Pandas

- Is a open source library built on top of NumPy, that looks a lot like R

- It allows for fast analysis and data cleaning and preparation

- It can work with data from a wide variety of sources

- If you have installed Anaconda you have also Pandas already installed, otherwise you can install it from the command prompt:

  pip install pandas

Series

- A Series is very similar to a NumPy array (in fact it is built on top of the NumPy array object).

- What differentiates the NumPy array from a Series, is that a Series can have axis labels, meaning it can be indexed by a label, instead of just a number location. It also doesn't need to hold numeric data, it can hold any arbitrary Python Object.

- Labels need not be unique but must be any hashable type. The object supports both integer- and label-based indexing and provides a host of methods for performing operations involving the index.
- Statistical methods from ndarray (n-dimensional NumPy array) have been overridden to automatically exclude missing data (currently represented as NaN)
- Operations between Series (+, -, /, , *) align values based on their associated index values– they need not be the same length. The result index will be the sorted union of the two indexes.

- The Series is the building block for the dataframe, the most used Pandas datatype

Creating a Series

You can convert a list, numpy array, or dictionary to a Series:

```
import numpy as np
import pandas as pd
labels = ['a','b','c']
my_list = [10,20,30]
arr = np.array([100,200,300])
d = {'x':1000,'y':2000,'z':3000}

pd.Series(data=my_list)
0    10
1    20
2    30
dtype: int64

pd.Series(data=my_list, index=labels)
a     10
b     20
c     30
dtype: int64

pd.Series(my_list,labels)
```

Creating a Series

```
pd.Series(arr,labels)
a    100
b    200
c    300
dtype: int64

pd.Series(d)
x    1000
y    2000
z    3000
dtype: int64
```

A pandas Series can hold a variety of (also heterogeneous) object types:

```
pd.Series(data=labels)
0    a
1    b
2    c
dtype: object

pd.Series([10.2,'d',len])
0                        10.2
1                           d
2    <built-in function len>
dtype: object
```

Indexing a Series

The key to using a Series is understanding its index. Pandas makes use of these index names or numbers by allowing for fast look ups of information (works like a hash table or dictionary).

```
ser1 = pd.Series([1,2,3,4],index = ['USA', 'Germany','USSR', 'Japan'])
ser2 = pd.Series([1,2,5,4],index = ['USA', 'Germany','Italy', 'Japan'])
ser1['USA']
1
```

Operations are then also done based off of index:

```
ser1 + ser2
Germany    4.0
Italy      NaN
Japan      8.0
USA        2.0
USSR       NaN
dtype: float64
```

Dataframes

Data frames in Python are very similar to the Data frames in R: they are defined as a **two-dimensional labeled data structures** with columns of (potentially different) types.

Each row of these grids corresponds to measurements or values of an instance, while each column is a vector containing data for a specific variable. This means that data frame's rows do not need to contain, but can contain, the same type of values: they can be numeric, character, logical, etc.

We can think of a DataFrame as a bunch of Series objects put together to share the same index.

```
import pandas as pd
import numpy as np
from numpy.random import randn
np.random.seed(102)
df = pd.DataFrame(randn(5,4),index='A B C D E'.split(),columns='W X Y
Z'.split())
df
   W          X          Y          Z
A  1.668068  0.925862  1.057997 -0.920339
B  1.299748  0.331183 -0.509845 -0.903099
C -0.130016 -2.238203  0.973165 -0.024185
D -0.484928 -1.109264 -0.558975  1.042387
E -1.712263  0.136120 -0.464444  0.050980
```

Selection and indexing

```
df['W']            #Type pandas.core.series.Series
A     1.668068
B     1.299748
C    -0.130016
D    -0.484928
E    -1.712263
Name: W, dtype: float64
```

	W	X	Y	Z
A	1.668068	0.925862	1.057997	-0.920339
B	1.299748	0.331183	-0.509845	-0.903099
C	-0.130016	-2.238203	0.973165	-0.024185
D	-0.484928	-1.109264	-0.558975	1.042387
E	-1.712263	0.136120	-0.464444	0.050980

```
df[['W','Z']]      #Type pandas.core.frame.DataFrame
          W         Z
A  1.668068 -0.920339
B  1.299748 -0.903099
C -0.130016 -0.024185
D -0.484928  1.042387
E -1.712263  0.050980

df['new'] = df['W'] + df['Y']  #Creates a new column
df.drop('new',axis=1)          #Removes a column
df.drop('E', axis=0)           #Removes a row
```

The remove operations are not *in place*, unless specified, so **the last two lines do not modify df**

Selection and indexing

	W	X	Y	Z
A	1.668068	0.925862	1.057997	-0.920339
B	1.299748	0.331183	-0.509845	-0.903099
C	-0.130016	-2.238203	0.973165	-0.024185
D	-0.484928	-1.109264	-0.558975	1.042387
E	-1.712263	0.136120	-0.464444	0.050980

```
df.loc['A']       #Label selection
df.iloc[0]        #Index selection
df.ix[0]          #pandas.core.series.Series
W      1.668068
X      0.925862
Y      1.057997
Z     -0.920339
Name: A, dtype: float64
```

```
df.loc[['A','B'],['W','Y']]   #subsetting
          W         Y
A  1.668068  1.057997
B  1.299748 -0.509845
```

```
df[df>0]                      #Conditional selection with dataframe
          W         X         Y         Z
A  1.668068  0.925862  1.057997       NaN
B  1.299748  0.331183       NaN       NaN
C       NaN       NaN  0.973165       NaN
D       NaN       NaN       NaN  1.042387
E       NaN  0.136120       NaN  0.050980
```

