Lab Session II
Introduction to Pandas

DR. JASMEET SINGH
ASSISTANT PROFESSOR, CSED
TIET

Introduction to Pandas

- **Pandas** is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- It supports two basic data structures
  - Pandas Series
  - Pandas DataFrame
- Pandas can be installed as: pip install pandas
- Once it has been installed, it can be imported as: import pandas as pd

Pandas 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.
- The basic method to create a Series is to call:

```
pandas.Series(data, index=index)
```

- Here, data can be many different things:
  - a Python dict
  - an ndarray
  - a scalar value (like 5)
- The passed index is a list of axis labels. Thus, this separates into a few cases depending on what data is.

Pandas Series (Contd….)

- **From ndarray**
  - If data is an ndarray, index must be the same length as data.
  - If no index is passed, one will be created having values [0, ..., len(data) - 1]
  - Examples: s = pd.Series(np.random.rand(5), index=["a", "b", "c", "d", "e"]),
    pd.Series(np.random.randn(5))
- **From dictionary**
  - When the data is a dict, and an index is not passed, the Series index will be ordered by the dict's insertion order.
  - If an index is passed, the values in data corresponding to the labels in the index will be pulled out.
  - Examples: d = {"a": 0.0, "b": 1.0, "c": 2.0}, pd.Series(d, index=["b", "c“, "d", "a"])

Pandas Series (Contd….)

- **From scaler values**
  - If data is a scalar value, an index must be provided.
  - The value will be repeated to match the length of index.
  - pd.Series(5.0, index=["a", "b", "c", "d", "e"])

Pandas Series is ndarray-like

- Pandas series acts very similarly to a ndarray, and is a valid argument to most NumPy functions.
- However, operations such as slicing will also slice the index.
- Examples:
  - s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
  - np.mean(s)
  - s[s>np.mean(s)]
  - s[:3]
- While Series is ndarray-like, if you need an actual ndarray, then use Series.to_numpy(). For example , s.to_numpy()

Pandas Series is dict-like

- A Series is like a fixed-size dict in that we can get and set values by index label.
- Examples,
  - s=pd.Series({"a": 0.0, "b": "e", "c": 2.0})
  - s["a"]
  - s['c']=5
  - "e" in s

Pandas DataFrame

- DataFrame is a 2-dimensional labeled data structure with columns of potentially different types.
- We can think of it like a spreadsheet or SQL table, or a dict of Series objects.
- It is generally the most commonly used pandas object.
- Like Series, DataFrame accepts many different kinds of input:
  - Dict of 1D ndarrays, lists, dicts, or Series
  - 2-D numpy.ndarray
  - A Series
  - Another DataFrame
  - Reading files from disk

Pandas DataFrame

- A Pandas DataFrame is created using following syntax:
- Syntax: pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=None)
  - data can be Dict of 1D ndarrays, lists, dicts, or Series, 2D numpy arrays, series.
  - Index: index or array-like: index to use for resulting frame. Will default to RangeIndex if no indexing information part of input data and no index provided.
  - Columns: index or array-like: Column labels to use for resulting frame when data does not have them, defaulting to RangeIndex(0, 1, 2, …, n). If data contains column labels, will perform column selection instead.
  - dtype:dtype, default None: Data type to force. Only a single dtype is allowed. If None, infer.
  - copy: bool or None, default None: Copy data from inputs. For dict data, the default of None behaves like copy=True. For DataFrame or 2d ndarray input, the default of None behaves like copy=False.

Reading Files From Disk

- The pandas I/O API is a set of top level reader functions accessed like pandas.read_csv() that generally return a pandas object. The corresponding writer functions are object methods that are accessed like DataFrame.to_csv().
- Below is a table containing available readers and writers.

Format	Data Description	Reader	writer
Text	csv	read_csv	to_csv
Text	JSON	read_json	to_json
Text	HTML	read_html	to_html
Text	XML	read_xml	to_xml
Binary	MS Excel	read_excel	to_excel
Binary	Pickle	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql

read_csv

- read_csv is the most common method used to read datasets in Data Analytics and Machine learning experiments.
- The most common attributes used with read_csv are:
  - filepath_or_buffer: *various*
  - *sep: str, defaults to ',' for read_csv(), \t for read_table()*
  - *header: int, default 'infer'*
  - names: array-like, default None
  - usecols: list-like or callable, default None
  - skiprows: list-like or integer, default None
  - nrows: int, default None

to_csv

- The Series and DataFrame objects have an instance method to_csv which allows storing the contents of the object as a comma-separated-values file.
- The function takes a number of arguments. Only the first is required.
  - path_or_buf: A string path to the file to write or a file object.
  - sep : Field delimiter for the output file (default “,”)
  - columns: Columns to write (default None)
  - header: Whether to write out the column names (default True)
  - index: whether to write row (index) names (default True)
  - encoding: a string representing the encoding to use if the contents are non-ASCII, for Python versions prior to 3

Pandas DataFrame Attributes

Attribute	Function	Example
at	Access a single value for a row/column label pair.	df = pd.DataFrame([[0, 2, 3], [0, 4, 1], [10, 20, 30]],index=[4, 5, 6], columns=['A', 'B', 'C']) df.at[4, 'B']
iat	Access a single value for a row/column pair by integer position.	df.iat[1,1]
axes	Return a list representing the axes of the DataFrame.	df.axes
columns	Returns the column labels of the DataFrame	df.columns
dtypes	Return the dtypes in the DataFrame	df.dtypes
iloc	Purely integer-location based indexing for selection by position	df.iloc[1], df.iloc[0:2], df.iloc[0:2,1]
loc	Access a group of rows and columns by label(s) or a boolean array.	df.loc[6,'C']

