Data Analysis with Pandas

Tushar B. Kute,
http://tusharkute.com

Pandas

- Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.
- In 2008, developer Wes McKinney started developing pandas when in need of high performance, flexible tool for analysis of data.
- Prior to Pandas, Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyze.
- Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Features of Pandas

- Fast and efficient DataFrame object with default and customized indexing.
- Tools for loading data into in-memory data objects from different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of date sets.
- Label-based slicing, indexing and subsetting of large data sets.
- Columns from a data structure can be deleted or inserted.
- Group by data for aggregation and transformations.
- High performance merging and joining of data.
- Time Series functionality.

Installation

- Package managers of respective Linux distributions are used to install one or more packages in SciPy stack.
- For Ubuntu Users
  - `sudo apt-get install python-numpy python-pandas python-sympy python-nose`
- For Fedora Users
  - `sudo yum install numpyscipy python-matplotlibipython python-pandas sympy python-nose atlas-devel`

Pandas data structures

- Pandas deals with the following two data structures –
  - Series
  - DataFrame
- These data structures are built on top of Numpy array, which means they are fast.

Pandas data structures

- The best way to think of these data structures is that the higher dimensional data structure is a container of its lower dimensional data structure.
- For example, DataFrame is a container of Series
- Series
  - 1D labeled homogeneous array, size- immutable.
- Data Frames
  - General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.

Series

- Series is a one-dimensional array like structure with homogeneous data. For example, the following series is a collection of integers 10, 23, 56, …

10	23	56	17	52	61	73	90	26	72

- Key Points
  - Homogeneous data
  - Size Immutable
  - Values of Data Mutable

DataFrame

- DataFrame is a two-dimensional array with heterogeneous data. For example,

Roll	Name	Marks
1	Aneet	67.56
2	Asha	59.05
3	Anil	71.22

- The data is represented in rows and columns. Each column represents an attribute and each row represents a person.

Data types of columns

- The data types of the three columns are as follows –

Column	Type
Roll	Integer
Name	String
Marks	Float

- Key Points

  Heterogeneous data

  Size Mutable

  Data Mutable

Series object

- Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.).
- The axis labels are collectively called index.

```
pandas.Series
```

- A pandas Series can be created using the following constructor –

```
pandas.Series( data, index, dtype, copy)
```

Creating a Series

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
sr = pd.Series()
print sr

data = np.array(['a','b','c','d'])
sr = pd.Series(data)
print sr
```

Creating a Series

```
# Series using index
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[100,101,102,103])
print s

# Series from dict
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
print s

# Series from Scalar
s = pd.Series(5, index=[0, 1, 2, 3])
print s
```

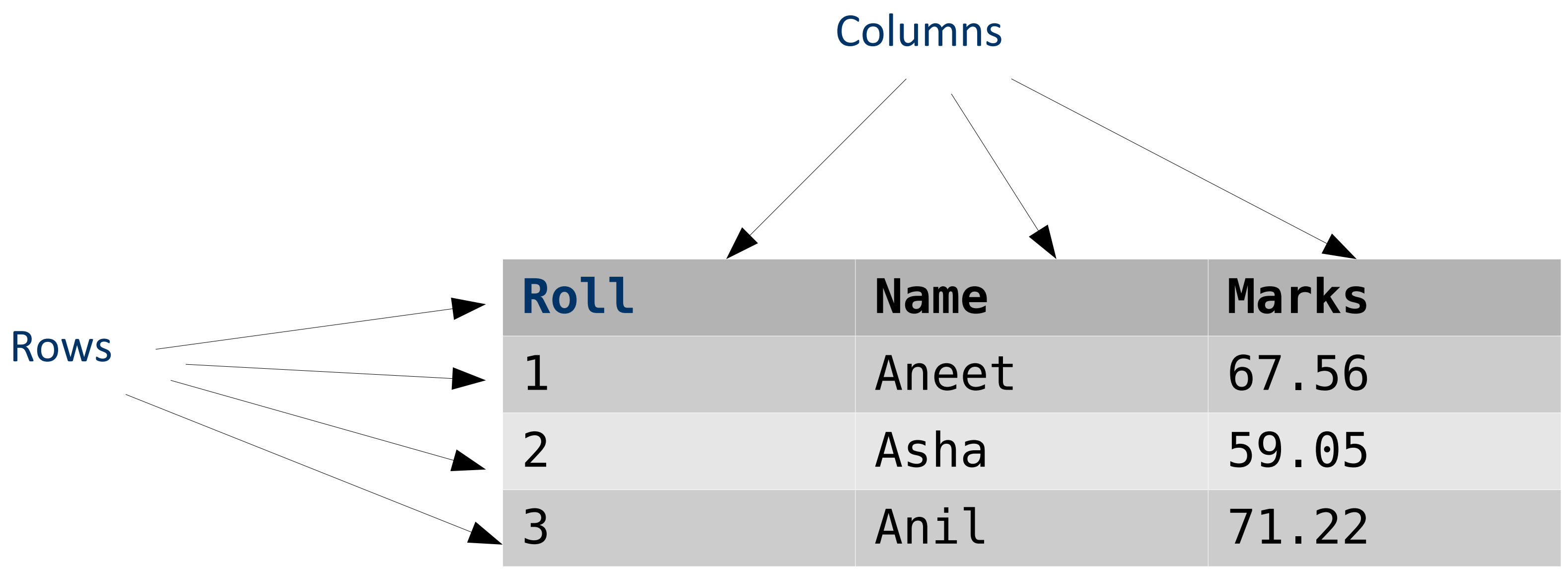
Accessing Series elements

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
#retrieve the first element
print s[0]
#retrieve the first three element
print s[:3]
#retrieve the last three element
print s[-3:]
#retrieve a single element
print s['a']
#retrieve multiple elements
print s[['a','c','d']]
#retrieve single elements [error]
print s['f']
```

