Objective of the class

Ethans
Learn from experts

Python Pandas

- What is Pandas
- Creating Series
- Creating Data Frames,
- Grouping, Sorting
- Plotting Data
- Data analysis with data set
- Practical use cases using data analysis.

What is Pandas?

- pandas is a Python package for providing fast, flexible, and expressive data structures.
- Data structures designed in pandas to make working with ‘relational’ or ‘labeled’ data
- Pandas is the most powerful and flexible open source data analysis / manipulation tool available in python.

- pandas is built on top of NumPy module and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.
- It handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering.

- Source - http://pandas.pydata.org/pandas-docs/stable/

Data Structure

- The two primary data structures of pandas are built on top of NumPy they are Series (1-dimensional) and DataFrame (2-dimensional).

These DS will help to do:

- Easy handling of missing data and adding/removing of columns
- Automatic data alignment and powerful, flexible group by functionality to perform operations on data sets, for both aggregating and transforming data
- Intelligent label-based slicing, fancy indexing, and sub setting of large data sets
- Easy merging and joining data sets
- Flexible reshaping and pivoting of data sets
- Robust IO tools for loading data from flat files.

Series

- Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.).
- The axis labels are collectively referred to as the **index**

 # Create pandas Series

```
>>> import pandas as pd
>>> S = pd.Series(data, index=index)
```

Data in above can be: List, Tuple, dict, NumPy array or any scalar value
Index – Should contains the label. **index** must be the same length as **data .**

```
>>> S = pd.Series('Ethans', index = ('Name',))
>>> S
Name     Ethans
```

Series – With other objects

```
>>> import pandas as pd
>>> S = pd.Series([1,2,3])
>>> S
0    1
1    2
2    3
dtype: int64
>>> S = pd.Series((1,2,3))
>>> S
0    1
1    2
2    3
dtype: int32
>>> S = pd.Series({1:2, 3:4})
>>> S
1    2
3    4
dtype: int64
```

Series – With numpy and LoL

Ethans
Learn from experts

```
>>> S = pd.Series(np.array([1,2,3]), index = (1,2,3))
>>> S
1    1
2    2
3    3
dtype: int32
>>>
>>> S = pd.Series(np.array([1,2,3]))
>>> S
0    1
1    2
2    3
dtype: int32
```

```
>>> employees = pd.Series([['ethans', 'Bob', 'Aaron'], [35, 40, 29]])
>>> employees
0    [ethans, Bob, Aaron]
1            [35, 40, 29]
dtype: object
```

Create Series: All example

```
# Creating a series in Pandas
import pandas as pd
s1 = pd.Series([1,2,3]) # from list
s1 = pd.Series((1,2,3)) # from tuple
s1 = pd.Series(list('pandas')) # from string
s1 = pd.Series((1,2,3), index=(1,2,3)) # from tuple, with indexes
s1 = pd.Series({1:2,3:4}) # from dictionary
s1 = pd.Series({1:2,3:4}, index= (1,2,3,4,5)) # from dictionary

import numpy as np
s1 = pd.Series(np.array([1,2,3]), index=(1,2,3)) # from np, with indexes
s1 = pd.Series([['Jatin','Aakash','Rahul'], [35, 28, 31]],
               index=('names', 'age')) # from list of list
```

Series – Get Index value

```
>>> d = {'Pune': 900, 'Mumbai': 1300, 'Delhi': 900, 'Bangalore': 1100,
'Goa': 450, 'Daman': None}

>>> cities = pd.Series(d)
>>> cities
Bangalore    1100.0
Daman           NaN
Delhi         900.0
Goa           450.0
Mumbai       1300.0
Pune          900.0
dtype: float64
```

```
>>> cities['Pune']
900.0
>>> cities[['Pune', 'Mumbai', 'Goa']]
Pune         900.0
Mumbai      1300.0
Goa          450.0
dtype: float64
```

Series – Call basic functions

```
>>> cities.isnull()
Bangalore    False
Daman         True
Delhi        False
Goa          False
Mumbai       False
Pune         False
dtype: bool
>>> cities.notnull()
Bangalore     True
Daman        False
Delhi         True
Goa           True
Mumbai        True
Pune          True
dtype: bool
>>> cities[cities.isnull()]
Daman   NaN
```

```
>>> cities < 1000
Bangalore    False
Daman        False
Delhi         True
Goa           True
Mumbai       False
Pune          True
dtype: bool
>>> cities[cities < 1000]
Delhi    900.0
Goa      450.0
Pune     900.0
dtype: float64
>>> cities[cities == 900]
Delhi    900.0
Pune     900.0
dtype: float64
```

