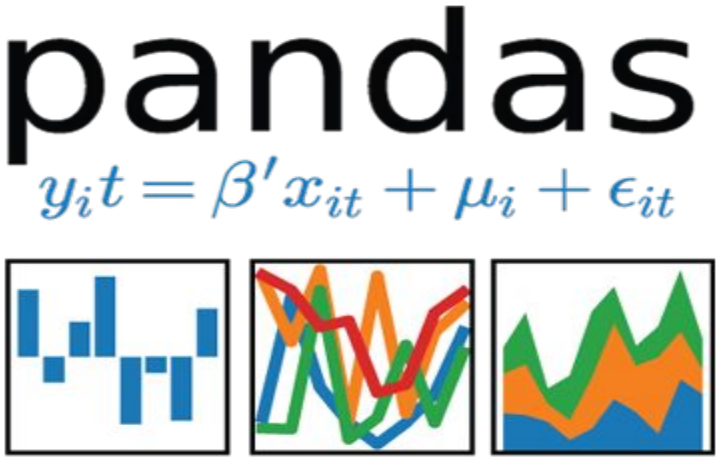
Programming

with python™

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

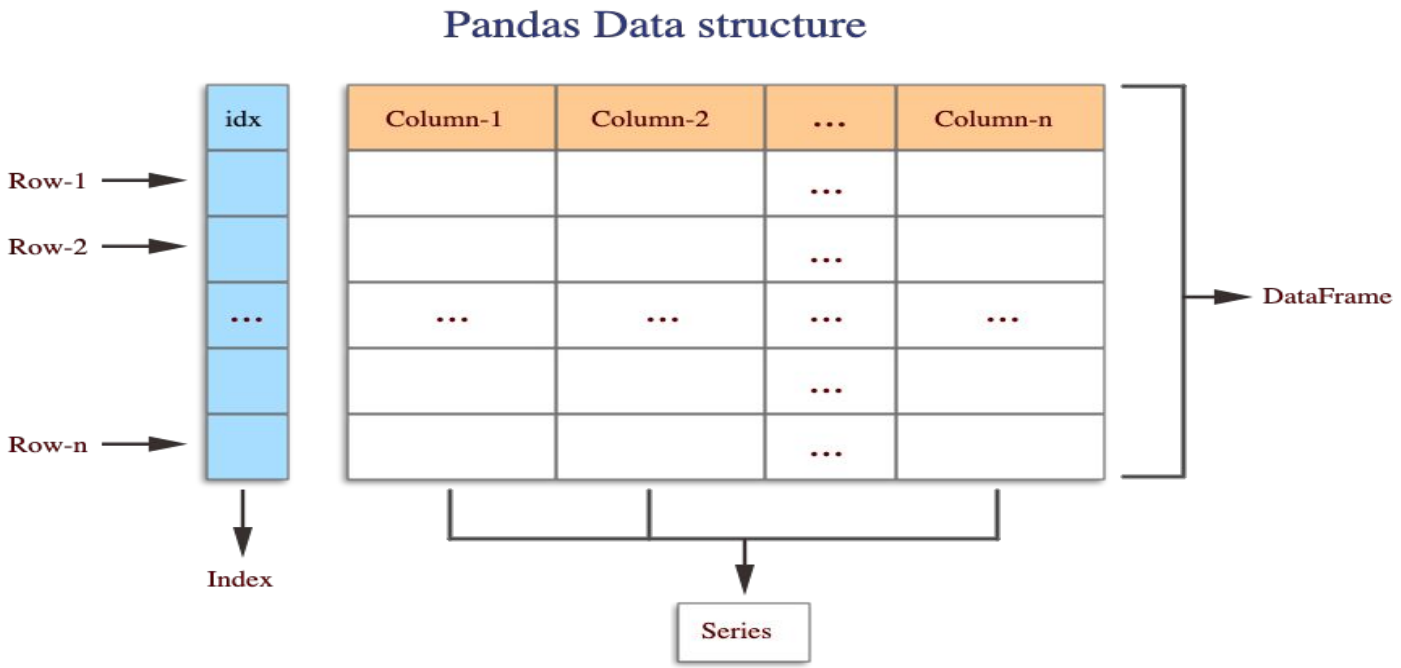
Day-9 Agenda



Part Two

Python Pandas *just to remember*

- Python package providing **powerful data structures**
- Designed to work with **“tabular”** data in an easier way



Updating a DataFrame

- Updating a DataFrame could be by adding new columns, changing values, renaming column names (etc.)
- Let's start with creating an empty DataFrame that can be defined by the DataFrame(), with empty arguments:

```
>>> df = pd.DataFrame()
```

