Spark Basics & Python

project with the latest technology

Lee Hae Joon

Reviews

What is RDD and Strong Points ?

Python

INSTALL JUPYTER NOTEBOOK

https://www.digitalocean.com/community/tutorials/how-to-set-up-a-jupyter-notebook-to-run-ipython-on-ubuntu-16-04

Python

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable.

1. **Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.

2. **Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs

3. **Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

4. **Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Python

Reserved Keywords

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Python Datatype

list - Like almost everything in Python, lists are objects. There are many methods associated to them. Some of which are presented here below.

```
In [1]: list_ = [1,2,3,'a','b','c']
        list_[0]

Out[1]: 1

In [2]: list_.append('d')
        list_

Out[2]: [1, 2, 3, 'a', 'b', 'c', 'd']

In [3]: list_append = [4,5,6]
        list_.append(list_append)
        list_

Out[3]: [1, 2, 3, 'a', 'b', 'c', 'd', [4, 5, 6]]

In [4]: list_extend = [7,8,9]
        list_.extend(list_extend)
        list_

Out[4]: [1, 2, 3, 'a', 'b', 'c', 'd', [4, 5, 6], 7, 8, 9]

In [5]: list_.index('b')

Out[5]: 4
```

```
In [5]: list_.index('b')
Out[5]: 4

In [7]: list_.insert(1,'c')
        list_
Out[7]: [1, 'c', 'c', 2, 3, 'a', 'b', 'c', 'd', [4, 5, 6], 7, 8, 9]

In [8]: ## slicing
        list_[2:4]
Out[8]: ['c', 2]

In [46]: list_.sort()
         list_
Out[46]: [1, 2, 3, 7, 8, 9, [4, 5, 6], 'a', 'b', 'c', 'c', 'c', 'd']

In [10]: ## list comprehension
         evens = []
         for i in list_:
             if list_.index(i)%2 == 0:
                 evens.append(i)

         evens
Out[10]: [1, 3, 'b', 'd', 7, 9]
```

Tuples - This is a data structure very similar to the list data structure. The main difference being that tuple manipulation are faster than list because tuples are immutable.

```
In [11]: ### Constructing tuples
         tuple_ = (1,2,3)
         tuple_[0]

Out[11]: 1

In [13]: tuple_[0] = 4
         tuple_

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-a7fa442c0cba> in <module>()
----> 1 tuple_[0] = 4
      2 tuple_

TypeError: 'tuple' object does not support item assignment
```

```
In [14]: tuple_2 = 4,5,6 # constructed by using comma
         tuple_2

Out[14]: (4, 5, 6)

In [16]: tuple_3 = tuple_2 * 3
         tuple_3

Out[16]: (4, 5, 6, 4, 5, 6, 4, 5, 6)

In [17]: ### interaction with dictionary
         dict_ = dict([('one',1), ('two',2), ('three',3)])
         dict_

Out[17]: {'one': 1, 'three': 3, 'two': 2}
```

Dictionary

```
In [23]: dict_ = {'one':1, 'two':2, 'three':3}
         dict_.keys()
         dict_.values()

Out[23]: [3, 2, 1]

In [24]: # iterating from dictionary
         for ele in dict_.items():
             print ele

         ('three', 3)
         ('two', 2)
         ('one', 1)

In [26]: dict_.has_key('one')

Out[26]: True

In [27]: dict_.get('one')

Out[27]: 1
```

```
In [27]: dict_.get('one')
Out[27]: 1

In [28]: [x for x in dict_.itervalues()] # iterkeys(), iteritems()
Out[28]: [3, 2, 1]

In [33]: ### combining dictionaries
         dict_1 = {'a':1}
         dict_2 = {'b':2}
         dict_1.update(dict_2)
         dict_1
Out[33]: {'a': 1, 'b': 2}

In [34]: # deleting item from dictionary
         del dict_1['a']
         dict_1
Out[34]: {'b': 2}
```

Sets, constructed from a sequence (or some other iterable object). Since sets cannot have duplicated, there are usually used to build sequence of unique items (e.g., set of identifiers).

