

| Module | Description | Example | Script |
|--------|---|---|-------------|
| core | dictionary, adding a new entry | <code>co['po'] = 'CO'</code> | g05/demo.py |
| core | dictionary, creating | <code>co = {'name':'Colorado', 'capital':'Denver'}</code> | g05/demo.py |
| core | dictionary, creating via comprehension | <code>fips_cols = {col:str for col in fips_vars}</code> | g13/demo.py |
| core | dictionary, looking up a value | <code>name = ny['name']</code> | g05/demo.py |
| core | dictionary, making a list of | <code>list1 = [co,ny]</code> | g05/demo.py |
| core | dictionary, obtaining a list of keys | <code>names = super_dict.keys()</code> | g05/demo.py |
| core | f-string, grouping with commas | <code>print(f'Total population: {tot_pop:,}')</code> | g11/demo.py |
| core | f-string, using a formatting string | <code>print(f"PV of {payment} with T={year} and r={r} is \${p...}</code> | g07/demo.py |
| core | file, closing | <code>fh.close()</code> | g02/demo.py |
| core | file, opening for reading | <code>fh = open('states.csv')</code> | g05/demo.py |
| core | file, opening for writing | <code>fh = open(filename,"w")</code> | g02/demo.py |
| core | file, output using print | <code>print("It was written during",year,file=fh)</code> | g02/demo.py |
| core | file, output using write | <code>fh.write("Where was this file was written?\n")</code> | g02/demo.py |
| core | file, print without adding spaces | <code>print('\nOuter:\n', join_o['_merge'].value_counts(), s...</code> | g14/demo.py |
| core | file, reading one line at a time | <code>for line in fh:</code> | g05/demo.py |
| core | for, looping through a list | <code>for n in a_list:</code> | g04/demo.py |
| core | for, looping through a list of tuples | <code>for number,name in div_info:</code> | g13/demo.py |
| core | function, calling | <code>d1_ssq = sumsq(d1)</code> | g06/demo.py |
| core | function, calling with an optional argument | <code>sample_function(100, 10, r=0.07)</code> | g07/demo.py |
| core | function, defining | <code>def sumsq(values):</code> | g06/demo.py |
| core | function, defining with optional argument | <code>def sample_function(payment,year,r=0.05):</code> | g07/demo.py |
| core | function, returning a result | <code>return values</code> | g06/demo.py |
| core | list, appending an element | <code>a_list.append("four")</code> | g03/demo.py |
| core | list, create via comprehension | <code>cubes = [n**3 for n in a_list]</code> | g04/demo.py |
| core | list, creating | <code>a_list = ["zero", "one", "two", "three"]</code> | g03/demo.py |
| core | list, determining length | <code>n = len(b_list)</code> | g03/demo.py |
| core | list, extending with another list | <code>a_list.extend(a_more)</code> | g03/demo.py |
| core | list, generating a sequence | <code>b_list = range(1,6)</code> | g04/demo.py |
| core | list, joining with spaces | <code>a_string = " ".join(a_list)</code> | g03/demo.py |
| core | list, selecting an element | <code>print(a_list[0])</code> | g03/demo.py |
| core | list, selecting elements 0 to 3 | <code>print(a_list[:4])</code> | g03/demo.py |
| core | list, selecting elements 1 to 2 | <code>print(a_list[1:3])</code> | g03/demo.py |

| Module | Description | Example | Script |
|-----------|---------------------------------------|---|-------------|
| core | list, selecting elements 1 to the end | <code>print(a_list[1:])</code> | g03/demo.py |
| core | list, selecting last 3 elements | <code>print(a_list[-3:])</code> | g03/demo.py |
