CONTINUING EDUCATION CENTRE

Data Analytics and Machine Learning Using Python

Data Structure and Analysis Using Pandas

CONTINUING EDUCATION CENTRE

Outline

- Basic functionalities of Pandas (Series and Frame)
- Data manipulation (indexing, slicing, sorting, and grouping)
- Data analytic and statistic
- Big Data Infrastructure (NoSQL, Hadoop, Spark, dsb)

CONTINUING EDUCATION CENTRE

Intro

- Pandas is an open-source Python library providing high-performance, easy-to-use data structures and data analysis tools
- Main features:
  - Fast and efficient DataFrame object
  - Loading data from many different file formats
  - Easy data alignment and handling
  - Label-based slicing, indexing and grouping
  - Time series functionality

Data Structure

- Pandas has two important data structures:
  - Series
  - DataFrame

Data Structure	Description
Series	1D, homogeneous array, sizeimmutable.
DataFrame	2D, size-mutable, heterogeneously columns.

- Additionally, a “Panel” data structure can be created on top of DataFrame
- These data structures are built on top of Numpy array

CONTINUING EDUCATION CENTRE

Data Structure::Series

```
In [1]: import pandas as pd
In [2]: S1 = pd.Series([1,3,5])
In [3]: S1
Out[3]:
0    1
1    3
2    5
dtype: int64
In [4]: S1.index
Out[4]: RangeIndex(start=0, stop=6, step=1)
```

CONTINUING EDUCATION CENTRE

Data Structure::Series

```
In [2]: import numpy as np

In [3]: S2 = pd.Series(np.random.randint(1,100,10))

In [4]: S2

Out[4]:
0    28
1    87
2    58
3    26
4    54
5    63
6    85
7    30
8    85
9    67
dtype: int32
```

CONTINUING EDUCATION CENTRE

Data Structure::Series

```
In [5]: jam_kerja =
pd.Series([7,7,6,8,5,3,0],index=['senin','selasa','rabu','kamis','jumat','sabtu','minggu'])


In [6]: jam_kerja
Out[6]:
senin     7
selasa    7
rabu      6
kamis     8
jumat     5
sabtu     3
minggu    0
dtype: int64
```

CONTINUING EDUCATION CENTRE

Data Structure::Series

#Importing from standard python list

In [3]: S3 = pd.Series(range(1,100))

#Importing from standard python dictionary

In [4]: data = {'a' : 0., 'b' : 1., 'c' : 2.}

In [5]: S4 = pd.Series(data)

In [6]: S4

Out[6]:

a   0.0

b   1.0

c   2.0

dtype: float64

Data Structure::DataFrame

```
In [1]: import pandas as pd
In [2]: daftar = [['Adi',10],['Budi',12],['Cica',13]]
In [3]: umur = pd.DataFrame(daftar,columns=['Nama', 'Umur'])
In [4]: umur
Out[4]:
   Nama  Umur
0   Adi    10
1  Budi    12
2  Cica    13
```

Data Structure::DataFrame

```
In [1]: import pandas as pd
In [2]: daftar = {‘Nama’:[‘Adi', ‘Budi', ‘Cica', ‘Danu‘,’Edwin’], ’Umur’:[10,12,13,12,14]}
In [3]: umur = pd.DataFrame(daftar)


# Adding and removing a new column
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
df['three']=pd.Series([10,20,30],index=['a','b','c'])
del df['two']
```

CONTINUING EDUCATION CENTRE

Data Manipulation::Indexing

- Series or DataFrame can be indexed using numeric or key, depends on how they were created:

```
In [2]: daftar = {'Nama':['Adi', 'Budi', 'Cica', 'Danu','Edwin'],
'Umur':[10,12,13,12,14]}
In [3]: umur = pd.DataFrame(daftar)
In [4]: anak2 = umur['Nama'][1]
```

- Note: Columns can be selected using the attribute operator '.'

```
In [5]: print(umur.Umur)
```

Data Manipulation::Indexing

- Pandas also provides alternative indexing method:

Indexing	Description
.loc()	Labe based
.iloc()	Integer based
.ix()	Both label and integer based

In [1]: bmi = pd.DataFrame(np.random.randn(20,12),columns=[‘jan’,‘feb',‘mar’,’apr’,’may’,’jun’,’jul’,’aug’,’sep’,’oct’,’nov’,’dec’])

In [2]: bmi.loc[:10,['jan','feb','mar']]

In [3]: bmi.loc[:,’jan’:’aug’]

Data Manipulation::Reindexing

Reindexing can be used to extract particular data:

```
In [1]: N=20
In [2]: df = pd.DataFrame({
   'A': pd.date_range(start='2016-01-01',periods=N,freq='D'),
   'x': np.linspace(0,stop=N-1,num=N),
   'y': np.random.rand(N),
   'C': np.random.choice(['Low','Medium','High'],N).tolist(),
   'D': np.random.normal(100, 10, size=(N)).tolist()
})
In [3]: my_df = df.reindex(index=[0,2,5], columns=['A', 'C', 'B'])
```

Data Manipulation::Renaming index

The rename() method can be used to relabel an axis based on some mapping (a dict or Series) or an arbitrary function.