Pandas DataFrame Attributes

Attribute	Function	Example
index	The index (row labels) of the DataFrame.	df.index
ndim	Return an int representing the number of axes / array dimensions	df.ndim
shape	Return a tuple representing the dimensionality of the DataFrame.	df.shape
size	Return an int representing the number of elements in this object.	df.size
values	Return a Numpy representation of the DataFrame	df.values

Indexing and Iteration

- Along with at, iat, loc, and iloc used in indexing following methods are also used for iterating the DataFrame. The most common functions are explained below:

Function	Description	Syntax
Head	Return the first n rows.	DataFrame.head(***n*=5**)
Tail	Return the last n rows	DataFrame.tail(n=5)
Insert	Insert column into DataFrame at specified location	DataFrame.insert(loc, column, value, allow_duplicates=False)
Items	Iterate over (column name, Series) pairs.	DataFrame.items()
Iteritems	Iterate over (column name, Series) pairs.	DataFrame.iteritems()
Keys	This gives index for Series, columns for DataFrame.	DataFrame.keys()
Iterrows	Iterate over DataFrame rows as (index, Series) pairs.	DataFrame.iterrows()

Indexing and Iteration

Function	Description	Syntax
Where	Replace values where the condition is False.	DataFrame.where(cond, other=nan, inplace=False) Example: df = pd.DataFrame(np.arange(10).reshape(-1, 2), columns=['A', 'B']) df.where(df%3==0, -df)
Row Selection	Conditional Row Selection	df[condition] df[df['Sex']=='female'] or df[(df['Sex']=='female') &(df['Age']>='65')]
Unique values	Unique Values	print(df['Sex'].unique()),print(df['Sex'].nunique()) print(df['Sex'].value_counts())
Drop	Deleting Columns	df.drop(['Age'],axis=1) df.drop(df.columns[1],axis=1)
Delete Rows	Deleting rows/duplicate rows	df[df['Sex']!='male'] or df.drop_duplicates()

Binary Operations

Function	Description	Syntax	Example
add	Get Addition of dataframe and other, element-wise.	DataFrame.add(other, axis='columns', level=None, fill_value=None)	df = pd.DataFrame({'angles': [0, 3, 4],'degrees': [360, 180, 360]}) df.add(1)
subtract	Get Subtraction of dataframe and other, element-wise	DataFrame.sub(other[, axis, level, fill_value]	df.subtract(1)
multiplication	Get Multiplication of dataframe and other, element-wise	DataFrame.mul(other[, axis, level, fill_value])	df[‘angles’].mul(2)
Divison	Get Floating division of dataframe and other, element-wise	DataFrame.div(other[, axis, level, fill_value])	df['angles']=df['angles'].div(2)
Mod	Get Modulo of dataframe and other, element-wise	DataFrame.mod(other[, axis, level, fill_value])	df['degrees'].mod(df['angles'])
Power	Get Exponential power of dataframe and other, element-wise	DataFrame.pow(other[, axis, level, fill_value])	df['degrees'].pow(2)
Dot	Compute the matrix multiplication between the DataFrame and other.	DataFrame.dot(other)	df['angles'].dot(df['degrees'])

Binary Operations (Contd.....)

Function	Description	Syntax
Greater than	Get greater than of dataframe and other, element-wise	DataFrame.lt(other[, axis, level])
Less than	Get Less than of dataframe and other, element-wise	DataFrame.lt(other[, axis, level])
Less than equal to	Get Less than equal to of dataframe and other, element-wise	DataFrame.le(other[, axis, level])
Greater than equal to	Get Greater than or equal to of dataframe and other, element-wise	DataFrame.ge(other[, axis, level])
Not equal to	Get Not equal to of dataframe and other, element-wise	DataFrame.ne(other[, axis, level])
Equal to	Get Equal to of dataframe and other, element-wise	DataFrame.eq(other[, axis, level])

Function application & GroupBy

Function	Description	Syntax	Example
Apply	Apply a function along an axis of the DataFrame.	DataFrame.apply(func, axis=0, raw=False, result_type=None)	df.apply(np.sqrt)
Applymap	Apply a function to a Dataframe elementwise	DataFrame.applymap(func, na_action=None)	df.applymap(lambda x: len(str(x)))
Agg or Aggregate	Aggregate using one or more operations over the specified axis	DataFrame.agg(func=None, axis=0)	df.agg('min')
Groupby	Group DataFrame using a mapper or by a Series of columns.	DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True)	df.groupby(by='Pclass').mean()

Binning/Discretization

- The Binning method is used for data smoothing.
- In this method the data is first sorted and then the sorted values are distributed into a number of *buckets* or *bins*.
- pndas.cut is used for binning- Bin values into discrete intervals.

Syntax: pandas.cut(x, bins, right=True, labels=None, retbins=False, precision=3, include_lowest=False, duplicates='raise')

Example:

```
df=pd.read_csv('C:/Users/jasme/Desktop/titanic.csv')
bins=np.linspace(min(df['Age']),max(df['Age']),4)
group_names=['child','young','old']
df['Age']=pd.cut(df['Age'],bins,labels=group_names,include_lowest=True)
```