| core | list, selecting the last element | <code>print(a_list[-1])</code> | g03/demo.py |
| core | list, sorting | <code>c_sort = sorted(b_list)</code> | g03/demo.py |
| core | list, summing | <code>tot_inc = sum(incomes)</code> | g08/demo.py |
| core | math, raising a number to a power | <code>a_cubes.append(n**3)</code> | g04/demo.py |
| core | math, rounding a number | <code>rounded = round(ratio,2)</code> | g05/demo.py |
| core | sets, computing difference | <code>print(name_states - pop_states)</code> | g13/demo.py |
| core | sets, creating | <code>name_states = set(name_data['State'])</code> | g13/demo.py |
| core | sets, of tuples | <code>tset1 = set([(1,2), (2,3), (1,3), (2,3)])</code> | g13/demo.py |
| core | string, concatenating | <code>name = s1+" "+s2+" "+s3</code> | g02/demo.py |
| core | string, converting to an int | <code>values.append(int(line))</code> | g06/demo.py |
| core | string, creating | <code>filename = "demo.txt"</code> | g02/demo.py |
| core | string, including a newline character | <code>fh.write(name+"!\n")</code> | g02/demo.py |
| core | string, matching partial string | <code>is_gas = trim['DBA Name'].str.contains('XPRESS')</code> | g27/demo.py |
| core | string, matching start | <code>is_big = trim['DBA Name'].str.startswith(store)</code> | g27/demo.py |
| core | string, splitting on a comma | <code>parts = line.split(',')</code> | g05/demo.py |
| core | string, splitting on whitespace | <code>b_list = b_string.split()</code> | g03/demo.py |
| core | string, stripping blank space | <code>clean = [item.strip() for item in parts]</code> | g05/demo.py |
| core | type, obtaining for a variable | <code>print('\nraw_states is a DataFrame object:', type(raw_...</code> | g09/demo.py |
| csv | setting up a DictReader object | <code>reader = csv.DictReader(fh)</code> | g08/demo.py |
| fiona | importing the module | <code>import fiona</code> | g24/demo.py |
| fiona | list layers in a geopackage | <code>layers = fiona.listlayers(demo_file)</code> | g24/demo.py |
| geopandas | adding a heatmap legend | <code>slices.plot('s_pop',edgecolor='yellow',linewidth=0.2,le...</code> | g26/demo.py |
| geopandas | clip a layer | <code>zips_clip = zips.clip(county,keep_geom_type=True)</code> | g24/demo.py |
| geopandas | combine all geographies in a layer | <code>water_dis = water_by_name.dissolve()</code> | g24/demo.py |
| geopandas | combine geographies by attribute | <code>water_by_name = water.dissolve('FULLNAME')</code> | g24/demo.py |
| geopandas | computing areas | <code>zips['z_area'] = zips.area</code> | g26/demo.py |
| geopandas | construct a buffer | <code>near_water = water_dis.buffer(1600)</code> | g24/demo.py |
| geopandas | constructing centroids | <code>centroids['geometry'] = tracts.centroid</code> | g27/demo.py |

| Module | Description | Example | Script |
|------------|---|--|--------------------------|
| geopandas | drawing a heatmap | <code>near_wv.plot("mil", cmap='Blues', legend=True, ax=ax)</code> | <code>g22/demo.py</code> |
| geopandas | extracting geometry from a geodataframe | <code>wv_geo = wv['geometry']</code> | <code>g22/demo.py</code> |
| geopandas | importing the module | <code>import geopandas as gpd</code> | <code>g21/demo.py</code> |
| geopandas | join to nearest object | <code>served_by = centroids.sjoin_nearest(geo, how='left', dist. . .</code> | <code>g27/demo.py</code> |
| geopandas | merging data onto a geodataframe | <code>conus = conus.merge(trim, on='STATEFP', how='left', valida. . .</code> | <code>g22/demo.py</code> |