Series – Call basic functions

```
>>> cities * 2
Bangalore     2200.0
Daman            NaN
Delhi         1800.0
Goa            900.0
Mumbai        2600.0
Pune          1800.0
dtype: float64
>>> np.square(cities)
Bangalore     1210000.0
Daman               NaN
Delhi          810000.0
Goa            202500.0
Mumbai        1690000.0
Pune           810000.0
dtype: float64
```

```
>>> cities['Pune'] = 800
>>> cities[cities > 1000] = 800
>>> cities
Bangalore     800.0
Daman           NaN
Delhi         900.0
Goa           450.0
Mumbai        800.0
Pune          800.0
dtype: float64
>>> 'Chennai' in cities
False
>>> 'Daman' in cities
True
```

Data Frame

Ethans
Learn from experts

- Data Frame is two-dimensional labeled array capable of holding any data type. DataFrame is a tabular data structure comprised of rows and columns, like a spreadsheet, database table, or R's data.frame object.

 # Create pandas Data Frame

```
>>> import pandas as pd
>>> df = pd.DataFrame({'names':['Ethan', 'Bob', 'Aaron'],
                       'Age':[30, 35, 40],
                       'City':['Pune', 'New York', 'Texas']})
>>>
>>> df
   Age       City  names
0   30       Pune  Ethan
1   35   New York    Bob
2   40      Texas  Aaron
```

Data Frame – with Series

```
>>> d = {'one' : pd.Series([1., 2., 3.], index=['a', 'b', 'c']),
   'two' : pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}

>>>
>>> df1 = pd.DataFrame(d)
>>> df1
   one  two
a  1.0  1.0
b  2.0  2.0
c  3.0  3.0
d  NaN  4.0
```

```
>>> df1 = pd.DataFrame(d, index = ['d', 'c', 'a'])
>>> df1
   one  two
d  NaN  4.0
c  3.0  3.0
a  1.0  1.0
```

```
>>> df1 = pd.DataFrame(d, index = ['d', 'c', 'a'],
    columns = ['one', 'newOne'])
>>> df1
   one newOne
d  NaN    NaN
c  3.0    NaN
a  1.0    NaN
```

Data Frame – with LoD

```
>>> d = [{'a': 1, 'b': 2}, {'a': 3, 'b': 4, 'c': 5}]
>>> df = pd.DataFrame(d)
>>> df
   a  b    c
0  1  2  NaN
1  3  4  5.0
>>> pd.DataFrame(d, index=['first', 'second'])
        a  b    c
first   1  2  NaN
second  3  4  5.0
>>> pd.DataFrame(d, columns=['a', 'b'])
   a  b
0  1  2
1  3  4
```

selection, addition and deletion

```
>>> D = {'India':[1, .4,'Delhi'], 'China': [1.5, .2, 'Beijing']}
>>> df = pd.DataFrame(D, index = ['Population', 'PD', 'Capital'])
>>> df
            China  India
Population    1.5      1
PD            0.2    0.4
Capital     Beijing  Delhi
>>> df['India']
Population      1
PD            0.4
Capital     Delhi
Name: India, dtype: object
>>>
>>> df['India']['PD']
0.4
>>> df['Brazil'] = [.5, .4, 'Brasilia']
>>> df
            China  India    Brazil
Population    1.5      1       0.5
PD            0.2    0.4       0.4
Capital     Beijing  Delhi  Brasilia
```

```
>>> del df['China']
>>> df
            India    Brazil
Population      1       0.5
PD            0.4       0.4
Capital     Delhi  Brasilia
```

Functions

```
>>> data = {'Country':['India', 'China', 'Brazil'],
        'Year':[2013, 2014, 2015],
        'Population':[1,1.5, .5]}
>>>
>>> df = pd.DataFrame(data)
>>> df
  Country  Population  Year
0   India         1.0  2013
1   China         1.5  2014
2  Brazil         0.5  2015
```

```
>>> df.describe()
       Population     Year
count        3.00      3.0
mean         1.00   2014.0
std          0.50      1.0
min          0.50   2013.0
25%          0.75   2013.5
50%          1.00   2014.0
75%          1.25   2014.5
max          1.50   2015.0
```

```
>>> df.sum()
Country         IndiaChinaBrazil
Population                     3
Year                        6042
dtype: object
>>> df.mean()
Population      1.0
Year         2014.0
dtype: float64
>>> df.max()
Country       India
Population      1.5
Year           2015
```

```
>>> df.head(1)
  Country  Population  Year
0   India         1.0  2013
>>> df.tail(1)
  Country  Population  Year
2  Brazil         0.5  2015
```