In [35]:
```
a = set([1, 2, 3, 4])
b = set([3, 4, 5, 6])
```

In [37]:
```
# uniton
a | b
# interaction
a & b
```

Out[37]: {3, 4}

In [39]:
```
# difference
a - b
```

Out[39]: {1, 2}

```
In [39]: # difference
         a - b
Out[39]: {1, 2}

In [40]: # symmetric difference
         a ^ b
Out[40]: {1, 2, 5, 6}

In [44]: c = a.intersection(b) # c = a&b
         c
Out[44]: {3, 4}
```

DataFrame, Pandas

Next to Matplotlib and NumPy, Pandas is one of the most widely used Python libraries in data science. It is mainly used for data munging, and with good reason: it's very powerful and flexible, among many other things. It makes the least sexy part of the "sexiest job of the 21st Century" a bit more pleasant.

```
In [11]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt

In [2]:  s = pd.Series([1,3,5,np.nan,6,8])

In [12]: dates = pd.date_range('20170101', periods = 6)

In [8]:  dates

Out[8]:  <class 'pandas.tseries.index.DatetimeIndex'>
         [2017-01-01, ..., 2017-01-06]
         Length: 6, Freq: D, Timezone: None

In [ ]:

In [13]: df = pd.DataFrame(np.random.randn(6,4), index = dates, columns = list('ABCD'))

In [10]: df

Out[10]:
```

	A	B	C	D
2017-01-01	-0.671602	-1.199595	1.250647	-1.291698
2017-01-02	-0.837777	-1.087355	-1.751561	0.098106
2017-01-03	-0.874211	-0.671499	-0.059741	-1.065709
2017-01-04	1.860088	-0.056245	0.469709	0.226871
2017-01-05	-0.262813	0.279909	1.087980	-0.080228
2017-01-06	-0.901976	-1.081591	-1.451922	-1.438884

6 rows × 4 columns

In [13]:
```
df.head(1)
```

Out[13]:

	A	B	C	D
2017-01-01	-0.671602	-1.199595	1.250647	-1.291698

1 rows × 4 columns

In [12]:
```
df.tail(2)
```

Out[12]:

	A	B	C	D
2017-01-05	-0.262813	0.279909	1.087980	-0.080228
2017-01-06	-0.901976	-1.081591	-1.451922	-1.438884

2 rows × 4 columns

In [14]:
```
df.describe()
```

Out[14]:

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.281382	-0.636063	-0.075815	-0.591924
std	1.075608	0.615821	1.273858	0.753644
min	-0.901976	-1.199595	-1.751561	-1.438884
25%	-0.865103	-1.085914	-1.103876	-1.235201
50%	-0.754690	-0.876545	0.204984	-0.572969
75%	-0.365011	-0.210058	0.933412	0.053523
max	1.860088	0.279909	1.250647	0.226871

8 rows × 4 columns

```
In [15]: df.T
```

Out[15]:

	2017-01-01 00:00:00	2017-01-02 00:00:00	2017-01-03 00:00:00	2017-01-04 00:00:00	2017-01-05 00:00:00	2017-01-06 00:00:00
A	-0.671602	-0.837777	-0.874211	1.860088	-0.262813	-0.901976
B	-1.199595	-1.087355	-0.671499	-0.056245	0.279909	-1.081591
C	1.250647	-1.751561	-0.059741	0.469709	1.087980	-1.451922
D	-1.291698	0.098106	-1.065709	0.226871	-0.080228	-1.438884

4 rows × 6 columns

```
In [52]: df.sort_index(axis=0, ascending = False)
         #df.sort_index(axis=1, ascending = False)
```

Out[52]:

