
```
df = pd.DataFrame( json_normalize(pcap_data) )
```

...	TCP.seq	TCP.sport	TCP.urgptr	TCP.window	L
...	951057939.0	3372.0	0.0	8760.0	
...	290218379.0	80.0	0.0	5840.0	
...	951057940.0	3372.0	0.0	9660.0	
...	951057940.0	3372.0	0.0	9660.0	
...	290218380.0	80.0	0.0	6432.0	



METIS

Manipulating a DataFrame

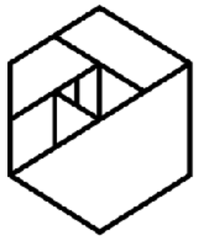


METIS

```
df = data['column1']
```

Returns a series

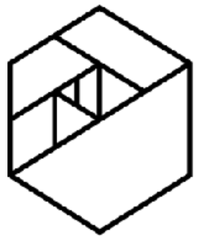
```
df['ip'].value_counts().head()
```



METIS

```
df = data[['column1','column2','column3']]
```

Returns a DataFrame



METIS

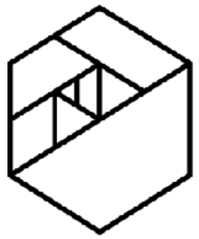
Filtering a DataFrame

```
df[<boolean condition>]
```

```
df[['col1','col2']][col3 > 5]
```

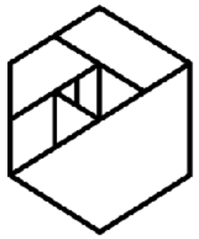
Columns

Filter



METIS

```
data.loc[<index>]
data.loc[<list of indexes>]
data.sample(<n>)
```

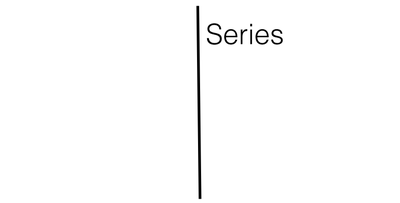


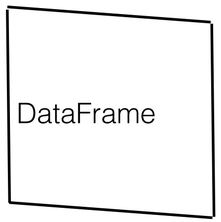
METIS

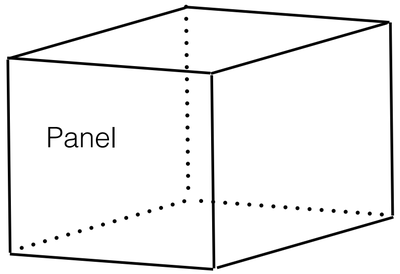
`data.apply( <function> )`

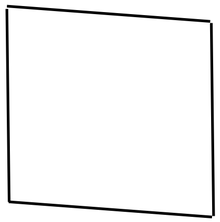
IF
Call Apply On...

THEN
Function
Receives









METIS

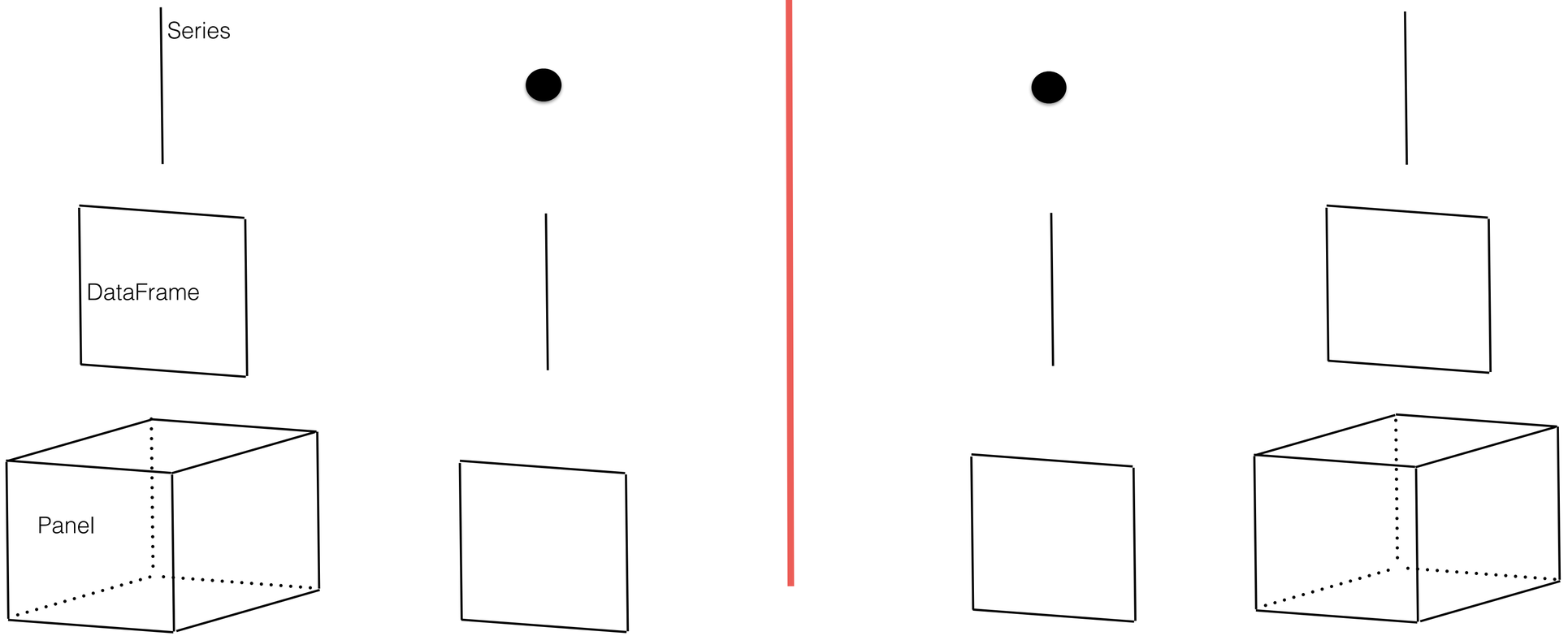
`data.apply( <function> )`

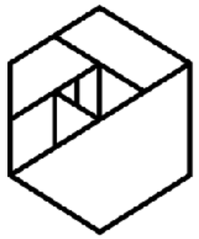
IF
Call Apply On...

THEN
Function
Receives

IF
Function
Returns

THEN
Apply
Returns...



