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

Module	Description	Example	Script
core	list, selecting last 3 elements	print(a_list[-3:])	g03/demo.py
core	list, selecting the last element	print(a_list[-1])	g03/demo.py
core	list, sorting	c_sort = sorted(b_list)	g03/demo.py
core	list, summing	tot_inc = sum(incomes)	g08/demo.py
core	math, raising a number to a power	a_cubes.append(n**3)	g04/demo.py
core	math, rounding a number	rounded = round(ratio, 2)	g05/demo.py
core	sets, computing difference	print(name_states - pop_states)	g13/demo.py
core	sets, creating	$name_states = set(\ name_data[`State']\)$	g13/demo.py
core	sets, of tuples	tset1 = set([(1,2), (2,3), (1,3), (2,3)])	g13/demo.py
core	string, concatenating	name = $s1+""+s2+""+s3$	g02/demo.py
core	string, converting to an int	values.append(int(line))	${ m g06/demo.py}$
core	string, creating	filename = "demo.txt"	g02/demo.py
core	string, including a newline character	$fh.write(name+"!\n")$	g02/demo.py
core	string, splitting on a comma	parts = line.split(',')	${\sf g05/demo.py}$
core	string, splitting on whitespace	$b_list = b_string.split()$	g03/demo.py
core	string, stripping blank space	$clean = [item.strip() \; for \; item \; in \; parts]$	g05/demo.py
core	type, obtaining for a variable	<pre>print('\nraw_states is a DataFrame object:', type(raw</pre>	g09/demo.py
CSV	setting up a DictReader object	reader = csv.DictReader(fh)	g08/demo.py
fiona	importing the module	import fiona	g24/demo.py
fiona	list layers in a geopackage	${\sf layers} = {\sf fiona.listlayers(demo_file)}$	g24/demo.py
geopandas	clip a layer	zips_clip = zips.clip(county,keep_geom_type=True)	g24/demo.py
geopandas	combine all geographies in a layer	$water_dis = water_by_name.dissolve()$	g24/demo.py
geopandas	combine geographies by attribute	$water_by_name = water.dissolve('FULLNAME')$	g24/demo.py
geopandas	construct a buffer	$near_water = water_dis.buffer(1600)$	g24/demo.py
geopandas	extracting geometry from a geodataframe	$wv_geo = wv['geometry']$	g22/demo.py
geopandas	importing the module	import geopandas as gpd	${\sf g21/demo.py}$
geopandas	merging data onto a geodataframe	$conus = conus.merge(trim, on = `STATEFP', how = `left', valida. \ . \ .$	g22/demo.py
geopandas	obtaining coordinates	<pre>print('Number of points:', len(wv_geo.exterior.coords)</pre>	g22/demo.py
geopandas	plot with categorical coloring	sel.plot('NAME',cmap='Dark2',ax=ax1)	g22/demo.py
geopandas	plotting a boundary	syr.boundary.plot(color='gray',linewidth=1,ax=ax1)	g21/demo.py

geopandas project a layer country = country = country to_crs(epsg=utm18n)	Module	Description	Example	Script
geopandas reading a file states = gpd.read_file("tl_2016_36_place-syracuse.zip") g22/demo.py geopandas reading a shapefile states = gpd.read_file("cb_2019_us_state_500k.zip") g22/demo.py geopandas setting transparency via alpha near_clip.plot(color='tan_ax=ax1) g24/demo.py geopandas setting transparency via alpha near_clip.plot(alpha=0.25,ax=ax1) g24/demo.py geopandas spatial join, contains c_contains_z = country,sjoin(zips.how='right'.predicate='.c. g25/demo.py geopandas spatial join, rorsese i_crosses_z = inters.join(zips.how='right'.predicate='.c. g25/demo.py geopandas spatial join, vibrage z_overlaps_c= zips.sjoin(county.how='left'.predicate='.c. g25/demo.py geopandas spatial join, vibrage z_overlaps_c= zips.sjoin(county.how='left'.predicate='.c. g25/demo.py geopandas spatial join, within c_z zips.sjoin(county.how='left'.predicate='.c. g25/demo.py geopandas spatial join, within c_z zips.sjoin(county.how='left'.predicate='.c. g25/demo.py geopandas testing if rows touch a geometry touches_w= conus.touches(w_geo) g22/demo.py geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/demo.py geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/demo.py goopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/demo.py goopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/demo.py g05/demo.py ipon using to print an object nicely print(json.dumps(list1.indent=4)) g05/demo.py g05/demo.py print(json.dumps(list1.indent=4)) g05/demo.py g05	geopandas	project a layer	county = county.to_crs(epsg=utm18n)	g24/demo.py
geopandas reading a shapefile states = gpd.read_file("ch_2019_us_state_500k.zip") g22/demo.py geopandas setting the color of a plot county.plot(color='tan',ax=ax1) g24/demo.py geopandas setting transparency via alpha near_clip.plot(alpha=0.25,ax=ax1) g24/demo.py geopandas spatial join, contains c_contains_z = county.sjoin(zips,how='right',predicate=-: g25/demo.py geopandas spatial join, crosses i_crosses_z = intersect_s = zips.sjoin(county,how='left',predicate=-: g25/demo.py geopandas spatial join, intersects z_intersect_c = zips.sjoin(county,how='left',predicate=-: g25/demo.py geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate=-tou g25/demo.py geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate=-tou g25/demo.py geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate=-tou g25/demo.py geopandas writing a layer to a geodatabase conus.touches_ww= conus.touches_wv_geopandas writing a layer to a geodatabase conus.touches_ww= conus.touches_wv_geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/demo.py g22/demo.p	geopandas	reading a file	syr = gpd.read_file("tl_2016_36_place-syracuse.zip")	g21/demo.py
geopandas setting transparency via alpha cardina plot (alpha=0.25, ax=ax1) ge24/demo, py geopandas spatial join, contains contains contains, and contains spatial join, corosses i crosses_z = inter.sjoin(zips.how=right', predicate= g25/demo.py geopandas spatial join, intersects z_intersect_c = zips.sjoin(county,how=left', predicate= g25/demo.py geopandas spatial join, touches z_overlaps_c = zips.sjoin(county,how=left', predicate= g25/demo.py geopandas spatial join, touches z_touch_c = zips.sjoin(county,how=left', predicate= g25/demo.py geopandas spatial join, within z_within_c = zips.sjoin(county,how=left', predicate='tou g25/demo.py geopandas testing if rows touch a geometry touches_w= conus.touches(ww_geo) g22/demo.py geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/demo.py geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/demo.py g22/demo.py g2	geopandas	reading a shapefile	states = gpd.read_file("cb_2019_us_state_500k.zip")	
geopandas spatial join, contains c_contains_z = county, sjoin(zips, how="right", predicate= g25/demo, py geopandas spatial join, crosses i_crosses_z = inter. sjoin(zips, how="right", predicate="c g25/demo, py geopandas spatial join, vitersects z_intersect_c = zips, sjoin(county, how="left", predicate= g25/demo, py geopandas spatial join, overlaps z_overlaps_c = zips, sjoin(county, how="left", predicate=-' g25/demo, py geopandas spatial join, vithin z_within_c = zips. sjoin(county, how="left", predicate=-' g25/demo, py geopandas spatial join, vithin z_within_c = zips. sjoin(county, how="left", predicate=-'wi g25/demo, py geopandas spatial join, vithin z_within_c = zips. sjoin(county, how="left", predicate=-wi g25/demo, py geopandas testing if rows touch a geometry touches_wv = conus. touches(wv_geo) g22/demo, py geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg", layer="states") g22/demo, py geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg", layer="states") g25/demo, py goopandas writing a layer to a geodatabase conus.to_file("conus.gpkg", layer="states") g25/demo, py goopandas using to print an object nicely print(json.dumps(list1, indent=4)) g25/demo, py goopandas point an object nicely print(json.dumps(list1, indent=4)) g25/demo, py goopandas asset, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/demo, py g11/demo, py g21/demo, py	geopandas	setting the color of a plot	county.plot(color='tan',ax=ax1)	g24/demo.py
