Programming

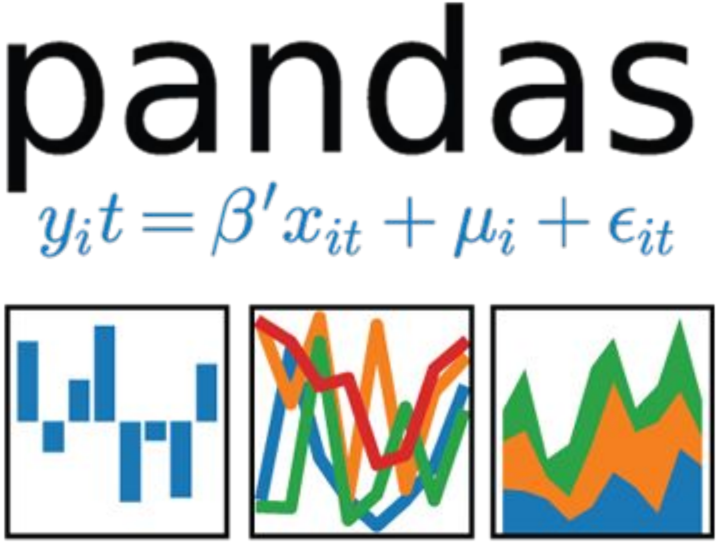
with python™

By

Rami Tailakh

Senior Software Engineer and Data Science Practitioner
MSc in Applied Computing and Information Technology

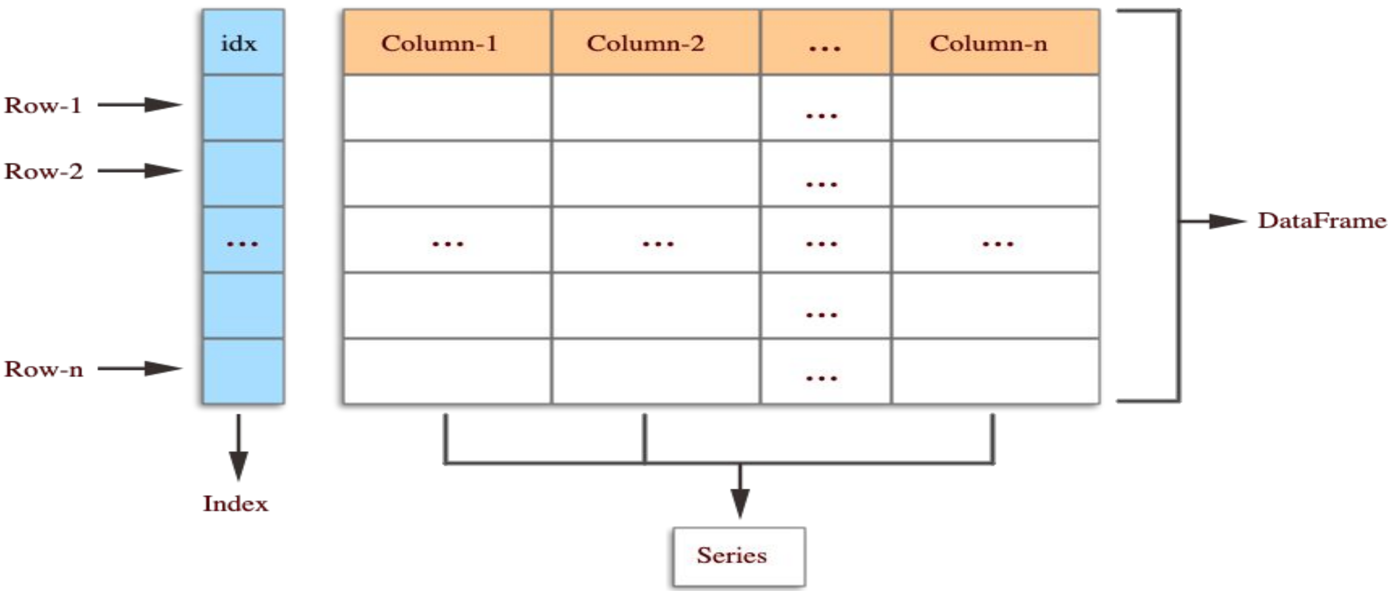
Day-8 Agenda



Python Pandas

- It is a Python package providing powerful data structures
- It is designed to work with “tabular” data in an easier way

Pandas Data structure



© w3resource.com

Python Pandas/Continue

- It is important for doing data pre-processing and analysis in Python
- Pandas provides rich set of functions to process various types of data
- Pandas provides two very useful data structures to process the data; Series and DataFrame.

Pandas Data structures

- **Series** is a one-dimensional array that can store various data types, including mix data types.
- **DataFrame** is the widely used data structure of pandas, which can be used with two dimensional arrays.
- A Series has a raw identifier called **index**
- To start with pandas we import it as follows:

```
>>> import pandas as pd
```

Pandas Data structures-Series

- **Series** is a one-dimensional array that can store various data types, including mix data types.
- Any *list*, *tuple* and *dictionary* can be converted into Series, as follows:

pd.Series(<list/tuple/dictionary>)

- A list/tuple conversion, by default, it sets the index to 0, 1, 2 and so on.