	A	B	C	D
2017-01-06	-0.901976	-1.081591	-1.451922	-1.438884
2017-01-05	-0.262813	0.279909	1.087980	-0.080228
2017-01-04	1.860088	-0.056245	0.469709	0.226871
2017-01-03	-0.874211	-0.671499	-0.059741	-1.065709
2017-01-02	-0.837777	-1.087355	-1.751561	0.098106
2017-01-01	-0.671602	-1.199595	1.250647	-1.291698

6 rows × 4 columns

```
In [21]: df[:3] # selected rows label
```

Out[21]:

	A	B	C	D
2017-01-01	-0.671602	-1.199595	1.250647	-1.291698
2017-01-02	-0.837777	-1.087355	-1.751561	0.098106
2017-01-03	-0.874211	-0.671499	-0.059741	-1.065709

3 rows × 4 columns

In [21]:
```
df[:3] # selected rows label
```

Out[21]:

	A	B	C	D
2017-01-01	-0.671602	-1.199595	1.250647	-1.291698
2017-01-02	-0.837777	-1.087355	-1.751561	0.098106
2017-01-03	-0.874211	-0.671499	-0.059741	-1.065709

3 rows × 4 columns

In [27]:
```
df.loc['2017-01-01'] # label
df.loc[dates[0]] # select by label 'row'
```

Out[27]:
```
A   -0.671602
B   -1.199595
C    1.250647
D   -1.291698
Name: 2017-01-01 00:00:00, dtype: float64
```

In [32]:
```
df.loc[:,'A':'C']  # row label, column label
df.loc[:, ['A','C']]
```

Out[32]:

	A	C
2017-01-01	-0.671602	1.250647
2017-01-02	-0.837777	-1.751561
2017-01-03	-0.874211	-0.059741
2017-01-04	1.860088	0.469709
2017-01-05	-0.262813	1.087980
2017-01-06	-0.901976	-1.451922

6 rows × 2 columns

In [34]:
```
# selection by position
df.iloc[3]
df.iloc[3:5,0:2]
```

Out[34]:

	A	B
2017-01-04	1.860088	-0.056245
2017-01-05	-0.262813	0.279909

2 rows × 2 columns

In [37]:
```
df_filter = df[df>0]
df_filter.fillna('zero') # df_filter.dropna(how='any)
```

Out[37]:

	A	B	C	D
2017-01-01	zero	zero	1.250647	zero
2017-01-02	zero	zero	zero	0.09810621
2017-01-03	zero	zero	zero	zero
2017-01-04	1.860088	zero	0.4697087	0.2268709
2017-01-05	zero	0.2799088	1.08798	zero
2017-01-06	zero	zero	zero	zero

6 rows × 4 columns

In [14]:
```
df_copy = df.copy()
df_copy['copy'] = ['one', 'two','three', np.nan, np.nan, 'six']
df_copy
```

Out[14]:

	A	B	C	D	copy
2017-01-01	1.617212	-2.285564	-0.132234	0.273070	one
2017-01-02	-0.352639	0.357600	-0.571745	0.515890	two
2017-01-03	-1.097020	2.993803	-0.682899	-0.272870	three
2017-01-04	-0.296105	-1.641046	0.571109	-0.245860	NaN
2017-01-05	-0.672826	-0.015981	0.001862	-1.756203	NaN
2017-01-06	0.157500	-2.489007	-0.395584	-1.312094	six

In [15]:
```python
df_copy['A+10'] = df_copy['A'].apply(lambda x : x + 10)
df_copy
```

Out[15]:

	A	B	C	D	copy	A+10
2017-01-01	1.617212	-2.285564	-0.132234	0.273070	one	11.617212
2017-01-02	-0.352639	0.357600	-0.571745	0.515890	two	9.647361
2017-01-03	-1.097020	2.993803	-0.682899	-0.272870	three	8.902980
2017-01-04	-0.296105	-1.641046	0.571109	-0.245860	NaN	9.703895
2017-01-05	-0.672826	-0.015981	0.001862	-1.756203	NaN	9.327174
2017-01-06	0.157500	-2.489007	-0.395584	-1.312094	six	10.157500