geopandas spatial join, crosses i_crosses_z = inter.sjoin(zips,how='right',predicate='c g25/demo.py geopandas spatial join, intersects z_intersect_c = zips.sjoin(county,how='left',predicate=' g25/demo.py geopandas spatial join, touches z_coverlaps_c = zips.sjoin(county,how='left',predicate=' g25/demo.py geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate='tou g25/demo.py geopandas testing if rows touch a geometry touches_w=c conus.touches(ww_geo) g22/demo.py geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/demo.py geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/demo.py geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/demo.py g05/demo.py json using to print an object nicely print(json.dumps(list1,indent=4)) g05/demo.py g11/demo.py g11/demo.p	geopandas	setting transparency via alpha	near_clip.plot(alpha=0.25,ax=ax1)	g24/demo.py
geopandas spatial join, intersects z_intersect_c = zips.sjoin(county,how='left', predicate= g25/demo.py geopandas spatial join, overlaps z_overlaps_c = zips.sjoin(county,how='left', predicate= g25/demo.py geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left', predicate=-!ou g25/demo.py geopandas spatial join, within z_within_c = zips.sjoin(county,how='left', predicate=-!ou g25/demo.py geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/demo.py writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/demo.py json importing the module import json using to print an object nicely print(json.dumps(list1,indent=4)) g05/demo.py json using to print an object nicely print(json.dumps(list1,indent=4)) g12/demo.py matplotlib axes, adding a vertical line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/demo.py matplotlib axes, albeling the X axis ax1.set_xlabel("Millions") g11/demo.py matplotlib axes, labeling the Y axis ax1.set_xlabel("Millions") g11/demo.py matplotlib axes, setting a title ax.set_ylabel("Population, Millions") g11/demo.py matplotlib axes, sxkd palette syr.plot(color=kxcd:lightblue*,ax=ax1) g11/demo.py matplotlib colors, xkcd palette syr.plot(color=kxcd:lightblue*,ax=ax1) g11/demo.py matplotlib figure, adding a title fig2.suptitle("Pooled Data') g12/demo.py matplotlib figure, saving fig1.savefig("figure.png") g11/demo.py matplotlib figure, saving fig1.savefig("figure.png") g11/demo.py matplotlib figure, saving the layout fig1.savefig("figure.png") g11/demo.py matplotlib figure, working with a list of axes fig. ax21.ax3 p1.suplots(1,2,figsize=(12,6)) g20/demo.py matplotlib figure, working with a list of axes fig.1.savefig("figure.png") g11/demo.py matplotlib figure, working with a list of axes fig.1.savefig("figure.png") g10/demo.py matplotlib importing pyplot importi	geopandas	spatial join, contains	c_contains_z = county.sjoin(zips,how='right',predicate=	g25/demo.py
geopandas spatial join, overlaps z_overlaps_c = zips.sjoin(county,how="left', predicate=" g25/demo.py geopandas spatial join, touches z_touch_c = zips.sjoin(county,how="left', predicate="tou g25/demo.py geopandas spatial join, within z_within_c = zips.sjoin(county,how="left', predicate="tou g25/demo.py geopandas spatial join, within z_within_c = zips.sjoin(county,how="left', predicate="wi g25/demo.py geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/demo.py geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/demo.py g2	geopandas	spatial join, crosses	<pre>i_crosses_z = inter.sjoin(zips,how='right',predicate='c</pre>	g25/demo.py
geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate='tou g25/demo.py geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate='wi g25/demo.py geopandas testing if rows touch a geometry touches_ww_geo) g22/demo.py geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/demo.py json importing the module import json g05/demo.py json using to print an object nicely print(json.dumps(list1,indent=4)) g05/demo.py matplotlib axes, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/demo.py matplotlib axes, adding a vertical line ax21.axhline(medians['inc'], c='r', ls='-', lw=1) g12/demo.py matplotlib axes, labeling the X axis ax1.set_xlabel("Millions') g11/demo.py matplotlib axes, setting a title ax1.set_title("Population, Millions") g11/demo.py matplotlib axes, setting a title ax.set_ylabel("Population,') g11/demo.py matplotlib colors, xkcd palette syr.plot(color='xkcd:lightblue',ax=ax1) g21/demo.py matplotlib figure, dour panel grid fig2.suptitle("Pooled Data') figure, four panel grid fig3, axs = plt.subplots(2,2.sharex=True,sharey=True) g12/demo.py matplotlib figure, setting the size fig. (ax21,ax22) = plt.subplots(1,2, figsize=(12,6)) g20/demo.py matplotlib figure, setting the layout fig1.tight_layout() g11/demo.py matplotlib figure, working with a list of axes for ax in axs: import fig1, ax1 = plt.subplots() g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py g11/demo.py	geopandas	spatial join, intersects	<pre>z_intersect_c = zips.sjoin(county,how='left',predicate=</pre>	g25/demo.py
geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate='tou g25/demo.py geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate='wi g25/demo.py geopandas testing if rows touch a geometry touches_ww_geo) g22/demo.py geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/demo.py json importing the module import json g05/demo.py json using to print an object nicely print(json.dumps(list1,indent=4)) g05/demo.py matplotlib axes, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/demo.py matplotlib axes, adding a vertical line ax21.axhline(medians['inc'], c='r', ls='-', lw=1) g12/demo.py matplotlib axes, labeling the X axis ax1.set_xlabel("Millions') g11/demo.py matplotlib axes, setting a title ax1.set_title("Population, Millions") g11/demo.py matplotlib axes, setting a title ax.set_ylabel("Population,') g11/demo.py matplotlib colors, xkcd palette syr.plot(color='xkcd:lightblue',ax=ax1) g21/demo.py matplotlib figure, dour panel grid fig2.suptitle("Pooled Data') figure, four panel grid fig3, axs = plt.subplots(2,2.sharex=True,sharey=True) g12/demo.py matplotlib figure, setting the size fig. (ax21,ax22) = plt.subplots(1,2, figsize=(12,6)) g20/demo.py matplotlib figure, setting the layout fig1.tight_layout() g11/demo.py matplotlib figure, working with a list of axes for ax in axs: import fig1, ax1 = plt.subplots() g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py g11/demo.py	geopandas	spatial join, overlaps		g25/demo.py
geopandas geopandas testing if rows touch a geometry geopandas writing a layer to a geodatabase conus.touches(wv_geo) g22/demo.py geopandas writing a layer to a geodatabase conus.to_mile("conus.gpkg",layer="states") g22/demo.py g22/de	_			g25/demo.py
geopandas geopandas vriting a layer to a geodatabase conus.touches(wv_geo) g22/demo.py g22/demo.py g22/demo.py writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/demo.py g22/demo.py g22/demo.py g05/demo.py g11/demo.py g12/demo.py g11/demo.py	geopandas	spatial join, within		g25/demo.py
geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/demo.py json importing the module import json g05/demo.py matplotlib axes, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) matplotlib axes, adding a vertical line ax21.axvline(medians['inc'], c='r', ls='-', lw=1) matplotlib axes, labeling the X axis ax1.axetvlabel("Hollions") matplotlib axes, labeling the Y axis ax1.setvlabel("Population, Millions") matplotlib axes, setting a title ax1.setvlabel("Population, Millions") matplotlib axes, setting a title ax1.setvlabel(None) matplotlib axes, urning off the label ax.setvlabel(None) matplotlib figure, adding a title fig2.suptitle("Pooled Data') matplotlib figure, four panel grid fig3, axs = plt.subplots(2,2,sharex=True,sharey=True) matplotlib figure, setting the size fig3, axs = plt.subplots(1,2) matplotlib figure, setting the size fig3.axs = plt.subplots(1,2,figsize=(12,6)) matplotlib figure, setting the size fig3.tightlayout() matplotlib figure, working with a list of axes matplotlib figure, working with a list of axes matplotlib setting the default resolution matplotlib setting the default resolution plt.rcParams[figure.dpi'] = 300 g11/demo.py matplotlib using subplots to set up a figure fig1, ax1 = plt.subplots() fig1.subplots() fig1.subplots() fig1.demo.py matplotlib setting the default resolution plt.rcParams[figure.dpi'] = 300 g24/demo.py matplotlib using subplots to set up a figure fig1.demo.py matplotlib() fig1.demo.py matplotlib setting the default resolution plt.rcParams[figure.dpi'] = 300 g24/demo.py matplotlib using subplots to set up a figure fig1.ax1 = plt.subplots()				
matplotlib axes, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/demo.py matplotlib axes, adding a vertical line ax21.axvline(medians['inc'], c='r', ls='-', lw=1) g12/demo.py matplotlib axes, labeling the X axis ax1.set_xlabel('Millions') g11/demo.py matplotlib axes, labeling the Y axis ax1.set_ylabel('Population, Millions'') g11/demo.py matplotlib axes, setting a title ax1.set_title('Population') axes, turning off the label ax.set_ylabel(None) g13/demo.py matplotlib colors, xkcd palette syr.plot(color='xkcd:lightblue',ax=ax1) g21/demo.py matplotlib figure, adding a title fig2.suptitle('Pooled Data') g12/demo.py matplotlib figure, four panel grid fig3, axs = plt.subplots(2,2,sharex=True,sharey=True) g12/demo.py matplotlib figure, left and right panels fig2, (ax21,ax22) = plt.subplots(1,2) g12/demo.py matplotlib figure, sexting the size fig, axs = plt.subplots(1,2,figsize=(12,6)) g20/demo.py matplotlib figure, tuning the layout fig1.tight_layout() g11/demo.py matplotlib figure, working with a list of axes for ax in axs: g20/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib using subplots to set up a figure fig1, ax1 = plt.subplots()	geopandas	writing a layer to a geodatabase	conus.to_file("conus.gpkg",layer="states")	
matplotlib axes, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/demo.py matplotlib axes, labeling the X axis ax1.set_xlabel('Millions') g11/demo.py matplotlib axes, labeling the Y axis ax1.set_ylabel('Population, Millions') g11/demo.py matplotlib axes, setting a title ax1.set_title('Population, Millions') g11/demo.py matplotlib axes, setting a title ax1.set_title('Population') g11/demo.py matplotlib axes, turning off the label ax.set_ylabel(None) g13/demo.py matplotlib figure, adding a title fig2.suptitle('Pooled Data') g12/demo.py matplotlib figure, four panel grid fig3, axs = plt.subplots(2,2,sharex=True,sharey=True) g12/demo.py matplotlib figure, left and right panels fig2, (ax21,ax22) = plt.subplots(1,2) g12/demo.py matplotlib figure, setting the size fig, axs = plt.subplots(1,2,figsize=(12,6)) g20/demo.py matplotlib figure, tuning the layout fig1.tight_layout() g11/demo.py matplotlib figure, working with a list of axes for ax in axs: g20/demo.py matplotlib importing pyplot import matplotlib.pyplot as plt g11/demo.py matplotlib setting the default resolution plt.rcParams[figure.dpi'] = 300 g11/demo.py matplotlib using subplots to set up a figure fig1, ax1 = plt.subplots()	json	importing the module	import json	g05/demo.py
matplotlib axes, adding a vertical line ax21.axvline(medians['inc'], c='r', ls='-', lw=1) g12/demo.py matplotlib axes, labeling the X axis ax1.set_xlabel('Millions') g11/demo.py matplotlib axes, labeling the Y axis ax1.set_ylabel('Population, Millions'') g11/demo.py matplotlib axes, setting a title ax1.set_title('Population') g11/demo.py matplotlib axes, turning off the label ax.set_ylabel(None) g13/demo.py matplotlib colors, xkcd palette syr.plot(color='xkcd:lightblue',ax=ax1) g21/demo.py matplotlib figure, adding a title fig2.suptitle('Pooled Data') g12/demo.py matplotlib figure, four panel grid fig3, axs = plt.subplots(2,2,sharex=True,sharey=True) g12/demo.py matplotlib figure, setting the panels fig2. (ax21,ax22) = plt.subplots(1,2) g12/demo.py matplotlib figure, setting the size fig. axs = plt.subplots(1,2,figsize=(12,6)) g20/demo.py matplotlib figure, tuning the layout fig1.tight_layout() g11/demo.py matplotlib figure, working with a list of axes for ax in axs: g20/demo.py matplotlib importing pyplot import matplotlib, pyplot as plt g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib using subplots to set up a figure fig1, ax1 = plt.subplots() g24/demo.py	json	using to print an object nicely	<pre>print(json.dumps(list1,indent=4))</pre>	g05/demo.py
matplotlib axes, labeling the X axis ax1.set_xlabel('Millions') g11/demo.py matplotlib axes, labeling the Y axis ax1.set_ylabel("Population, Millions") g11/demo.py matplotlib axes, setting a title ax1.set_title('Population') g11/demo.py matplotlib axes, turning off the label ax.set_ylabel(None) g13/demo.py matplotlib colors, xkcd palette syr.plot(color='xkcd:lightblue',ax=ax1) g21/demo.py matplotlib figure, adding a title fig2.suptitle('Population') g12/demo.py matplotlib figure, four panel grid fig3, axs = plt.subplots(2,2,sharex=True,sharey=True) g12/demo.py matplotlib figure, left and right panels fig2, (ax21,ax22) = plt.subplots(1,2) g11/demo.py matplotlib figure, saving fig1.savefig('figure.png') g11/demo.py matplotlib figure, setting the size fig, axs = plt.subplots(1,2,figsize=(12,6)) g20/demo.py matplotlib figure, tuning the layout fig1.tight_layout() g11/demo.py matplotlib figure, working with a list of axes for ax in axs: g20/demo.py matplotlib importing pyplot import matplotlib.pyplot as plt g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib using subplots to set up a figure os.remove(out_file) g24/demo.py	matplotlib	axes, adding a horizontal line	ax21.axhline(medians['etr'], c='r', ls='-', lw=1)	g12/demo.py
matplotlib axes, labeling the Y axis ax1.set_ylabel("Population, Millions") g11/demo.py matplotlib axes, setting a title ax1.set_title('Population') g11/demo.py matplotlib axes, turning off the label ax.set_ylabel(None) g13/demo.py matplotlib colors, xkcd palette syr.plot(color='xkcd:lightblue',ax=ax1) g21/demo.py matplotlib figure, adding a title fig2.suptitle('Pooled Data') g12/demo.py matplotlib figure, four panel grid fig3, axs = plt.subplots(2,2,sharex=True,sharey=True) g12/demo.py matplotlib figure, left and right panels fig2, (ax21,ax22) = plt.subplots(1,2) g12/demo.py matplotlib figure, saving fig1.savefig('figure.png') g11/demo.py matplotlib figure, setting the size fig, axs = plt.subplots(1,2,figsize=(12,6)) g20/demo.py matplotlib figure, uning the layout fig1.tight_layout() g11/demo.py matplotlib figure, working with a list of axes for ax in axs: g20/demo.py matplotlib importing pyplot import matplotlib.pyplot as plt g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib using subplots to set up a figure fig1, ax1 = plt.subplots() g24/demo.py	matplotlib	axes, adding a vertical line	ax21.axvline(medians['inc'], c='r', ls='-', lw=1)	g12/demo.py