DataFrames

- A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.
- Features of DataFrame
  - Potentially columns are of different types
  - Size – Mutable
  - Labeled axes (rows and columns)
  - Can Perform Arithmetic operations on rows and columns

DataFrame Structure



Roll	Name	Marks
1	Aneet	67.56
2	Asha	59.05
3	Anil	71.22

Creating DataFrames

- A pandas DataFrame can be created using the following constructor –

  `pandas.DataFrame( data, index, columns, dtype, copy)`
- A pandas DataFrame can be created using various inputs like –
  - Lists
  - dict
  - Series
  - Numpy ndarrays
  - Another DataFrame

Creating DataFrames

```
import pandas as pd
# Empty dataframe
df = pd.DataFrame()
print df

# Create df from list
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print df

# Create df from list
data = [['Ashok',10],['Ana',12],['Asha',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print df
```

Creating DataFrames

```
# Create df from list
data = [['Ashok',10],['Ana',12],['Asha',13]]
df = pd.DataFrame(data,columns=['Name','Age'], dtype=float)
print df

# Create df from dict
data = {'Name':['Ashok','Ana','Anil'],'Age':[28,34,29]}
df = pd.DataFrame(data)
print df

data = {'Name':['Ashok','Ana','Anil'],'Age':[28,34,29]}
df = pd.DataFrame(data, index = ['row1','row2','row3'])
print df

# Create df from list of dict
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print df
```

Creating DataFrames

```
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]

#With two column indices, values same as dictionary keys
df1 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b'])

#With two column indices with one index with other name
df2 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b1'])
print df1
print df2
```

Row selection

```
# Row selection by location
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([5, 6, 7, 8], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print df.loc['b']

# Row selection by integer location
print df.iloc[2]
# Slicing
print df[2:4]
print df[:4]
print df[2:]
```

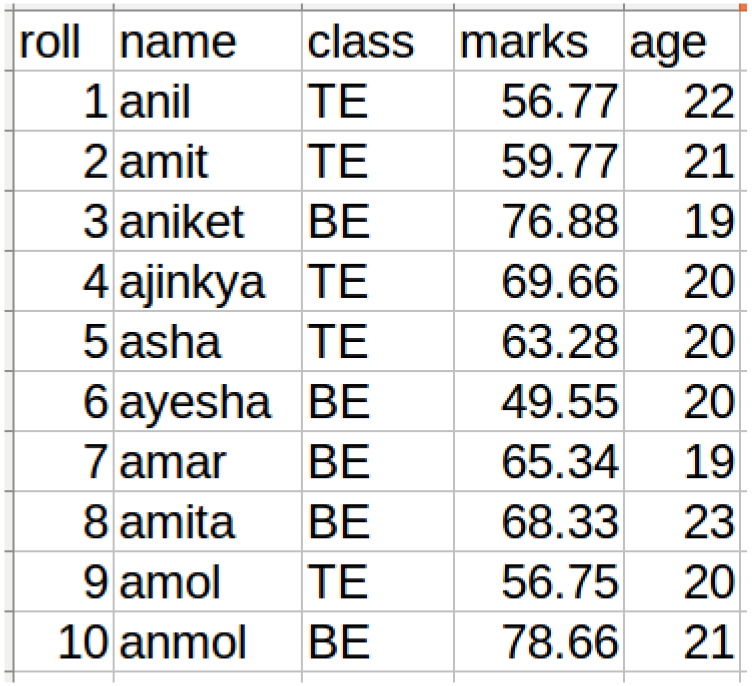
Row deletion

```
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])
# Additions of rows
df = df.append(df2)
print df

# Drop rows with label 0
df = df.drop(0)
print df
```