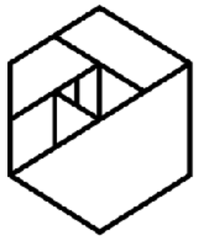


METIS

1	2	3
4	5	6
7	8	9

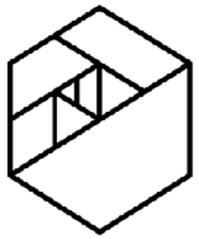
`data.T`

1	4	7
2	5	8
3	6	9



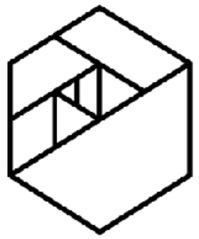
METIS

```
#Gets you the sum of columns
data.sum( axis=0 )

#Gets you the sum of the rows
data.sum( axis=1 )
```



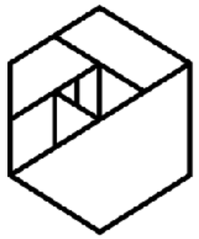
METIS

```
DataFrame.drop(labels,
 axis=0,
 level=None,
 inplace=False,
 errors='raise')¶
```



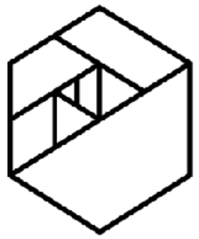
METIS

Merging Data Sets



METIS

```
combinedSeries = pd.concat(
[series1, series2],
ignore_index=True )
```

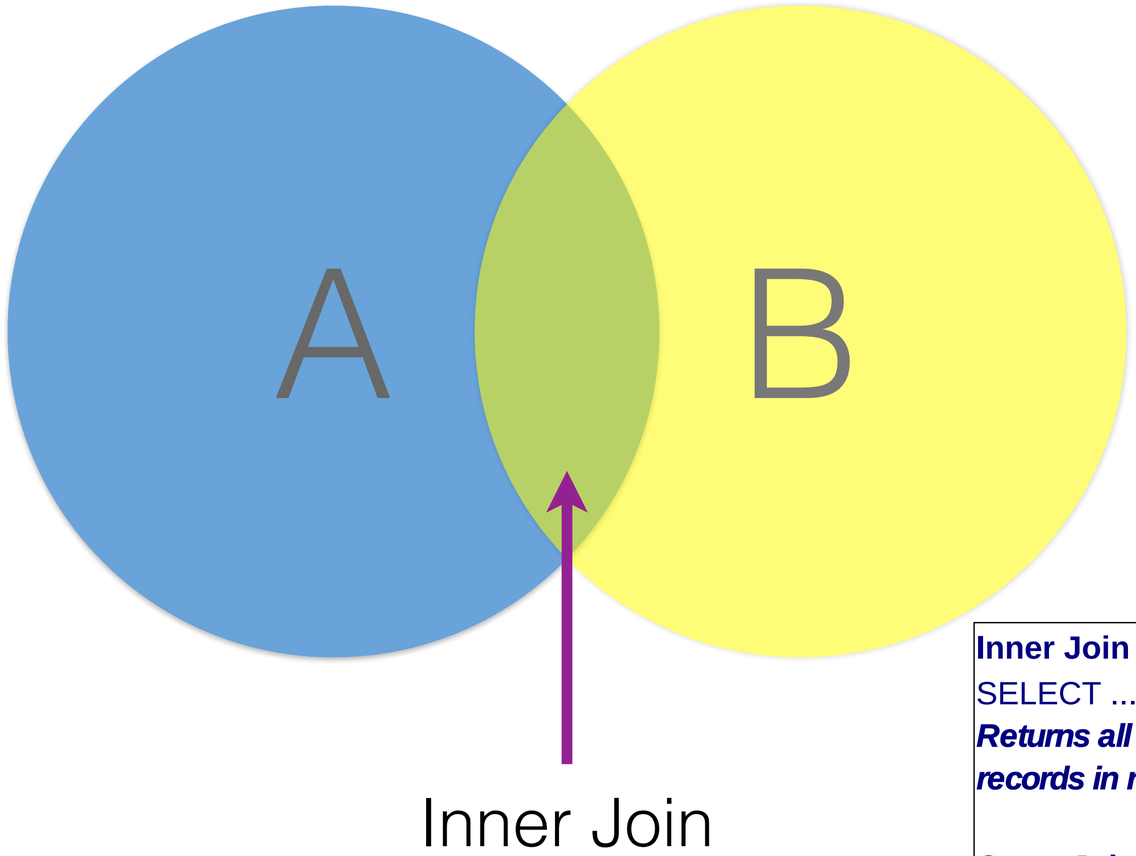


METIS

Joins

METIS

Inner Join (Intersection)



Inner Join

SELECT ... FROM a INNER JOIN b ON a.key = b.key;

Returns all records from left hand table a that have matching records in right hand table b.

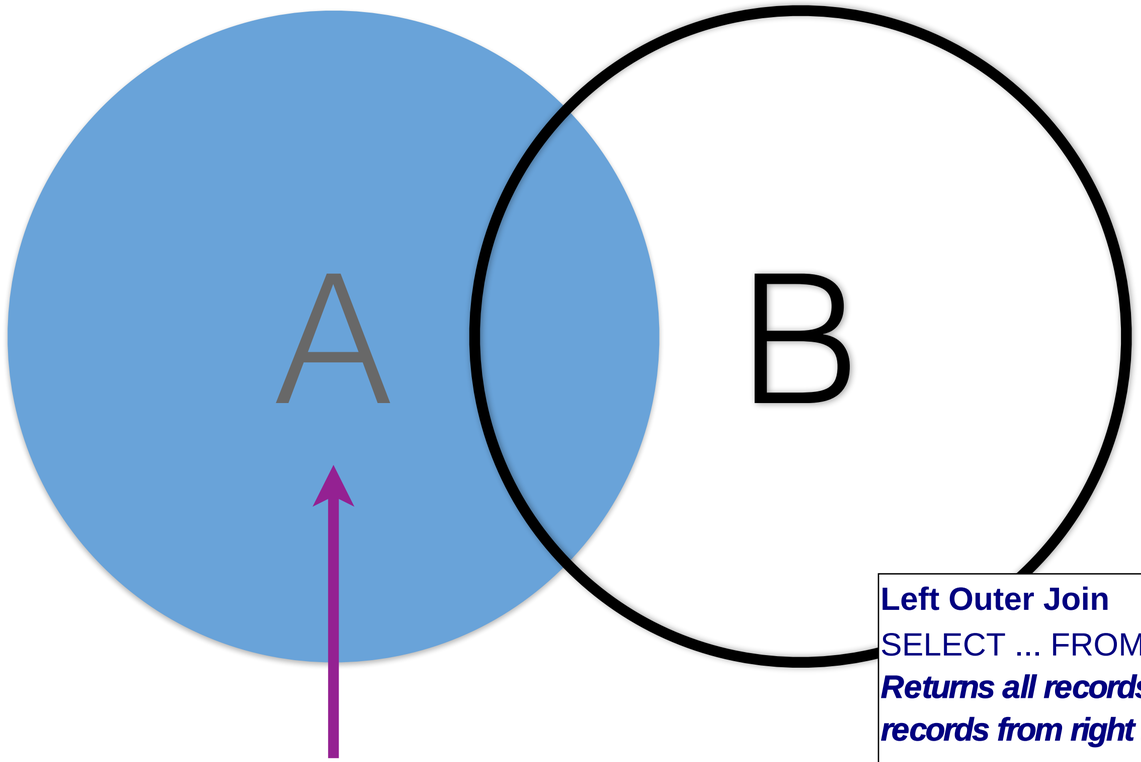
Outer Join

SELECT ... FROM a OUTER JOIN b ON a.key = b.key;

Returns all records from left hand table a, and from right hand table b, joining those records that match.

METIS

Left Join



Left Outer Join

SELECT ... FROM a LEFT OUTER JOIN b ON a.key = b.key;

Returns all records from left hand table a, including/joining those records from right hand table b that match records in table a.

