Pandas Cheat Sheet

by KarimAchaibou (KarimAchaibou) via cheatography.com/121484/cs/22373/

Lists [ ]

A list is an ordered an mutable (you can change it) Python container

creating a list: [ ]

```
numbers = [1,2,3,4]
cities = ["Bruges", "Rome"]
```

or mix of different types as wel as duplicated elements

list() constructor:

of a string: `list("Karim")` --> ["K","a","r","i","m"]

of tuple: `list( ( "Bruges", "Rome"))` --> ["Bruges", "Rome"]

of a dictionary `list({"hydrogen": 1,"helium":2})` --> [hydrogen","helium"]

of a set `list({"Bruges", "Rome"})` --> ["Bruges", "Rome"]

of a numpy array `list(np.array([1,2,3]))` --> [1,2,3]

accessing:

starting index = 0; last element = -1

`cities[0]` --> ["Bruges"]

`cities[-2]`--> ["Bruges"]

accessing multiple elements

[start:stop:step]

start index is inclusive

end index is exclusive

default value for step is 1; other values can be omitted = include all

modifying items

replace second item: `cities[1] ="Gent"`

replace first two items: `cities[:2] = ["Paris" ,"London"]`

Removing elements (del, pop, remove)

del[ ] keyword --> delete first element: `del cities[0]`

list.pop(x) methode: removes the item at the given index, and returns it --> `remove_cities = cities.pop(1)`

list.remove(x) methode: deletes the first matching element from a the list, and returns None --> `cities.remove("Bruges")`

Inserting elements

list.insert(i,x) --> insert an element x (numbers, booleans, lists) at index i and returns none

list.append(x) --> adds an item to the end of the list - equivalent to list.insert(len(list),x)

Sorting

Lists [ ] (cont)

function: sorted(iterable[, key][, reverse]) --> returns a sorted list => add to variable

methode: list.sort(key=…, reverse=) --> sorts the list in-place

arguments:

- reverse : default = False = ascending
- key: sort a list based on the value returned by the function (def or lambda) provided in the key parameter

Reversing

function: reversed(seq) --> to get a list use the list() constructor ex.: products_reversed = list(reversed(products))

methode: list.reverse() --> reverses the list in-place returning **None**

Concatenate list

+ operator

list.extend(iterable) --> extends the list by appending all the items from the iterable

Check if an element exists in a list

in → Evaluates to True if the object on the left side is included in the object on the right side.

not in → Evaluates to True if the object on the left side is not included in the object on the right side.

basics

import the package

```
import pandas as pd
```

check version

```
pd.__version__
```

show all rows of dataframe (None = diplay all rows or fill in a number instead)

```
pd.set_option('display.max_rows', None)
```

pandas.DataFrame.set_option

Copy data from clipboard

```
df = pd.read_clipboard()
```

import data from csv-file (.. = up one level)

```
df = pd.read_csv("path/file.csv")
```

Copy a data frame

basics (cont)

`df_copy = df.copy()`

head() - show first 5 rows (default) or X rows

`df.head()` or `df.head(10)`

tail() - show last 5 rows (default) or X rows

`df.tail()` or `df.tail(10)`

info() - This method prints information about a DataFrame including the index dtype and column dtypes, non-null values and memory usage

`pd.info()`

describe() - Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.

`pd.describe()` chain `.round(2)` to clean up the table

pandas.DataFrame.describe

column names

`df.columns`

size of the dataframe

`df.shape`

Quantile() (like describe() but you can define your own values). Default axis = 0 => row-wise

`df.quantile( [0.1 ,0.4,0.7, 0.8, 0.9])`

Mean, Standard Deviation, Variance, Count, Median, Min, and Max on column level

`df["column name"].mean()` or other function, native or self-made

renaming columns

`df.rename (columns= {'oldName1': 'newName1', 'oldName2': 'newName2'}, inplace=True)`

Using the argument, **inplace** = True => save dataframe into itself. If we don't state **inplace** = True you need to add result to a new or same dataframe with the "=" operator

reorder columns - pass a list as a list and index

order we want:

`cols = ['col_name_4', 'col_name_2' ,'col_name_3', 'col_name_1']`

overwrite the old dataframe with the same dataframe but new column order:

`df= df[cols]`

basics (cont)

adding new columns

`df["new_column_name"] = ...`

... = [list] or a function applied to an other column or ...

Count unique rows

`len(df ['column_name'].unique()` or `df['column_-name'].nunique

Get count of (unique) values for a particular column

`df.`*`column_name`*`.value_counts()`

transform dataframe to list

chain with `.tolist()` --> `df.columns.tolist()`

making a dataframe

format: df = pd.DataFrame(*data,index values,column names*)

Creating df from list:

```
lst = ['This', 'is', 'a', 'nice', 'cheat', 'sheet']
df1 = pd.DataFrame (lst)
```

Creating df form dict:

```
dict= {"First Column Name":  ["First value", " Secon
}
df2 = pd.DataFrame (dict)
```

another example:

```
df3 = pd.DataFrame (np.random.randn(6, 4), inde
```

Index

link: Pandas Index Explained

get index values (strings) - rows ("0", "1", "2" , ...)

`df.index`

>>> RangeIndex(start=0, stop=32561, step=1)

get column index values

`df.columns`

naming index (rows)

`df.index.name = "`*`name_of_choice`*`"`

reset index

`df_new = df.reset_index()` (the df has already been sliced otherwise the old and new index will be the same)

Index (cont)

Resetting the index will make it a column and recreate another default index.