In [44]:
```python
s = pd.Series([1,3,5,np.nan,6,8], index=dates).shift(2)
s
```

Out[44]:
```
2017-01-01    NaN
2017-01-02    NaN
2017-01-03      1
2017-01-04      3
2017-01-05      5
2017-01-06    NaN
Freq: D, dtype: float64
```

In [51]:
```python
df.sub(s,axis = 0) # axis = 'index'
```

Out[51]:

	A	B	C	D
2017-01-01	NaN	NaN	NaN	NaN
2017-01-02	NaN	NaN	NaN	NaN
2017-01-03	-1.874211	-1.671499	-1.059741	-2.065709
2017-01-04	-1.139912	-3.056245	-2.530291	-2.773129
2017-01-05	-5.262813	-4.720091	-3.912020	-5.080228
2017-01-06	NaN	NaN	NaN	NaN

6 rows × 4 columns

In [60]:
```
df.apply(np.cumsum, axis=1)
df.apply(np.cumsum, axis=0)
```

Out[60]:

	A	B	C	D
2017-01-01	-0.671602	-1.199595	1.250647	-1.291698
2017-01-02	-1.509379	-2.286950	-0.500913	-1.193592
2017-01-03	-2.383590	-2.958449	-0.560654	-2.259301
2017-01-04	-0.523503	-3.014694	-0.090946	-2.032430
2017-01-05	-0.786316	-2.734785	0.997034	-2.112658
2017-01-06	-1.688292	-3.816376	-0.454887	-3.551542

6 rows × 4 columns

In [61]:
```
df.apply(lambda x : x.max() - x.min())
```

Out[61]:
```
A    2.762064
B    1.479504
C    3.002208
D    1.665755
dtype: float64
```

In [77]:
```
df = pd.DataFrame({'A' : ['one', 'one', 'two', 'three'] * 3,'B' : ['A', 'B', 'C'] * 4,
                   'C' : ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] * 2,'D' : np.random.randn(12),
                   'E' : np.random.randn(12)})
df
```

Out[77]:

	A	B	C	D	E
0	one	A	foo	0.453737	0.836380
1	one	B	foo	0.281438	-2.513328
2	two	C	foo	0.796836	1.051851
3	three	A	bar	-0.610079	0.423231
4	one	B	bar	-1.447885	0.359370
5	one	C	bar	1.511768	-1.312828
6	two	A	foo	1.102993	0.990530
7	three	B	foo	-0.238541	-1.701424
8	one	C	foo	-0.015979	0.922675
9	one	A	bar	0.228019	-2.041269
10	two	B	bar	2.906439	0.038860
11	three	C	bar	-1.683287	0.361368

12 rows × 5 columns

In [17]:
```
df_val = pd.pivot_table(df, values='D', index=['A', 'B'], columns=['C'])
df_val
```

Out[17]:

	C	-0.682899448303	-0.571744507826	-0.395584066434	-0.132233839532	0.00186176615742	0.571109205329
A	B						
-1.097020	2.993803	-0.27287	NaN	NaN	NaN	NaN	NaN
-0.672826	-0.015981	NaN	NaN	NaN	NaN	-1.756203	NaN
-0.352639	0.357600	NaN	0.51589	NaN	NaN	NaN	NaN
-0.296105	-1.641046	NaN	NaN	NaN	NaN	NaN	-0.24586
0.157500	-2.489007	NaN	NaN	-1.312094	NaN	NaN	NaN
1.617212	-2.285564	NaN	NaN	NaN	0.27307	NaN	NaN