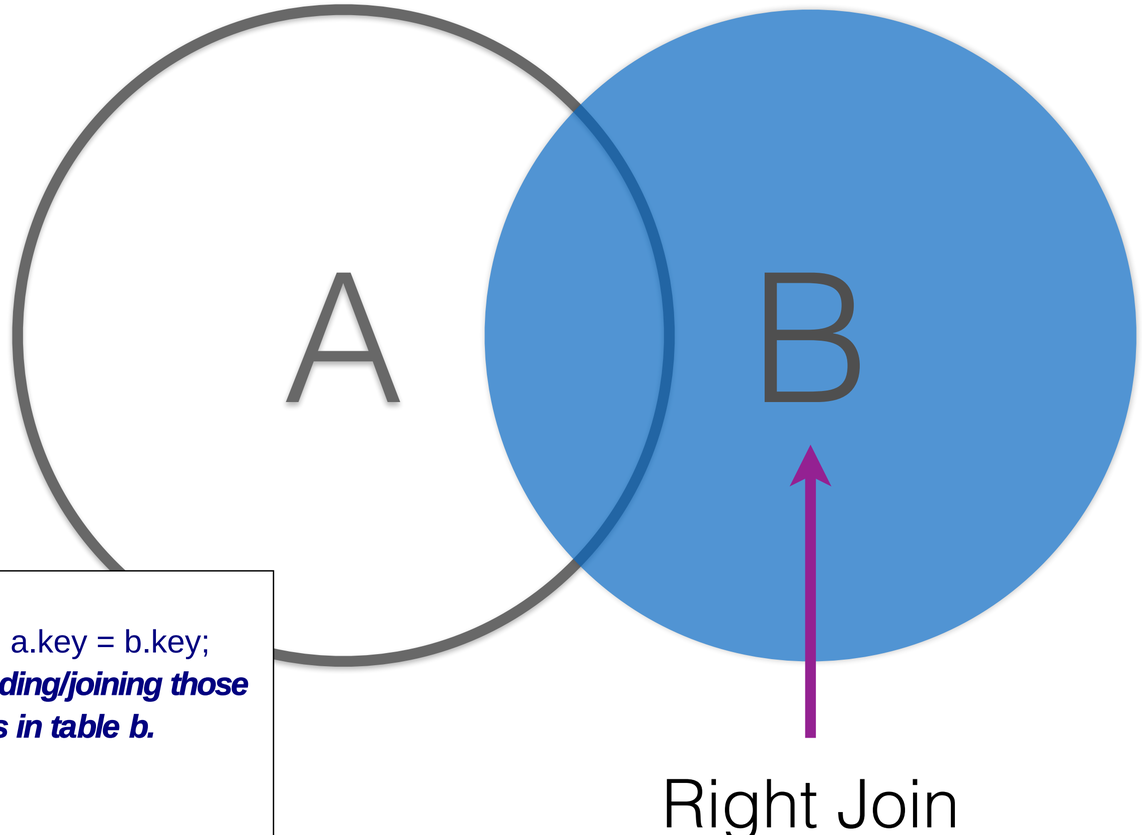
Left Exclusive Join

SELECT ... FROM a LEFT OUTER JOIN b ON a.key = b.key
WHERE b.key IS NULL;

Returns all records from left hand table a that do not match records from right hand table b.

METIS

Right Join



Right Outer Join

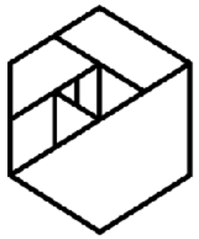
SELECT ... FROM a RIGHT OUTER JOIN b ON a.key = b.key;

Returns all records from right hand table b, including/joining those records from left hand table a that match records in table b.

Right Exclusive Join

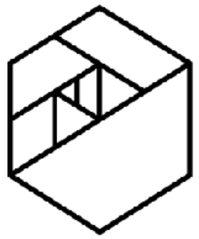
SELECT ... FROM a RIGHT OUTER JOIN b ON a.key = b.key
WHERE a.key IS NULL;

Returns all records from right hand table b that do not match records from left hand table a.



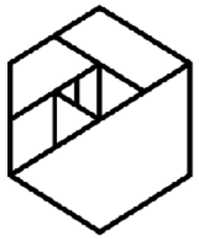
METIS

```
pd.merge( leftData, rightData )
```



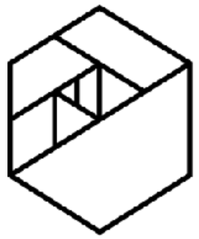
METIS

```
pd.merge( leftData, rightData, how="<join type>" )
```



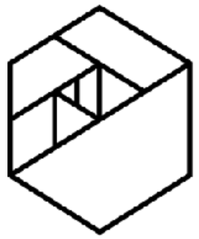
METIS

```
pd.merge( leftData, rightData,
how="<join type>",
on=<field list> )
```



METIS

Union

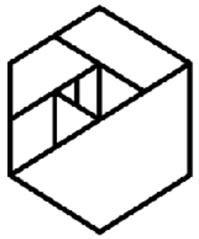


METIS

Union

Series 1

Index	Value
0	6
1	4
2	2
3	3



METIS

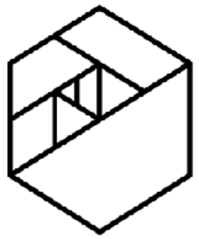
Union

Series 1

Index	Value
0	6
1	4
2	2
3	3

Series 2

Index	Value
0	7
1	5
2	3
3	4



METIS

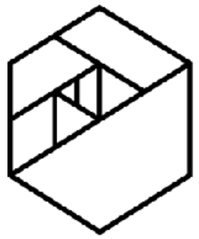
Union

combinedSeries

Index	Value
0	6
1	4
2	2
3	3
4	7
5	5
6	3
7	4

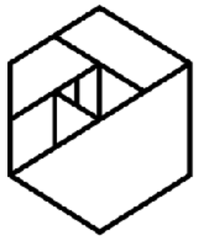
Joins combine results horizontally, i.e. return a group of columns arranged into a row
[col col ... col]

By contrast, set operations combine results vertically, i.e. return a group of rows arranged into a column.
[row
  row
  ...
  row]



METIS

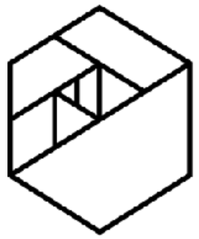
Grouping and Aggregating Data



METIS

```
df_grouped = df.groupby(
['Protocol', 'Source','Destination'])
print(df_grouped.size())
```

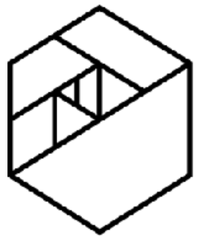
```
Protocol  Source              Destination
ARP       MoxaTech_27:8c:37   Westermo_1a:61:83      610
          Pegatron_3a:0d:e8   Ruggedco_64:85:c2      871
          Ruggedco_64:85:c2   Pegatron_3a:0d:e8      871
          SiemensN_27:64:11   Siemens-_89:59:82      428
          Westermo_1a:61:83   MoxaTech_27:8c:37      610
DNS       192.168.88.1        192.168.88.61         9938
          192.168.88.61       192.168.88.1          9937
          192.168.89.2        8.8.8.8               2194
ICMP      192.168.89.1        192.168.89.2          2194
LOOP      CiscoInc_95:1d:8b   CiscoInc_95:1d:8b     1992
S7COMM    10.10.10.10         10.10.10.20          19913
          10.10.10.20         10.10.10.10          19914
TCP       10.10.10.10         10.10.10.20          23409
          10.10.10.20         10.10.10.10          59739
Name: Protocol, dtype: int64
```