Pandas Data structures-Series/Continue

Example:

```
>>> d = ['Adam', 'Engineer', 1000]

>>> pd.Series(d)



>>> d = ('Adam', 'Engineer', 1000)

>>> pd.Series(d)



>>> d = {'name': 'Adam', 'job': 'Engineer', 'salary': 1000}

>>> pd.Series(d)
```

Pandas Data structures-Series/Continue

- A list/tuple conversion, by default, it sets the index to 0, 1, 2 and so on. Custom index names can be provided as follows:

```
>>> d = ['Adam', 'Engineer', 1000]

>>> pd.Series(d, index = ['name', 'job', 'salary'])


>>> d = ('Adam', 'Engineer', 1000)

>>> pd.Series(d, index = ['name', 'job', 'salary'])
```

Pandas Data structures-Series/Continue

- Elements of a Series can be accessed using index name as follows:

```
>>> d = ['Adam', 'Engineer', 1000]

>>> s = pd.Series(d)

>>> s[0]



>>> d = ['Adam', 'Engineer', 1000]

>>> s = pd.Series(d, index = ['name', 'job', 'salary'])

>>> s['name']
```

Pandas Data structures-DataFrame

- **DataFrame** is the widely used data structure of pandas.
- It can be used with two dimensional arrays; it has two different index: column-index and row-index.
- A DataFrame can be created of a dictionary of **equal-length list**.
- CSV’s, spreadsheets, and text files can be read by pandas as DataFrame.

Pandas Data structures-DataFrame/Continue

- Basic example

```
>>> data = { 'name': ['Adam', 'Sam', 'Bob'],
              'job': ['Engineer', 'Accountant', 'Salesman'],
           'salary': [2200, 1200]
          }
>>> df = pd.DataFrame(data)
>>> df
```

Pandas Data structures-DataFrame/Continue

- Additional columns can be added after defining a DataFrame as shown below:

```
>>> df['status'] = 'active'


>>> df
```

Pandas Data structures-DataFrame/Continue

- Data in a DataFrame can be accessed in either row or column indexes

```
# To access one column
>>> df['name']

# To access more than one column
>>> df[['name', 'job']]

# To access data by slicing
>>> df[0:1]
>>> df.iloc[0:2]
```

Pandas Data structures-DataFrame/Continue

- Data in a DataFrame can be accessed by row and column indexes

```
# To access data field by an index
>>> df.at[0,'name']

# To access data field by an index
>>> df.loc[0][['name','job']]
```

DataFrame Index Reset

- Indexes can be changed.

```
>>> df.index = ['A', 'B', 'C']
```

- A column (or more) can be set as index

```
>>> df.set_index('name')
# Have the set_index had any effect on df? What should we do?
>>> df

>>> df = df.set_index('name')
>>> df
```

Drop a Column/DataFrame

- A column can be dropped by the **drop** method or the **del** command

```
>>> df.drop('salary', axis=1)
# Have the drop method had any effect on df? What should we do?
>>> df

>>> df = df.drop('salary', axis=1)
>>> df


>>> del df['status']
```

Note: The **del** keyword is used to delete objects; variables, lists, or parts of a list, dataframes (etc.).

Reading Files

- Pandas provides various functionalities
- For instance, reading files of different formats such as csv, excel, html, json and others. It can also read data from the clipboard.
- To read from clipboard, copy cells from a spreadsheet, and, then, execute the following line:

```
>>> pd.read_clipboard()
```

Reading Files/Continue

- To read from a file, the “sms_spam.csv” and “fake_news.csv” files [1][2]
- Reading a file:

```
>>> df_sms = pd.read_csv('sms_spam.csv’)

>>> df_fake = pd.read_csv('fake_news.csv’)
```

[1]: https://www.kaggle.com/hdza1991/sms-spam
[2]: https://www.kaggle.com/mrisdal/fake-news/download

Pandas Data Operations

- Pandas offers various useful data operations for DataFrame:

  - Row and column selection
  - Data Filtering
  - Sorting
  - Counting
  - Grouping
  - String Operations

Data Operations-Data Selection

- Viewing data in a DataFrame just requires typing its name

```
>>> df_sms
```

- This will not show all rows and column, but it can be limited by the following line :

```
>>> pd.set_option('max_rows', 10, 'max_columns', 10)
```

Data Operations-Data Selection/Continue

- The following is to view the first 5 rows of the DataFrame

```
>>> df_sms.head()
```

- Total number of lines to be changed as follows:

```
>>> df_sms.head(3)
```

- The following is to view the last 5 rows of the DataFrame

```
>>> df_sms.tail()
```

- The following line of code shows the total number of rows

```
>>> len(df_sms)
```