Pandas IO Basics - Tools

Ethans
Learn from experts

Reader functions

- *read_csv*
- *read_excel*
- *read_hdf*
- *read_sql*
- *read_json*
- *read_msgpack* (experimental)
- *read_html*
- *read_gbq* (experimental)
- *read_stata*
- *read_sas*
- *read_clipboard*
- *read_pickle*

Writer functions

- *to_csv*
- *to_excel*
- *to_hdf*
- *to_sql*
- *to_json*
- *to_msgpack* (experimental)
- *to_html*
- *to_gbq* (experimental)
- *to_stata*
- *to_clipboard*
- *to_pickle*

Read csv

```
df = pd.read_csv(r'C:\ethans\Training\Python\India_Population.csv')
```

```
>>> df.head()
         Date      Value
0  2020-12-31  1380.007
1  2019-12-31  1362.087
2  2018-12-31  1344.401
3  2017-12-31  1326.944
4  2016-12-31  1309.713
>>> df.tail()
         Date     Value
36 1984-12-31   747.000
37 1983-12-31   731.000
38 1982-12-31   715.563
39 1981-12-31   699.938
40 1980-12-31   685.688
```

```
>>> df.describe()
             Value
count    41.000000
mean   1026.812854
std     210.235406
min     685.688000
25%     847.438000
50%    1029.188000
75%    1195.063000
max    1380.007000
```

```
>>> len(df)
41
>>> df.columns
Index([u'Date', u'Value'], dtype='object')
>>> df.set_index('Date', inplace=True)
```

```
>>> df.head()
                Value
Date
2020-12-31  1380.007
2019-12-31  1362.087
2018-12-31  1344.401
2017-12-31  1326.944
2016-12-31  1309.713
```

Write html

```
>>> df.columns = ['Population']
>>> df.head()
            Population
Date
2020-12-31    1380.007
2019-12-31    1362.087
2018-12-31    1344.401
2017-12-31    1326.944
2016-12-31    1309.713
>>> df.to_html('IndiaPopulation.html')
```

```
>>> df.iloc[1]
Value    1362.087
Name: 2019-12-31 00:00:00, dtype: float64
>>> df[df.Value > 1000]
```

```
>>> df[df.Value > 1000].tail(1)
                Value
Date
1999-12-31   1010.188
```

Date	Population
2020-12-31	1380.007
2019-12-31	1362.087
2018-12-31	1344.401
2017-12-31	1326.944
2016-12-31	1309.713
2015-12-31	1292.707
2014-12-31	1275.921
2013-12-31	1259.353
2012-12-31	1243.000
2011-12-31	1217.438

Concatenating

Ethans
Learn from experts

```
import pandas as pd

df1 = pd.DataFrame({'FSI':[80,85,88,85],
                    'Interest_rate':[2, 3, 2, 2],
                    'PSR':[500, 550, 650, 650]},
                   index = [2001, 2002, 2003, 2004])

df2 = pd.DataFrame({'FSI':[80,85,88,85],
                    'Interest_rate':[2, 3, 2, 2],
                    'PSR':[709, 750, 802, 890]},
                   index = [2005, 2006, 2007, 2008])

df3 = pd.DataFrame({'FSI':[80,85,88,85],
                    'Interest_rate':[2, 3, 2, 2],
                    'Govt_circle_rate':[450, 520, 570, 590]},
                   index = [2001, 2002, 2003, 2004])
```

```
# Concanating df1 and df2, columns are common
concat = pd.concat([df1,df2])
print concat

# Concanating df1 and df3, columns are common
concat = pd.concat([df1,df3])
print concat

# Concatinating df1, df2 and df3, columns are different
concat = pd.concat([df1,df2,df3])
print(concat)
```

Appending

```
import pandas as pd

df1 = pd.DataFrame({'FSI':[80,85,88,85],
                    'Interest_rate':[2, 3, 2, 2],
                    'PSR':[500, 550, 650, 650]},
                   index = [2001, 2002, 2003, 2004])

df2 = pd.DataFrame({'FSI':[80,85,88,85],
                    'Interest_rate':[2, 3, 2, 2],
                    'PSR':[709, 750, 802, 890]},
                   index = [2005, 2006, 2007, 2008])

df3 = pd.DataFrame({'FSI':[80,85,88,85],
                    'Interest_rate':[2, 3, 2, 2],
                    'Govt_circle_rate':[450, 520, 570, 590]},
                   index = [2001, 2002, 2003, 2004])
```

```
# Same columns appending
df4 = df1.append(df2)
print(df4)

# Different columns appending
df5 = df1.append(df3)
print(df5)
```