matplotlib axes, setting a title ax1.set_title('Population') g11/demo.py matplotlib axes, turning off the label ax.set_ylabel(None) g13/demo.py matplotlib colors, xkcd palette syr.plot(color='xkcd:lightblue',ax=ax1) g21/demo.py matplotlib figure, adding a title fig2.suptitle('Pooled Data') g12/demo.py matplotlib figure, four panel grid fig3, axs = plt.subplots(2,2,sharex=True,sharey=True) g12/demo.py matplotlib figure, left and right panels fig2. (ax21,ax22) = plt.subplots(1,2) g12/demo.py matplotlib figure, saving fig1.saves[g'(figure.png') g11/demo.py matplotlib figure, setting the size fig, axs = plt.subplots(1,2,figsize=(12,6)) g20/demo.py matplotlib figure, tuning the layout fig1.tight_layout() g11/demo.py matplotlib figure, working with a list of axes for ax in axs: g20/demo.py matplotlib importing pyplot import matplotlib.pyplot as plt matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib using subplots to set up a figure os.remove(out_file) g24/demo.py	matplotlib	axes, labeling the X axis	ax1.set_xlabel('Millions')	g11/demo.py
matplotlibaxes, turning off the labelax.set_ylabel(None)g13/demo.pymatplotlibcolors, xkcd palettesyr.plot(color='xkcd:lightblue',ax=ax1)g21/demo.pymatplotlibfigure, adding a titlefig2.suptitle('Pooled Data')g12/demo.pymatplotlibfigure, four panel gridfig3, axs = plt.subplots(2,2,sharex=True,sharey=True)g12/demo.pymatplotlibfigure, left and right panelsfig2, (ax21,ax22) = plt.subplots(1,2)g12/demo.pymatplotlibfigure, savingfig1.savefig('figure.png')g11/demo.pymatplotlibfigure, setting the sizefig, axs = plt.subplots(1,2,figsize=(12,6))g20/demo.pymatplotlibfigure, tuning the layoutfig1.tight_layout()g11/demo.pymatplotlibfigure, working with a list of axesfor ax in axs:g20/demo.pymatplotlibimporting pyplotimport matplotlib.pyplot as pltg11/demo.pymatplotlibsetting the default resolutionplt.rcParams['figure.dpi'] = 300g11/demo.pymatplotlibusing subplots to set up a figurefig1, ax1 = plt.subplots()g11/demo.pyosdelete a fileos.remove(out_file)g24/demo.py	matplotlib	axes, labeling the Y axis	ax1.set_ylabel("Population, Millions")	g11/demo.py
matplotlibaxes, turning off the labelax.set_ylabel(None)g13/demo.pymatplotlibcolors, xkcd palettesyr.plot(color='xkcd:lightblue',ax=ax1)g21/demo.pymatplotlibfigure, adding a titlefig2.suptitle('Pooled Data')g12/demo.pymatplotlibfigure, four panel gridfig3, axs = plt.subplots(2,2,sharex=True,sharey=True)g12/demo.pymatplotlibfigure, left and right panelsfig2, (ax21,ax22) = plt.subplots(1,2)g12/demo.pymatplotlibfigure, savingfig1.savefig('figure.png')g11/demo.pymatplotlibfigure, setting the sizefig, axs = plt.subplots(1,2,figsize=(12,6))g20/demo.pymatplotlibfigure, tuning the layoutfig1.tight_layout()g11/demo.pymatplotlibfigure, working with a list of axesfor ax in axs:g20/demo.pymatplotlibimporting pyplotimport matplotlib.pyplot as pltg11/demo.pymatplotlibsetting the default resolutionplt.rcParams['figure.dpi'] = 300g11/demo.pymatplotlibusing subplots to set up a figurefig1, ax1 = plt.subplots()g11/demo.pyosdelete a fileos.remove(out_file)g24/demo.py	matplotlib	axes, setting a title	ax1.set_title('Population')	g11/demo.py
matplotlib figure, adding a title fig2.suptitle('Pooled Data') g12/demo.py matplotlib figure, four panel grid fig3, axs = plt.subplots(2,2,sharex=True,sharey=True) g12/demo.py matplotlib figure, left and right panels fig2, (ax21,ax22) = plt.subplots(1,2) g12/demo.py matplotlib figure, saving fig1.savefig('figure.png') g11/demo.py matplotlib figure, setting the size fig. axs = plt.subplots(1,2,figsize=(12,6)) g20/demo.py matplotlib figure, tuning the layout fig1.tight_layout() g11/demo.py matplotlib figure, working with a list of axes for ax in axs: g20/demo.py matplotlib importing pyplot import matplotlib.pyplot as plt g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib using subplots to set up a figure fig1, ax1 = plt.subplots() g24/demo.py	matplotlib	axes, turning off the label		g13/demo.py
matplotlib figure, four panel grid fig3, axs = plt.subplots(2,2,sharex=True,sharey=True) matplotlib figure, left and right panels fig2, (ax21,ax22) = plt.subplots(1,2) matplotlib figure, saving fig1.savefig('figure.png') matplotlib figure, setting the size fig., axs = plt.subplots(1,2,figsize=(12,6)) matplotlib figure, tuning the layout fig1.tight_layout() matplotlib figure, working with a list of axes for ax in axs: matplotlib importing pyplot import matplotlib.pyplot as plt matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 matplotlib using subplots to set up a figure fig1, ax1 = plt.subplots() os.remove(out_file) g12/demo.py g11/demo.py g20/demo.py g11/demo.py g11/demo.py g11/demo.py g11/demo.py g11/demo.py g11/demo.py g11/demo.py g24/demo.py	matplotlib	colors, xkcd palette	syr.plot(color='xkcd:lightblue',ax=ax1)	g21/demo.py
matplotlib figure, left and right panels fig2, (ax21,ax22) = plt.subplots(1,2) g12/demo.py matplotlib figure, saving fig1.savefig('figure.png') g11/demo.py matplotlib figure, setting the size fig, axs = plt.subplots(1,2,figsize=(12,6)) g20/demo.py matplotlib figure, tuning the layout fig1.tight_layout() g11/demo.py matplotlib figure, working with a list of axes for ax in axs: g20/demo.py matplotlib importing pyplot import matplotlib.pyplot as plt g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib using subplots to set up a figure fig1, ax1 = plt.subplots() g24/demo.py os.remove(out_file)	matplotlib	figure, adding a title	fig2.suptitle('Pooled Data')	g12/demo.py
matplotlib figure, saving fig1.savefig('figure.png') g11/demo.py matplotlib figure, setting the size fig, axs = plt.subplots(1,2,figsize=(12,6)) g20/demo.py matplotlib figure, tuning the layout fig1.tight_layout() g11/demo.py matplotlib figure, working with a list of axes for ax in axs: g20/demo.py matplotlib importing pyplot import matplotlib.pyplot as plt g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib using subplots to set up a figure fig1, ax1 = plt.subplots() g24/demo.py g24/demo.py	matplotlib	figure, four panel grid	fig3, $axs = plt.subplots(2,2,sharex=True,sharey=True)$	g12/demo.py
matplotlib figure, saving fig1.savefig('figure.png') g11/demo.py matplotlib figure, setting the size fig, axs = plt.subplots(1,2,figsize=(12,6)) g20/demo.py matplotlib figure, tuning the layout fig1.tight_layout() g11/demo.py matplotlib figure, working with a list of axes for ax in axs: g20/demo.py matplotlib importing pyplot import matplotlib.pyplot as plt g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib using subplots to set up a figure fig1, ax1 = plt.subplots() g24/demo.py g24/demo.py	matplotlib	figure, left and right panels	fig2, $(a\times21,a\times22) = plt.subplots(1,2)$	g12/demo.py
$\begin{array}{llllllllllllllllllllllllllllllllllll$	matplotlib		fig1.savefig('figure.png')	g11/demo.py
$\begin{array}{llllllllllllllllllllllllllllllllllll$	matplotlib	figure, setting the size	fig, $axs = plt.subplots(1,2,figsize=(12,6))$	g20/demo.py
matplotlib importing pyplot import matplotlib.pyplot as plt g11/demo.py matplotlib setting the default resolution plt.rcParams['figure.dpi'] = 300 g11/demo.py matplotlib using subplots to set up a figure fig1, ax1 = plt.subplots() g11/demo.py os delete a file os.remove(out_file) g24/demo.py	matplotlib	figure, tuning the layout	fig1.tight_layout()	g11/demo.py
$\begin{array}{llllllllllllllllllllllllllllllllllll$	matplotlib	figure, working with a list of axes	for ax in axs:	g20/demo.py
matplotlib using subplots to set up a figure fig1, ax1 = plt.subplots() g11/demo.py os delete a file os.remove(out_file) g24/demo.py	matplotlib	importing pyplot	import matplotlib.pyplot as plt	
matplotlib using subplots to set up a figure fig1, ax1 = plt.subplots() g11/demo.py os delete a file os.remove(out_file) g24/demo.py	matplotlib	setting the default resolution	plt.rcParams['figure.dpi'] = 300	g11/demo.py
$\langle - \rangle$	matplotlib	using subplots to set up a figure		
os importing the module import os g24/demo.py	os	delete a file	os.remove(out_file)	g24/demo.py