Advanced DataFrame Pandas

In [3]:

```
# Create a dataframe with dates as your index
States = ['NY', 'NY', 'NY', 'NY', 'FL', 'FL', 'GA', 'GA', 'FL', 'FL']
data = [1.0, 2, 3, 4, 5, 6, 7, 8, 9, 10]
idx = pd.date_range('1/1/2012', periods=10, freq='MS')
df1 = pd.DataFrame(data, index=idx, columns=['Revenue'])
df1['State'] = States
df1
```

Out[3]:

	Revenue	State
2012-01-01	1.0	NY
2012-02-01	2.0	NY
2012-03-01	3.0	NY
2012-04-01	4.0	NY
2012-05-01	5.0	FL
2012-06-01	6.0	FL
2012-07-01	7.0	GA
2012-08-01	8.0	GA
2012-09-01	9.0	FL
2012-10-01	10.0	FL

https://pandas.pydata.org/pandas-docs/stable/timeseries.html

```
In [4]: # Create a second dataframe
data2 = [10.0, 10.0, 9, 9, 8, 8, 7, 7, 6, 6]
idx2 = pd.date_range('1/1/2013', periods=10, freq='MS')
df2 = pd.DataFrame(data2, index=idx2, columns=['Revenue'])
df2['State'] = States
df2
```

Out[4]:

	Revenue	State
2013-01-01	10.0	NY
2013-02-01	10.0	NY
2013-03-01	9.0	NY
2013-04-01	9.0	NY
2013-05-01	8.0	FL
2013-06-01	8.0	FL
2013-07-01	7.0	GA
2013-08-01	7.0	GA
2013-09-01	6.0	FL
2013-10-01	6.0	FL

In [5]:
```python
df = pd.concat([df1,df2])
df
```

Out[5]:

	Revenue	State
2012-01-01	1.0	NY
2012-02-01	2.0	NY
2012-03-01	3.0	NY
2012-04-01	4.0	NY
2012-05-01	5.0	FL
2012-06-01	6.0	FL
2012-07-01	7.0	GA
2012-08-01	8.0	GA
2012-09-01	9.0	FL
2012-10-01	10.0	FL
2013-01-01	10.0	NY
2013-02-01	10.0	NY
2013-03-01	9.0	NY
2013-04-01	9.0	NY
2013-05-01	8.0	FL
2013-06-01	8.0	FL
2013-07-01	7.0	GA
2013-08-01	7.0	GA
2013-09-01	6.0	FL
2013-10-01	6.0	FL

```
In [7]: # Method 1

        # make a copy of original df
        newdf = df.copy()

        newdf['x-Mean'] = abs(newdf['Revenue'] - newdf['Revenue'].mean())
        newdf['1.96*std'] = 1.96*newdf['Revenue'].std()
        newdf['Outlier'] = abs(newdf['Revenue'] - newdf['Revenue'].mean()) > 1.96*newdf['Revenue'].std()
        newdf
```

Out[7]:

