**#Reading flat file and indexing**

**#Using data frame methods**

**#Reading CVS file into data frame and accessing info and exploration.**

**#Acceing data frame just like a matrix**

**#Sub-setting based on column (categorical numerical)value, concat & Sorting**

**#merging data frames**

**#Renaming, Adding, and Removing Columns**

**#Handling Missing Values and updating data based on condition**

**#Other functions modify or reformat data, like Map,Replace, String, \_datetime, astype**

**#Creating data frames from other data types**

**#Other less frequently used functions/methods**

import pandas as pd

# #Reading flat file and indexing

## # can read a file from local computer or directly from a URL

pd.read\_table('u.user')

pd.read\_table('https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u.user')

## # read 'u.user' into 'users'

users = pd.read\_table('u.user', sep='|', index\_col='user\_id')

## #reading data with desired columns and sorting and indexing

import pandas as pd

url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data'

col\_names = ['id','ri','na','mg','al','si','k','ca','ba','fe','glass\_type']

glass = pd.read\_csv(url, names=col\_names, index\_col='id')

glass.sort('al', inplace=True)

glass.head()

# #Using data frame methods

# # examine the users data

users # print the first 30 and last 30 rows

type(users) # DataFrame

users.head() # print the first 5 rows

users.head(10) # print the first 10 rows

users.tail() # print the last 5 rows

users.index # "the index" (aka "the labels")

users.columns # column names (which is "an index")

users.dtypes # data types of each column

users.shape # number of rows and columns

users.values # underlying numpy array

## # select a column

users['gender'] # select one column

type(users['gender']) # Series

users.gender # select one column using the DataFrame attribute

# summarize (describe) the DataFrame

users.describe() # describe all numeric columns

users.describe(include=['object']) # describe all object columns

users.describe(include='all') # describe all columns

# summarize a Series

users.gender.describe() # describe a single column

users.age.mean() # only calculate the mean

# count the number of occurrences of each value

users.gender.value\_counts() # most useful for categorical variables

users.age.value\_counts() # can also be used with numeric variables

# detecting duplicate rows(less frequently used )

users.duplicated() # True if a row is identical to a previous row

users.duplicated().sum() # count of duplicates

users[users.duplicated()] # only show duplicates

users.drop\_duplicates() # drop duplicate rows

users.age.duplicated() # check a single column for duplicates

users.duplicated(['age', 'gender', 'zip\_code']).sum() # specify columns for finding duplicates

# #Reading CVS file into data frame and accessing info and exploration.

# read drinks.csv into a DataFrame called 'drinks'

drinks = pd.read\_table('drinks.csv', sep=',')

drinks = pd.read\_csv('drinks.csv') # assumes separator is comma

# print the head and the tail

drinks.head()

drinks.tail()

# examine the default index, data types, and shape

drinks.index

drinks.dtypes

drinks.shape

# print the 'beer\_servings' Series

drinks['beer\_servings']

drinks.beer\_servings

# calculate the mean 'beer\_servings' for the entire dataset

drinks.describe() # summarize all numeric columns

drinks.beer\_servings.describe() # summarize only the 'beer\_servings' Series

drinks.beer\_servings.mean() # only calculate the mean

# Count the number of occurrences of each 'continent' value and see if it looks correct

drinks.continent.value\_counts()

# BONUS: display only the number of rows of the 'users' DataFrame

users.shape[0]

# BONUS: display the 3 most frequent occupations in 'users'

users.occupation.value\_counts().head(3)

users.occupation.value\_counts()[:3]

# BONUS: create the 'users' DataFrame from the u.user\_original file (which lacks a header row)

# Hint: read the pandas.read\_table documentation

user\_cols = ['user\_id', 'age', 'gender', 'occupation', 'zip\_code']

users = pd.read\_table('u.user\_original', sep='|', header=None, names=user\_cols, index\_col='user\_id')