	os	importing the module	import os	g24/demo.py

pandas RE, replacing a digit or space unit_part = values.str.replace(r'\d \s',",regex=True) value_pandas RE, replacing a non-digit or space value_part = values.str.replace(r'\d \s',",regex=True) value_pandas RE, replacing a non-word character units = units.str.replace(r'\d \s',",regex=True) value_part = values.str.replace(r'\d \s',",regex=True} values.str.replace(r'\d \s',",regex=True} values.str.replace(r'\d \s',",regex=True} values.str.replace(r'\d \s',",regex=True} values.str.replace(r'\d \s',",regex=True} values.str.replace(r'\d \s',"regex=True} value	g24/demo.py g23/demo.py g23/demo.py g23/demo.py g23/demo.py g09/demo.py g09/demo.py g10/demo.py g09/demo.py g09/demo.py g09/demo.py
RE, replacing a non-digit or space pandas RE, replacing a non-word character RE, replacing a non-word character pandas columns, dividing with explicit alignment pandas columns, listing names columns, listing names pandas columns, renaming columns, retrieving one by name pandas columns, retrieving several by name pandas dataframe, appending pandas dataframe, dropping a column pandas dataframe, dropping duplicates pandas dataframe, dropping missing data pandas dataframe, getting a block of rows via index pandas dataframe, linner join pandas dataframe, left n:1 merge pandas dataframe, making a copy pandas dataframe, outer 1:1 merge pandas dataframe, reading zipped pickle format value_part = values.str.replace(r'\D \s',",regex=True) value_part = values.tr.replace(r'\D \s's',",regex=True) value_part = values.tr.replace(r'\D \s's',",regex=True) value_part = values.tr.replace(r'\D \s's',",regex=True) value_part = values.tr.replace(r'\D \s's'),"aregex=True) value_part = values.tr.replace(r) value_part = values.tr.replace(r) value_part = values.tr.replace(r) value_pare = values.tr.ealums: value = values.tr.ealums: value = values.tr.ealums: values.tr.ealums: value = va	g23/demo.py g23/demo.py g09/demo.py g09/demo.py g10/demo.py g09/demo.py g09/demo.py
RE, replacing a non-word character pandas columns, dividing with explicit alignment pandas columns, listing names pandas columns, renaming pandas columns, retrieving one by name pandas columns, retrieving several by name pandas dataframe, appending pandas dataframe, dropping a column pandas dataframe, dropping missing data pandas dataframe, getting a block of rows via index pandas dataframe, inner 1:1 merge pandas dataframe, left 1:1 merge pandas dataframe, making a copy pandas dataframe, making a copy pandas dataframe, making a copy pandas dataframe, making a copy pandas dataframe, making a copy pandas dataframe, reading zipped pickle format normed2 = 100*states.div(pa_row,axis='columns') print('\nColumns:', list(raw_states.columns) print('\nColumns:', list(raw_states.columns) print('\nColumns:', list(raw_states.columns) pop = states['pop'] pop = states['pop'] print(pop[some_states]/1e6) gen_all = pd.concat([gen_oswego, gen_onondaga]) print(trim[has_AM], "\n") both = both.drop(columns='_merge') flood = flood.drop_duplicates(subset='TAX_ID') trim = demo.dropna(subset="Days") dups = parcels.duplicated(subset='TAX_ID' , keep=False sel = merged.loc[number] join_i = parcels.merge(flood, merged = name_data.merge(pop_data,left_on="State",right pandas dataframe, left n:1 merge poth = parcels.merge(flood, pandas dataframe, making a copy pandas dataframe, reading zipped pickle format sample2 = pd.read_pickle('sample_pkl.zip')	g23/demo.py g09/demo.py g09/demo.py g10/demo.py g09/demo.py g09/demo.py
pandas columns, dividing with explicit alignment columns, listing names columns, listing names pandas columns, retrieving one by name pop = states['pop'] print('\nColumns:', list(raw_states.columns)) county = county.rename(columns={'B01001_001E':'pop'}) pandas columns, retrieving one by name pop = states['pop'] print(pop[some_states]/1e6) pandas dataframe, appending gen_all = pd.concat([gen_oswego, gen_onondaga]) print(trim[has_AM], "\n") pandas dataframe, dropping a column both = both.drop(columns='_merge') pandas dataframe, dropping missing data trim = demo.dropna(subset="TAX_ID') trim = demo.dropna(subset="TAX_ID') trim = demo.dropna(subset="TAX_ID') keep=False sel = merged.loc[number] join_i = parcels.merge(flood, merged = name_data.merge(pop_data,left_on="State",right	g09/demo.py g09/demo.py g10/demo.py g09/demo.py g09/demo.py
pandas columns, listing names print('\nColumns:', list(raw_states.columns)) pandas columns, renaming county = county.rename(columns={'B01001_001E':'pop'}) pandas columns, retrieving one by name pop = states['pop'] pandas columns, retrieving several by name print(pop[some_states]/1e6) pandas dataframe, appending gen_all = pd.concat([gen_oswego, gen_onondaga]) pandas dataframe, dropping a column both = both.drop(columns='_merge') pandas dataframe, dropping duplicates flood = flood.drop_duplicates(subset='TAX_ID') pandas dataframe, finding duplicate records dataframe, getting a block of rows via index pandas dataframe, inner join dataframe, left 1:1 merge join_i = parcels.merge(flood, pandas dataframe, left m:1 merge both = gen_all.merge(plants, pandas dataframe, making a copy trim = trim.copy() pandas dataframe, reading zipped pickle format sample2 = pd.read_pickle('sample_pkl.zip')	g09/demo.py g10/demo.py g09/demo.py g09/demo.py
columns, renaming county = county.rename(columns={'B01001_001E':'pop'}) pandas columns, retrieving one by name pop = states['pop'] pandas columns, retrieving several by name pop = states['pop'] pandas dataframe, appending gen_all = pd.concat([gen_oswego, gen_onondaga]) pandas dataframe, dropping a column both = both.drop(columns='_merge') pandas dataframe, dropping missing data trim = demo.dropna(subset="TAX_ID') pandas dataframe, getting a block of rows via index pandas dataframe, inner 1:1 merge join_i = parcels.merge(flood, merged = name_data.merge(pop_data,left_on="State",right pandas dataframe, making a copy trim = trim.copy() pandas dataframe, reading zipped pickle format county = county.rename(columns={'B01001_001E':'pop'}) pop = states['pop'] pop = s	g10/demo.py g09/demo.py g09/demo.py
pandas columns, retrieving one by name pandas columns, retrieving several by name print(pop[some_states]/1e6) pandas dataframe, appending gen_all = pd.concat([gen_oswego, gen_onondaga]) pandas dataframe, boolean row selection print(trim[has_AM], "\n") pandas dataframe, dropping a column both = both.drop(columns='_merge') pandas dataframe, dropping missing data trim = demo.dropna(subset='TAX_ID') pandas dataframe, finding duplicate records dups = parcels.duplicated(subset='TAX_ID', keep=False pandas dataframe, inner 1:1 merge join_i = parcels.merge(flood, pandas dataframe, left 1:1 merge join_l = parcels.merge(flood, pandas dataframe, making a copy trim = trim.copy() pandas dataframe, reading zipped pickle format sample2 = pd.read_pickle('sample_pkl.zip')	g09/demo.py g09/demo.py
pandas columns, retrieving several by name print(pop[some_states]/1e6) pandas dataframe, appending gen_all = pd.concat([gen_oswego, gen_onondaga]) pandas dataframe, boolean row selection print(trim[has_AM], "\n") pandas dataframe, dropping a column both = both.drop(columns='_merge') pandas dataframe, dropping missing data trim = demo.dropna(subset="TAX_ID') pandas dataframe, finding duplicate records dups = parcels.duplicated(subset='TAX_ID' , keep=False pandas dataframe, inner 1:1 merge join_i = parcels.merge(flood, merged = name_data.merge(pop_data,left_on="State",right pandas dataframe, left 1:1 merge join_l = parcels.merge(flood, both = gen_all.merge(plants, trim = trim.copy()) pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, sample2 = pd.read_pickle('sample_pkl.zip'))	g09/demo.py