In [1]: bmi = pd.DataFrame(np.random.randn(6,3),columns=['col1','col2','col3'])

In [2]: bmi.rename(columns={'col1' : ‘Januari', 'col2' : ‘Februari', 'col2' : ‘Maret'},index = {0 : ‘Adi', 1 : ‘Budi', 2 : ‘Cica'})

Data Manipulation::Slicing

The slicing mechanism for Series or DataFrame is pretty much the same as in the standard python (or even easier).

```
In [1]: bmi =
pd.DataFrame(np.random.randn(20,12),columns=['jan','feb','mar'
,'apr','may','jun','jul','aug','sep','oct','nov','dec'])
In [2]: print(bmi['jan'])
In [3]: print(bmi['jan'][:10]
In [4]: print(bmi[10:][['jan','feb']])
```

Data Manipulation::Droping

Droping can be used to unselect particular data in the DataFrame

In [1]: bmi = pd.DataFrame(np.random.randn(20,12),columns=[‘jan’,‘feb',‘mar’,’apr’,’may’,’jun’,’jul’,’aug’,’sep’,’oct’,’nov’,’dec’])

In [2]: print(bmi.drop(1))

In [3]: print(bmi.drop([1,4,10])

In [4]: print(bmi.drop(['jan','aug'],axis=1))

Data Manipulation::Sorting

There are two kinds of sorting:

- By label
- By Actual Value

```
df=pd.DataFrame(np.random.randn(10,2),index=[1,4,6,2,3,5,9,8,0,7],
columns=['col2','col1'])
        col2       col1
1  -2.063177   0.537527
4   0.142932  -0.684884
6   0.012667  -0.389340
2  -0.548797   1.848743
...
```

Data Manipulation::Sorting by Label

Using the sort_index() method, by passing the axis arguments and the order of sorting

```
#sorting the rows
df_s1 = df.sort_index()
df_s2 = df.sort_index(ascending=False)

#sorting the columns
df_s3 = df.soft_index(axis=1)
```

Data Manipulation::Sorting by Value

Using the sort_values() method and 'by' argument which will use the column name

```
df = pd.DataFrame({'col1':[2,1,1,1],'col2':[1,3,2,4]})
#sorting the first column only:
df_s1 = df.sort_values(by='col1')

#sorting both columns
df_s12 = df.sort_values(by=['col1','col2'])
```

CONTINUING EDUCATION CENTRE

Reshaping Data

The structure of Pandas’ DataFrame can be altered using reshaping mechanism. Basically, it spreads rows into columns.

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

→

Type	a	b	c
Date			
2016-03-01	11.432	NaN	20.784
2016-03-02	1.303	13.031	NaN
2016-03-03	99.906	NaN	20.784

df_new = df.pivot(index=‘Date’, columns=‘Type’, values=‘Value’)

Reshaping Data

We might run into problems using pivot.

ix	Item	CType	USD	EU
0	Item0	Gold	1$	1€
1	Item0	Bronze	2$	2€
2	Item0	Gold	3$	3€
3	Item1	Silver	4$	4€

ix=Item	Bronze	Gold	Silver
Item0	2$	*1 or 3$?*	NaN
Item1	NaN	NaN	4$

d.pivot(index='Item', columns='CType', values='USD')

Value Error: index contains duplicate entries, cannot reshape

<u>Solution</u>: d.pivot_table(index='item', columns='Ctype', values='USD', aggfunc=np.mean)

Combining Data

Pandas provides merge() function to combine two DataFrame.

data1

X1	X2
a	11.432
b	1.303
c	99.906

data2

X1	X3
a	20.784
b	NaN
d	20.784

```
>>> pd.merge(data1,
             data2,
             how='left',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN

```
>>> pd.merge(data1,
             data2,
             how='right',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
d	NaN	20.784

Or: data1.joint(data2,how='right')

Combining Data

Pandas provides merge() function to combine two DataFrame.

data1

X1	X2
a	11.432
b	1.303
c	99.906

data2

X1	X3
a	20.784
b	NaN
d	20.784

```
>>> pd.merge(data1,
             data2,
             how='inner',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN

```
>>> pd.merge(data1,
             data2,
             how='outer',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN
d	NaN	20.784

Or: data1.joint(data2,how='outer')