| geopandas | obtaining coordinates | <code>print('Number of points:', len(wv_geo.exterior.coords). . .</code> | <code>g22/demo.py</code> |
| geopandas | overlaying a layer using union | <code>slices = zips.overlay(county, how='union', keep_geom_type. . .</code> | <code>g26/demo.py</code> |
| geopandas | plot with categorical coloring | <code>sel.plot('NAME', cmap='Dark2', ax=ax1)</code> | <code>g22/demo.py</code> |
| geopandas | plotting a boundary | <code>syr.boundary.plot(color='gray', linewidth=1, ax=ax1)</code> | <code>g21/demo.py</code> |
| geopandas | project a layer | <code>county = county.to_crs(epsg=utm18n)</code> | <code>g24/demo.py</code> |
| geopandas | reading a file | <code>syr = gpd.read_file("tl_2016_36_place-syracuse.zip")</code> | <code>g21/demo.py</code> |
| geopandas | reading a shapefile | <code>states = gpd.read_file("cb_2019_us_state_500k.zip")</code> | <code>g22/demo.py</code> |
| geopandas | reading data in WKT format | <code>coords = gpd.GeoSeries.from_wkt(big['Georeference'])</code> | <code>g27/demo.py</code> |
| geopandas | setting the color of a plot | <code>county.plot(color='tan', ax=ax1)</code> | <code>g24/demo.py</code> |
| geopandas | setting transparency via alpha | <code>near_clip.plot(alpha=0.25, ax=ax1)</code> | <code>g24/demo.py</code> |
| geopandas | spatial join, contains | <code>c_contains_z = county.sjoin(zips, how='right', predicate=. . .</code> | <code>g25/demo.py</code> |
| geopandas | spatial join, crosses | <code>i_crosses_z = inter.sjoin(zips, how='right', predicate='c. . .</code> | <code>g25/demo.py</code> |
| geopandas | spatial join, intersects | <code>z_intersect_c = zips.sjoin(county, how='left', predicate=. . .</code> | <code>g25/demo.py</code> |
| geopandas | spatial join, overlaps | <code>z_overlaps_c = zips.sjoin(county, how='left', predicate=' . . .</code> | <code>g25/demo.py</code> |
| geopandas | spatial join, touches | <code>z_touch_c = zips.sjoin(county, how='left', predicate='tou. . .</code> | <code>g25/demo.py</code> |
| geopandas | spatial join, within | <code>z_within_c = zips.sjoin(county, how='left', predicate='wi. . .</code> | <code>g25/demo.py</code> |
| geopandas | testing if rows touch a geometry | <code>touches_wv = conus.touches(wv_geo)</code> | <code>g22/demo.py</code> |
| geopandas | writing a layer to a geodatabase | <code>conus.to_file("conus.gpkg", layer="states")</code> | <code>g22/demo.py</code> |
| json | importing the module | <code>import json</code> | <code>g05/demo.py</code> |
| json | using to print an object nicely | <code>print(json.dumps(list1, indent=4))</code> | <code>g05/demo.py</code> |
| matplotlib | axes, adding a horizontal line | <code>ax21.axhline(medians['etr'], c='r', ls='-', lw=1)</code> | <code>g12/demo.py</code> |
| matplotlib | axes, adding a vertical line | <code>ax21.axvline(medians['inc'], c='r', ls='-', lw=1)</code> | <code>g12/demo.py</code> |
| matplotlib | axes, labeling the X axis | <code>ax2.set_xlabel('Millions')</code> | <code>g11/demo.py</code> |
| matplotlib | axes, labeling the Y axis | <code>ax1.set_ylabel("Millions")</code> | <code>g11/demo.py</code> |
| matplotlib | axes, turning off the label | <code>ax.set_ylabel(None)</code> | <code>g13/demo.py</code> |
| matplotlib | axis, turning off | <code>ax1.axis('off')</code> | <code>g26/demo.py</code> |
| matplotlib | changing marker size | <code>geo.plot(color='blue', markersize=1, ax=ax1)</code> | <code>g27/demo.py</code> |
| matplotlib | colors, xkcd palette | <code>syr.plot(color='xkcd:lightblue', ax=ax1)</code> | <code>g21/demo.py</code> |
