**Pandas Dataframes Notes**

**October 28**

From pandas import dataframe

**Data frame creation**

Method 1: From a dictionary

Can specify the sequence of columns using columns parameter.

popdf = DataFrame(popdat, columns=['year','state', 'pop'])

popdat: dictionary

if you give a column name that's not a key in the dictionary, you get NaN's (like Series index)

Warning: Changing column names afterward is like assigning new names, not rearranging!

popdf.index

popdf.columns

popdf.values -> gives a 2d array

(Dataframe\_name).shape -> (# of rows, # of columns)

Method 2: From a nested dictionary

The outer keys will be column names, and the inner will be indexes.

indexes will be lined up, and NaN's will be used to fill missing values.

Popdf2.T to get the indexes as columns and columns as indexes. Does not change original.

Method 3: From a 2D numpy array

You can specify names of rows and columns using something like:

randdf.index = ['one', 'two', 'three']

randdf.columns = ['first', 'second']

you can also do this in creation time for method 3.

**Columns**

Checking column membership: 'debt' in popdf.columns

Each column is a series.

name of a column Series is simply the column name

A column can be referenced as an attribute of the dataframe.

Dataframe\_name[‘colName’] = Dataframe\_name.colName

Both are equivalent and are series.

You can get a subset of columns with a list.

popdf[['state','pop']]

**INDEXING AND MANIPULATING ROWS AND COLUMNS**

Row of a dataframe is a series.

loc: useful for row indexing by position.

popdf.loc[1]

Range of rows: popdf.loc[1:3]

(end value 3 is inclusive)

Getting a subset of rows

Popdf.loc[[2010, 2021]]

Subset of rows, subset of columns:

popdf.loc[[0,2],['state','pop']] # like with ndarray

Adding a column:

# same value for all rows

popdf['debt'] = 1.5

# Different value for each row

popdf['debt'] = np.arange(1,6)

dept column gets values [1, 2, 3, 4, 5]

np.arange(a,b): gives 1d array from a to b-1

# Different value for each row

popdf2['NJ'] = [8.2, 8.4, 8.6]

#generating column from series

debts = Series([1.2, 1.5, 1.7])

popdf['debt'] = debts

Nan’s used to pad insufficient values

#creating a new column with values as a function of the other columns

Ex: randdf['third'] = randdf['first'] > randdf['second']

Randdf[‘third’] gets Boolean values

Row (index) membership: 'three' in randdf.index

randdf.loc[1:2] -> will not work for dataframes indexed by labels.

In this case you can do something like:

randdf.loc['two':'three']

Using loc to add a row:

popdf2.loc[2020] = [7.2, 8.6, 8.9]

Deleting a column using del operation: del popdf['debt']

This actually changes popdf.

Indexing a dataframe using iloc (integer indices)

Popdf.loc(1, ‘year’): gives the value at numeric index 1 and in column ‘year.’

‘year’ is the 0th column.

popdf.iloc[1,0] # use index for rows and columns

this gives same value as Popdf.loc(1, ‘year’).

popdf.iloc[1:4] -> similar to popdf.loc[1:4] but does not include the 4th row, and also works even if the indexes are not numeric

With iloc you can only use integer indexes for either rows or columns.

Specifying subset of rows and columns using iloc.

popdf2.iloc[2:,[0,2]]

all rows from and including index 2.

Columns at indices 0 and 2.

**November 2nd**

Typically, dataframes are created from csv files.

Ex:

mpgfile = open("auto\_mpg\_original.csv")

mpgs = pd.read\_csv(mpgfile)

Automatically, missing or null values are given a value of NaN in the dataframe.

mpgs[mpgs['mpg'].isnull()]

#all the rows where mpg column is null

# summary stats for numeric columns in dataset

mpgs.describe()

np.abs(data\_frame\_name) -> a new dataframe with the absolute values from the original. The original is not modified.

Alternatively: df.abs(). The original is not modified.

dfabs: dataframe

dfabs.mean() # default is axis=0, so column wise mean. This is a series. Indices are column names.

dfabs.mean(axis=1) # row-wise mean. This is a series. Indices are index/row names.

Other functions:

Cumsum

Sum

For these functions, Nan values are skipped.

dfabs2.mean(skipna=False) # but they can be included if needed

In this case, columns with Nan’s will get a value of 0.

Dfabs.argmax() does not work.

dfabs['C'].argmax()

* Gives you the numeric index of the row that has maximum value in column ‘C’

dfabs.loc['Three'].argmax()

* Gives you the numeric index of the column that has maximum value in row index ‘Three’

Drop rows that have Nan in any column:

datf.dropna()

Original is not modified. Use parameter ‘inplace = True’ for that.

To only drop those rows that have nan in all columns:

datf.dropna(how='all')

To do similar things for dropping columns, add a parameter, axis = 1.

Filling missing (NaN) values

datf.fillna(0): All nan’s replaced with 0.

You can specify different arguments in fillna:

* Method: ‘ffill’, ‘bfill’ or ‘backfill’
* Axis: default (column-wise), 1 (row-wise)

datf.fillna({1: 2.5, 2: 1.5})

#Nan's in column 1 are replaced with 2.5, etc..

Again you may have to specify inplace = True

datfc[2].fillna(1.5,inplace=True)

datfc

So the actual column 2 of datfc is modified.

Similarly, we can use loc to fill nas of a row and modify the original dataframe.

Filling in the null values of a column with the columns mean:

mpgs2['mpg'] = mpgs2['mpg'].fillna(mpgs2['mpg'].mean())

General dataframe manipulation:

Column returned on indexing is a VIEW, so modifications will affect underlying dataframe column. If you don’t want this you need to copy explicitly.

Adding dataframes together: indexes are arranged in lexographic order and NaN values are used as needed.

REINDEXING A SERIES

In reindexing, the value and index change positions.

If you are adding extra indices then fill\_value or method may be useful.

REINDEXING A DATAFRAME

**NOVEMBER 4**

Apply function

Dataframe\_name.apply(func)

Default axis: 0 [column-wise]

Each column series is sent in as an input to func

# for built-in Python functions, apply is generally unnecessary

dfabs.round(2)

Use applymap for one item of a dataframe at a time.

Map is used particularly for series. Map does not work for a dataframe.

Iterating over rows and columns of a dataframe.

For rows: iterrows

For columns: iteritems

Grouping

Value Counts

Dataframe\_name.index.tolist() returns a list of the indices.

Dropping Rows or Columns

Drop function to drop rows

Del function to drop columns

Drop function for columns needs axis = 1

Use inplace = ‘True’ to modify the original.