pandas dataframe, appending gen_all = pd.concat([gen_oswego, gen_onondaga]) pandas dataframe, boolean row selection print(trim[has_AM], "\n") pandas dataframe, dropping a column both = both.drop(columns='_merge') pandas dataframe, dropping duplicates flood = flood.drop_duplicates(subset='TAX_ID') pandas dataframe, finding duplicate records dups = parcels.duplicated(subset='TAX_ID', keep=False pandas dataframe, inner 1:1 merge join_i = parcels.merge(flood, pandas dataframe, left 1:1 merge join_l = parcels.merge(flood, pandas dataframe, left m:1 merge both = gen_all.merge(plants, pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, pandas dataframe, reading zipped pickle format sample2 = pd.read_pickle('sample_pkl.zip')	
pandas dataframe, boolean row selection print(trim[has_AM], "\n") pandas dataframe, dropping a column both = both.drop(columns='_merge') pandas dataframe, dropping duplicates flood = flood.drop_duplicates(subset='TAX_ID') pandas dataframe, dropping missing data trim = demo.dropna(subset="Days") pandas dataframe, finding duplicate records dups = parcels.duplicated(subset='TAX_ID', keep=False pandas dataframe, inner 1:1 merge join_i = parcels.merge(flood, pandas dataframe, inner join merged = name_data.merge(pop_data,left_on="State",right pandas dataframe, left 1:1 merge join_I = parcels.merge(flood, pandas dataframe, left m:1 merge both = gen_all.merge(plants, pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, pandas dataframe, reading zipped pickle format sample2 = pd.read_pickle('sample_pkl.zip')	g15/demo.pv
pandas dataframe, dropping a column both = both.drop(columns='_merge') pandas dataframe, dropping duplicates flood = flood.drop_duplicates(subset='TAX_ID') pandas dataframe, dropping missing data trim = demo.dropna(subset="Days") pandas dataframe, finding duplicate records dups = parcels.duplicated(subset='TAX_ID', keep=False sel = merged.loc[number] pandas dataframe, inner 1:1 merge join_i = parcels.merge(flood, pandas dataframe, left 1:1 merge join_l = parcels.merge(pop_data,left_on="State",right pandas dataframe, left m:1 merge both = gen_all.merge(plants, pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, pandas dataframe, reading zipped pickle format sample2 = pd.read_pickle('sample_pkl.zip')	
dataframe, dropping duplicates pandas pandas dataframe, dropping missing data pandas pandas dataframe, finding duplicate records pandas dataframe, finding duplicate records pandas pandas dataframe, getting a block of rows via index pandas pandas dataframe, inner 1:1 merge pandas dataframe, inner join pandas dataframe, left 1:1 merge pandas pandas dataframe, left m:1 merge pandas dataframe, left m:1 merge pandas dataframe, making a copy pandas dataframe, outer 1:1 merge pandas dataframe, reading zipped pickle format flood = flood.drop_duplicates(subset='TAX_ID') trim = demo.dropna(subset="Days") trim = demo.dropna(subset="Days") sel = merged.loc[number] join_i = parcels.merge(flood, merged = name_data.merge(pop_data,left_on="State",right. join_l = parcels.merge(flood, both = gen_all.merge(plants, trim = trim.copy() join_o = parcels.merge(flood, sample2 = pd.read_pickle('sample_pkl.zip')	g12/demo.py
pandas dataframe, dropping missing data trim = demo.dropna(subset="Days") dataframe, finding duplicate records dups = parcels.duplicated(subset='TAX_ID', keep=False pandas dataframe, getting a block of rows via index sel = merged.loc[number] pandas dataframe, inner 1:1 merge join_i = parcels.merge(flood, pandas dataframe, left 1:1 merge join_l = parcels.merge(pop_data,left_on="State",right pandas dataframe, left m:1 merge join_l = parcels.merge(flood, both = gen_all.merge(plants, trim = trim.copy() pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, sample2 = pd.read_pickle('sample_pkl.zip')	g15/demo.py
dataframe, finding duplicate records dups = parcels.duplicated(subset='TAX_ID', keep=False sel = merged.loc[number] pandas dataframe, inner 1:1 merge join_i = parcels.merge(flood, pandas dataframe, inner join merged = name_data.merge(pop_data,left_on="State",right pandas dataframe, left 1:1 merge join_l = parcels.merge(flood, pandas dataframe, left m:1 merge join_l = parcels.merge(flood, both = gen_all.merge(plants, trim = trim.copy() pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, sample2 = pd.read_pickle('sample_pkl.zip')	g14/demo.py
pandas dataframe, getting a block of rows via index pandas dataframe, inner 1:1 merge join_i = parcels.merge(flood, merged = name_data.merge(pop_data,left_on="State",right. pandas dataframe, left 1:1 merge join_l = parcels.merge(flood, both = gen_all.merge(plants, trim = trim.copy()) pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, sample2 = pd.read_pickle('sample_pkl.zip'))	g12/demo.py
pandas dataframe, inner 1:1 merge join_i = parcels.merge(flood, pandas dataframe, inner join merged = name_data.merge(pop_data,left_on="State",right pandas dataframe, left 1:1 merge join_l = parcels.merge(flood, pandas dataframe, left m:1 merge both = gen_all.merge(plants, pandas dataframe, making a copy trim = trim.copy() pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, pandas dataframe, reading zipped pickle format sample2 = pd.read_pickle('sample_pkl.zip')	g14/demo.py
pandas dataframe, inner join merged = name_data.merge(pop_data,left_on="State",right pandas dataframe, left 1:1 merge join_l = parcels.merge(flood, pandas dataframe, left m:1 merge both = gen_all.merge(plants, pandas dataframe, making a copy trim = trim.copy() pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, pandas dataframe, reading zipped pickle format sample2 = pd.read_pickle('sample_pkl.zip')	g13/demo.py
pandas dataframe, left 1:1 merge join_l = parcels.merge(flood, pandas dataframe, left m:1 merge both = gen_all.merge(plants, pandas dataframe, making a copy trim = trim.copy() pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, pandas dataframe, reading zipped pickle format sample2 = pd.read_pickle('sample_pkl.zip')	g14/demo.py
pandas dataframe, left m:1 merge both = gen_all.merge(plants, pandas dataframe, making a copy trim = trim.copy() pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, pandas dataframe, reading zipped pickle format sample2 = pd.read_pickle('sample_pkl.zip')	. g13/demo.py
pandas dataframe, making a copy trim = trim.copy() pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, pandas dataframe, reading zipped pickle format sample2 = pd.read_pickle('sample_pkl.zip')	g14/demo.py
pandas dataframe, outer 1:1 merge join_o = parcels.merge(flood, pandas dataframe, reading zipped pickle format sample2 = pd.read_pickle('sample_pkl.zip')	g15/demo.py
pandas dataframe, reading zipped pickle format sample2 = pd.read_pickle('sample_pkl.zip')	g12/demo.py
	g14/demo.py
Later Communication of the Control o	g16/demo.py
pandas dataframe, resetting the index hourly = hourly.reset_index()	g17/demo.py
pandas dataframe, right 1:1 merge join_r = parcels.merge(flood,	g14/demo.py
pandas dataframe, saving in zipped pickle format sample.to_pickle('sample_pkl.zip')	g16/demo.py
pandas dataframe, selecting rows by list indexing print(low_to_high[-5:])	g09/demo.py
pandas dataframe, selecting rows via boolean dup_rec = flood[dups]	g14/demo.py
pandas dataframe, selecting rows via query trimmed = county.query("state == '04' or state == '36' ")	g10/demo.py
pandas dataframe, set index keeping the column states = states.set_index('STUSPS',drop=False)	g22/demo.py