Reading a csv file

- Sample: student.csv



roll	name	class	marks	age
1	anil	TE	56.77	22
2	amit	TE	59.77	21
3	aniket	BE	76.88	19
4	ajinkya	TE	69.66	20
5	asha	TE	63.28	20
6	ayesha	BE	49.55	20
7	amar	BE	65.34	19
8	amita	BE	68.33	23
9	amol	TE	56.75	20
10	anmol	BE	78.66	21

```
df = read_csv ('student.csv')
```

Reading a csv file

```
import pandas as pd
import numpy as np
df=pd.read_csv("student.csv")
print df

df=pd.read_csv("student.csv",index_col=['roll'])
print df

df=pd.read_csv("student.csv", dtype={'age': np.float64})
print df

df=pd.read_csv("student.csv", names=['a','b','c','d','e'])
print df
```

DataFrame attributes

```
df=pd.read_csv("student.csv")
print df
# Print Transpose
print "Transpose:\n", df.T
# Print axes
print "Axes: \n", df.axes
# Print data types
print "Dtypes: \n", df.dtypes
# Print empty
print "Empty: ", df.empty
# Print dimensions
print "Dimensions: ", df.ndim
# Print dimensionality
print "Dimensionality: ", df.shape
# Print total elements
print "Total Elements: ", df.size
# Print Values
print "Values: ", df.values
```

DataFrame functions

```
# First 5 elements
print df.head()
# First 3 elements
print df.head(3)
# Last 5 elements
print df.tail()
# Last 2 elements
print df.tail(2)
# Addition of all column elements
print "Sum:\n", df.sum()
# Addition of all row elements
print "Sum:\n", df.sum(axis=1)
```

DataFrame: Functions

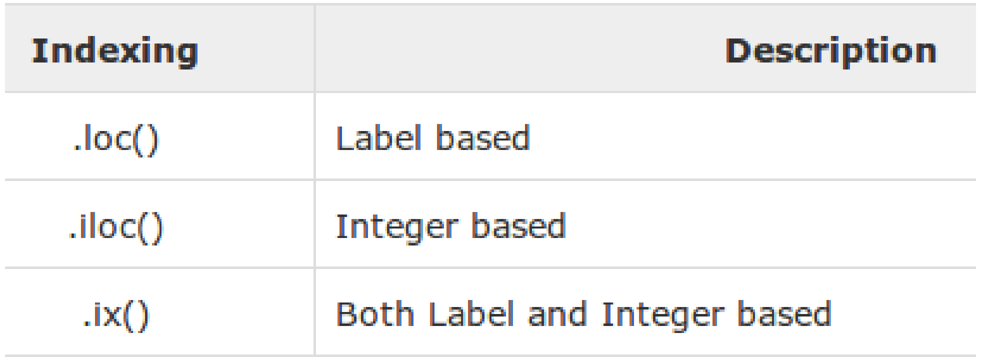
S.No.	Function	Description
1	count()	Number of non-null observations
2	sum()	Sum of values
3	mean()	Mean of Values
4	median()	Median of Values
5	mode()	Mode of values
6	std()	Standard Deviation of the Values
7	min()	Minimum Value
8	max()	Maximum Value
9	abs()	Absolute Value
10	prod()	Product of Values
11	cumsum()	Cumulative Sum
12	cumprod()	Cumulative Product

Summarizing

```
# Summary of numbers
print "Summary:\n", df.describe()
# Summary of strings
print "Summary:\n", df.describe(include=['object'])
# Summary of all
print "Summary:\n", df.describe(include='all')
```

Indexing and slicing

- The Python and NumPy indexing operators "[ ]" and attribute operator "." provide quick and easy access to Pandas data structures across a wide range of use cases.
- However, since the type of the data to be accessed isn’t known in advance, directly using standard operators has some optimization limits.
- Pandas now supports three types of Multi-axes indexing; the three types are mentioned in the following table –



Indexing	Description
.loc()	Label based
.iloc()	Integer based
.ix()	Both Label and Integer based

.loc()

- Pandas provide various methods to have purely label based indexing. When slicing, the start bound is also included. Integers are valid labels, but they refer to the label and not the position. .loc() has multiple access methods like –
  - A single scalar label
  - A list of labels
  - A slice object
  - A Boolean array
- loc takes two single/list/range operator separated by ','. The first one indicates the row and the second one indicates columns.

