Cheatography

DataFrame Python Cheat Sheet

by San (Bhartik) via cheatography.com/153407/cs/33011/

DATAFRAME

Definition

Pandas module in python provides a two-dimensional data structure with labeled rows and columns similar to excel sheet or a table in relational database. This data structure in pandas is called DataFrame.

ADVANTAGES

• DataFrame is the most flexible constructor and allows its creation from many sources as discussed above.

• DataFrame with its row/column names, index selection and slicing methods provide a very flexible way to access and manipulate data.

• Presence of indexes allow for search, filter and merge which are very fast operations in a DataFrame.

• Pandas is built on of numpy so it allows for very efficient matrix and vectorized operations on the data stored in DataFrame.

DISADVANTAGES

• The flexibility provided for access comes at a cost for bit higher cost in terms of higher learning curve for its users. Different ways to access data can be bit overwhelming and intimidating for new users initially.

• Pandas DataFrame can handle data that can fit in the memory. Additionally, DataFrame indexes and structure use additional memory to take care of bookkeeping needed to maintain such a flexible data structure.

BEST PRACTICES TO USE DATAFRAME

• Many new programmers coming from other programming languages try to use iterator to loop over data inside DataFrame. Iterator (“iter”) functions are expensive operations so should only be used if every other option has been exhausted. Built-in function provides by pandas/numpy have been highly optimized in many cases vectorized and thus are many times faster than a simple for-loop.

• Many new programmers coming from other programming languages try to use iterator to loop over data inside DataFrame. Iterator (“iter”) functions are expensive operations so should only be used if every other option has been exhausted. Built-in function provides by pandas/numpy have been highly optimized in many cases vectorized and thus are many times faster than a simple for-loop.

• Create proper index: Choice of index could have significant impact on the performance while fetching and performing operation on data stored in DataFrame. As an example, its best to change data type to date or datetime for timeseries data. This choice of index allows for much flexible and fast slicing of data needed to perform timeseries analysis on the stored data.

• Create relevant data type most suitable for data in each column of the DataFrame. As an example, after creation of DataFrame from a csv, some of the columns could be type “o” (pandas object). It is best practice to change the column type to float or integer depending upon data stored. This should allow more efficient operations to be performed on the data.

• As it’s known “code is read much more often than it is written” so care should be taken to write code which is descriptive with self-explanatory variable names and properly documented. Readability plays a key role in understanding of code by others as well as long term maintainability of the project as it grows in scope and codebase.

• One of the main components of ease of readability and maintainability of code is following naming and formatting conventions. This truly applies to column/row headers and index names stored in the DataFrame to make access to the data meaningful.

Creation

DataFrame can be created using following constructor with flexibility to provide data, index name, column names and column data type as follows:

```
pd.Dat aFrame(     data=None,      index: 'Axes | None' = None,      columns: 'Axes | None' = None,      dtype:
```

Select row with .loc[]

```
#Select row with row index
df.iloc[1]
```

Create DataFrame from list

One of the quickest ways to generate DataFrame in real-life would be to load data directly from SQL database, CSV or excel file. DataFrame can be created in different ways from various data sources.

```
import pandas as pd
# list of strings
list1 = ['One', 'Two', 'Three', 'Four', 'Five', 'Six', 'Seven']
#list of numbers list2 = [1,2,3 ,4, 5,6,7]
 # Calling DataFrame constr uctor on lists
df = pd.Dat aFr ame ({' Wor ds' :list1, 'Numbe r': list2})
print(df)
```

Shallow copy

```
#Access shallo w/r efe rence copy which generates " Set tin gWi thC opy War nin -
g" warning
df1 = df
df1['Words'][0] = 'One only'
```

Deep Copy

Data contained in the DataFrame can be modfied by slicing and using various selection methods

```
#Access data and manipulate using " dee p" copy instead of shallow view change

df1 = df[['W ord s'] ].c opy()
df1.iloc[0] = 'One only'
df1
```

Select Column

```
#Subset: select one of the columns
df['Words']
```

Select row with row index

```
#Select row with row index
df.iloc[1]
```