Cheat Sheet: The pandas DataFrame Object

Preliminaries

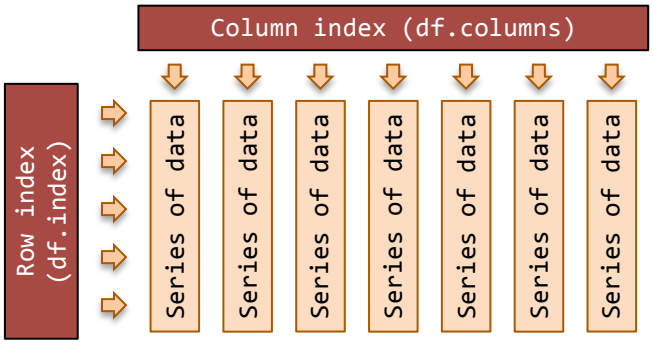
Start by importing these Python modules

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pandas import DataFrame, Series
```

Note: these are the recommended import aliases

The conceptual model

DataFrame object: The pandas DataFrame is a two-dimensional table of data with column and row indexes. The columns are made up of pandas Series objects.



Series object: an ordered, one-dimensional array of data with an index. All the data in a Series is of the same data type. Series arithmetic is vectorised after first aligning the Series index for each of the operands.

```
s1 = Series(range(0,4))  # -> 0, 1, 2, 3
s2 = Series(range(1,5))  # -> 1, 2, 3, 4
s3 = s1 + s2             # -> 1, 3, 5, 7
s4 = Series(['a','b'])*3 # -> 'aaa','bbb'
```

The index object: The pandas Index provides the axis labels for the Series and DataFrame objects. It can only contain hashable objects. A pandas Series has one Index; and a DataFrame has two Indexes.

```
# --- get Index from Series and DataFrame
idx = s.index
idx = df.columns    # the column index
idx = df.index      # the row index

# --- some Index attributes
b = idx.is_monotonic_decreasing
b = idx.is_monotonic_increasing
b = idx.has_duplicates
i = idx.nlevels     # multi-level indexes

# --- some Index methods
a = idx.values()    # get as numpy array
l = idx.tolist()    # get as a python list
idx = idx.astype(dtype)# change data type
b = idx.equals(o)   # check for equality
idx = idx.union(o) # union of two indexes
i = idx.nunique()   # number unique labels
label = idx.min()   # minimum label
label = idx.max()   # maximum label
```

Get your data into a DataFrame

Load a DataFrame from a CSV file

```
df = pd.read_csv('file.csv')# often works
df = pd.read_csv('file.csv', header=0,
      index_col=0, quotechar='"',sep=':',
      na_values = ['na', '-', '.', ''])
```

Note: refer to pandas docs for all arguments

From inline CSV text to a DataFrame

```
from StringIO import StringIO # python2.7
#from io import StringIO      # python 3
data = """, Animal, Cuteness, Desirable
row-1,      dog,    8.7,      True
row-2,      bat,    2.6,      False"""
df = pd.read_csv(StringIO(data),
      header=0, index_col=0,
      skipinitialspace=True)
```

Note: *skipinitialspace=True* allows a pretty layout

Load DataFrames from a Microsoft Excel file

```
# Each Excel sheet in a Python dictionary
workbook = pd.ExcelFile('file.xlsx')
dictionary = {}
for sheet_name in workbook.sheet_names:
    df = workbook.parse(sheet_name)
    dictionary[sheet_name] = df
```

Note: the parse() method takes many arguments like read_csv() above. Refer to the pandas documentation.

Load a DataFrame from a MySQL database

```
import pymysql
from sqlalchemy import create_engine
engine = create_engine('mysql+pymysql://'
    +'USER:PASSWORD@localhost/DATABASE')
df = pd.read_sql_table('table', engine)
```

Data in Series then combine into a DataFrame

```
# Example 1 ...
s1 = Series(range(6))
s2 = s1 * s1
s2.index = s2.index + 2# misalign indexes
df = pd.concat([s1, s2], axis=1)

# Example 2 ...
s3 = Series({'Tom':1, 'Dick':4, 'Har':9})
s4 = Series({'Tom':3, 'Dick':2, 'Mar':5})
df = pd.concat({'A':s3, 'B':s4 }, axis=1)
```

Note: 1st method has in integer column labels
Note: 2nd method does not guarantee col order
Note: index alignment on DataFrame creation

Get a DataFrame from data in a Python dictionary