| matplotlib | figure, adding a title | <code>fig2.suptitle('Pooled Data')</code> | <code>g12/demo.py</code> |
| matplotlib | figure, four panel grid | <code>fig3, axs = plt.subplots(2,2, sharex=True, sharey=True)</code> | <code>g12/demo.py</code> |

| Module | Description | Example | Script |
|------------|--|---|--------------------------|
| matplotlib | figure, left and right panels | <code>fig2, (ax21,ax22) = plt.subplots(1,2)</code> | <code>g12/demo.py</code> |
| matplotlib | figure, saving | <code>fig2.savefig('figure.png')</code> | <code>g11/demo.py</code> |
| matplotlib | figure, setting the size | <code>fig, axs = plt.subplots(1,2,figsize=(12,6))</code> | <code>g20/demo.py</code> |
| matplotlib | figure, tuning the layout | <code>fig2.tight_layout()</code> | <code>g11/demo.py</code> |
| matplotlib | figure, working with a list of axes | <code>for ax in axs:</code> | <code>g20/demo.py</code> |
| matplotlib | importing pyplot | <code>import matplotlib.pyplot as plt</code> | <code>g11/demo.py</code> |
| matplotlib | setting an edge color | <code>slices.plot('COUNTYFP',edgecolor='yellow',linewidth=0.2...</code> | <code>g26/demo.py</code> |
| matplotlib | setting the default resolution | <code>plt.rcParams['figure.dpi'] = 300</code> | <code>g11/demo.py</code> |
| matplotlib | using subplots to set up a figure | <code>fig1, ax1 = plt.subplots()</code> | <code>g11/demo.py</code> |
| os | delete a file | <code>os.remove(out_file)</code> | <code>g24/demo.py</code> |
| os | importing the module | <code>import os</code> | <code>g24/demo.py</code> |
| os | test if a file or directory exists | <code>if os.path.exists(out_file):</code> | <code>g24/demo.py</code> |
| pandas | RE, replacing a digit or space | <code>unit_part = values.str.replace(r'\d \s', "", regex=True)</code> | <code>g23/demo.py</code> |
| pandas | RE, replacing a non-digit or space | <code>value_part = values.str.replace(r'\D \s', "", regex=True)</code> | <code>g23/demo.py</code> |
| pandas | RE, replacing a non-word character | <code>units = units.str.replace(r'\W', "", regex=True)</code> | <code>g23/demo.py</code> |
| pandas | columns, dividing with explicit alignment | <code>normed2 = 100*states.div(pa_row,axis='columns')</code> | <code>g09/demo.py</code> |
| pandas | columns, listing names | <code>print('\nColumns:', list(raw_states.columns))</code> | <code>g09/demo.py</code> |
| pandas | columns, renaming | <code>county = county.rename(columns={'B01001_001E':'pop'})</code> | <code>g10/demo.py</code> |
| pandas | columns, retrieving one by name | <code>pop = states['pop']</code> | <code>g09/demo.py</code> |
| pandas | columns, retrieving several by name | <code>print(pop[some_states]/1e6)</code> | <code>g09/demo.py</code> |
| pandas | dataframe, appending | <code>gen_all = pd.concat([gen_oswego, gen_onondaga])</code> | <code>g15/demo.py</code> |
| pandas | dataframe, boolean row selection | <code>print(trim[has_AM], "\n")</code> | <code>g12/demo.py</code> |
| pandas | dataframe, dropping a column | <code>both = both.drop(columns='_merge')</code> | <code>g15/demo.py</code> |
| pandas | dataframe, dropping duplicates | <code>flood = flood.drop_duplicates(subset='TAX_ID')</code> | <code>g14/demo.py</code> |
| pandas | dataframe, dropping missing data | <code>trim = demo.dropna(subset="Days")</code> | <code>g12/demo.py</code> |
| pandas | dataframe, finding duplicate records | <code>dups = parcels.duplicated(subset='TAX_ID', keep=False...</code> | <code>g14/demo.py</code> |