Merging

```
#-------------------------------------------------------------------
# Merging
raw_data = {
        'subjectID': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
        'Marks': [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]}
df1 = pd.DataFrame(raw_data)
print 'Subjects and Marks ', '--' * 30
print df1

raw_data = {
        'subjectID': ['1', '2', '3', '4', '5'],
        'firstname': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
        'lastname': ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']}
df2 = pd.DataFrame(raw_data)
print 'Names and SubjectID ', '--' * 30
print df2
```

```
#Join
print 'Inner Join ', '--' * 30
print pd.merge(df1, df2, on='subjectID')

#right join
print 'Right Join', '--' * 30
print pd.merge(df1, df2, on='subjectID', how='right')

#Left join
print 'Left Join', '--' * 30
print pd.merge(df1, df2, on='subjectID', how='left')
```

Data Analysis – On movie Data

Ethans
Learn from experts

MovieLens

GroupLens Research has collected and made available rating data sets from the MovieLens web site (http://movielens.org). The data sets were collected over various periods of time, depending on the size of the set. Before using these data sets, please review their README files for the usage licenses and other details.

Help our research lab: Please take a short survey about the MovieLens datasets

MovieLens 100K Dataset

Stable benchmark dataset. 100,000 ratings from 1000 users on 1700 movies. Released 4/1998.

- README.txt
- ml-100k.zip (size: 5 MB, checksum)
- Index of unzipped files

Permalink: http://grouplens.org/datasets/movielens/100k/

Analyzing Data Set – User Data

User Data:

'UserId', 'Age', 'Sex', 'Occ', 'Zip'

```
1|24|M|technician|85711
2|53|F|other|94043
3|23|M|writer|32067
4|24|M|technician|43537
5|33|F|other|15213
6|42|M|executive|98101
7|57|M|administrator|91344
8|36|M|administrator|05201
9|29|M|student|01002
10|53|M|lawyer|90703
11|39|F|other|30329
12|28|F|other|06405
```

```
user_col = ['UserId', 'Age', 'Sex', 'Occ', 'Zip']
users = pd.read_csv('u.user', sep='|', names = user_col)
```

Data Set – Rating Data

Rating Data:

'UserId', 'MovieId', 'rating', 'timeStamp'

```
196 242 3   881250949
186 302 3   891717742
22  377 1   878887116
244 51  2   880606923
166 346 1   886397596
298 474 4   884182806
115 265 2   881171488
253 465 5   891628467
305 451 3   886324817
```

```
data_col = ['UserId', 'MovieId', 'rating', 'time']
ratingData = pd.read_csv('u.data', sep='\t', names = data_col)
```

Data Set – Movie Data

Movie Data:

'MovieId', 'title', 'release', 'videoRelease', 'url'

```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20Story%20
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?GoldenEye%20(1
3|Four Rooms (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Four%20Rooms%
4|Get Shorty (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Get%20Shorty%
5|Copycat (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Copycat%20(1995)
6|Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)|01-Jan-1995||http://us.i
```

```
movie_col = ['MovieId', 'title', 'release', 'videoRelease', 'url']
MovieData = pd.read_csv('u.item', sep='|', names = movie_col, usecols = range(5))
```

Problem Statements and Solutions

Ethans
Learn from experts

1 - Find the 5 top rated movies in the list.

```
movie_rating = pd.merge(MovieData, ratingData)
data = pd.merge(movie_rating, users)

print data.head(5)

most_rated = data.groupby('title').size().sort_values(ascending=False)[:5]
print most_rated

most_rated.plot()
show()
```

2 – Which age group users provide the maximum ratings?

```
####-------------------------------------------------------------
###2
labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79']
data['age_group'] = pd.cut(data.Age, range(0, 81, 10), right = False, labels = labels)
print data.head(5)
print data.groupby('age_group').size().sort_values(ascending=False)[:1]
```

pandas_datareader

Ethans
Learn from experts

```
__author__ = "Ethan's"

import pandas as pd
import datetime
from pandas_datareader import data
import matplotlib.pyplot as pyplot

startDate = datetime.datetime(2015, 1, 1)
endDate = datetime.datetime(2016, 1, 1)
df = data.DataReader("AAPL", "google", startDate, endDate)

print(df.head())
df['High'].plot()
pyplot.legend()
pyplot.show()
```