```
# default --- assume data is in columns
df = DataFrame({
        'col0' : [1.0, 2.0, 3.0, 4.0],
        'col1' : [100, 200, 300, 400]
    })
```

Get a DataFrame from data in a Python dictionary

```
# --- use helper method for data in rows
df = DataFrame.from_dict({ # data by row
        'row0' : {'col0':0, 'col1':'A'},
        'row1' : {'col0':1, 'col1':'B'}
    }, orient='index')

df = DataFrame.from_dict({ # data by row
        'row0' : [1, 1+1j, 'A'],
        'row1' : [2, 2+2j, 'B']
    }, orient='index')
```

Create play/fake data (useful for testing)

```
# --- simple
df = DataFrame(np.random.rand(50,5))

# --- with a time-stamp row index:
df = DataFrame(np.random.rand(500,5))
df.index = pd.date_range('1/1/2006',
       periods=len(df), freq='M')

# --- with alphabetic row and col indexes
import string
import random
r = 52 # note: min r is 1; max r is 52
c = 5
df = DataFrame(np.random.randn(r, c),
       columns = ['col'+str(i) for i in
                  range(c)],
       index = list((string.uppercase +
           string.lowercase)[0:r]))
df['group'] = list(
    ''.join(random.choice('abcd')
        for _ in range(r))
    )
```

Saving a DataFrame

Saving a DataFrame to a CSV file

```
df.to_csv('name.csv', encoding='utf-8')
```

Saving DataFrames to an Excel Workbook

```
from pandas import ExcelWriter
writer = ExcelWriter('filename.xlsx')
df1.to_excel(writer,'Sheet1')
df2.to_excel(writer,'Sheet2')
writer.save()
```

Saving a DataFrame to MySQL

```
import pymysql
from sqlalchemy import create_engine
e = create_engine('mysql+pymysql://' +
    'USER:PASSWORD@localhost/DATABASE')
df.to_sql('TABLE',e, if_exists='replace')
```

Note: if_exists → 'fail', 'replace', 'append'

Saving a DataFrame to a Python dictionary

```
dictionary = df.to_dict()
```

Saving a DataFrame to a Python string

```
string = df.to_string()
```

Note: sometimes may be useful for debugging

Working with the whole DataFrame

Peek at the DataFrame contents

```
df.info()           # index & data types
n = 4
dfh = df.head(n)    # get first n rows
dft = df.tail(n)    # get last n rows
dfs = df.describe() # summary stats cols
top_left_corner_df = df.iloc[:5, :5]
```

DataFrame non-indexing attributes

```
dfT = df.T    # transpose rows and cols
l = df.axes   # list row and col indexes
(r, c) = df.axes  # from above
s = df.dtypes # Series column data types
b = df.empty  # True for empty DataFrame
i = df.ndim   # number of axes (2)
t = df.shape  # (row-count, column-count)
(r, c) = df.shape # from above
i = df.size   # row-count * column-count
a = df.values # get a numpy array for df
```

DataFrame utility methods

```
dfc = df.copy() # copy a DataFrame
dfr = df.rank() # rank each col (default)
dfs = df.sort() # sort each col (default)
dfc = df.astype(dtype) # type conversion
```

DataFrame iteration methods

```
df.iteritems()# (col-index, Series) pairs
df.iterrows() # (row-index, Series) pairs

# example ... iterating over columns
for (name, series) in df.iteritems():
    print('Col name: ' + str(name))
    print('First value: ' +
        str(series.iat[0]) + '\n')
```

Maths on the whole DataFrame (not a complete list)

```
df = df.abs()  # absolute values
df = df.add(o) # add df, Series or value
s = df.count() # non NA/null values
df = df.cummax()  # (cols default axis)
df = df.cummin()  # (cols default axis)
df = df.cumsum()  # (cols default axis)
df = df.cumprod() # (cols default axis)
df = df.diff() # 1st diff (col def axis)
df = df.div(o) # div by df, Series, value
df = df.dot(o) # matrix dot product
s = df.max()   # max of axis (col def)
s = df.mean()  # mean (col default axis)
s = df.median()# median (col default)
s = df.min()   # min of axis (col def)
df = df.mul(o) # mul by df Series val
s = df.sum()   # sum axis (cols default)
```

Note: The methods that return a series default to working on columns.

DataFrame filter/select rows or cols on label info