Selection and indexing

	W	X	Y	Z
A	1.668068	0.925862	1.057997	-0.920339
B	1.299748	0.331183	-0.509845	-0.903099
C	-0.130016	-2.238203	0.973165	-0.024185
D	-0.484928	-1.109264	-0.558975	1.042387
E	-1.712263	0.136120	-0.464444	0.050980

```
df['W']<0
A    False
B    False
C     True
D     True
E     True
Name: W, dtype: bool

df[df['W']<0]                  #Conditional selection with series (no NaN)
          W         X         Y         Z
C -0.130016 -2.238203  0.973165 -0.024185
D -0.484928 -1.109264 -0.558975  1.042387
E -1.712263  0.136120 -0.464444  0.050980

df[df['W']<0][['Y','X']]
          Y         X
C  0.973165 -2.238203
D -0.558975 -1.109264
E -0.464444  0.136120

df[(df['W']<0) & (df['Y'] > 0)]    #Do NOT use and/or
          W         X         Y         Z
C -0.130016 -2.238203  0.973165 -0.024185
```

Dataframe vs Array

```
A=np.array([[1,2],[3,4]])
A[0,0]
1

df[0,0]
KeyError: (0, 0)          #df[0] is valid IF df has a column with index 0

M=df.as_matrix()
type(M)
numpy.ndarray
```

Missing data in Pandas

```
df = pd.DataFrame({'A':[1,2,np.nan],
                   'B':[5,np.nan,np.nan],
                   'C':[1,2,3]})
     A    B  C
0  1.0  5.0  1
1  2.0  NaN  2
2  NaN  NaN  3

df.dropna()
     A    B  C
0  1.0  5.0  1

df.dropna(thresh = 2)
     A    B  C
0  1.0  5.0  1
1  2.0  NaN  2

df['A'].fillna(value=df['A'].mean())
0  1.0
1  2.0
2  1.5
Name: A, dtype: float64

df['A'].fillna(value=df['A'].mean(),inplace=True)
```

Operations

```
df = pd.DataFrame({'col1':[1,2,3,4],
                   'col2':[444,555,666,444],
                   'col3':['abc','def','ghi','xyz']})

df['col2'].unique()              #Info on unique values
array([444, 555, 666])
df['col2'].nunique()             #number of unique values
3
df['col2'].value_counts()
444 2
555 1
666 1
Name: col2, dtype: int64

def times2(x): return x*2        #Applying functions (like a map method)
df['col1'].apply(times2)
0  2
1  4
2  6
3  8
Name: col3, dtype: int64

df['col1*col2'] = df.apply(lambda row: row['col1']* row['col2'], axis=1)
df['col3'].apply(len)
df['col1'].sum()
```

Groupby

```
import pandas as pd

# Create dataframe
data = {'Company':['INFN','INFN','CNR','CNR','ENEA','ENEA'],
'Person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
'Pubblications':[200,120,340,124,243,350]}
df = pd.DataFrame(data)

#You can use groupby to group ROWS toghether based off of a column name
#On this object you can perform aggregate methods that return dataframes
by_comp = df.groupby("Company")
by_comp.mean()
by_comp.std()
by_comp.min()
by_comp.max()
by_comp.count()
by_comp.describe()

#You can also group by multiple columns
```

Pivot table

```
import pandas as pd

# Create dataframe
data = {'A':['foo','foo','foo','bar','bar','bar'],
        'B':['one','one','two','two','one','one'],
        'C':['x','y','x','y','x','y'],
        'D':[1,3,2,5,4,1]}
df = pd.DataFrame(data)
df
     A    B  C  D
0  foo  one  x  1
1  foo  one  y  3
2  foo  two  x  2
3  bar  two  y  5
4  bar  one  x  4
5  bar  one  y  1

#multi index creation
df.pivot_table(values='D',index=['A', 'B'],columns=['C'])
C        x    y
A   B
bar one  4.0  1.0
    two  NaN  5.0
foo one  1.0  3.0
    two  2.0  NaN
```

Dataframe exercise!

In this exercise we will be using the SF Salaries Dataset from Kaggle:
https://www.kaggle.com/kaggle/sf-salaries
For your convenience you can find the csv file also here:
`http://bit.do/salaries-csv`
This data contains the names, job title, and compensation for San Francisco city employees on an annual basis from 2011 to 2014.

Import Pandas as pd and read salaries.csv as a dataframe called sal

Check the head of the dataframe

How many entries there are in the dataframe?

What is the average Basepay?

What is the job title of JOSEPH DRISCOLL ? (Use all caps)

What is the name of highest paid person (including benefits)?

How many unique job titles are there?

What are the top 3 most common jobs?

How many Job Titles were represented by only one person in 2013?

How many people have the word Chief in their job title?