Data Analytic and Statistic

Pandas provides basic descriptive statistic methods:

Function	Description
abs()	Absolute value
count()	Number of non-null
cumprod(), comsum()	Cummulative product/sum
max(), min()	Maximum and minimum value
mean(), median(), mode()	Mean, median, and mode value
prod()	Product of values
sum()	Sum of values
std()	Standard Deviation of values

The describe() function is provided to compute a summary of statistics pertaining to the DataFrame columns

CONTINUING EDUCATION CENTRE

Data Analytic and Statistic::describe

In [1]: d = {'Name' : pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack', 'Lee','David','Gasper','Betina','Andres']), 'Age' : pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]), 'Rating' : pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

In [2]: df = pd.DataFrame(d)

In [3]: print (df.describe())

Out [3]:

	Age	Rating
count	12.000000	12.000000
mean	31.833333	3.743333
std	9.232682	0.661628
min	23.000000	2.560000
25%	25.000000	3.230000
50%	29.500000	3.790000
75%	35.500000	4.132500
max	51.000000	4.800000

Data Analytic and Statistic::pct_change

- The function pct_change() compares every element with its **<u>prior</u>** element and computes the change percentage.

```
In [1]: s = pd.Series([1,2,3,4,5,4]); print s.pct_change()
Out [1]:
0        NaN
1   1.000000
2   0.500000
3   0.333333
4   0.250000
5  -0.200000
dtype: float64
```

Note: by default, the pct_change() operates on columns; to apply row wise, use ‘axis=1’ argument.

CONTINUING EDUCATION CENTRE

Data Analytic and Statistic::cov

- The Series object has a method cov() to compute covariance between series objects.

  Note:NA will be excluded automatically.

```
In [1]: s1 = pd.Series(np.random.randn(10))
In [2]: s2 = pd.Series(np.random.randn(10))
In [3]: print s1.cov(s2)
Out [3]:
-0.12978405324

# when applied on a DataFrame, cov is computed between all the columns.
In [4]: df = pd.DataFrame(np.random.randn(10, 5), columns=['a', 'b', 'c', 'd', 'e'])
In [5]: print df['a'].cov(df['b'])
In [6]: print df.cov()
```

Data Analytic and Statistic::corr

- Correlation shows the linear relationship between any two array of values (series).

```
In [1]: df = pd.DataFrame(np.random.randn(10, 5), columns=['a', 'b', 'c', 'd', 'e'])
In [2]: print (df['a'].corr(df['b'])
Out [2]:
-0.383712785514
In [3]: print df.corr()
Out [3]:
          a          b          c          d          e
a  1.000000  -0.383713  -0.145368   0.002235  -0.104405
b -0.383713   1.000000   0.125311  -0.372821   0.224908
c -0.145368   0.125311   1.000000  -0.045661  -0.062840
d  0.002235  -0.372821  -0.045661   1.000000  -0.403380
e -0.104405   0.224908  -0.062840  -0.403380   1.000000
```

Pandas I/O

- Pandas I/O provides convenience methods to work with files. Two common format: csv and excel
- Examples:
  - df = pd.read_csv(‘myfile.csv’)
  - df = pd.read_csv(‘myfile.csv’,names=[‘Nama’,’Umur’,’Tinggi’],header=0)
  - df.to_csv(‘result.csv’)
  - df = pd.read_excel(‘myfile.xlsx’)
  - df = pd.read_excel(‘myfile.xlsx’,sheet_name=‘Sheet1’)
  - df.to_excel(‘result.xlsx’, sheet_name=‘Hasil’)

Pandas I/O::SQL Database

- There are various way to access SQL database, depend on the provier and how the server is accessed (eq. ODBC, Oracle, MySQL, etc.)
- Examples format using SQLAlchemy:

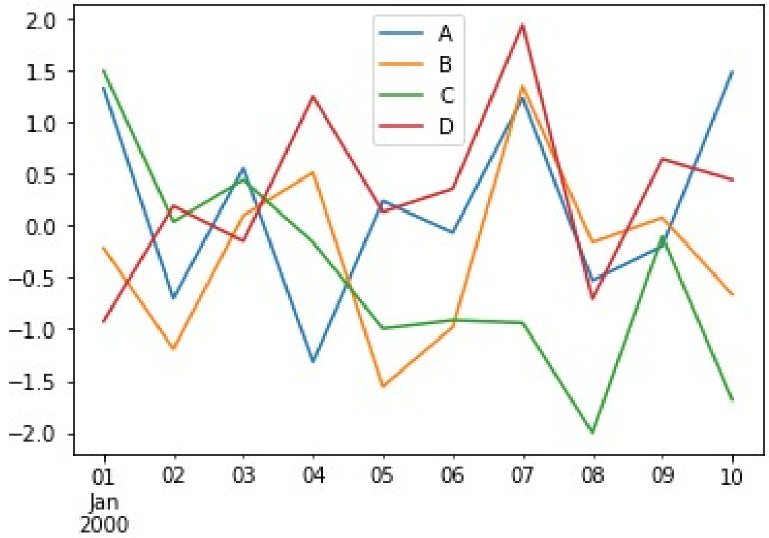
```
from sqlalchemy import create_engine
import pandas as pd
engine = create_engine('mysql+mysqldb://<user>:<pass>@<host>[:<port>]/<db_name>', echo=False)
f = pd.read_sql('SELECT * FROM table_name', engine)
```