Example:

```
import pandas as pd
import numpy as np

df=pd.read_csv("student.csv")
print df

print df.loc[:,'name']
print df.loc[:,['name','marks']]
print df.loc[[2,3,5,6,9],'name']
print df.loc[[2,3,6,9],['name','age']]
print df.loc[2:6]
print df.loc[3] > 60
```

.iloc()

- Pandas provide various methods in order to get purely integer based indexing. Like python and numpy, these are 0-based indexing.
- The various access methods are as follows:
  - An Integer
  - A list of integers
  - A range of values

Example:

```
df=pd.read_csv("student.csv")
print df

print df.iloc[:4]
print df.iloc[3:8]
print df.iloc[2:6,1:3]
print df.iloc[:7,:]
print df.iloc[:7,2]
print df.iloc[2:7,3]
print df.iloc[:,:3]
print df.iloc[[3,4,8,9],[2,3]]
```

Table wise function

```
df=pd.read_csv("student.csv")
df = df.loc[:,['age','marks']]

def adder(ele1,ele2):
    return ele1+ele2

print "Original:\n", df
print "Adding :\n", df.pipe(adder,2)
print "Mean:\n", df.apply(np.mean)

print "Row-wise:\n", df.apply(np.mean, axis=1)

print "Lambda Expression:"
print df.apply(lambda x: x.max() - x.min())
print df.apply(np.mean)
```

Apply function

```
# My custom function
print df['marks'].map(lambda x:x/10.0)
print df.apply(np.mean)

print "Apply Map:\n", df.applymap(lambda x:x*10)
print df.apply(np.mean)
```

Renaming and reindexing

```
df=pd.read_csv("student.csv")
print df
# Re-indexing
df_r = df.reindex(index=[0,3,8], columns=['roll','name','marks'])
print df_r

# Re-naming
print ("After renaming the rows and columns:")
print df.rename(columns={'roll' : 'id', 'marks' : 'cgpa'},
index = {0 : 'Zeroth', 1 : 'First', 2 : 'Second'})
```

Iterating the dataframe

- The behavior of basic iteration over Pandas objects depends on the type. When iterating over a Series, it is regarded as array-like, and basic iteration produces the values.
- Other data structures, like DataFrame and Panel, follow the dict-like convention of iterating over the keys of the objects.
- In short, basic iteration (for i in object) produces –
  - Series – values
  - DataFrame – column labels

Iterating through dataframes

```
df=pd.read_csv("student.csv")
print df

print "Column Names:"
for col in df:
    print col,

print "Individual Columns"
for key,value in df.iteritems():
    print key,"\n", value

print "Individual Rows"
for row_index,row in df.iterrows():
    print row_index,row

print "Individual Tuples"
for row in df.itertuples():
   print row
```

Sorting

- There are two kinds of sorting available in Pandas. They are –
  - By label
  - By Actual Value

Sorting by label

```
df=pd.read_csv("student.csv")
print df

# Sorting with row index
sorted_df=df.sort_index()
print sorted_df
# Sorting with row index with ascending
sorted_df=df.sort_index(ascending=False)
print sorted_df
# Sorting with column names
sorted_df=df.sort_index(axis=1)
print sorted_df
```

Sorting by values

```
df=pd.read_csv("student.csv")
print df

# Sort by column names
sorted_df = df.sort_values(by='name')
print sorted_df

# Sort by multiple column names
sorted_df = df.sort_values(by=['age','name',])
print sorted_df

# Sort by column names
sorted_df = df.sort_values(by='marks',kind='mergesort')
print sorted_df
```

Functions

- `lower()`
  - Converts strings in the Series/Index to lower case.
- `upper()`
  - Converts strings in the Series/Index to upper case.
- `len()`
  - Computes String length().
- `strip()`
  - Helps strip whitespace(including newline) from each string in the Series/index from both the sides.
- `split(' ')`
  - Splits each string with the given pattern.
- `cat(sep=' ')`
  - Concatenates the series/index elements with given separator.