METIS

```
Stats_Packets = df_grouped['Length'].agg({'No
Packets': len, 'Volume': sum,\
        'SD': lambda x: np.std(x, ddof=1) if
len(x)>2 else 0})
```

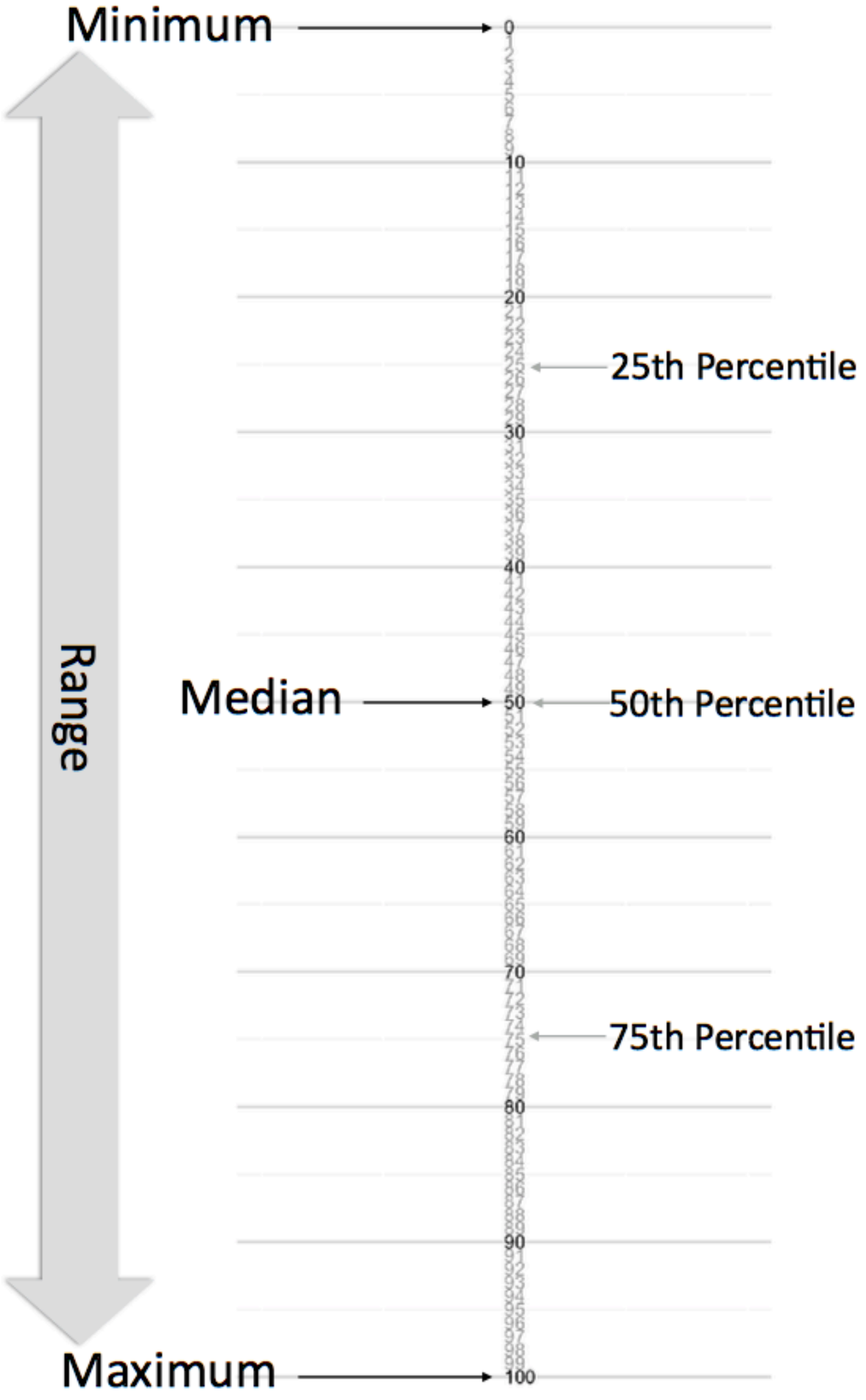
Protocol	Source	Destination	Volume	SD	No Packets
ARP	MoxaTech_27:8c:37	Westermo_1a:61:83	45140	0.000000	610
	Pegatron_3a:0d:e8	Ruggedco_64:85:c2	52260	0.000000	871
	Ruggedco_64:85:c2	Pegatron_3a:0d:e8	52260	0.000000	871
	SiemensN_27:64:11	Siemens-_89:59:82	25680	0.000000	428
	Westermo_1a:61:83	MoxaTech_27:8c:37	36600	0.000000	610
DNS	192.168.88.1	192.168.88.61	725474	0.000000	9938
	192.168.88.61	192.168.88.1	725401	0.000000	9937
	192.168.89.2	8.8.8.8	152391	1.442727	2194
ICMP	192.168.89.1	192.168.89.2	213823	1.442727	2194
LOOP	CiscoInc_95:1d:8b	CiscoInc_95:1d:8b	119520	0.000000	1992
S7COMM	10.10.10.10	10.10.10.20	2070952	0.000000	19913
	10.10.10.20	10.10.10.10	3046842	0.000000	19914
TCP	10.10.10.10	10.10.10.20	2280712	15.683145	23409
	10.10.10.20	10.10.10.10	5436249	43.840987	59739

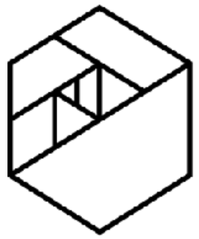


METIS

Questions?

Explore Your Data


Minimum
0
10
20
25th Percentile
30
40
Median
50
50th Percentile
60
70
75th Percentile
80
90
Maximum
100
Range



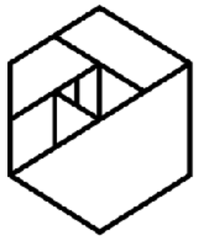
METIS

`Series.abs()`	Absolute Value of the Series
`Series.count()`	Returns number of non-empty values in the series
`Series.max()`	Returns maximum value in the Series
`Series.mean()`	Returns the mean of a Series
`Series.median()`	Returns the median of a Series
`Series.min()`	Returns the minimum value in a Series
`Series.mode()`	Take a guess..
`Series.quantile([q])`	Returns the quantiles of a Series
`Series.sum`	Returns the sum of a series
`Series.std`	Returns the standard deviation of a Series

METIS

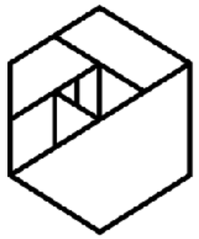
Tukey 5 Number Summary

- **Minimum:** The smallest value in the dataset
- **Lower Quartile:** Smallest 25% of the dataset
- **The Median:** The middle value of the dataset
- **Upper Quartile:** The largest 25% of the dataset
- **Maximum:** The largest value in the dataset