Pandas Data Visualization

- Pandas provides a simple wrapper around the matplotlib libraries plot() method
- Example:

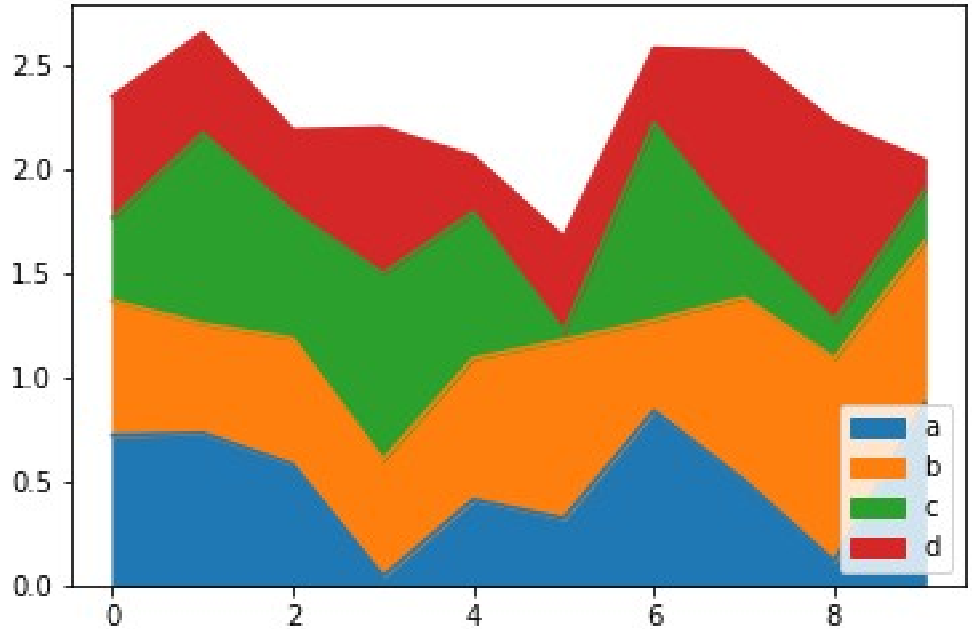
In [1]: df = pd.DataFrame(np.random.randn(10,4),index=pd.date_range('1/1/2000', periods=10), columns=list('ABCD'))

In [2]: df.plot()



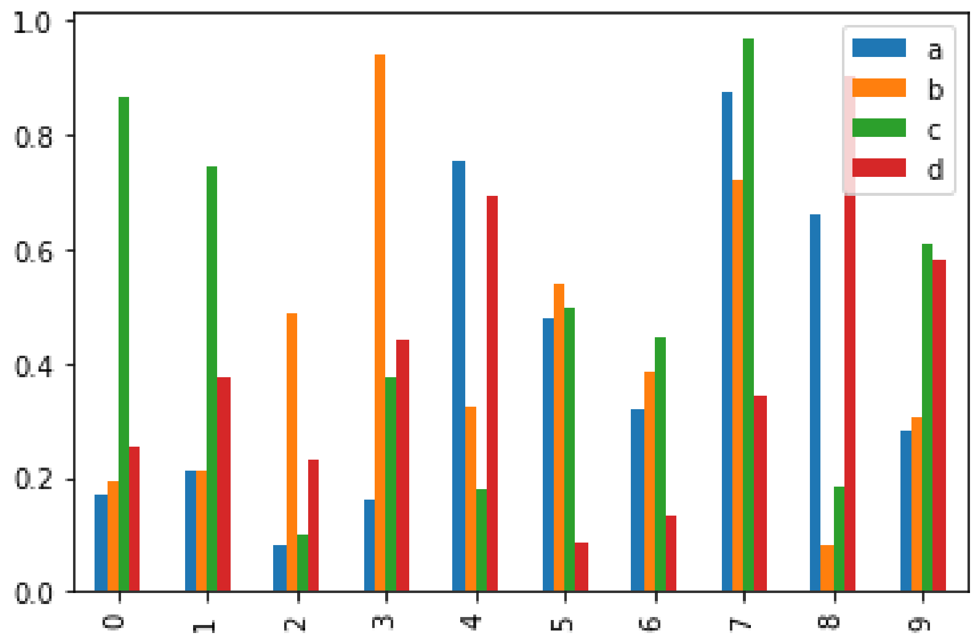
Pandas Data Visualization::Area Plot

```
In [1]: df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])
In [2]: df.plot.area()
```