```
df = df.filter(items=['a', 'b']) # by col
df = df.filter(items=[5], axis=0) #by row
df = df.filter(like='x') # keep x in col
df = df.filter(regex='x') # regex in col
df = df.select(crit=(lambda x:not x%5))#r
```

Note: select takes a Boolean function, for cols: axis=1
Note: filter defaults to cols; select defaults to rows

Working with Columns

A DataFrame column is a pandas Series object

Get column index and labels

```
idx = df.columns          # get col index
label = df.columns[0]     # 1st col label
lst = df.columns.tolist() # get as a list
```

Change column labels

```
df.rename(columns={'old':'new'},
    inplace=True)
df = df.rename(columns={'a':1,'b':'x'})
```

Selecting columns

```
s = df['colName'] # select col to Series
df = df[['colName']] # select col to df
df = df[['a','b']]    # select 2 or more
df = df[['c','a','b']]# change order
s = df[df.columns[0]] # select by number
df = df[df.columns[[0, 3, 4]] # by number
s = df.pop('c') # get col & drop from df
```

Selecting columns with Python attributes

```
s = df.a            # same as s = df['a']
# cannot create new columns by attribute
df.existing_col = df.a / df.b
df['new_col'] = df.a / df.b
```

Trap: column names must be valid identifiers.

Adding new columns to a DataFrame

```
df['new_col'] = range(len(df))
df['new_col'] = np.repeat(np.nan,len(df))
df['random'] = np.random.rand(len(df))
df['index_as_col'] = df.index
df1[['b','c']] = df2[['e','f']]
df3 = df1.append(other=df2)
```

Trap: When adding an indexed pandas object as a new column, only items from the new series that have a corresponding index in the DataFrame will be added. The receiving DataFrame is not extended to accommodate the new series. To merge, see below.
Trap: when adding a python list or numpy array, the column will be added by integer position.

Swap column contents – change column order

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

Dropping columns (mostly by label)

```
df = df.drop('col1', axis=1)
df.drop('col1', axis=1, inplace=True)
df = df.drop(['col1','col2'], axis=1)
s = df.pop('col') # drops from frame
del df['col'] # even classic python works
df.drop(df.columns[0], inplace=True)
```

Vectorised arithmetic on columns

```
df['proportion']=df['count']/df['total']
df['percent'] = df['proportion'] * 100.0
```

Apply numpy mathematical functions to columns

```
df['log_data'] = np.log(df['col1'])
df['rounded'] = np.round(df['col2'], 2)
```

Note: Many more mathematical functions

Columns value set based on criteria

```
df['b']=df['a'].where(df['a']>0,other=0)
df['d']=df['a'].where(df.b!=0,other=df.c)
```

Note: where other can be a Series or a scalar

Data type conversions

```
s =  df['col'].astype(str) # Series dtype
na = df['col'].values      # numpy array
pl = df['col'].tolist()    # python list
```

Note: useful dtypes for Series conversion: int, float, str
Trap: index lost in conversion from Series to array or list

Common column-wide methods/attributes

```
value = df['col'].dtype  # type of data
value = df['col'].size   # col dimensions
value = df['col'].count()# non-NA count
value = df['col'].sum()
value = df['col'].prod()
value = df['col'].min()
value = df['col'].max()
value = df['col'].mean()
value = df['col'].median()
value = df['col'].cov(df['col2'])
s =     df['col'].describe()
s =     df['col'].value_counts()
```

Find index label for min/max values in column

```
label =  df['col1'].idxmin()
label =  df['col1'].idxmax()
```

Common column element-wise methods

```
s = df['col'].isnull()
s = df['col'].notnull() # not isnull()
s = df['col'].astype(float)
s = df['col'].round(decimals=0)
s = df['col'].diff(periods=1)
s = df['col'].shift(periods=1)
s = df['col'].to_datetime()
s = df['col'].fillna(0) # replace NaN w 0
s = df['col'].cumsum()
s = df['col'].cumprod()
s = df['col'].pct_change(periods=4)
s = df['col'].rolling_sum(periods=4,
                          window=4)
```

Note: also rolling_min(), rolling_max(), and many more.

Append a column of row sums to a DataFrame

```
df['Total'] = df.sum(axis=1)
```

Note: also means, mins, maxs, etc.

Multiply every column in DataFrame by Series

```
df = df.mul(s, axis=0) # on matched rows
```

Note: also add, sub, div, etc.

Selecting columns with .loc, .iloc and .ix

```
df = df.loc[:, 'col1':'col2'] # inclusive
df = df.iloc[:, 0:2]          # exclusive
```

Get the integer position of a column index label