	Revenue	State	x-Mean	1.96*std	Outlier
2012-01-01	1.0	NY	5.75	5.200273	True
2012-02-01	2.0	NY	4.75	5.200273	False
2012-03-01	3.0	NY	3.75	5.200273	False
2012-04-01	4.0	NY	2.75	5.200273	False
2012-05-01	5.0	FL	1.75	5.200273	False
2012-06-01	6.0	FL	0.75	5.200273	False
2012-07-01	7.0	GA	0.25	5.200273	False
2012-08-01	8.0	GA	1.25	5.200273	False
2012-09-01	9.0	FL	2.25	5.200273	False
2012-10-01	10.0	FL	3.25	5.200273	False
2013-01-01	10.0	NY	3.25	5.200273	False
2013-02-01	10.0	NY	3.25	5.200273	False
2013-03-01	9.0	NY	2.25	5.200273	False
2013-04-01	9.0	NY	2.25	5.200273	False
2013-05-01	8.0	FL	1.25	5.200273	False
2013-06-01	8.0	FL	1.25	5.200273	False
2013-07-01	7.0	GA	0.25	5.200273	False
2013-08-01	7.0	GA	0.25	5.200273	False
2013-09-01	6.0	FL	0.75	5.200273	False
2013-10-01	6.0	FL	0.75	5.200273	False

```
In [26]: # Method 2
         # Group by item

         # make a copy of original df
         newdf = df.copy()

         State = newdf.groupby('State')
         State

         for idx, data in State: # DataFrameGroupBy
             print idx, data
```

```
FL             Revenue State
2012-05-01       5.0    FL
2012-06-01       6.0    FL
2012-09-01       9.0    FL
2012-10-01      10.0    FL
2013-05-01       8.0    FL
2013-06-01       8.0    FL
2013-09-01       6.0    FL
2013-10-01       6.0    FL
GA             Revenue State
2012-07-01       7.0    GA
2012-08-01       8.0    GA
2013-07-01       7.0    GA
2013-08-01       7.0    GA
NY             Revenue State
2012-01-01       1.0    NY
2012-02-01       2.0    NY
2012-03-01       3.0    NY
2012-04-01       4.0    NY
2013-01-01      10.0    NY
2013-02-01      10.0    NY
2013-03-01       9.0    NY
2013-04-01       9.0    NY
```

In [8]:
```python
# Method 2
# Group by multiple items

# make a copy of original df
newdf = df.copy()

StateMonth = newdf.groupby(['State', lambda x: x.month])

newdf['Outlier'] = StateMonth.transform( lambda x: abs(x-x.mean()) > 1.96*x.std() )
newdf['x-Mean'] = StateMonth.transform( lambda x: abs(x-x.mean()) )
newdf['1.96*std'] = StateMonth.transform( lambda x: 1.96*x.std() )
newdf
```

Out[8]:

	Revenue	State	Outlier	x-Mean	1.96*std
2012-01-01	1	NY	False	4.5	12.473364
2012-02-01	2	NY	False	4.0	11.087434
2012-03-01	3	NY	False	3.0	8.315576
2012-04-01	4	NY	False	2.5	6.929646
2012-05-01	5	FL	False	1.5	4.157788
2012-06-01	6	FL	False	1.0	2.771859
2012-07-01	7	GA	False	0.0	0.000000
2012-08-01	8	GA	False	0.5	1.385929
2012-09-01	9	FL	False	1.5	4.157788
2012-10-01	10	FL	False	2.0	5.543717
2013-01-01	10	NY	False	4.5	12.473364
2013-02-01	10	NY	False	4.0	11.087434
2013-03-01	9	NY	False	3.0	8.315576
2013-04-01	9	NY	False	2.5	6.929646
2013-05-01	8	FL	False	1.5	4.157788
2013-06-01	8	FL	False	1.0	2.771859
2013-07-01	7	GA	False	0.0	0.000000
2013-08-01	7	GA	False	0.5	1.385929
2013-09-01	6	FL	False	1.5	4.157788
2013-10-01	6	FL	False	2.0	5.543717

20 rows × 5 columns

```
In [9]: # Method 3
        # Group by item

        # make a copy of original df
        newdf = df.copy()

        State = newdf.groupby('State')

        def s(group):
            group['x-Mean'] = abs(group['Revenue'] - group['Revenue'].mean())
            group['1.96*std'] = 1.96*group['Revenue'].std()
            group['Outlier'] = abs(group['Revenue'] - group['Revenue'].mean()) > 1.96*group['Revenue'].std()
            return group

        Newdf2 = State.apply(s)
        Newdf2
```

Out[9]:

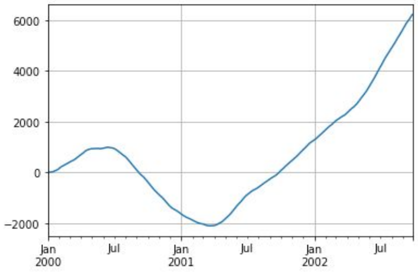
	Revenue	State	x-Mean	1.96*std	Outlier
2012-01-01	1	NY	5.00	7.554813	False
2012-02-01	2	NY	4.00	7.554813	False
2012-03-01	3	NY	3.00	7.554813	False
2012-04-01	4	NY	2.00	7.554813	False
2012-05-01	5	FL	2.25	3.434996	False
2012-06-01	6	FL	1.25	3.434996	False
2012-07-01	7	GA	0.25	0.980000	False
2012-08-01	8	GA	0.75	0.980000	False
2012-09-01	9	FL	1.75	3.434996	False
2012-10-01	10	FL	2.75	3.434996	False
2013-01-01	10	NY	4.00	7.554813	False
2013-02-01	10	NY	4.00	7.554813	False
2013-03-01	9	NY	3.00	7.554813	False
2013-04-01	9	NY	3.00	7.554813	False
2013-05-01	8	FL	0.75	3.434996	False
2013-06-01	8	FL	0.75	3.434996	False
2013-07-01	7	GA	0.25	0.980000	False
2013-08-01	7	GA	0.25	0.980000	False
2013-09-01	6	FL	1.25	3.434996	False

```
In [8]: file_name = "./analyzed_american.csv"
        Newdf2.to_csv(file_name)
```

analyzed_american.csv 2 minutes ago

Matplotlib

```
In [6]: ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=1000))
        ts

Out[6]: 2000-01-01    1.075219
        2000-01-02    0.436073
        2000-01-03   -0.901465
        2000-01-04   -0.133734
        2000-01-05   -1.400580
        2000-01-06    1.506402
        2000-01-07    2.166358
        2000-01-08    0.526718
        2000-01-09    0.148402
        2000-01-10   -1.341403

In [1]: %matplotlib inline

In [16]: ts = ts.cumsum()
         ts.plot()
         ts.hist()

         #matplotlib reference - https://matplotlib.org/tutorials/introductory/sample_plots.html#sphx-glr-tutorials-introductory

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x113312d50>
```



https://matplotlib.org/tutorials/introductory/sample_plots.html#sphx-glr-tutorials-introductory-sample-plots-py

Practice

room_id	host_id	room_type	borough	neighborhood	reviews	overall_satisfaction	accommodates	bedrooms	price	minstay	latitude	longitude	last_modified
6291807	16375951	Entire home/apt		Jamaica Plain	1		4	2	119	14	42.29816	-71.11153	2016-05-19 02:58:16.563871
2656568	13597630	Entire home/apt		Back Bay	0		2	1	600	1	42.348072	-71.076639	2016-05-19 02:58:06.015822
10723203	15913699	Private room		Allston	2		2	1	96	1	42.350588	-71.129477	2016-05-19 02:57:39.074104
10034592	20399668	Private room		Dorchester	13	5	2	1	55	1	42.317168	-71.040483	2016-05-19 02:57:28.669274
5454513	4962900	Entire home/apt		Back Bay	13	4.5	5	2	276		42.346598	-71.080123	2016-05-19 02:56:56.182103
335730	290698	Private room		East Boston	34	3.5	9	1	70	1	42.39001	-70.996161	2016-05-19 02:56:44.188597
7635616	22348222	Entire home/apt		Beacon Hill	3	5	2	0	155	7	42.359601	-71.063908	2016-05-19 02:56:18.932562
12808014	28197086	Private room		Allston	0		1	1	65	1	42.351883	-71.130772	2016-05-19 02:56:12.350833
6793913	30283594	Entire home/apt		Fenway	2		5	2	571	3	42.343305	-71.101489	2016-05-19 02:55:56.997256
586994	2894162	Entire home/apt		Beacon Hill	47	4.5	2	1	160	4	42.358926	-71.063109	2016-05-19 02:54:56.750137