Parameters:

drop = True (default = False) paramater won’t create that as column in the dataframe.

inplace = True (default = False)

crosstabs has also index values

```
cross = pd.crosstab( df_new.col_name_1, df_new.col_name_2)
cross.index
>>>Index( ['value_1_of_col_1', 'value_2_of_col_1', ...], dtype= 'object', name=' col_name_1')
```

individual items can be accessed like:

```
cross.loc ["value_?_of_col_1"]
```

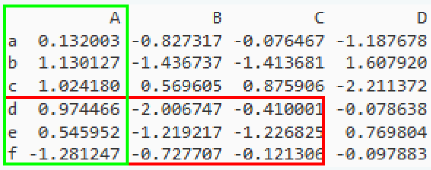
using the old index (index befor reseting) to access initial dataframe

`df[" col_name"] [new_df.index_old]` --> index_old (see before name_of_choice where we give our index a name)

Filtering a complementary set from the data

`df_new =df[df.index.isin(df_sub.index)]` --> tilde sign :negate data (True becomes False ...)

df3



```
          A         B         C         D
a  0.132003 -0.827317 -0.076467 -1.187678
b  1.130127 -1.436737 -1.413681  1.607920
c  1.024180  0.569605  0.875906 -2.211372
d  0.974466 -2.006747 -0.410001 -0.078638
e  0.545952 -1.219217 -1.226825  0.769804
f -1.281247 -0.727707 -0.121306 -0.097883
```

Selecting

slicing = getting and setting of subsets of the data set (3 ways)

.loc is primarily label based

.iloc is primarily integer position based

.loc, .iloc, and also [ ] indexing can accept a callable as indexer

`df.loc [row_indexer ,column_indexer]` --> : is the null slice

selecting column(s):

```
df['colname'] or through a list of columns df[['coln ame1', 'colname2']]
```

or directly as an attribute

```
df.colname
```

swapping columns

```
df[['B', 'A']] = df[['A', 'B']]
```

swapping column values on a subset (**you have to swap the raw data !**)

df.loc[:, ['B', 'A']] = df[['A', 'B']].to_numpy()

Selecting (cont)

create new column A with value 0 --> length of df

```
df['A'] = list(range(len( df.index)))
```

slicing using the **[ ]** operator --> [ ] slices the rows

[start:end:step] -->[2:5] --> starts at row 3 (row 2 not included); stops at row 5 (included); default step = 1

If step is negatief = start from the last element

.loc - Selection by label (labels can NOT be integer values)

`df.loc ['index_label_x ':' index_label_y'']` --> index_labels are row labels.

When slicing with .loc **both the start bound AND the stop bound are included**

Select all rows starting from row d, select all columns A to C

`df3.loc['d':, 'A':'C']` --> red square

getting values with a boolean array

`df3.loc[:, df3.loc['a'] > 0]` --> all rows and columns where row a >0 --> green square

select numeric columns (column names)

```
df_numeric = df.select_dtypes(include = [np.numbe
])
numeric_cols = df_numeric.columns.values
```

select non numeric columns (column names)

```
df_non_numeric = df.select_dtypes(exclude=[np.n
mber])
non_numeric_cols = df_non_numeric.columns.va
ues
```

setup environment

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib
plt.style.use("ggplot")
from matplotlib.pyplot import figure
%matplotlib inline
matplotlib.rcParams["figure.figsize"] =
(12,8)
pd.options.mode.chained_assignment = None
```

dropping and filling

drop columns

1) focus on columns to keep (add columns to a new dateframe)

```
df_new = df[['col_1_to_keep', 'col_2_to_keep', ... ]]
```

2) focus on columns to drop

```
df_new = df.drop(['col_1_to_drop','col_2_to_drop','col_3_to_drop'
, ...], axis=1)
```

fill NaN with some value x

```
df.fillna(x)
```

datetime

Import statement

`from datetime import datetime` --> python's default library for handling date and time

Creating datetime object

```
datetime(year=2020, month=4, day=11)
```

>>> datetime.datetime(2020, 4, 11, 0, 0)

arguments: year;month;day;hour;minute;second;millisecond

Now()

```
current_time = datetime.now()
```

Converting: string to datetime object

```
datetime.strptime("11-04-2020, 20:58:15", "%d-%m-%Y, %H:%M:%S")
```

>>>datetime.datetime(2020, 4, 11, 20, 58, 15)

formating arguments

Converting: datetime to string object

```
datetime.strftime(datetime(year=2020, month=4, day=11), "%d/%m/%
Y")
```

>>> '11/04/2020'

Data range in Pandas

```
pd.date_range(start=datetime(year=2020, month=4, day=11),periods
=3, freq='D')
```

>>>DatetimeIndex(['2020-04-11', '2020-04-12', '2020-04-13'], dtype='datetime64[ns]', freq='D')

frequency aliases

start argument can also be like: '2020-04-11' or '2020/04/11' or '2020, may 11'

apply function

under construction

missing data

heatmap

`cols = df.columns[:30]` --> select first 30 columns (names)

`colours = ["blue", "yellow"]` --> missing data will be disp

```
sns.heatmap(df[cols].isnull(), cmap=sns.c
```

false's. If value is NA then isnull() returns true = 1

data percentage list

```
missing = {}
for col in df.columns:
 pct_missing = np.mean(df[col].isnull())
 missing[col] = round(pct_missing*100)
missing_sorted = {key: value for key, value in sor
se=True)}
```