| pandas | dataframe, getting a block of rows via index | <code>sel = merged.loc[number]</code> | <code>g13/demo.py</code> |
| pandas | dataframe, inner 1:1 merge | <code>join_i = parcels.merge(flood, how='inner', on="TAX_ID",...</code> | <code>g14/demo.py</code> |
| pandas | dataframe, inner join | <code>merged = name_data.merge(pop_data,left_on="State",right...</code> | <code>g13/demo.py</code> |
| pandas | dataframe, left 1:1 merge | <code>join_l = parcels.merge(flood, how='left', on="TAX_ID",...</code> | <code>g14/demo.py</code> |
| pandas | dataframe, left m:1 merge | <code>both = gen_all.merge(plants, how='left', on='Plant Code...</code> | <code>g15/demo.py</code> |
| pandas | dataframe, making a copy | <code>trim = trim.copy()</code> | <code>g12/demo.py</code> |
| pandas | dataframe, outer 1:1 merge | <code>join_o = parcels.merge(flood, how='outer', on="TAX_ID",...</code> | <code>g14/demo.py</code> |

| Module | Description | Example | Script |
|--------|--|---|--------------------------|
| pandas | dataframe, reading zipped pickle format | <code>sample2 = pd.read_pickle('sample_pkl.zip')</code> | <code>g16/demo.py</code> |
| pandas | dataframe, resetting the index | <code>hourly = hourly.reset_index()</code> | <code>g17/demo.py</code> |
| pandas | dataframe, right 1:1 merge | <code>join_r = parcels.merge(flood, how='right', on="TAX_ID",...</code> | <code>g14/demo.py</code> |
| pandas | dataframe, saving in zipped pickle format | <code>sample.to_pickle('sample_pkl.zip')</code> | <code>g16/demo.py</code> |
| pandas | dataframe, selecting rows by list indexing | <code>print(low_to_high[-5:])</code> | <code>g09/demo.py</code> |
| pandas | dataframe, selecting rows via boolean | <code>dup_rec = flood[dups]</code> | <code>g14/demo.py</code> |
| pandas | dataframe, selecting rows via query | <code>trimmed = county.query("state == '04' or state == '36' ")</code> | <code>g10/demo.py</code> |
| pandas | dataframe, set index keeping the column | <code>states = states.set_index('STUSPS',drop=False)</code> | <code>g22/demo.py</code> |
| pandas | dataframe, shape attribute | <code>print('number of rows, columns:', conus.shape)</code> | <code>g22/demo.py</code> |
| pandas | dataframe, sorting by a column | <code>county = county.sort_values('pop')</code> | <code>g10/demo.py</code> |
| pandas | dataframe, sorting by index | <code>summary = summary.sort_index(ascending=False)</code> | <code>g15/demo.py</code> |
| pandas | dataframe, summing a boolean | <code>print('\nduplicate parcels:', dups.sum())</code> | <code>g14/demo.py</code> |
| pandas | dataframe, unstacking an index level | <code>bymo = bymo.unstack('month')</code> | <code>g17/demo.py</code> |
| pandas | dataframe, using a multilevel column index | <code>means = grid['mean']</code> | <code>g20/demo.py</code> |
| pandas | dataframe, using xs to select a subset | <code>print(county.xs('04',level='state'))</code> | <code>g10/demo.py</code> |
| pandas | dataframe, using xs with columns | <code>c1 = grid.xs('c1',axis='columns',level=1)</code> | <code>g20/demo.py</code> |
| pandas | dataframe, writing to a CSV file | <code>merged.to_csv('demo-merged.csv')</code> | <code>g13/demo.py</code> |
| | | | |
| pandas | datetime, building via to_datetime() | <code>date = pd.to_datetime(recs['ts'])</code> | <code>g14/demo.py</code> |
| pandas | datetime, building with a format | <code>ymd = pd.to_datetime(sample['TRANSACTION_DT'], format=...</code> | <code>g16/demo.py</code> |