Pandas Data Visualization::Bar

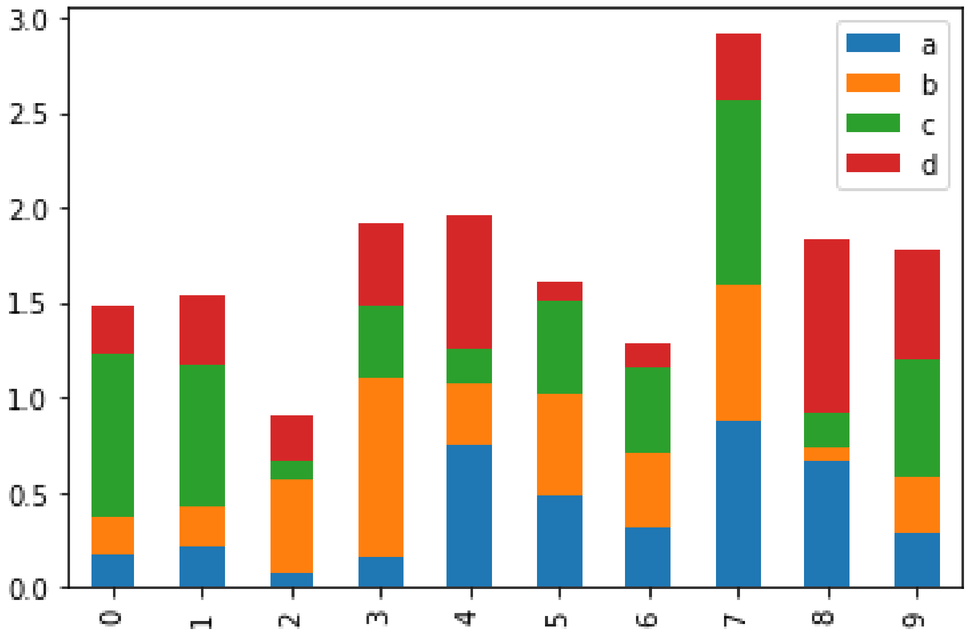
```
In [1]: df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])
In [2]: df.plot.bar()
```



Pandas Data Visualization::Bar

In [1]: df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d‘])

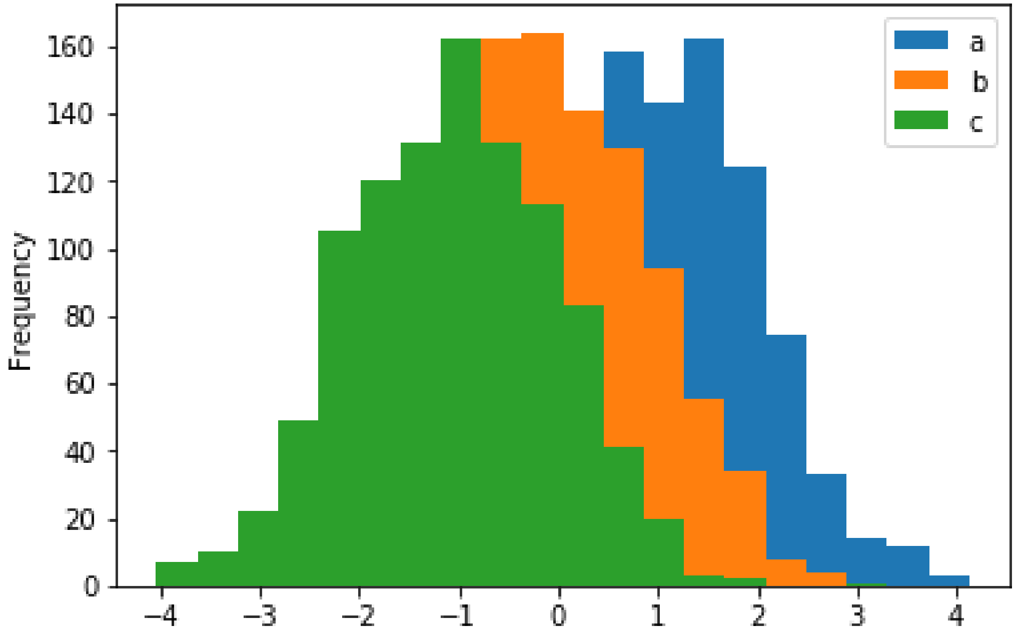
In [2]: df.plot.bar(stacked=True)