Functions

- **`get_dummies()`**
  - Returns the DataFrame with One-Hot Encoded values.
- **`contains(pattern)`**
  - Returns a Boolean value True for each element if the substring contains in the element, else False.
- **`replace(a,b)`**
  - Replaces the value a with the value b.
- **`repeat(value)`**
  - Repeats each element with specified number of times.
- **`count(pattern)`**
  - Returns count of appearance of pattern in each element.
- **`startswith(pattern)`**
  - Returns true if the element in the Series/Index starts with the pattern.

Functions

- **`endswith(pattern)`**
  - Returns true if the element in the Series/Index ends with the pattern.
- **`find(pattern)`**
  - Returns the first position of the first occurrence of the pattern.
- **`findall(pattern)`**
  - Returns a list of all occurrence of the pattern.
- **`swapcase()`**
  - Swaps the case lower/upper.
- **`islower() /  isupper()  /  isnumeric()`**
  - Checks whether all characters in each string in the Series/Index in lower case / upper case / numeric or not. Returns Boolean

Functions

```
import pandas as pd
import numpy as np

df=pd.read_csv("student.csv")
print df

print 'Upper names:\n', df['name'].str.upper()
print 'Lengths:\n', df['name'].str.len()
print 'Strip:\n', df['name'].str.strip()
print 'Split:\n', df['name'].str.split(' ')
print 'Cat:\n', df['name'].str.cat(sep='_')
print 'Dummies:\n', df['class'].str.get_dummies()
print 'Contains:\n', df['name'].str.contains('i')
```

Functions

```
print 'Replace\n', df['name'].str.replace('a','A')
print 'Repeat:\n', df['class'].str.repeat(2)
print 'Count:\n', df['name'].str.count('a')
print 'Starts:\n', df['name'].str.startswith('a')
print 'Ends:\n', df['name'].str.endswith('a')
print 'Find:\n', df['name'].str.find('e')
print 'Findall:\n', df['name'].str.findall('e')
print 'Swapcase:\n', df['name'].str.swapcase()
print 'Is Upper:\n', df['name'].str.isupper()
print 'Is Lower:\n', df['name'].str.islower()
print 'Is Numeric:\n', df['name'].str.isnumeric()
```

Options and Customization

- Pandas provide API to customize some aspects of its behavior, display is being mostly used.
- The API is composed of five relevant functions. They are –
  - get_option()
  - set_option()
  - reset_option()
  - describe_option()
  - option_context()

Example and use

```
import pandas as pd
print pd.get_option("display.max_rows")
print pd.get_option("display.max_columns"
pd.set_option("display.max_rows",80)
print pd.get_option("display.max_rows"))
pd.set_option("display.max_columns",30)
print pd.get_option("display.max_columns")
pd.reset_option("display.max_rows")
print pd.get_option("display.max_rows")
pd.describe_option("display.max_rows")
```

Creating DataFrames

- Statistical methods help in the understanding and analyzing the behavior of data.
  - pct_change()
  - cov()
  - corr()
  - rank()

Percentage change

- Series, DataFrames and Panel, all have the function pct_change().
- This function compares every element with its prior element and computes the change percentage.
- By default, the pct_change() operates on columns; if you want to apply the same row wise, then use axis=1() argument.

```
print df['marks'].pct_change()
```

Covariance

- Covariance is applied on series data. The Series object has a method cov to compute covariance between series objects.
- NA will be excluded automatically.
- Covariance method when applied on a DataFrame, computes cov between all the columns.

```
print df['age'].cov(df['marks'])
```

Correlation

- Correlation shows the linear relationship between any two array of values (series). There are multiple methods to compute the correlation like pearson(default), spearman and kendall.
- If any non-numeric column is present in the DataFrame, it is excluded automatically.

```
print df['age'].corr(df['marks'])
print df.corr()
```

Data Ranking

- Data Ranking produces ranking for each element in the array of elements. In case of ties, assigns the mean rank.
- Rank optionally takes a parameter ascending which by default is true; when false, data is reverse-ranked, with larger values assigned a smaller rank.
- Rank supports different tie-breaking methods, specified with the method parameter –
  - average – average rank of tied group
  - min – lowest rank in the group
  - max – highest rank in the group
  - first – ranks assigned in the order they appear in the array

```
print df.rank()
```

Rolling

- This function can be applied on a series of data. Specify the window=n argument and apply the appropriate statistical function on top of it.