- We, then, add 'customer_id', 'name' and 'job' columns as follows:

```
>>> df['customer_id'] = ['A-0001', 'A-0002', 'A-0003']
>>> df['name'] = ['Adam', 'Bob', 'Alice']
>>> df['job'] = ['Accountant', 'Engineer', 'Manager']
>>> df
```

Updating a DataFrame

- Adding a new column with data applied according to specific criteria
- The **.loc** is used to add/update (new) columns as follows:

```
>>> df.loc[df.job == 'Manager','extra_benefits'] = 1000
>>> df

>>> df.loc[df.job == 'Engineer','extra_benefits'] = 100
>>> df
```

Note that when it is **not matched** the value is **NaN.**

Updating a DataFrame

- Duplicates in data should be handled
- The **duplicated()** is to check for duplicated rows as follows:

```
>>> df[df.duplicated()]
```

- To drop duplicated rows we use the **drop_duplicates()**:

```
>>> df.drop_duplicates()
# Have the drop method had any effect on df? What should we do?
```

- To commit the dropping, do either:

```
>>> df.drop_duplicates(inplace=True)
# OR
>>> df = df.drop_duplicates()
```

Updating a DataFrame

- The **drop_duplicates()** can be performed based on specific criteria
- Again re-run all the steps started by creating an empty DataFrame and do the dropping as follows:

```
>>> df = df.drop_duplicates(subset=['customer_id'])
>>> df
```

- Re-run the steps again, but try with the following:

```
>>> df = df.drop_duplicates(subset=['customer_id'], keep="last")
```

- Note that ‘first’ is the default

Updating a DataFrame

- Missing values in data should also be handled
- The decision of what value to impute with missing values is not a programming problem

- The **isnull()** is to check for missing values as follows:

```
>>> df.isnull().sum()
```

- The **fillna()** can be used as follows

```
>>> df['extra_benefits'].fillna(50)
```

- We can also build our custom imputation function

Updating a DataFrame

- (A) column name(s) can be renamed by th **rename()** operation as follows:

```
>>> df.rename(columns={'extra_benefits': 'benefits'})

# Have the drop method had any effect on df? What should we do?
>>> df
```

- To commit the renaming, do either:

```
>>> df = df.rename(columns={'extra_benefits': 'benefits'})

# OR

>>> df.rename(columns={'extra_benefits': 'benefits'}, inplace=True)
```

Updating a DataFrame

- We can also create new columns from existing
- Back to SMS data, we can create a column of the number of words if the text
- The **apply() function (**and **lambda expressions)** are used as follows:

```
>>> df_sms['words_count'] = df_sms['text'].apply(lambda x: len(x.split()))
>>> df_sms['chars_count'] = df_sms['text'].apply(len)
>>> df_sms
```

- In the above example, we created new columns representing the number of words and characters of the **'text'** column.

DataFrame Merge-Append

- New rows can be added from a DataFrame to another
- In adding rows, the columns of each DataFrame should be identical
- Adding rows can be done by the **append()/concat()**

```
>>> df1 = pd.DataFrame([[1, 'Math'], [2, 'Science']], columns=['id', 'module'])
>>> df2 = pd.DataFrame([[3, 'Reading'], [4, 'History']], columns=['id', 'module'])
>>> df1.append(df2)
```

- To reset the index, do it as follows:

```
>>> df1.append(df2, ignore_index=True)
```

- This can also be done by the concat() functions as follows:

```
>>> pd.concat([df1,df2], ignore_index=True)
```

- The appending result should be saved into a new DataFrame

DataFrame Merge-Concat

- New columns can be added from a DataFrame to another

- In adding columns, the rows should be identical

- Adding columns can be done by on of the following functions and methods:
  - **concat()**
  - **join()**
  - or **merge()**

DataFrame Merge-Concat

- The **concat()** function is to concatenate DataFrame or Series objects

- Example:

```
>>> df1 = pd.DataFrame({'module': ['math', 'history']})
>>> df2 = pd.DataFrame({'mark': [78, 81]})

>>> df3 = pd.concat([df1,df2], axis=1)
>>> df3
```

DataFrame Merge-Join

- The **join()** method is an SQL-style by which it joins the columns in a DataFrame with the columns on another
- The rows have identical values (relational keys)
- The join can be: 'left', 'right', 'outer', 'inner'
- Example:

```
>>> df1 = pd.DataFrame({'module': ['math', 'history']})
>>> df2 = pd.DataFrame({'mark': [78, 81]})

>>> df3 = df1.join(df2)
>>> df3
```