In [4]:
```python
import pandas as pd
import numpy as np
```

In [5]:
```python
basic_folder = '/Users/phil/python_work/venv/'
file_name = basic_folder + 'airbnb.csv'
df_air = pd.read_csv(file_name)
```

In [6]:
```python
df_air
```

Out[6]:

	room_id	host_id	room_type	borough	neighborhood	reviews	overall_satisfaction	accommodates	bedrooms	price	minstay	latitude	longitude
0	6291807	16375951	Entire home/apt	NaN	Jamaica Plain	1	NaN	4	2.0	119.0	14.0	42.298160	-71.111530
1	2656568	13597630	Entire home/apt	NaN	Back Bay	0	NaN	2	1.0	600.0	1.0	42.348072	-71.076639
2	10723203	15913699	Private room	NaN	Allston	2	NaN	2	1.0	96.0	1.0	42.350588	-71.129477
3	10034592	20399668	Private room	NaN	Dorchester	13	5.0	2	1.0	55.0	1.0	42.317168	-71.040483
4	5454513	4962900	Entire home/apt	NaN	Back Bay	13	4.5	5	2.0	276.0	NaN	42.346598	-71.080123
5	335730	290698	Private room	NaN	East Boston	34	3.5	9	1.0	70.0	1.0	42.390010	-70.996161
6	7635616	22348222	Entire home/apt	NaN	Beacon Hill	3	5.0	2	0.0	155.0	7.0	42.359601	-71.063908

```
In [8]: df_air['overall_satisfaction'] = df_air.overall_satisfaction.fillna(3)
```

```
In [9]: df_air
```

3270	6248970	32449720	Private room	NaN	Jamaica Plain	52	5.0	1	1.0	65.0	3.0	42.321720	-71.103459
3271	7549896	39568178	Private room	NaN	Chinatown	3	5.0	2	1.0	100.0	7.0	42.347990	-71.064152
3272	8519646	15098486	Private room	NaN	North End	16	4.5	2	1.0	116.0	1.0	42.364245	-71.052945
3273	6574771	16881770	Private room	NaN	Dorchester	9	5.0	1	1.0	40.0	3.0	42.323962	-71.058513
3274	1321422	6608084	Private room	NaN	Dorchester	168	5.0	3	1.0	45.0	1.0	42.308380	-71.046943
3275	12590656	51449558	Private room	NaN	Dorchester	0	3.0	3	1.0	55.0	2.0	42.321141	-71.056032
3276	1178371	6430732	Private room	NaN	Dorchester	66	5.0	1	1.0	45.0	2.0	42.294568	-71.066813

3277 rows × 14 columns

```
In [10]: df_neigh = df_air.set_index(['neighborhood'])
         df_neigh
```

Out[10]:

neighborhood	room_id	host_id	room_type	borough	reviews	overall_satisfaction	accommodates	bedrooms	price	minstay	latitude	longitude	last
Jamaica Plain	6291807	16375951	Entire home/apt	NaN	1	3.0	4	2.0	119.0	14.0	42.298160	-71.111530	2 02:58:
Back Bay	2656568	13597630	Entire home/apt	NaN	0	3.0	2	1.0	600.0	1.0	42.348072	-71.076639	2 02:58:
Allston	10723203	15913699	Private room	NaN	2	3.0	2	1.0	96.0	1.0	42.350588	-71.129477	2 02:57:
Dorchester	10034592	20399668	Private room	NaN	13	5.0	2	1.0	55.0	1.0	42.317168	-71.040483	2 02:57:
Back Bay	5454513	4962900	Entire home/apt	NaN	13	4.5	5	2.0	276.0	NaN	42.346598	-71.080123	2 02:56:
East Boston	335730	290698	Private room	NaN	34	3.5	9	1.0	70.0	1.0	42.390010	-70.996161	2 02:56:
Beacon Hill	7635616	22348222	Entire	NaN	3	5.0	2	0.0	155.0	7.0	42.359601	-71.063908	2

Answer 4 questions