METIS

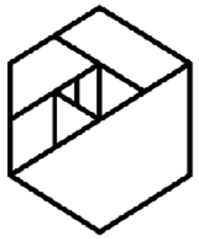
```
Series.describe()
```



METIS

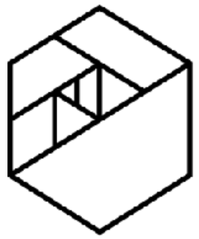
Series.describe()

```
>>> random_numbers.describe()
count     50.000000
mean      50.620000
std       30.102471
min        1.000000
25%       25.500000
50%       54.000000
75%       73.000000
max       99.000000
dtype: float64
```



METIS

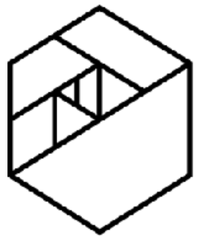
```
names = pd.Series(
 ['Jim','Bob',
 'Bob','Steve','Jim,'Jane','Steph','Emma','Rachel'])

names.describe()
```



METIS

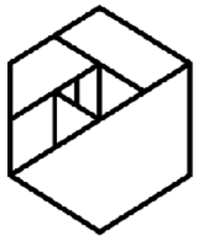
```
names = pd.Series(
 ['Jim','Bob',
 'Bob','Steve','Jim,'Jane','Steph','Emma','Rachel'])

names.describe()
```

```
count        9
unique       7
top        Jim
freq         2
dtype: object
```



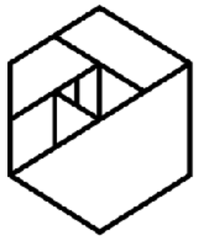
METIS

Finding Unique Values



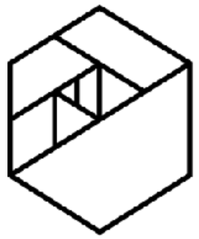
METIS

```
unique_nums = []
for key, value in random_numbers.iteritems():
    if value not in unique_nums:
        unique_nums.append(value)
```



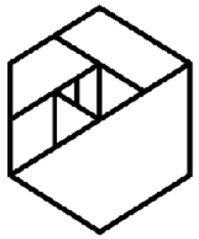
METIS

```
series.drop_duplicates()
series.unique()
```



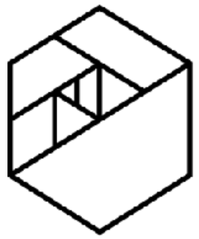
METIS

Finding Unique Values



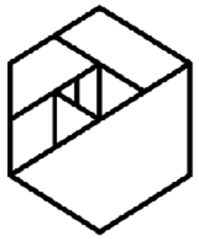
METIS

```
series.value_counts()
```



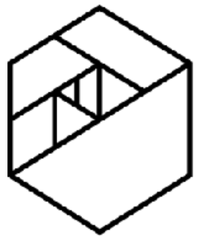
METIS

```
series.    value_counts(    bins=5  )
```



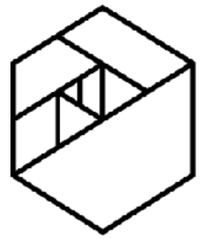
METIS

```
series.value_counts(bins=5,
                    normalize=True)
```