pandas dataframe, shape attribute print('number of rows, columns:', conus.shape)	g22/demo.py
pandas dataframe, sorting by a column county = county.sort_values('pop')	g10/demo.py
pandas dataframe, sorting by index summary = summary.sort_index(ascending=False)	g15/demo.py
pandas dataframe, summing a boolean print('\nduplicate parcels:', dups.sum())	g14/demo.py

Module	Description	Example	Script
pandas	dataframe, unstacking an index level	bymo = bymo.unstack('month')	g17/demo.py
pandas	dataframe, using a multilevel column index	means = grid['mean']	g20/demo.py
pandas	dataframe, using xs to select a subset	print(county.xs('04',level='state'))	g10/demo.py
pandas	dataframe, using xs with columns	c1 = grid.xs('c1',axis='columns',level=1)	g20/demo.py
pandas	dataframe, writing to a CSV file	merged.to_csv('demo-merged.csv')	g13/demo.py
pandas	datetime, building via to_datetime()	$date = pd.to_datetime(recs[`ts'])$	g14/demo.py
pandas	datetime, building with a format	$ymd = pd.to_datetime(\ sample[\text{`TRANSACTION_DT'}],\ format{=}.\ \ .$	g16/demo.py
pandas	datetime, extracting day attribute	recs[`day'] = date.dt.day	g14/demo.py
pandas	datetime, extracting hour attribute	recs['hour'] = date.dt.hour	g14/demo.py
pandas	general, display information about object	sample.info()	g16/demo.py
pandas	general, displaying all columns	pd.set_option('display.max_columns',None)	${\sf g16/demo.py}$
pandas	general, displaying all rows	pd.set_option('display.max_rows', None)	${\sf g09/demo.py}$
pandas	general, importing the module	import pandas as pd	g09/demo.py
pandas	general, using qcut to create deciles	$dec = pd.qcut(\ county['pop'],\ 10,\ labels = range(1,11)\)$	g10/demo.py
pandas	groupby, cumulative sum within group	${\sf cumulative_inc} = {\sf group_by_state['pop'].cumsum()}$	g10/demo.py
pandas	groupby, descriptive statistics	<pre>inc_stats = group_by_state['pop'].describe()</pre>	g10/demo.py
pandas	groupby, iterating over groups	for t,g in group_by_state:	${\sf g10/demo.py}$
pandas	groupby, median of each group	<pre>pop_med = group_by_state['pop'].median()</pre>	${\sf g10/demo.py}$
pandas	groupby, quantile of each group	$pop_25th = group_by_state['pop'].quantile(0.25)$	g10/demo.py
pandas	groupby, return group number	$groups = group_by_state.ngroup()$	${\sf g10/demo.py}$
pandas	groupby, return number within group	seqnum = group_by_state.cumcount()	g10/demo.py
pandas	groupby, return rank within group	rank_age = group_by_state['pop'].rank()	g10/demo.py
pandas	groupby, select first records	$first2 = group_by_state.head(2)$	g10/demo.py
pandas	groupby, select largest values	$largest = group_by_state['pop'].nlargest(2)$	g10/demo.py
pandas	groupby, select last records	$last2 = group_by_state.tail(2)$	g10/demo.py
pandas	groupby, size of each group	num_rows = group_by_state.size()	g10/demo.py
pandas	groupby, sum of each group	state = county.groupby(`state')[`pop'].sum()	g10/demo.py
pandas	index, creating with 3 levels	$county = county.set_index(['state', 'county', 'NAME'])$	g10/demo.py
pandas	index, listing names	<pre>print('\nIndex (rows):', list(raw_states.index))</pre>	${ m g09/demo.py}$
pandas	index, renaming values	div_pop = div_pop.rename(index=div_names)	g11/demo.py
pandas	index, retrieving a row by name	$pa_row = states.loc['Pennsylvania']$	g09/demo.py
pandas	index, retrieving first rows by location	print(low_to_high.iloc[0:10])	g09/demo.py
pandas	index, retrieving last rows by location	<pre>print(low_to_high.iloc[-5:])</pre>	g09/demo.py

pandas plotting, bar plot plotting, histogram hh_data['etr'].plot.hist(ax=ax0,bins=20,title='Distribu g11/de pandas plotting, horizontal bar plot div_pop.plot.barh(ax=ax1) g11/de pandas plotting, scatter clored by 3rd var tidy_data.plot.scatter(ax=ax4.x='lincome',y='ETR',c='typ g12/de pandas plotting, scatter plot hh_data.plot.scatter(ax=ax4.x='lincome',y='etr',title='ETR g12/de pandas plotting, scatter plot hh_data.plot.scatter(ax=ax21,x='inc',y='etr',title='ETR g12/de pandas plotting, turning off legend sel.plot.barh(x='Name',y='percent',ax=ax.legend=None) g13/de pandas reading, setting index column reading, using dtype dictionary county = pd.read_csv('state-data.csv') g09/de pandas reading, setting index column state_data = pd.read_csv('state-data.csv',index_col='na g11/de pandas series, automatic alignment by index series, contains RE or RE is_TT = trim['Number'].str.contains(r'*12") g12/de pandas series, contains an plain string has_AM = trim['Time'].str.contains(r'Tu Th'') g12/de pandas series, contains an RE has_AMPM = trim['Time'].str.contains("AM.*PM'') g12/de pandas series, converting to lower case name = mod['name'].str.loue(') g13/de pandas series, converting to lower case name = mod['name'].str.loue(') g23/de pandas series, converting to lower case name = mod['name'].str.loue(') g23/de pandas series, filling missing values mod['comb_units'] = mod['comb_units'].fillna('feet') g23/de pandas series, replacing values using a dictionary units = units.str.strip(') g23/de pandas series, replacing values units = units.str.strip(') g23/de pandas series, sort in decending order div_pop.sort_values(ascending=False) g11/de pandas series, sort	Module	Description	Example	Script
pandas plotting, histogram hh_data['etr'], plot hist(ax=ax0,bins=20,title='Distribu g12/de pandas plotting, horizontal bar plot div_pop.plot.barh(ax=ax1) g11/de jobting, scatter colored by 3rd var tidy_data.plot.scatter(ax=ax4_x='Income',y='ETR',c='typ g12/de pandas plotting, scatter plot hh_data.plot.scatter(ax=ax4_x='inc',y='etr',title='ETR g12/de pandas plotting, turning off legend sel.plot.barh(x='Name',y='percent',ax=ax_legend=None) g13/de pandas reading, setting index column state_data = pd.read_csv('state-data.csv') g09/de pandas reading, setting index column state_data = pd.read_csv('state-data.csv',index_col='na g11/de pandas series, applying a function to each element pandas series, applying a function to each element pandas series, automatic alignment by index merged('percent') = 100*merged('pop')/div_pop g13/de pandas series, contains RE or RE is_TT = trim['Days'].str.contains("TulTh") g12/de pandas series, contains a plain string has_AM = trim['Time'].str.contains("AM.") g12/de pandas series, converting to a list print (name_atal_State*_lots()) g13/de pandas series, converting to a list print (name_atal_State*_lots()) g13/de pandas series, converting to lower case name = mod['name'].str.lower() g23/de pandas series, converting to lower case name = mod['name'].str.lower() g23/de pandas series, converting to lower case name = mod['name'].str.lower() g23/de pandas series, converting to a list print (name_atal_S'state*_lots()) g13/de pandas series, converting to a list print (name_atal_S'state*_lots()) g13/de pandas series, converting to a list print (name_atal_S'state*_lots()) g13/de pandas series, converting to a list print (name_atal_S'state*_lots()) g23/de pandas series, streplacing values using a dictionary units = units.series(pellout) g23/de pandas series, replacing values using a dictionary units = units.series(pellout) g23/de pandas series, sort in decending order div_pop = div_pop.sort_value(ascending=False) g11/de pandas series, sort in decending order div_pop = div_pop.sort_valu	pandas	index, setting to a column	states = raw_states.set_index('name')	g09/demo.py