Pandas Data Visualization::Histogram

In [1]: df = pd.DataFrame({'a':np.random.randn(1000)+1,'b':np.random.randn(1000),'c': np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])

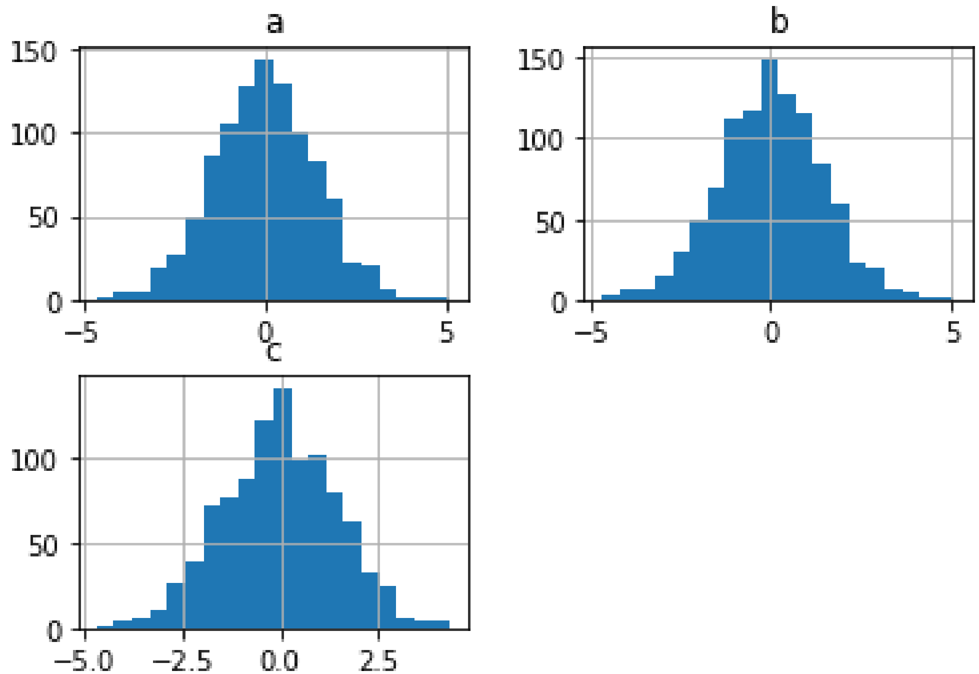
In [2]: df.plot.hist(bins=20)



Pandas Data Visualization::Histogram

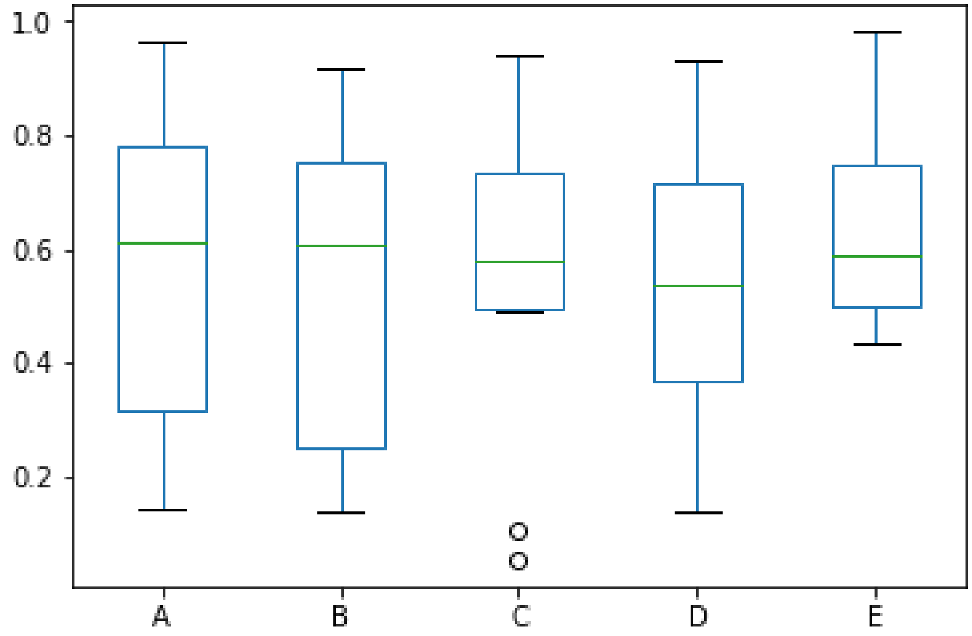
In [1]: df = pd.DataFrame({'a':np.random.randn(1000)+1,'b':np.random.randn(1000),'c': np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])

In [2]: df.diff().hist(bins=20)