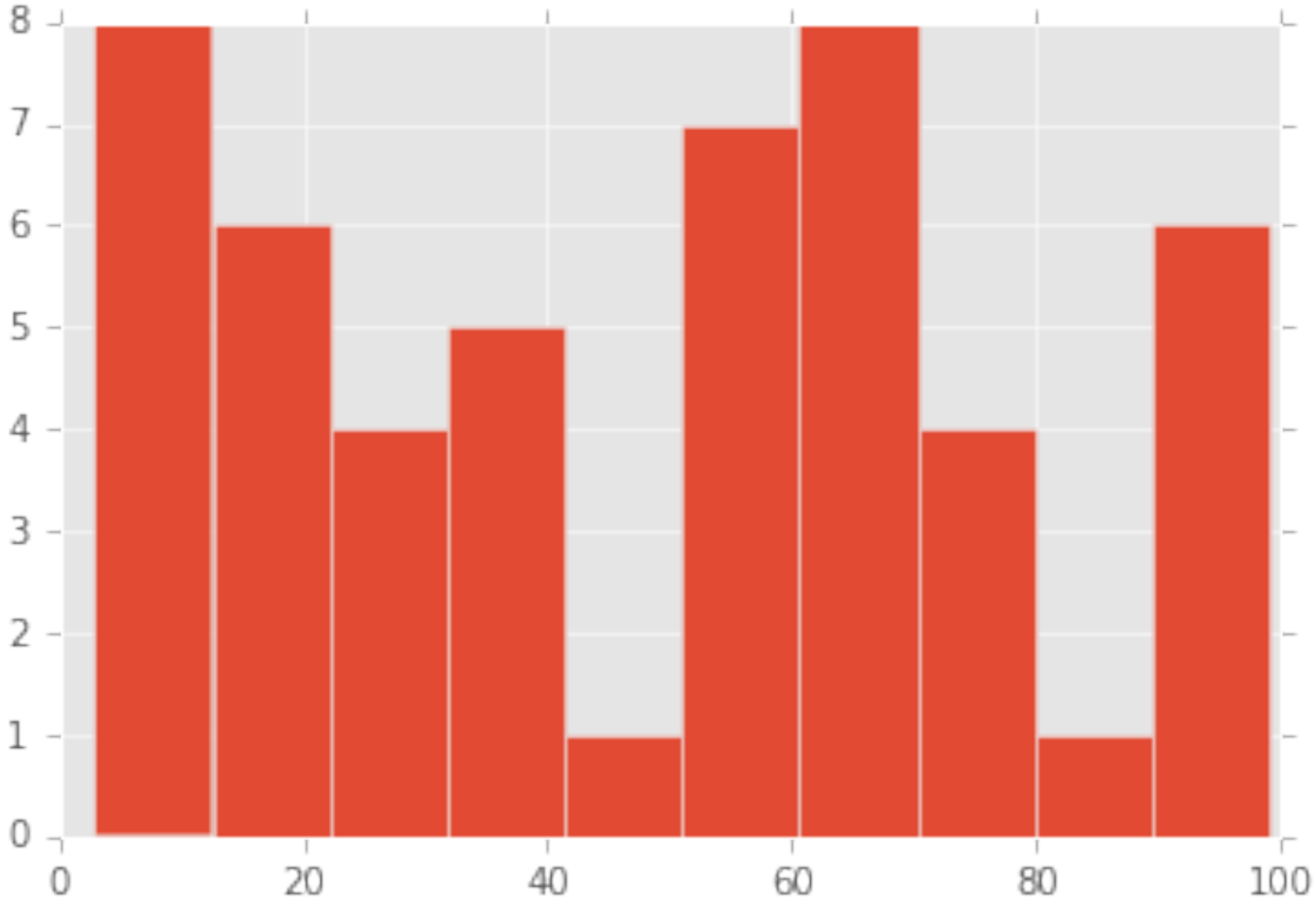
METIS

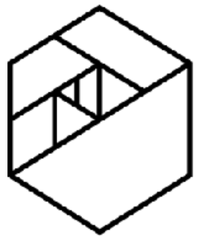
```
series.hist()
```



METIS

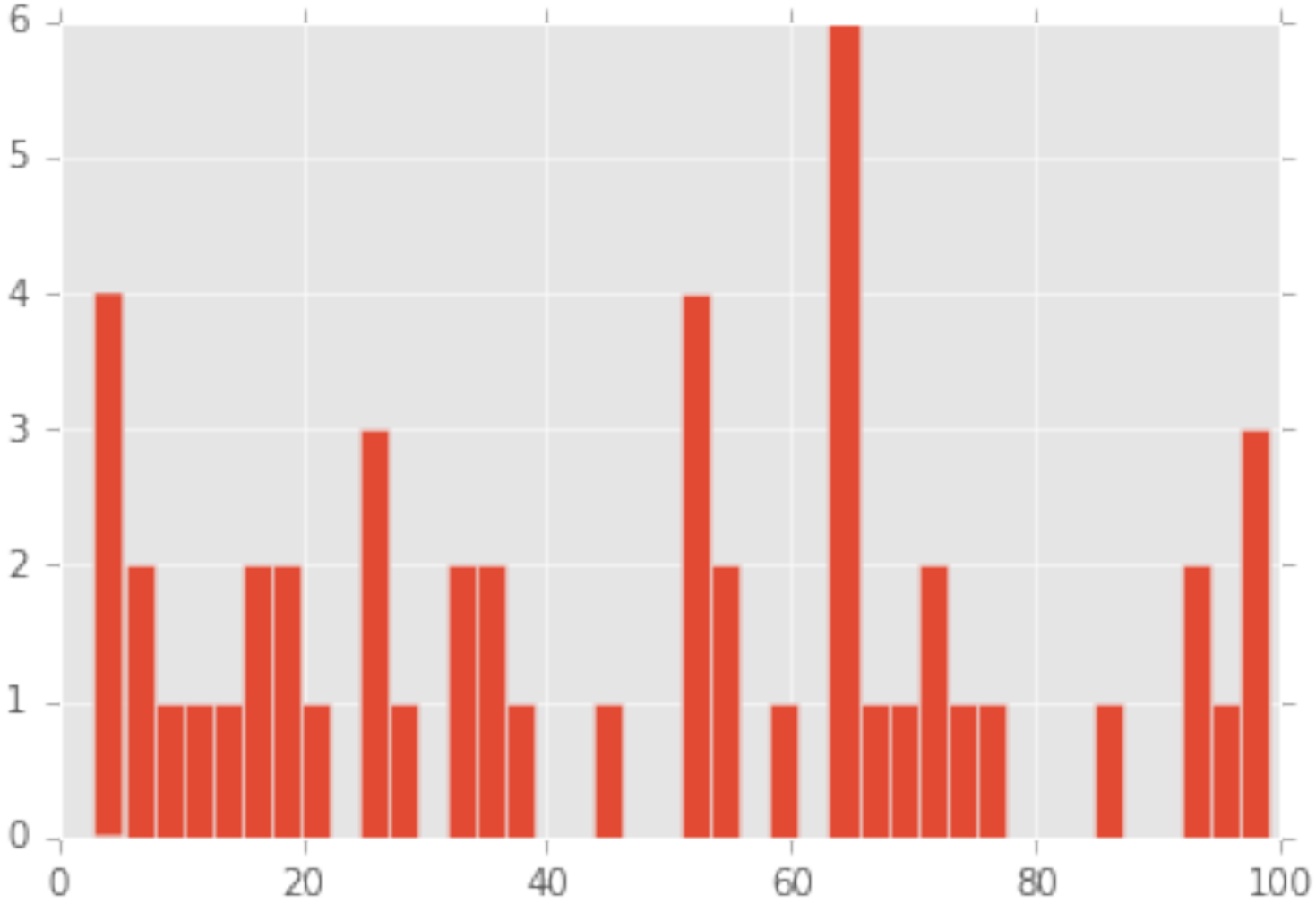
series.hist()

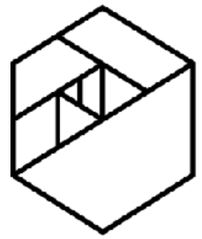




METIS

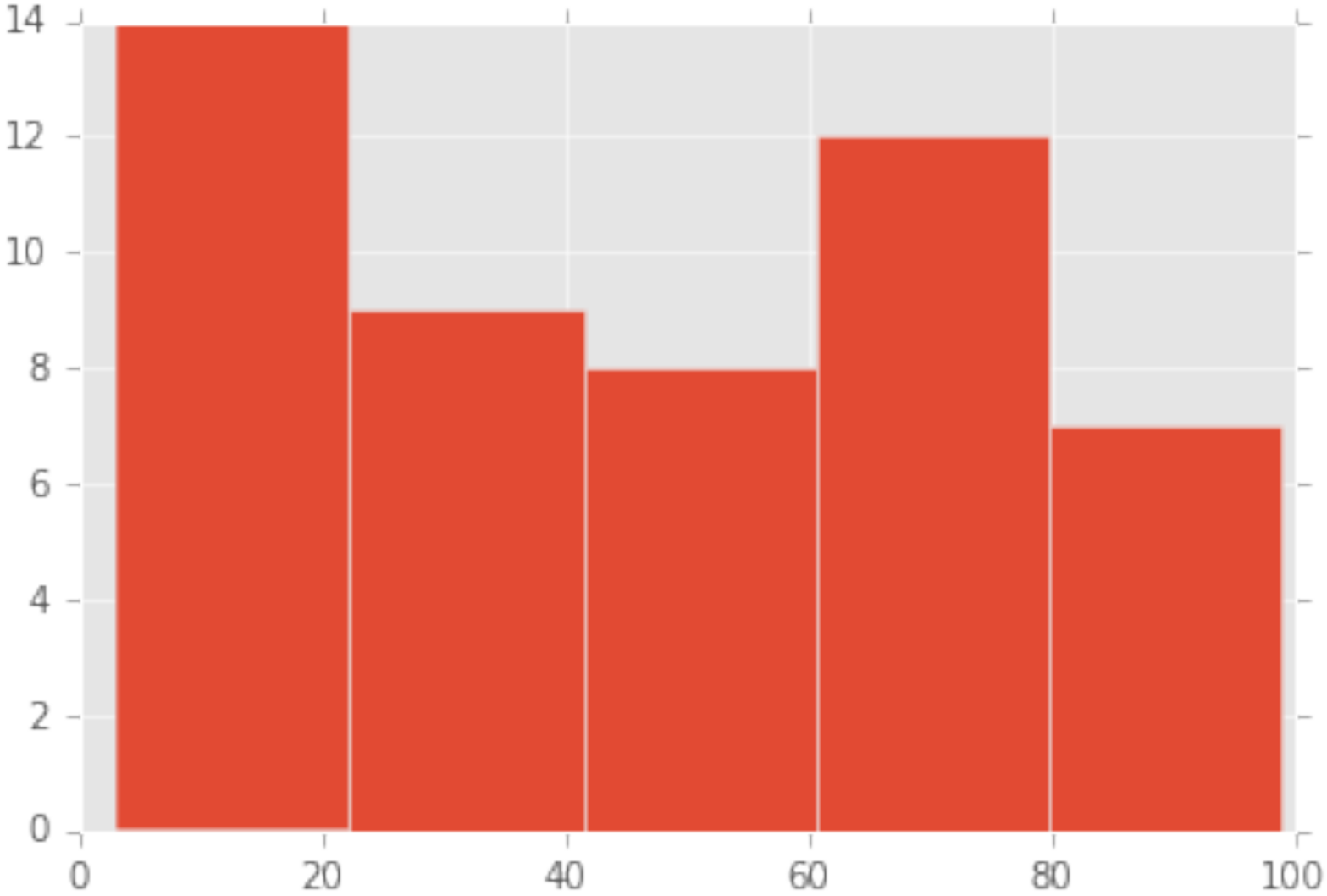
series.hist(bins=40)