# for each continent, calculate the mean beer servings

drinks.groupby('continent').beer.mean()

# for each continent, count the number of occurrences

drinks.continent.value\_counts()

# for each continent, describe beer servings

drinks.groupby('continent').beer.describe()

# similar, but outputs a DataFrame and can be customized

drinks.groupby('continent').beer.agg(['count', 'mean', 'min', 'max'])

drinks.groupby('continent').beer.agg(['count', 'mean', 'min', 'max']).sort('mean')

# if you don't specify a column to which the aggregation function should be applied,

# it will be applied to all numeric columns

drinks.groupby('continent').mean()

drinks.groupby('continent').describe()

# for each occupation, calculate the mean age

users.groupby('occupation').age.mean()

# BONUS: for each occupation, calculate the minimum and maximum ages

users.groupby('occupation').age.agg(['min', 'max'])

# BONUS: for each combination of occupation and gender, calculate the mean age

users.groupby(['occupation', 'gender']).age.mean()

# determine unique values in a column

users.occupation.nunique() # count the number of unique values

users.occupation.unique() # return the unique values

# #Accessing data frame just like a matrix

# select multiple columns

my\_cols = ['City', 'State'] # create a list of column names...

ufo[my\_cols] # ...and use that list to select columns

ufo[['City', 'State']] # or, combine into a single step

# use loc to select columns by name

ufo.loc[:, 'City'] # colon means "all rows", then select one column

ufo.loc[:, ['City', 'State']] # select two columns

ufo.loc[:, 'City':'State'] # select a range of columns

# loc can also filter rows by "name" (the index)

ufo.loc[0, :] # row 0, all columns

ufo.loc[0:2, :] # rows 0/1/2, all columns

ufo.loc[0:2, 'City':'State'] # rows 0/1/2, range of columns

# use iloc to filter rows and select columns by integer position

ufo.iloc[:, [0, 3]] # all rows, columns in position 0/3

ufo.iloc[:, 0:4] # all rows, columns in position 0/1/2/3

ufo.iloc[0:3, :] # rows in position 0/1/2, all columns

# #Sub-setting based on column (categorical numerical)value, concat & Sorting

# boolean filtering: only show users with age < 20

young\_bool = users.age < 20 # create a Series of booleans...

users[young\_bool] # ...and use that Series to filter rows

users[users.age < 20] # or, combine into a single step

users[users.age < 20].occupation # select one column from the filtered results

users[users.age < 20].occupation.value\_counts() # value\_counts of resulting Series

# boolean filtering with multiple conditions

users[(users.age < 20) & (users.gender=='M')] # ampersand for AND condition

users[(users.age < 20) | (users.age > 60)] # pipe for OR condition

# sorting

users.age.order() # sort a column

users.sort('age') # sort a DataFrame by a single column

users.sort('age', ascending=False) # use descending order instead

EXERCISE TWO

#Filter 'drinks' to only include European countries

drinks[drinks.continent=='EU']

# filter 'drinks' to only include European countries with wine\_servings > 300

drinks[(drinks.continent=='EU') & (drinks.wine\_servings > 300)]

# calculate the mean 'beer\_servings' for all of Europe

drinks[drinks.continent=='EU'].beer\_servings.mean()

# determine which 10 countries have the highest total\_litres\_of\_pure\_alcohol

drinks.sort('total\_litres\_of\_pure\_alcohol').tail(10)

# BONUS: sort 'users' by 'occupation' and then by 'age' (in a single command)

users.sort(['occupation', 'age'])

# BONUS: filter 'users' to only include doctors and lawyers without using a |

# Hint: read the pandas.Series.isin documentation

users[users.occupation.isin(['doctor', 'lawyer'])]

# concatenate two DataFrames (axis=0 for rows, axis=1 for columns)

drinks = pd.concat([drinks, continent\_dummies], axis=1)

# #merging data frames

A = pd.DataFrame({'color': ['green', 'yellow', 'red'], 'num':[1, 2, 3]})

