**Pandas**

**Introduction**

* Package is developed on top of numpy
* Pandas adds data structures to deal with different format of files based on **ndarray** of numpy

**Data Structures**

* Series
  + Like a one dimensional array of similar elements
  + If the data is missing at an index it will be represented as NaN (special value introduced by numpy)
  + NaN
    - Not a Number
    - the data type of NaN is ALWAYS number
* Data Frame
  + Multi-dimensional array (table)
  + Uses Series to create data frame
  + Collection of series

**Reading Files**

* read\_csv()
  + used to read the csv (comma separated values) file
  + structure
    - the 0th row is always considered as a title row
    - the 0th row values become the title of the columns
  + parameters
    - usecols
      * list of column names while reading the files
      * read\_csv(‘f1.csv’, usecols=[‘col1’, ‘col2’] # only col1 and col2 will be read from the file
    - index\_col
      * used to assign a column as an index column
    - squeeze (True/False)
      * used to decide whether the data will returned in data frame (squeeze = False) or a Series (squeeze = True)
* json: read\_json()
* xls: read\_xls()

**Series**

* Data structure which represents one dimensional array of similar values
* Series uses numpy array internally
* In series every value is associated with its index
* Creating Series
  + From list
    - pd.Series([1, 3, 4, 5, 6])
    - pd.Series([‘test’, ‘test1’])
    - pd.Series([True, False])
  + From dictionary
    - pd.Series( { ‘name’:’p1’, age: 40 } )
  + Using index and values separately
    - n1 = [ 40, 50, 30, 20]
    - n2 = [2, 4, 1, 5]
    - s1 = pd.Series(n1, index = n2)
    - output
      * 2 40
      * 4 50
      * 1 30
      * 5 20
    - print(s1[2]) # 40
    - print(s1[5]) # 20
  + From a file
* Statistical methods
  + mean()
    - used to calculate mean of the series
  + mode()
    - used to calculate mode of the series
  + std()
    - used to calculate standard deviation of the series
  + median()
    - used to calculate median of the series
  + var()
    - used to calculate variance of the series
* Attributes
  + values
    - returns the array of values
  + index
    - returns the indices used in the series
  + dtype
    - returns the datatype of every item in the series
  + shape
    - returns the shape of series (rows and columns)
  + ndim:
    - returns the dimensions of the series
  + itemsize:
    - returns the size of the value stored in the series
    - deprecated (don’t depend on the itemsize)
* Methods
  + head()
    - used to read only few (5) lines from top
  + tail()
    - used to read only few (5) lines from bottom
  + describe()
    - returns a summary of the data (values)
    - includes
      * mode (frequently repeated element)
      * frequency of most repeated element)
      * column name
      * dtype of column name
      * number of records in the values (array)
      * unique records from the values (array)
  + sort\_values()
    - returns the values in sorted order
    - pass ascending parameter to control the order
      * ascending = True (default)
      * ascending = False
  + apply() or map()
    - process every value in the series by applying lambda functions
    - s1 = pd.Series([1, 2, 3, 4])
    - print(s1.apply(lambda x : x \* x))
    - output
      * 0 1
      * 1 4
      * 2 9
      * 3 16
* Functions
  + max()
  + min()
  + list()
  + sorted()
  + len()
* Slicing
  + s1 = pd.Series([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
  + print(s1[0]) # 10
  + print(s1[0:4]) # series: [10, 20, 30, 40]
  + print(s1[5:]) # series: [60, 70, 80, 90, 100]
  + print(s1[:5]) # series: [10, 20, 30, 40, 50]
  + print(s1[ [0, 2, 4, 6] ] # series: [10, 30, 50, 70]
  + s2 = pd.Series([10, 20, 30], index = [“test1”, “test2”, “test3”])
  + print(s2[‘test1’]) # 10
  + print(s2[[‘test1’, ‘test2]]) # series: [10 20]
  + print(s2[‘test5’]) # KeyError exception will be raised
  + print(s2[[‘test1’, ‘test5’]]) # series: [10, NaN]
* Extract values from Series
  + Refer the python/R notes for further details
* Filtering values from Series
  + Refer the python/R notes for further details

**Data Frames**

* Multi-dimensional series (array) is called as data frame
* Similar to SQL table (in memory collection of rows and columns)
  + Collection of horizontal rows
  + Vertical columns
* Creating data frame
  + By reading values from different files
    - read\_csv()
* Attributes
  + shape
  + index
  + values
  + dtypes
  + columns
  + axes
    - return a collection of axes
    - the 0th position is always related to rows
    - the 1st position is always related to column
* Methods
  + head()
  + tail()
  + describe()
  + info()
    - similar to str() in R
    - returns information about the data like
    - number of records
    - number of non-null (non-NaN) records per column
    - memory usage of the data
  + get\_dtype\_counts()
    - return datatypes of columns in the data frame
  + isnull()
    - returns if the positionth value is NaN or non-null
  + dropna()
    - drop the columns/rows having NaN values
    - parameters
      * how:
        + any: delete a row if any of the columns has NaN value
        + all: delete a row if all of the columns have NaN values
  + drop():
    - drops a column(s) from the dataframe
    - parameter:
      * axis
        + 0: row
        + 1: column
  + fillna()
    - used to replace the NaN values with the required values
  + astype()
    - used to change the data type of a column
    - e.g.

#convert the float salaries in int salaries

df[‘Salary’] = df[‘Salary’].astype(‘int’)

* + unique()
    - used to retrieve the unique values from a column
    - e.g. df.<column name>.unique()
  + sort\_values()
    - used to sort the dataframe based on the sorting order of a column
    - e.g.

# salary in ascending

df.sort\_values(by = ‘Salary’)

# salary in descending

df.sort\_values(by = ‘Salary’, ascending = False)

# Salary and Age in ascending

df.sort\_values(by = [‘Salary’, ‘Age’])

# Salary and Age in descending

df.sort\_values(by = [‘Salary’, ‘Age’], ascending = False)

# Salary in ascending and Age in descending

df.sort\_values(by = [‘Salary’, ‘Age’], ascending=[True, False])

* + rank()
    - used to sort the columns based on ranking
    - e.g.
      * df.Salary.rank() # creating ranking based on salary in ascending order
      * df.Salary.rank(ascending = False) # creating ranking based on salary in descending order
* Adding Columns
  + Using []
    - The new column will be appended at the end
    - df[‘Bonus’] = df.Salary \* 0.05
  + Using insert()
    - Inserts a new column at the required position
    - Syntax
      * <df>.insert(<position>, <column name>, value = <value>)
    - df.insert(1, ‘Bonus’, value = df.Salary \* 0.05)
* Broadcasting operations
  + Add
  + Multiply
  + Divide
  + Subtract
* Aggregated methods
  + groupby()
* Slicing of data frame
* Extract data from a data frame
  + E.g.
    - print(d1[‘Name’]) # returns a series with Name and indices
    - print(d1.Name) # returns a series with Name and indices
* Filter dataframe
  + Create an object of required condition
    - condition1 = data.Salary > 1000000
  + Filter data using the condition
    - print(data[condition1]) # all records having salary > 1000000
  + multiple conditions
    - condition1 = data.Salary > 1000000
    - condition2 = data.Age > 20.0
    - print(data[condition1 & condition2]) # all records where salary > 100000 and Age > 20
  + for string values
    - print(data[data.Name.str.contains(‘Amir’)]) # records with Amir in their names