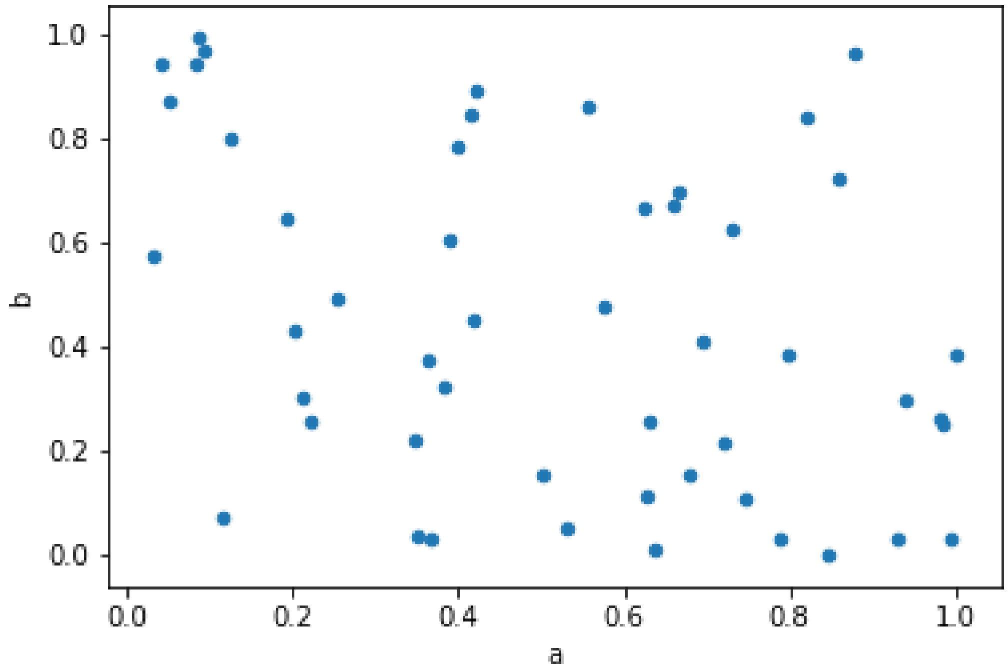
Pandas Data Visualization::Box Plots

```
In [1]: df = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C', 'D', 'E'])
In [2]: df.plot.box
```



Pandas Data Visualization::Scatter Plots

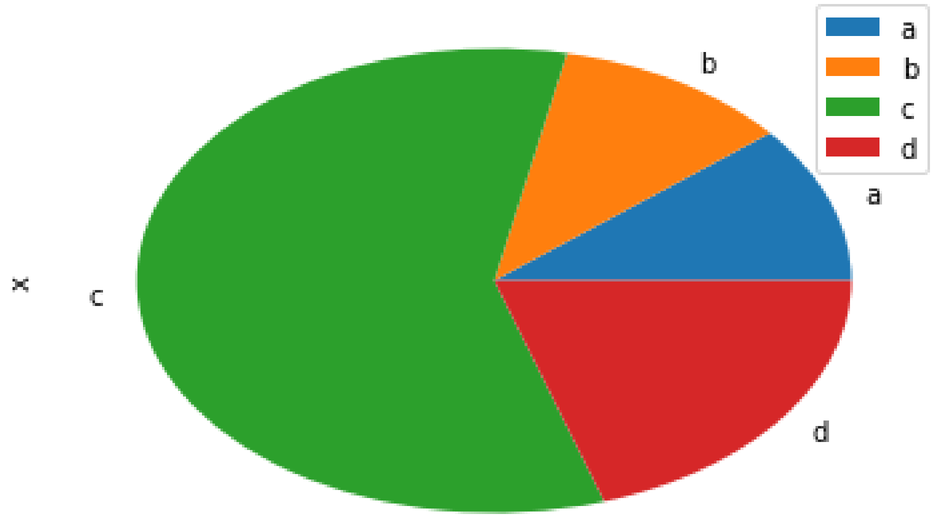
```
In [1]: df = pd.DataFrame(np.random.rand(50, 4), columns=['a', 'b', 'c', 'd'])
In [2]: df.plot.scatter(x='a', y='b')
```



Pandas Data Visualization::Pie Chart

In [1]: df = pd.DataFrame(3 * np.random.rand(4), index=['a', 'b', 'c', 'd'], columns=['x'])

In [2]: df.plot.pie(subplots=True)