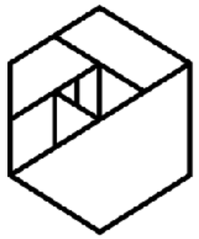




METIS

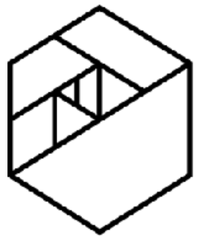
series.hist(bins=5)





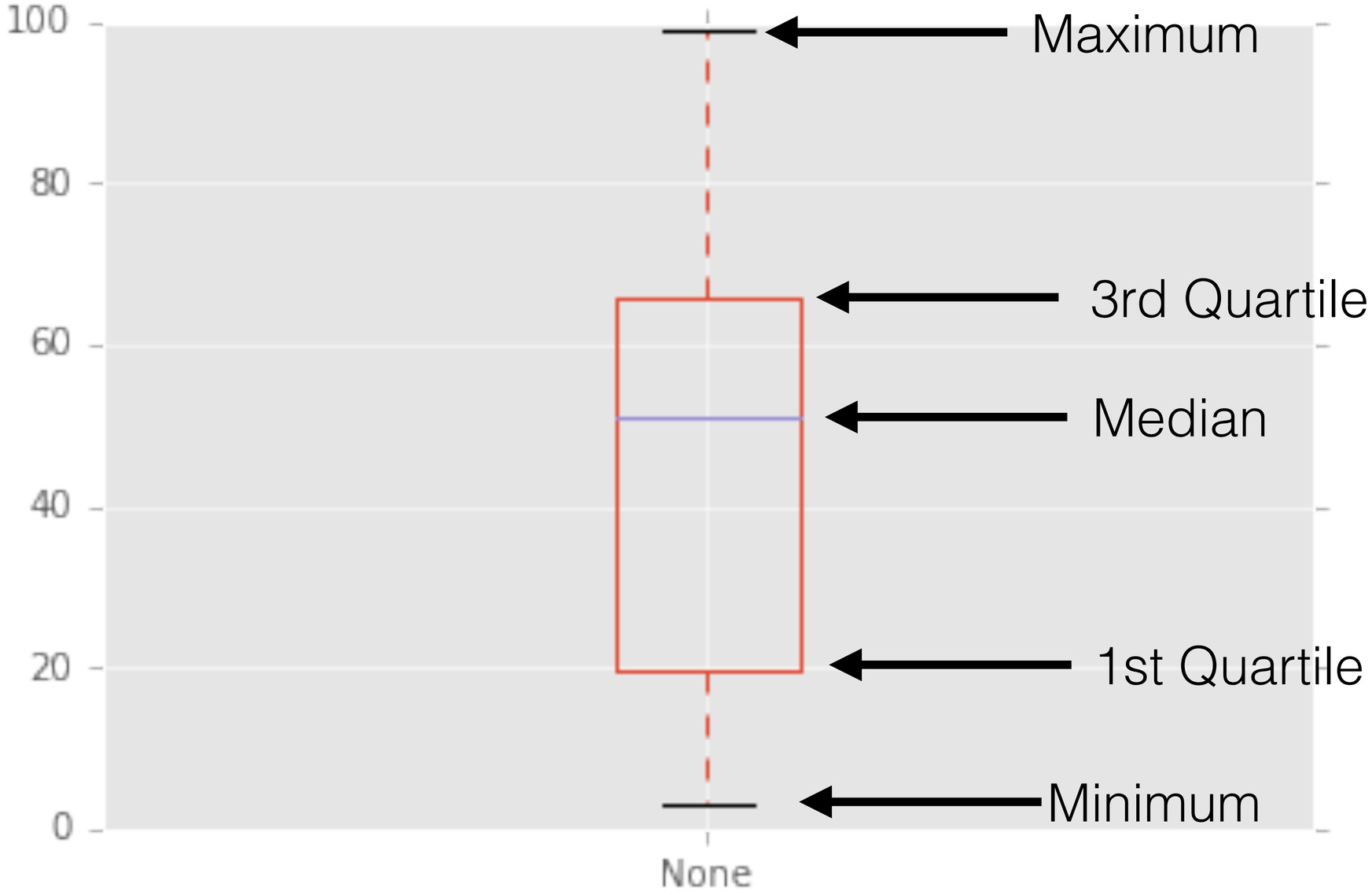
METIS

series.plot()



METIS

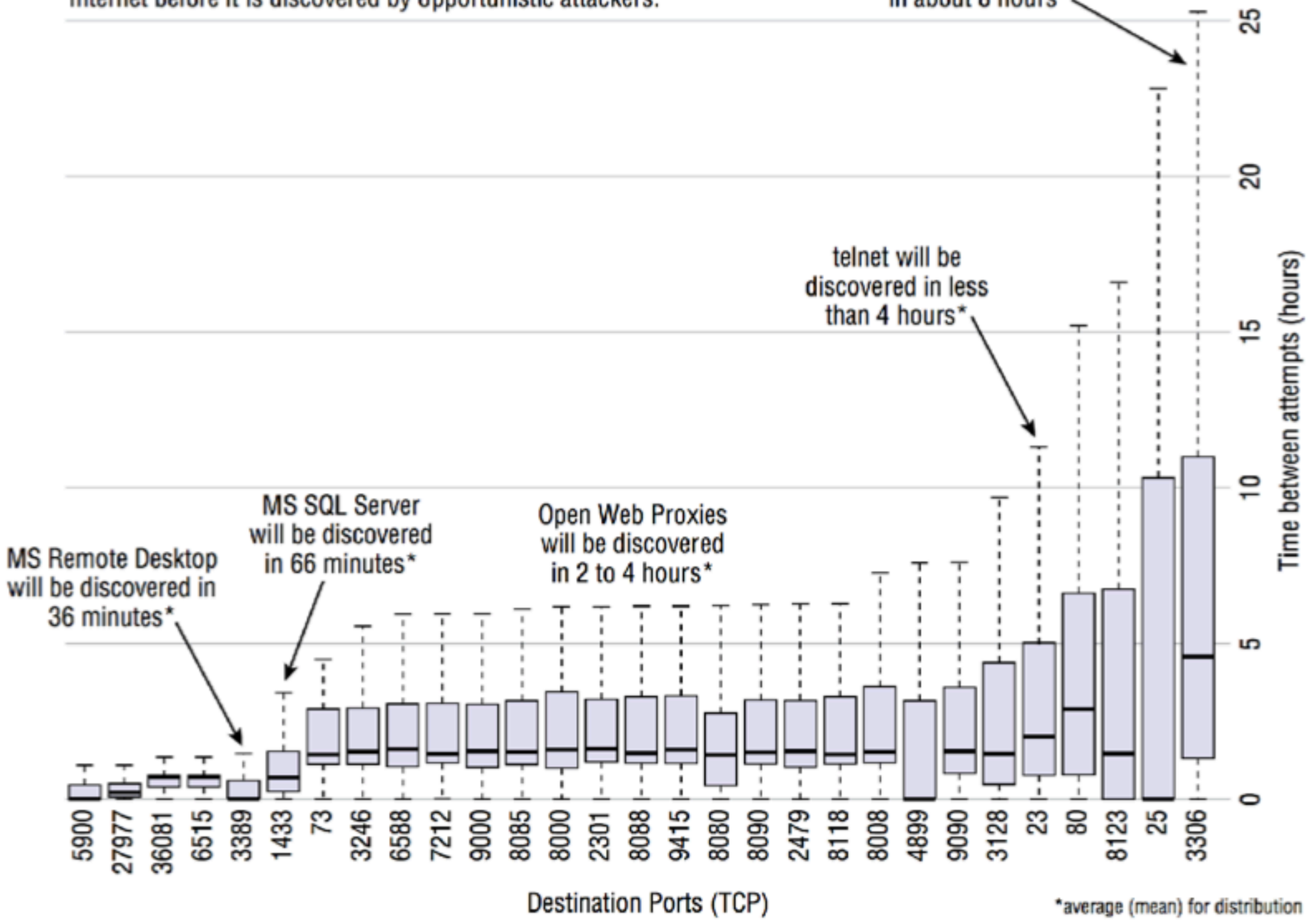
series.plot(kind='box')