Data Operations-Data Filtering

- Data can be filtered boolean expression in DataFrame
- Before we start filtering, we try to:
  - Get column names of the targeted DataFrame as follows:

    ```
    >>> df_sms.columns
    ```

  - Fetch categories in a given column using the **unique()** option, e.g.:

    ```
    >>> df_sms['type'].unique()
    ```

    ```
    Out: array(['ham', 'spam'], dtype=object)
    ```

- Then, we can filter which SMS's are ham or spam:

```
>>> df_sms[df_sms['type'] == 'spam']

>>> df_sms[df_sms['type'] == 'ham']
```

Data Operations-Data Filtering/Continue

- From ‘fake news’ dataset:
  - filter data with spam_score > 0.50

```
>>> df_fake[df_fake['spam_score'] > 0.50]
```

  - filter data with number of 'likes' <= 100

```
>>> df_fake[df_fake['likes'] <= 100]
```

Data Operations-Data Filtering/Continue

- From ‘fake news’ dataset:
  - filter data with number of 'likes' >= 500 and biased

```
>>> df_fake[(df_fake['likes'] >= 500) & (df_fake['type'] == 'bias')]
```

  - filter data with number of 'likes' or 'likes' > 750

```
>>> df_fake[(df_fake['likes'] > 750) | (df_fake['comments'] > 750)]
```

Data Operations-Sorting

- Sorting can be performed by the ‘sort_values’ or ‘sort_index’ method (Continue with ‘fake news’ dataset)

-

  - sort data according to the number of 'likes' (Ascending)

```
>>> df_fake.sort_values(by='likes')
```

  - sort data according to the number of 'likes' (Descending)

```
>>> df_fake.sort_values(by='likes', ascending=False)
```

Data Operations-Sorting/Continue

- Sorting can also be performed by the ‘sort_index’ method

```
>>> data = { 'name': ['Adam', 'Sam', 'Bob'],
          'job': ['Engineer', 'Accountant', 'Salesman'],
       'salary': [2200, 1200, None]
       }
>>> df = pd.DataFrame(data)
>>> df = df.set_index('name')
>>> df = df.sort_index()
>>> df
```

Data Operations-Counting

- Total number of occurrences can be counted by the ‘value_counts()’ method
-
  - Count rows of fake news according to the country

```
>>> df_fake['country'].value_counts()
```

  - Count SMS’s according to their type

```
>>> df_sms['type'].value_counts()
```

Data Operations-String Methods

- Pandas provides various string operations through '.str'
  - Find news that their titles contain 'Donald'

```
>>> df_fake[df_fake['title'].str.contains('Donald')]
```

  - Find SMS's that their text contain 'WIN'

```
>>> df_sms[df_sms.text.str.contains('WIN')]
```

- When a column contain **NaN** values apply the **fillna** method, e.g.:

```
>>> df_fake['title'].fillna('', inplace=True)
```

Data Operations-String Methods/Continue

- Find SMS’s with text more than 500 characters

```
>>> df_sms[df_sms.text.str.len() > 500]
```

- Find SMS’s that their whole text in upper case

```
>>> df_sms[df_sms.text.str.upper() == df_sms.text]
```

- Find SMS’s that their text only has 1 word (token)

```
>>> df_sms[df_sms.text.str.split().str.len() == 1]
```

DataFrame Statistics

Pandas provides various statistical functionalities:

- Describe: provides summary statistics for only the numerical columns

```
>>> df_fake.describe()
```

- Correlation: returns the correlation between numerical columns

```
>>> df_fake.corr()
```

- It also provides: mean(), max(), min(), median(), count() and std()

Data Grouping

- Pandas provides data grouping in which data can be grouped by column names.
- Data grouping adds more functionality to get more information about data
- Custom formats can be defined to group data
- The **groupby** method is used, as shown in the following examples

Data Grouping/Continue

- **size()** counts the total number for rows according to (a) specific column(s). Its result is as the same as **value_counts()**

```
>>> df_sms.groupby(['type']).size()

>>> df_fake.groupby(['country']).size()

>>> df_fake.groupby(['author', 'type']).count()
```

Data Grouping/Continue

- Advanced data grouping can be done using the **agg()** functionality

```
>>> df_fake[['type', 'likes']].groupby(['type']).agg({'likes':['min','max']})
```

```
>>> df_fake[['type', 'shares']].groupby(['type']).agg({'shares': ['sum']})
```

```
>>> df_fake[['type', 'likes', 'shares']].groupby(['type']).agg({'likes':['min','max'],
'shares': ['sum']})
```

Updating a DataFrame

Next Session!

Challenges

1. Using the books.xml document, parse the data and convert it into a DataFrame.

Hints:

- Parse the file using the ElementTree package.
- Convert the parsed data into a dictionary
- Convert the dictionary into a DataFrame

2. From the ‘SMS spam’ dataset, investigate what patterns spam SMS’s could be.

Hints:

- Display SMS’s classified as SPAM.
- Identify number of characters and words (tokens).
- Identify letter case.
- Which words are most frequently used?