B = pd.DataFrame({'color': ['green', 'yellow', 'pink'], 'size':['S', 'M', 'L']})

# ### Inner join

#

# Only include observations found in both A and B:

pd.merge(A, B, how='inner')

# ### Outer join

#

# Include observations found in either A or B:

pd.merge(A, B, how='outer')

# ### Left join

#

# Include all observations found in A:

pd.merge(A, B, how='left')

# ### Right join

#

# Include all observations found in B:

pd.merge(A, B, how='right')

# #Renaming, Adding, and Removing Columns

# rename one or more columns

drinks.rename(columns={'beer\_servings':'beer', 'wine\_servings':'wine'})

drinks.rename(columns={'beer\_servings':'beer', 'wine\_servings':'wine'}, inplace=True)

# replace all column names

drink\_cols = ['country', 'beer', 'spirit', 'wine', 'liters', 'continent']

drinks.columns = drink\_cols

# replace all column names when reading the file

drinks = pd.read\_csv('drinks.csv', header=0, names=drink\_cols)

# add a new column as a function of existing columns

drinks['servings'] = drinks.beer + drinks.spirit + drinks.wine

drinks['mL'] = drinks.liters \* 1000

# removing columns

drinks.drop('mL', axis=1) # axis=0 for rows, 1 for columns

drinks.drop(['mL', 'servings'], axis=1, inplace=True) # drop multiple columns

# replace any spaces in the column names with an underscore

ufo.rename(columns={'Colors Reported':'Colors\_Reported', 'Shape Reported':'Shape\_Reported'}, inplace=True)

# BONUS: redo the task above, writing generic code to replace spaces with underscores

# In other words, your code should not reference the specific column names

ufo.columns = [col.replace(' ', '\_') for col in ufo.columns]

ufo.columns = ufo.columns.str.replace(' ', '\_')

# BONUS: create a new column called 'Location' that includes both City and State

# For example, the 'Location' for the first row would be 'Ithaca, NY'

ufo['Location'] = ufo.City + ', ' + ufo.State

# #Handling Missing Values and updating data based on condition

# missing values are usually excluded by default

drinks.continent.value\_counts() # excludes missing values

drinks.continent.value\_counts(dropna=False) # includes missing values

# find missing values in a Series

drinks.continent.isnull() # True if missing

drinks.continent.notnull() # True if not missing

# use a boolean Series to filter DataFrame rows

drinks[drinks.continent.isnull()] # only show rows where continent is missing

drinks[drinks.continent.notnull()] # only show rows where continent is not missing

# side note: understanding axes

drinks.sum() # sums "down" the 0 axis (rows)

drinks.sum(axis=0) # equivalent (since axis=0 is the default)

drinks.sum(axis=1) # sums "across" the 1 axis (columns)

# side note: adding booleans

pd.Series([True, False, True]) # create a boolean Series

pd.Series([True, False, True]).sum() # converts False to 0 and True to 1

# find missing values in a DataFrame

drinks.isnull() # DataFrame of booleans

drinks.isnull().sum() # count the missing values in each column

# drop missing values

drinks.dropna() # drop a row if ANY values are missing

drinks.dropna(how='all') # drop a row only if ALL values are missing

# how many rows remain if you drop all rows with any missing values?

drinks.dropna().shape[0]

# fill in missing values

drinks.continent.fillna(value='NA', inplace=True) # fill in missing values with 'NA'

# turn off the missing value filter

drinks = pd.read\_csv('drinks.csv', header=0, names=drink\_cols, na\_filter=False)

#initializing and updating specific values in DF

movies['year'] = -1 ← Updates all values in column ‘years’ of movies DF as -1

movies.loc[0, 'year'] = 1994 ← Updates first row value in column ‘year’ of movies DF as -1