| pandas | datetime, extracting day attribute | <code>recs['day'] = date.dt.day</code> | <code>g14/demo.py</code> |
| pandas | datetime, extracting hour attribute | <code>recs['hour'] = date.dt.hour</code> | <code>g14/demo.py</code> |
| | | | |
| pandas | general, display information about object | <code>sample.info()</code> | <code>g16/demo.py</code> |
| pandas | general, displaying all columns | <code>pd.set_option('display.max_columns',None)</code> | <code>g16/demo.py</code> |
| pandas | general, displaying all rows | <code>pd.set_option('display.max_rows', None)</code> | <code>g09/demo.py</code> |
| pandas | general, importing the module | <code>import pandas as pd</code> | <code>g09/demo.py</code> |
| pandas | general, using copy_on_write mode | <code>pd.options.mode.copy_on_write = True</code> | <code>g16/demo.py</code> |
| pandas | general, using qcut to create deciles | <code>dec = pd.qcut(county['pop'], 10, labels=range(1,11))</code> | <code>g10/demo.py</code> |
| | | | |
| pandas | groupby, cumulative sum within group | <code>cumulative_inc = group_by_state['pop'].cumsum()</code> | <code>g10/demo.py</code> |
| pandas | groupby, descriptive statistics | <code>inc_stats = group_by_state['pop'].describe()</code> | <code>g10/demo.py</code> |
| pandas | groupby, iterating over groups | <code>for t,g in group_by_state:</code> | <code>g10/demo.py</code> |
| pandas | groupby, median of each group | <code>pop_med = group_by_state['pop'].median()</code> | <code>g10/demo.py</code> |
| pandas | groupby, quantile of each group | <code>pop_25th = group_by_state['pop'].quantile(0.25)</code> | <code>g10/demo.py</code> |
| pandas | groupby, return group number | <code>groups = group_by_state.ngroup()</code> | <code>g10/demo.py</code> |
| pandas | groupby, return number within group | <code>seqnum = group_by_state.cumcount()</code> | <code>g10/demo.py</code> |

| Module | Description | Example | Script |
|--------|---|---|-------------|
| pandas | groupby, return rank within group | <code>rank_age = group_by_state['pop'].rank()</code> | g10/demo.py |
| pandas | groupby, select first records | <code>first2 = group_by_state.head(2)</code> | g10/demo.py |
| pandas | groupby, select largest values | <code>largest = group_by_state['pop'].nlargest(2)</code> | g10/demo.py |
| pandas | groupby, select last records | <code>last2 = group_by_state.tail(2)</code> | g10/demo.py |
| pandas | groupby, size of each group | <code>num_rows = group_by_state.size()</code> | g10/demo.py |
| pandas | groupby, sum of each group | <code>state = county.groupby('state')['pop'].sum()</code> | g10/demo.py |
| pandas | index, creating with 3 levels | <code>county = county.set_index(['state','county', 'NAME'])</code> | g10/demo.py |
| pandas | index, listing names | <code>print('\nIndex (rows):', list(raw_states.index))</code> | g09/demo.py |
| pandas | index, renaming values | <code>div_pop = div_pop.rename(index=div_names)</code> | g11/demo.py |
| pandas | index, retrieving a row by name | <code>pa_row = states.loc['Pennsylvania']</code> | g09/demo.py |
| pandas | index, retrieving first rows by location | <code>print(low_to_high.iloc[0:10])</code> | g09/demo.py |
| pandas | index, retrieving last rows by location | <code>print(low_to_high.iloc[-5:])</code> | g09/demo.py |
| pandas | index, setting to a column | <code>states = raw_states.set_index('name')</code> | g09/demo.py |
| pandas | plotting, bar plot | <code>reg_pop.plot.bar(title='Population',ax=ax1)</code> | g11/demo.py |