Big Data

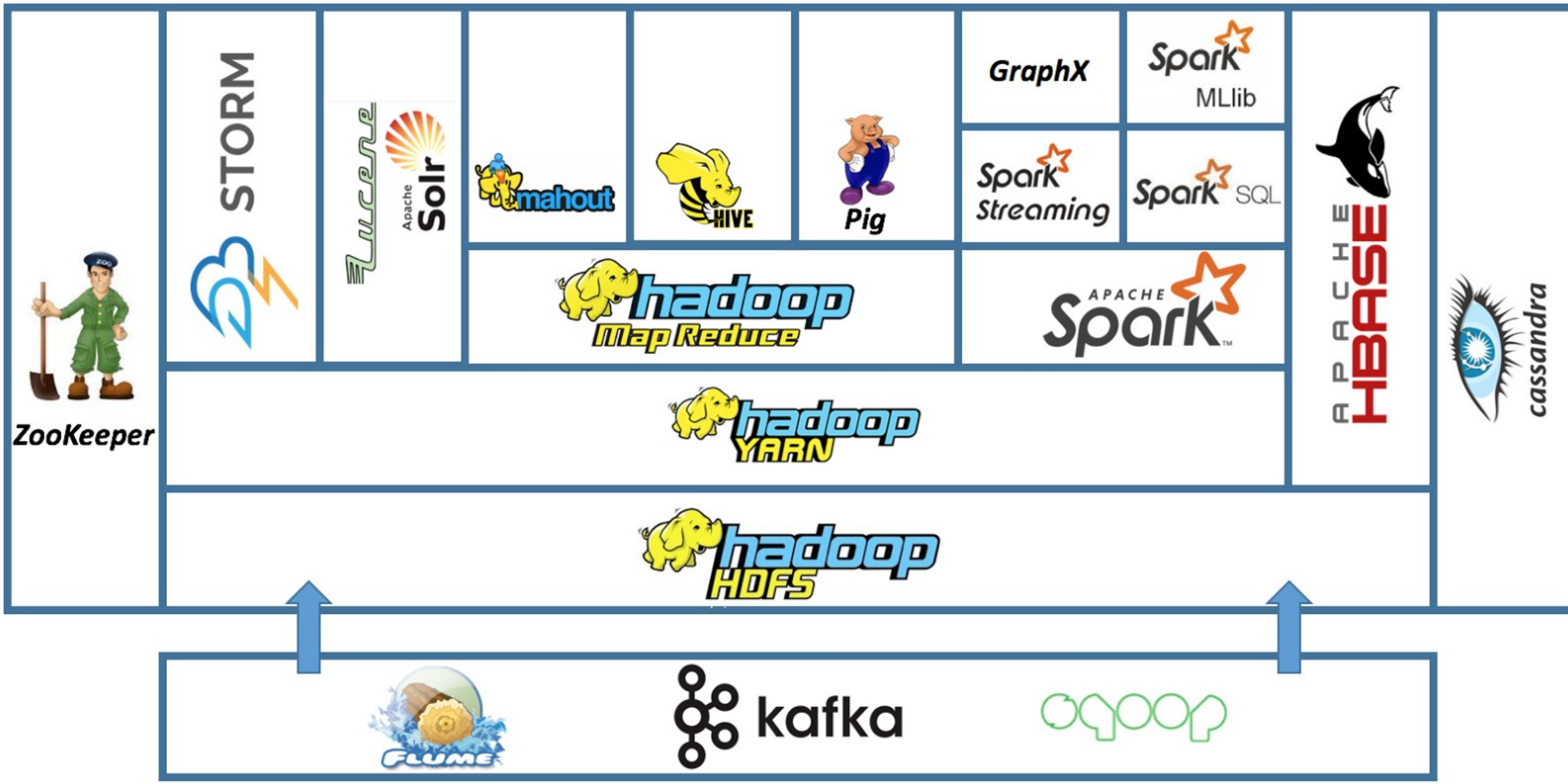
- Big Data – a term used to refer to data sets that are too large or complex for traditional data-processing application software to adequately deal with.
- Concern:
  - How big is big data?
  - How to process them?

Big Data::Hadoop

- Apache Hadoop is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation.
- It provides a software framework for distributed storage and processing of big data using the MapReduce programming model.

CONTINUING
EDUCATION
CENTRE

Big Data::Hadoop Ecosystem



Big Data::Hadoop Ecosystem

- Apache Mahout is a distributed machine learning algorithms focused primarily in the areas of collaborative filtering, clustering and classification, mainly running on top of Hadoop platform.
- Apache Spark is an open-source distributed general-purpose cluster-computing framework. Spark requires a cluster manager and a distributed storage system, and can work as standalone or alongside Hadoop.
- Apache Cassandra is an open-source, distributed, NoSQL database management system designed to handle large amounts of data across many commodity servers.

References

- https://pandas.pydata.org/pandas-docs/stable/tutorials.html
- https://www.tutorialspoint.com/python_pandas/
- https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python
- https://www.learnpython.org/en/Pandas_Basics