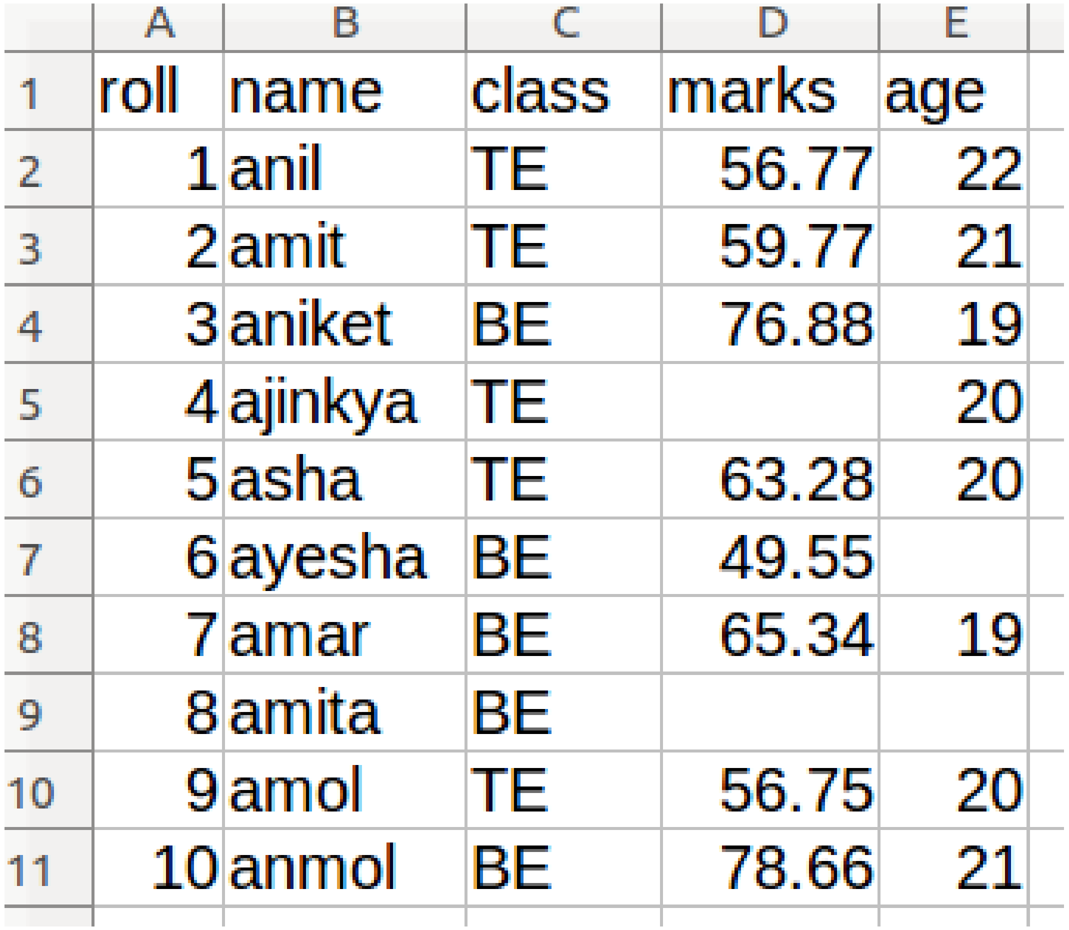
```
print df[['age','marks']].rolling(window=3).mean()
```

- Since the window size is 3, for first two elements there are nulls and from third the value will be the average of the n, n-1 and n-2 elements. Thus we can also apply various functions as mentioned above.

Missing Data

- Missing data is always a problem in real life scenarios.
- Areas like machine learning and data mining face severe issues in the accuracy of their model predictions because of poor quality of data caused by missing values.
- In these areas, missing value treatment is a major point of focus to make their models more accurate and valid.

Example:



	A	B	C	D	E
1	roll	name	class	marks	age
2	1	anil	TE	56.77	22
3	2	amit	TE	59.77	21
4	3	aniket	BE	76.88	19
5	4	ajinkya	TE		20
6	5	asha	TE	63.28	20
7	6	ayesha	BE	49.55	
8	7	amar	BE	65.34	19
9	8	amita	BE		
10	9	amol	TE	56.75	20
11	10	anmol	BE	78.66	21

Multiple filtering

```
filter1 = (m["rating"] > 3.6) & (m["year"] > 1990)
filter1

m[filter1]
```

Data Ranking

- Data Ranking produces ranking for each element in the array of elements. In case of ties, assigns the mean rank.
- Rank optionally takes a parameter ascending which by default is true; when false, data is reverse-ranked, with larger values assigned a smaller rank.
- Rank supports different tie-breaking methods, specified with the method parameter –
  - average – average rank of tied group
  - min – lowest rank in the group
  - max – highest rank in the group
  - first – ranks assigned in the order they appear in the array

```
print df.rank()
```

Rolling

- This function can be applied on a series of data. Specify the window=n argument and apply the appropriate statistical function on top of it.