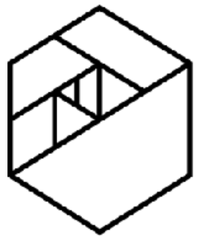


ME

How long will a service go undiscovered by opportunistic attackers?

By measuring the time between packets received, it is possible to estimate how long a service may be on the Internet before it is discovered by opportunistic attackers.



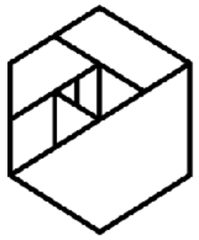


METIS

Correlations between Datasets

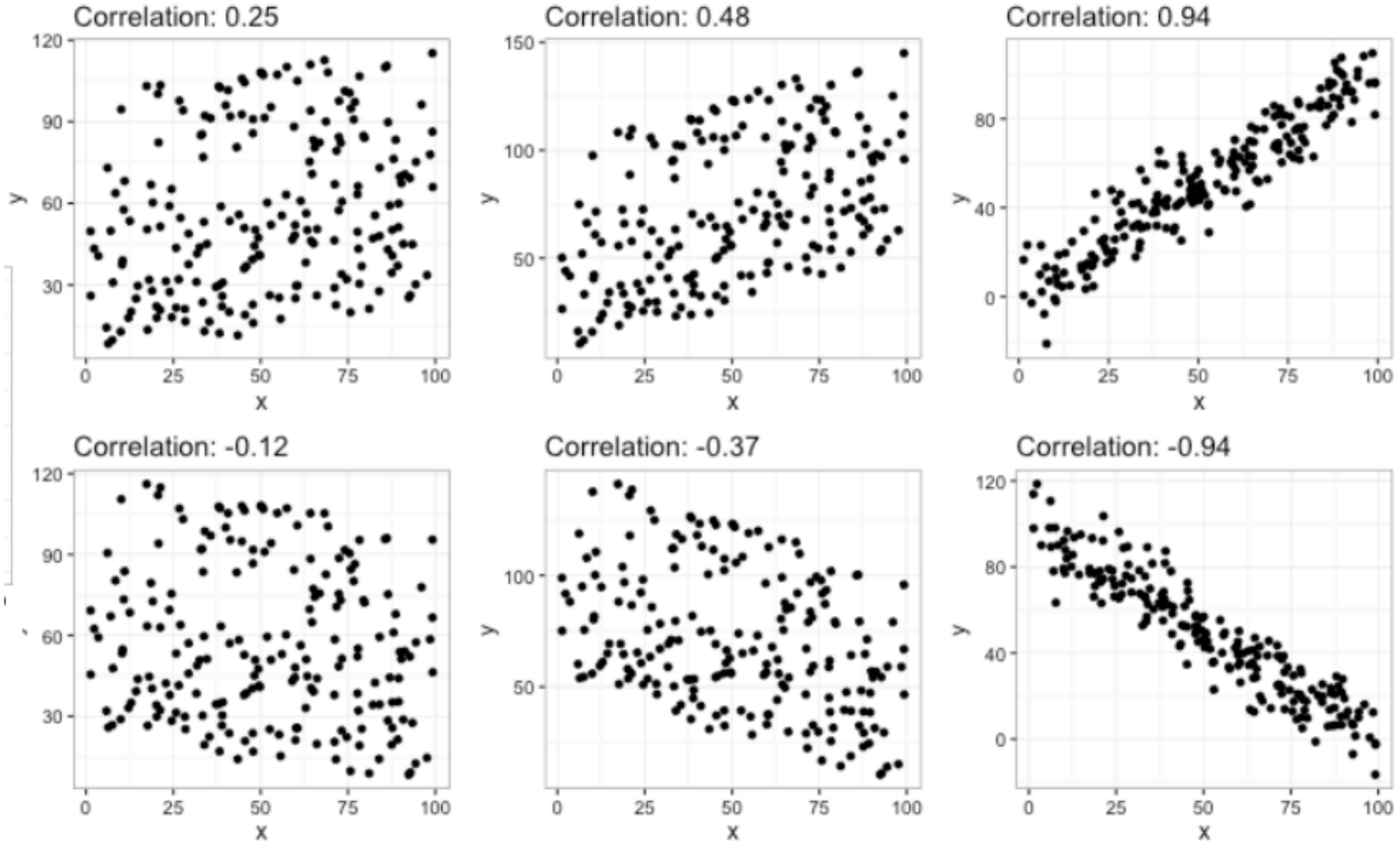
Correlations

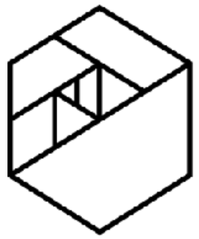
- Measurement of the relationship between **two continuous variables**
- Output is between -1 and 1, with 1 being perfect correlation, -1 is perfect negative correlation, 0 is no correlation.
- Correlation measurement is referred to as r.



METIS

Correlations

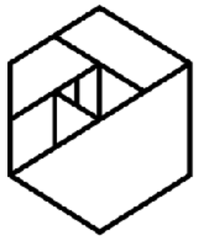




METIS

Correlations

```
series1.corr( series2 )
```



METIS

Case Study 1:

Please open ratings.dat, the MovieLens 1M dataset.

1. Everything is in a single column (so we can't separately look at ratings, user ids,movie ids, or timestamps)!

2. The first row is used as the name of the only column (we call this the **header**), which is no good, as the first record shouldn't be the header, but an actual record.

3. The timestamp is in a format that doesn't really tell us anything useful about when the ratings occurred.