DataFrame Merge-Join

- The **merge()** function is also an SQL-style by which it joins the columns in a DataFrame with the columns on another
- The rows have identical values (relational keys)
- The join can be: 'left', 'right', 'outer', 'inner'
- Example-1:

```
>>> df1 = pd.DataFrame({'module': ['math', 'history']})
>>> df2 = pd.DataFrame({'mark': [78, 81]})

>>> df3 = pd.merge(df1, df2, left_index=True, right_index=True)
>>> df3
```

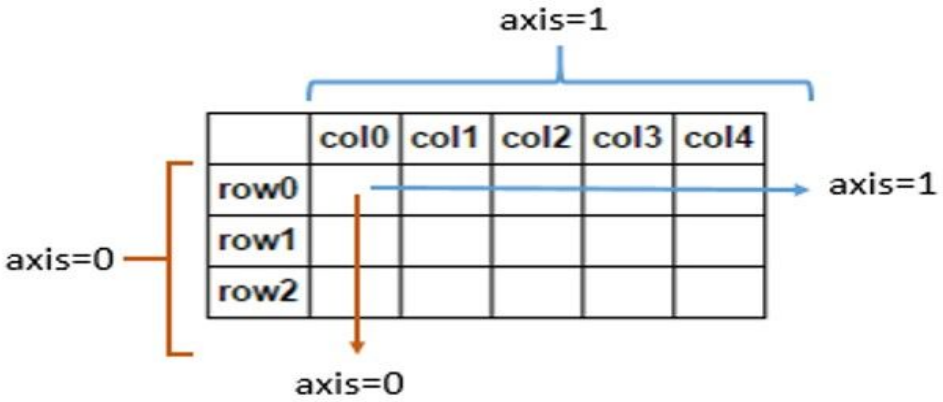
DataFrame Merge-Join

- Example-2: From the 2 dictionaries create two DataFrames and merge them into a new DataFrame

```
>>> customer_name = {'customer_id': ['C-001', 'C-002', 'C-003'],
                     'customer_name': ['Ben', 'Alice', 'Lee']}

>>> customer_jawwal = {'customer_id': ['C-001', 'C-002', 'C-003'],
                       'jawwal_number': ['0599111111', '0599111222', '0599111333']}

>>> customer_name_df = pd.DataFrame(customer_name)
>>> customer_jawwal_df = pd.DataFrame(customer_jawwal)

>>> customer_info_df = pd.merge(customer_name_df, customer_jawwal_df, on="customer_id")

>>> customer_info_df
```

The “axis” Concept in Pandas

- Visually, it is as follows:



- With “Series”, the object only has “axis 0” because it has only one dimension
- With “DataFrame”, the object has two axes: “axis 0” and “axis 1”
- “axis 0”, which is default, it goes along rows direction, whereas it goes along columns with ”axis=1”
- This thread is to find out more about where the concept came from

Iteration in Pandas

- Iteration in Pandas can be performed using **for** statement and **iterrows()** function
- With iteration each element of dataset can be accessed in a sequential manner
- An element can also be manipulated applying mathematical operations while iterating
- Also, **iteritems()** and **itertuples()** functions can be used

Iteration in Pandas/Continue

- **iterrows()** function can be used as follows:

```
>>> for row_index, row in df_sms.iterrows():
        print(row_index, row)
```

- **iteritems()** function can be used as follows:

```
>>> for key, values in df_sms.iteritems():
        print(key, values)
```

- **itertuples()** function can be used as follows:

```
>>> for row in df_sms.itertuples():
        print(row)
```

Iteration in Pandas/Continue

- Read the churn dataset. Calculate the number of month in contract. **Hint:** Divide TotalCharges by MonthlyCharges.

```
>>> for i, r in df_churn.iterrows():
        df_churn.at[i,'months_in_contract']=float(r['TotalCharges'])/float(r['MonthlyCharges'])

# Something went wrong? Execute the following:
>>> df_churn.info()
>>> df_churn[df_churn['TotalCharges'].isnull()]
>>> df_churn.loc[df_churn['TotalCharges'] == ' ', 'TotalCharges'] = np.NaN
>>> error_indexes = []
    for i, r in df_churn.iterrows():
        try:
            v = float(r['TotalCharges'])
        except:
            error_indexes.append(i)
>>> df_churn[df_churn.index.isin(error_indexes)]
```

Write to a File

- A **DataFrame** can be saved into hard drive in different formats, such as (but not limited to): csv, json, html, exel, clipboard, sql, and more.
- Syntax of saving into a CSV file is as follows

```
>>> <DataFrame>.to_csv('<file_name>', sep='<delimiter>')
```

Practice

- Download the customer churn dataset [1]
- Read it and save it into a DataFrame
- Let's mine the data frame

[1] Telecom Churn Dataset (IBM Watson Analytics)