```
j = df.columns.get_loc('col_name')
```

Test if column index values are unique/monotonic

```
if df.columns.is_unique: pass # ...
b = df.columns.is_monotonic_increasing
b = df.columns.is_monotonic_decreasing
```

Working with rows

Get the row index and labels

```
idx = df.index              # get row index
label = df.index[0]         # 1st row label
lst = df.index.tolist()     # get as a list
```

Change the (row) index

```
df.index = idx          # new ad hoc index
df.index = range(len(df)) # set with list
df = df.reset_index() # replace old w new
# note: old index stored as a col in df
df = df.reindex(index=range(len(df)))
df = df.set_index(keys=['r1','r2','etc'])
df.rename(index={'old':'new'},
    inplace=True)
```

Adding rows

```
df = original_df.append(more_rows_in_df)
```

Hint: convert to a DataFrame and then append. Both DataFrames should have same column labels.

Dropping rows (by name)

```
df = df.drop('row_label')
df = df.drop(['row1','row2']) # multi-row
```

Boolean row selection by values in a column

```
df = df[df['col2'] >= 0.0]
df = df[(df['col3']>=1.0) |
                      (df['col1']<0.0)]
df = df[df['col'].isin([1,2,5,7,11])]
df = df[~df['col'].isin([1,2,5,7,11])]
df = df[df['col'].str.contains('hello')]
```

Trap: bitwise "or", "and" "not" (ie. | & ~) co-opted to be Boolean operators on a Series of Boolean
Trap: need parentheses around comparisons.

Selecting rows using isin over multiple columns

```
# fake up some data
data = {1:[1,2,3], 2:[1,4,9], 3:[1,8,27]}
df = pd.DataFrame(data)

# multi-column isin
lf = {1:[1, 3], 3:[8, 27]} # look for
f = df[df[list(lf)].isin(lf).all(axis=1)]
```

Selecting rows using an index

```
idx = df[df['col'] >= 2].index
print(df.ix[idx])
```

Select a slice of rows by integer position
[inclusive-from : exclusive-to [: step]]
default start is 0; default end is len(df)

```
df = df[:]       # copy DataFrame
df = df[0:2]     # rows 0 and 1
df = df[-1:]     # the last row
df = df[2:3]     # row 2 (the third row)
df = df[:-1]     # all but the last row
df = df[::2]     # every 2nd row (0 2 ..)
```

Trap: a single integer without a colon is a column label for integer numbered columns.

Select a slice of rows by label/index
[inclusive-from : inclusive–to [: step]]

```
df = df['a':'c'] # rows 'a' through 'c'
```

Trap: doesn't work on integer labelled rows

Append a row of column totals to a DataFrame

```
# Option 1: use dictionary comprehension
sums = {col: df[col].sum() for col in df}
sums_df = DataFrame(sums,index=['Total'])
df = df.append(sums_df)

# Option 2: All done with pandas
df = df.append(DataFrame(df.sum(),
                    columns=['Total']).T)
```

Iterating over DataFrame rows

```
for (index, row) in df.iterrows(): # pass
```

Trap: row data type may be coerced.

Sorting DataFrame rows values

```
df = df.sort(df.columns[0],
                           ascending=False)
df.sort(['col1', 'col2'], inplace=True)
```

Random selection of rows

```
import random as r
k = 20 # pick a number
selection = r.sample(range(len(df)), k)
df_sample = df.iloc[selection, :]
```

Note: this sample is not sorted

Sort DataFrame by its row index

```
df.sort_index(inplace=True) # sort by row
df = df.sort_index(ascending=False)
```

Drop duplicates in the row index

```
df['index'] = df.index # 1 create new col
df = df.drop_duplicates(cols='index',
        take_last=True)# 2 use new col
del df['index']        # 3 del the col
df.sort_index(inplace=True)# 4 tidy up
```

Test if two DataFrames have same row index

```
len(a)==len(b) and all(a.index==b.index)
```

Get the integer position of a row or col index label

```
i = df.index.get_loc('row_label')
```

Trap: index.get_loc() returns an integer for a unique match. If not a unique match, may return a slice or mask.

Get integer position of rows that meet condition

```
a = np.where(df['col'] >= 2) #numpy array
```

Test if the row index values are unique/monotonic

```
if df.index.is_unique: pass # ...
b = df.index.is_monotonic_increasing
b = df.index.is_monotonic_decreasing
```