# #Other functions modify or reformat data, like Map,Replace, String, \_datetime, astype

# map existing values to a different set of values

users['is\_male'] = users.gender.map({'F':0, 'M':1})

# encode strings as integer values (automatically starts at 0)

users['occupation\_num'] = users.occupation.factorize()[0]

# replace all instances of a value in a column (must match entire value)

ufo.State.replace('Fl', 'FL', inplace=True)

# string methods are accessed via 'str'

ufo.State.str.upper() # converts to uppercase

ufo.Colors\_Reported.str.contains('RED', na='False') # checks for a substring

# convert a string to the datetime format

ufo['Time'] = pd.to\_datetime(ufo.Time)

ufo.Time.dt.hour # datetime format exposes convenient attributes

(ufo.Time.max() - ufo.Time.min()).days # also allows you to do datetime "math"

ufo[ufo.Time > pd.datetime(2014, 1, 1)] # boolean filtering with datetime format

# change the data type of a column

drinks['beer'] = drinks.beer.astype('float')

# change the data type of a column when reading in a file

pd.read\_csv('drinks.csv', dtype={'beer\_servings':float})

# setting and then removing an index

ufo.set\_index('Time', inplace=True)

ufo.reset\_index(inplace=True)

# sort a column by its index

ufo.State.value\_counts().sort\_index()

# create dummy variables for 'continent' and exclude first dummy column

continent\_dummies = pd.get\_dummies(drinks.continent, prefix='cont').iloc[:, 1:]

**#Creating data frames from other data types**

# create a DataFrame from a dictionary

pd.DataFrame({'capital':['Montgomery', 'Juneau', 'Phoenix'], 'state':['AL', 'AK', 'AZ']})

# create a DataFrame from a list of lists

pd.DataFrame([['Montgomery', 'AL'], ['Juneau', 'AK'], ['Phoenix', 'AZ']], columns=['capital', 'state'])

**#Other less frequently used functions/methods**

# display a cross-tabulation of two Series

pd.crosstab(users.occupation, users.gender)

# alternative syntax for boolean filtering (noted as "experimental" in the documentation)

users.query('age < 20') # users[users.age < 20]

users.query("age < 20 and gender=='M'") # users[(users.age < 20) & (users.gender=='M')]

users.query('age < 20 or age > 60') # users[(users.age < 20) | (users.age > 60)]

# display the memory usage of a DataFrame

ufo.info() # total usage

ufo.memory\_usage() # usage by column

# change a Series to the 'category' data type (reduces memory usage and increases performance)

ufo['State'] = ufo.State.astype('category')

# temporarily define a new column as a function of existing columns

drinks.assign(servings = drinks.beer + drinks.spirit + drinks.wine)

# limit which rows are read when reading in a file

pd.read\_csv('drinks.csv', nrows=10) # only read first 10 rows

pd.read\_csv('drinks.csv', skiprows=[1, 2]) # skip the first two rows of data

# write a DataFrame out to a CSV

drinks.to\_csv('drinks\_updated.csv') # index is used as first column

drinks.to\_csv('drinks\_updated.csv', index=False) # ignore index

# save a DataFrame to disk (aka 'pickle') and read it from disk (aka 'unpickle')

drinks.to\_pickle('drinks\_pickle')

pd.read\_pickle('drinks\_pickle')

# randomly sample a DataFrame

train = drinks.sample(frac=0.75, random\_state=1) # will contain 75% of the rows

test = drinks[~drinks.index.isin(train.index)] # will contain the other 25%

# change the maximum number of rows and columns printed ('None' means unlimited)

pd.set\_option('max\_rows', None) # default is 60 rows

pd.set\_option('max\_columns', None) # default is 20 columns

print drinks

# reset options to defaults

pd.reset\_option('max\_rows')

pd.reset\_option('max\_columns')

# change the options temporarily (settings are restored when you exit the 'with' block)

with pd.option\_context('max\_rows', None, 'max\_columns', None):

print drinks