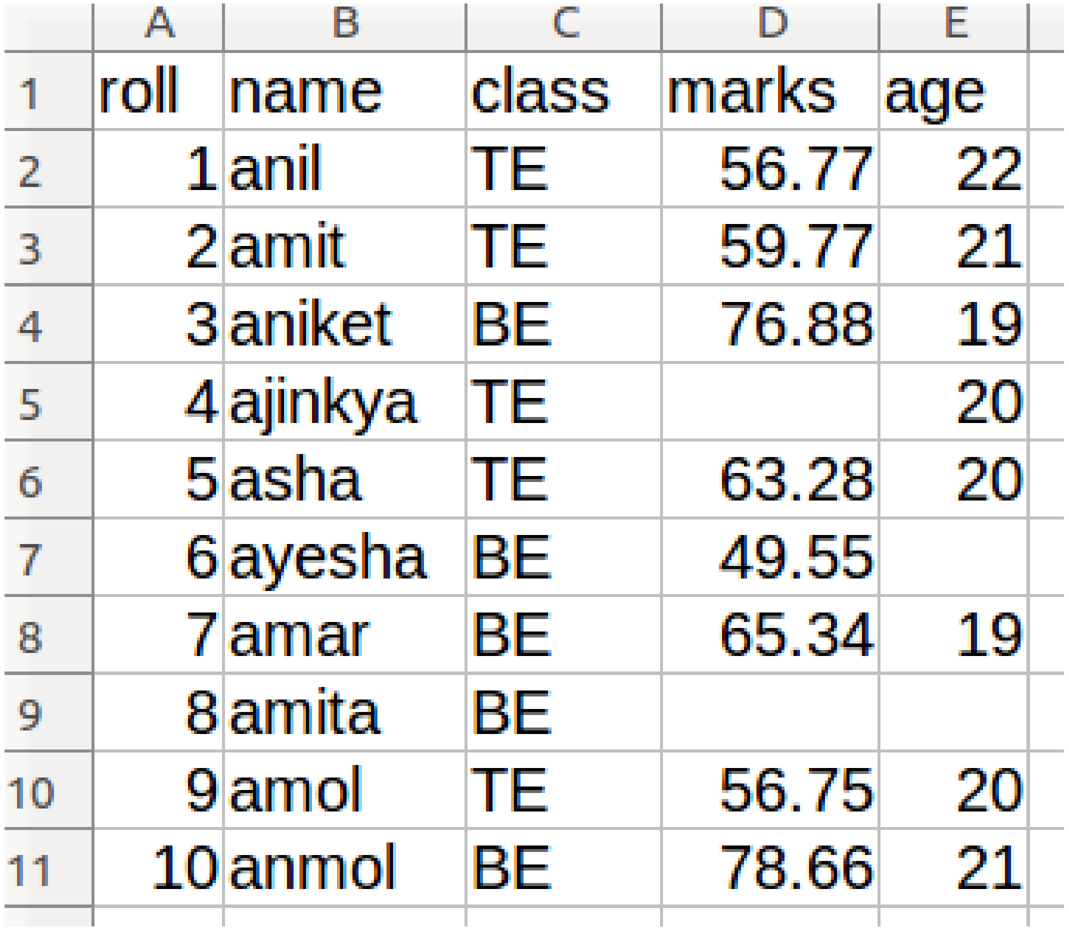
```
print df[['age','marks']].rolling(window=3).mean()
```

- Since the window size is 3, for first two elements there are nulls and from third the value will be the average of the n, n-1 and n-2 elements. Thus we can also apply various functions as mentioned above.

Missing Data

- Missing data is always a problem in real life scenarios.
- Areas like machine learning and data mining face severe issues in the accuracy of their model predictions because of poor quality of data caused by missing values.
- In these areas, missing value treatment is a major point of focus to make their models more accurate and valid.