| pandas | plotting, histogram | <code>hh_data['etr'].plot.hist(ax=ax1,bins=20,title='Distribu. . .</code> | g12/demo.py |
| pandas | plotting, horizontal bar plot | <code>div_pop.plot.barh(title='Population',ax=ax2)</code> | g11/demo.py |
| pandas | plotting, scatter colored by 3rd var | <code>tidy_data.plot.scatter(ax=ax4,x='Income',y='ETR',c='typ. . .</code> | g12/demo.py |
| pandas | plotting, scatter plot | <code>hh_data.plot.scatter(ax=ax21,x='inc',y='etr',title='ETR. . .</code> | g12/demo.py |
| pandas | plotting, turning off legend | <code>sel.plot.barh(x='Name',y='percent',ax=ax,legend=None)</code> | g13/demo.py |
| pandas | reading, csv data | <code>raw_states = pd.read_csv('state-data.csv')</code> | g09/demo.py |
| pandas | reading, from an open file handle | <code>gen_oswego = pd.read_csv(fh1)</code> | g15/demo.py |
| pandas | reading, setting index column | <code>state_data = pd.read_csv('state-data.csv',index_col='na. . .</code> | g11/demo.py |
| pandas | reading, using dtype dictionary | <code>county = pd.read_csv('county_pop.csv',dtype=fips)</code> | g10/demo.py |
| pandas | series, RE at start | <code>is_LD = trim['Number'].str.contains(r"12")</code> | g12/demo.py |
| pandas | series, applying a function to each element | <code>name_clean = name_parts.apply(' '.join)</code> | g23/demo.py |
| pandas | series, automatic alignment by index | <code>merged['percent'] = 100*merged['pop']/div_pop</code> | g13/demo.py |
| pandas | series, combining via where() | <code>mod['comb_units'] = unit_part.where(unit_part!="", mo. . .</code> | g23/demo.py |
| pandas | series, contains RE or RE | <code>is_TT = trim['Days'].str.contains(r"Tu Th")</code> | g12/demo.py |
| pandas | series, contains a plain string | <code>has_AM = trim['Time'].str.contains("AM")</code> | g12/demo.py |
| pandas | series, contains an RE | <code>has_AMPM = trim['Time'].str.contains("AM.*PM")</code> | g12/demo.py |
| pandas | series, converting strings to title case | <code>fixname = subset_view['NAME'].str.title()</code> | g16/demo.py |
| pandas | series, converting to a list | <code>print(name_data['State'].to_list())</code> | g13/demo.py |
| pandas | series, converting to lower case | <code>name = mod['name'].str.lower()</code> | g23/demo.py |

| Module | Description | Example | Script |
|----------|--|---|--------------------------|
| pandas | series, dropping rows using a list | <code>conus = states.drop(not_conus)</code> | <code>g22/demo.py</code> |
| pandas | series, element-by-element or | <code>is_either = is_ca is_tx</code> | <code>g16/demo.py</code> |
| pandas | series, filling missing values | <code>mod['comb_units'] = mod['comb_units'].fillna('feet')</code> | <code>g23/demo.py</code> |
| pandas | series, removing spaces | <code>units = units.str.strip()</code> | <code>g23/demo.py</code> |
| pandas | series, replacing values using a dictionary | <code>units = units.replace(spellout)</code> | <code>g23/demo.py</code> |
| pandas | series, retrieving an element | <code>print("\nFlorida's population:", pop['Florida']/1e6)</code> | <code>g09/demo.py</code> |
| pandas | series, sort in decending order | <code>div_pop = div_pop.sort_values(ascending=False)</code> | <code>g11/demo.py</code> |
| pandas | series, sorting by value | <code>low_to_high = normed['med_pers_inc'].sort_values()</code> | <code>g09/demo.py</code> |
| pandas | series, splitting strings on whitespace | <code>name_parts = name.str.split()</code> | <code>g23/demo.py</code> |