Example:

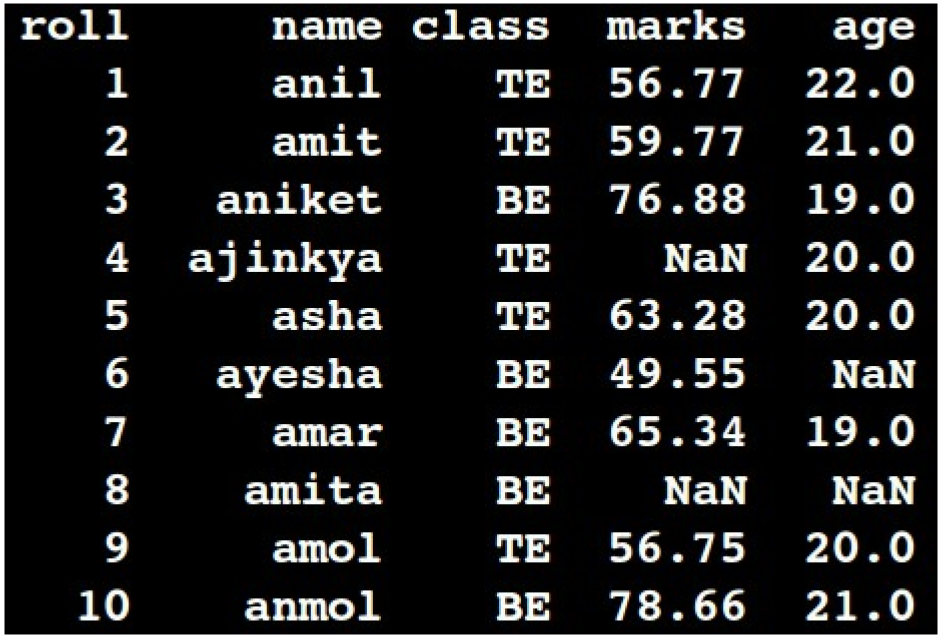


	A	B	C	D	E
1	roll	name	class	marks	age
2	1	anil	TE	56.77	22
3	2	amit	TE	59.77	21
4	3	aniket	BE	76.88	19
5	4	ajinkya	TE		20
6	5	asha	TE	63.28	20
7	6	ayesha	BE	49.55	
8	7	amar	BE	65.34	19
9	8	amita	BE		
10	9	amol	TE	56.75	20
11	10	anmol	BE	78.66	21

Reading the file

```python
import pandas as pd
import numpy as np

df=pd.read_csv("students.csv")
print df
```

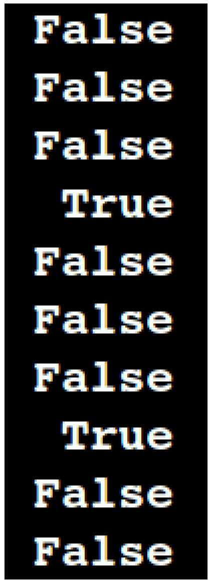


```
roll      name class  marks   age
   1      anil    TE  56.77  22.0
   2      amit    TE  59.77  21.0
   3    aniket    BE  76.88  19.0
   4   ajinkya    TE    NaN  20.0
   5      asha    TE  63.28  20.0
   6    ayesha    BE  49.55   NaN
   7      amar    BE  65.34  19.0
   8     amita    BE    NaN   NaN
   9      amol    TE  56.75  20.0
  10     anmol    BE  78.66  21.0
```

Checking for missing values

- To make detecting missing values easier (and across different array dtypes), Pandas provides the isnull() and notnull() functions, which are also methods on Series and DataFrame objects –

```
print df['marks'].isnull()
```



```
False
False
False
 True
False
False
False
 True
False
False
```

Cleaning or filling missing data

- Pandas provides various methods for cleaning the missing values. The fillna function can “fill in” NA values with non-null data in a couple of ways, which we have illustrated in the following sections.
- Replace NaN with a Scalar Value such as 0.

Missing Data

```
# Print sum of marks
print df['marks'].sum()

# Fill all NaN elements with 0
print df.fillna(0)

# Previous value will be replaced with NaN
print df.fillna(method='pad')

# Next value will be replaced with NaN
print df.fillna(method='backfill')

# Delete all NaN values
print df.dropna()

# Replacing values old:new
print df.replace({'TE':'TY','BE':'Final'})
```

Groupby

- Any groupby operation involves one of the following operations on the original object. They are –
  - Splitting the Object
  - Applying a function
  - Combining the results
- In many situations, we split the data into sets and we apply some functionality on each subset. In the apply functionality, we can perform the following operations –
  - Aggregation – computing a summary statistic
  - Transformation – perform some group-specific operation
  - Filtration – discarding the data with some condition

Thank you

@mitu_skillologies @mITuSkillologies @mitu_group @mitu-skillologies @MITUSkillologies

kaggle
@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in
tushar@tusharkute.com