| pandas | series, splitting via RE | <code>trim['Split'] = trim["Time"].str.split(r": - ")</code> | <code>g12/demo.py</code> |
| pandas | series, splitting with expand | <code>exp = trim["Time"].str.split(r": - ", expand=True)</code> | <code>g12/demo.py</code> |
| pandas | series, summing | <code>reg_pop = by_reg['pop'].sum()/1e6</code> | <code>g11/demo.py</code> |
| pandas | series, unstacking | <code>tot_wide = tot_amt.unstack('PGI')</code> | <code>g16/demo.py</code> |
| pandas | series, using <code>isin()</code> | <code>fixed = flood['TAX_ID'].isin(dup_rec['TAX_ID'])</code> | <code>g14/demo.py</code> |
| pandas | series, using <code>value_counts()</code> | <code>print('\nOuter:\n', join_o['_merge'].value_counts(), s...</code> | <code>g14/demo.py</code> |
| requests | calling the <code>get()</code> method | <code>response = requests.get(api,payload)</code> | <code>g18/demo.py</code> |
| requests | checking the URL | <code>print('url:', response.url)</code> | <code>g18/demo.py</code> |
| requests | checking the response text | <code>print(response.text)</code> | <code>g18/demo.py</code> |
| requests | checking the status code | <code>print('status:', response.status_code)</code> | <code>g18/demo.py</code> |
| requests | decoding a JSON response | <code>rows = response.json()</code> | <code>g18/demo.py</code> |
| requests | importing the module | <code>import requests</code> | <code>g18/demo.py</code> |
| scipy | calling newton's method | <code>cr = opt.newton(find_cube_root,xinit,maxiter=20,args=[y...</code> | <code>g07/demo.py</code> |
| scipy | importing the module | <code>import scipy.optimize as opt</code> | <code>g07/demo.py</code> |
| seaborn | adding a title to a grid object | <code>jg.fig.suptitle('Distribution of Hourly Load')</code> | <code>g17/demo.py</code> |
| seaborn | barplot | <code>hue='month',palette='deep',ax=ax1)</code> | <code>g17/demo.py</code> |
| seaborn | basic violin plot | <code>sns.violinplot(data=janjul,x="month",y="usage")</code> | <code>g17/demo.py</code> |
| seaborn | boxenplot | <code>sns.boxenplot(data=janjul,x="month",y="usage")</code> | <code>g17/demo.py</code> |
| seaborn | calling <code>tight_layout</code> on a grid object | <code>jg.fig.tight_layout()</code> | <code>g17/demo.py</code> |
| seaborn | drawing a heatmapped grid | <code>sns.heatmap(means,annot=True,fmt=".0f",cmap='Spectral',...</code> | <code>g20/demo.py</code> |
| seaborn | importing the module | <code>import seaborn as sns</code> | <code>g17/demo.py</code> |
| seaborn | joint distribution hex plot | <code>jg = sns.jointplot(data=bymo,x=1,y=7,kind='hex')</code> | <code>g17/demo.py</code> |
| seaborn | setting axis titles on a grid object | <code>jg.set_axis_labels('January','July')</code> | <code>g17/demo.py</code> |
| seaborn | setting the theme | <code>sns.set_theme(style="white")</code> | <code>g17/demo.py</code> |
| seaborn | split violin plot | <code>hue="month",palette='deep',split=True)</code> | <code>g17/demo.py</code> |

| Module | Description | Example | Script |
|---------|------------------------------|--|--------------------------|
| zipfile | importing the module | <code>import zipfile</code> | <code>g15/demo.py</code> |
| zipfile | opening a file in an archive | <code>fh1 = archive.open('generators-oswego.csv')</code> | <code>g15/demo.py</code> |
| zipfile | opening an archive | <code>archive = zipfile.ZipFile('generators.zip')</code> | <code>g15/demo.py</code> |
| zipfile | reading the list of files | <code>print(archive.namelist())</code> | <code>g15/demo.py</code> |