pandas plotting, horizontal bar plot div_pop_plot.barh(ax=ax1) g11/data.plotting, scatter colored by 3rd var bidy_data.plot.scatter(ax=ax4).x='lncome',y='ETR',c='typ g12/data.plot.scatter(ax=ax21,x='inc',y='etr',title='ETR g12/data.plot.scatter(ax=ax21,x='inc',y='etr',title='ETR g12/data.plot.scatter(ax=ax21,x='inc',y='etr',title='ETR g12/data.plot.scatter(ax=ax21,x='inc',y='etr',title='ETR g12/data.plot.scatter(ax=ax21,x='inc',y='etr',title='ETR g12/data.plot.scatter(ax=ax21,x='inc',y='etr',title='ETR g12/data.plot.scatter(ax=ax21,x='income',y='etr',title='ETR g12/data.plot.scatter(ax=ax21,x='income',y='etr',title='ETR g12/data.plot.scatter(ax=ax21,x='income',y='etr',title='ETR g12/data.plot.plot.plot.plot.plot.plot.plot.plot	pandas	plotting, bar plot	$reg_pop.plot.bar(ax=ax1)$	g11/demo.py
pandas plotting, scatter colored by 3rd var plotting, scatter plot hh_data.plot.scatter(ax=ax4,x='lncome',y='ETR',c='typ	pandas	plotting, histogram	hh_data['etr'].plot.hist(ax=ax0,bins=20,title='Distribu	g12/demo.py
pandas plotting, scatter plot plotting, turning off legend sel.plot.barh(x='Name',y='percent',ax=ax,legend=None) g13/dc sel.plot.barh(x='name',y='percent',ax=ax,legend=None,y='percent',ax=ax,legend=late,sv:filling sel.plot.barh(x='name,y='percent',y='percent',y='percent',y='percent',y='percent',y='percent',y='percent',y='percent',y='percent',y='percent',y='percent',y='percent',y='percent,y='perce	pandas	plotting, horizontal bar plot	div_pop.plot.barh(ax=ax1)	g11/demo.py
pandas plotting, turning off legend sel.plot.barh(x='Name',y='percent',ax=ax,legend=None) g13/de pandas reading, csv data raw_states = pd.read_csv('state-data.csv') g09/de pandas reading, setting index column state_data = pd.read_csv('state-data.csv',index_col='na g11/de pandas reading, using dtype dictionary county = pd.read_csv('county_pop.csv',dtype=fips) g10/de pandas series, RE at start is_LD = trim['Number'].str.contains(r"12") g12/de pandas series, applying a function to each element series, automatic alignment by index merged['percent'] = 100*merged['pop']/div_pop g13/de pandas series, combining via where() mod['comb_units'] = unit_part.where(unit_part!=" , mo g23/de pandas series, contains RE or RE is_TT = trim['Days'].str.contains("AM") g12/de pandas series, contains a plain string has_AM = trim['Time'].str.contains("AM") g12/de pandas series, converting strings to title case fixname = subset_view['NAME'].str.title() g16/de pandas series, converting to lower case name = mod['name'].str.lower() g23/de pandas series, dropping rows using a list conus = states.drop(not_conus) g22/de pandas series, element-by-element or is_either = is_ca is_tx g16/de pandas series, removing spaces units = units.replace(spellout) g23/de pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) g11/de pandas series, splitting strings on whitespace name_parts = name_str.split() g13/de pandas series, splitting win RE trim['Split'] = trim['Time'].str.split(r':] - ") g12/de pandas series, splitting win RE trim['Split'] = trim['Time'].str.split(r':] - ", expand=True) g12/de pandas series, splitting win RE trim['Split'] = trim['Time'].str.split(r':] - ", expand=True) g12/de pandas series, unstacking toto_wine experiment_state('PGI') g16/de pandas series, unstacking	pandas	plotting, scatter colored by 3rd var	$tidy_data.plot.scatter(ax=ax4,x=`Income',y=`ETR',c='typ$	g12/demo.py
pandas reading, csv data reading, setting index column state_data = pd.read_csv('state-data.csv') g09/dc pandas reading, setting index column state_data = pd.read_csv('state-data.csv',index_col='na g11/dc pandas reading, using dtype dictionary county = pd.read_csv('county_pop.csv',dtype=fips) g10/dc pandas series, applying a function to each element pandas series, applying a function to each element pandas series, combining via where() name_clean = name_parts.apply(' 'join) g23/dc pandas series, contains RE or RE is_TT = trim['Days'].str.contains(r'Tl_2") mod. g23/dc pandas series, contains RE or RE is_TT = trim['Days'].str.contains("AM") g12/dc pandas series, converting to a list print(name_data['State'].to_list()) g13/dc pandas series, converting to a list print(name_data['State'].to_list()) g13/dc pandas series, converting to lower case name = mod['name'].str.lower() g23/dc pandas series, converting to lower case name = mod['name'].str.lower() g23/dc pandas series, filling missing values mod['comb_units'] = mod['comb_units'].fillna('feet') g23/dc pandas series, retrieving an element or is_either = is_ca is_tx mod['comb_units'].fillna('feet') g23/dc pandas series, retrieving an element print("inflorida's population:", pop['Florida']/1e6) g09/dc pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) g11/dc pandas series, splitting strings on whitespace pandas series, splitting with expand series, splitting with expand series, unstacking tot_watch in tot_watch in the pandas series, unstacking tot_watch in the pandas series in the pandas series, unstackin	pandas	plotting, scatter plot	hh_data.plot.scatter(ax=ax21,x='inc',y='etr',title='ETR	g12/demo.py
pandas reading, setting index column reading, using dtype dictionary county = pd.read_csv('state-data.csv',index_col='na g11/de pandas reading, using dtype dictionary county = pd.read_csv('county_pop.csv',dtype=fips) g10/de pandas series, RE at start is_LD = trim['Number'].str.contains(r"1 2") g12/de pandas series, applying a function to each element pandas series, combining via where() mame_clean = name_parts.apply(' '.join) g23/de pandas series, combining via where() mod['comb_units'] = unit_part.where(unit_part!=" , mo	pandas	plotting, turning off legend	sel.plot.barh(x=`Name',y=`percent',ax=ax,legend=None)	g13/demo.py
pandas reading, using dtype dictionary county = pd.read_csv('county_pop.csv',dtype=fips) g10/dc pandas series, RE at start is_LD = trim['Number'].str.contains(r"12") g12/dc pandas series, applying a function to each element name_clean = name_parts.apply(' '.join) g23/dc pandas series, combining via where() mod['comb_units'] = unit_part.where(unit_part!=" , mo	pandas	reading, csv data		g09/demo.py
pandas series, RE at start is_LD = trim['Number'].str.contains(r'*12") g12/de pandas series, applying a function to each element pandas series, automatic alignment by index merged['percent'] = 100*merged['pop']/div_pop g13/de pandas series, combining via where() mod['comb_units'] = unit_part.where(unit_part!=" , mo	pandas	reading, setting index column	$state_data = pd.read_csv(`state-data.csv',index_col='na$	g11/demo.py
pandas series, applying a function to each element pandas series, automatic alignment by index merged['percent'] = 100*merged['pop']/div_pop g13/de pandas series, combining via where() mod['comb_units'] = unit_part.where(unit_part!=" , mo	pandas	reading, using dtype dictionary	county = pd.read_csv('county_pop.csv',dtype=fips)	g10/demo.py
pandas series, automatic alignment by index pandas series, combining via where() mod['comb_units'] = unit_part.where(unit_part!=" , mo	pandas	series, RE at start	$is_LD = trim['Number'].str.contains(r"1 2")$	g12/demo.py
pandas series, combining via where() mod['comb_units'] = unit_part.where(unit_part!=" , mo g23/de pandas series, contains RE or RE is_TT = trim['Days'].str.contains(r'Tu Th") g12/de pandas series, contains a plain string has_AM = trim['Time'].str.contains("AM") g12/de pandas series, contains an RE has_AMPM = trim['Time'].str.contains("AM.*PM") g12/de pandas series, converting strings to title case fixname = subset_view['NAME'].str.title() g16/de pandas series, converting to a list print(name_data['State'].to_list()) g13/de pandas series, dropping rows using a list conus = states.drop(not_conus) g22/de pandas series, element-by-element or is_either = is_ca is_tx g16/de pandas series, removing spaces units = units.str.strip() g23/de pandas series, replacing values using a dictionary pandas series, replacing values using a dictionary units = units.replace(spellout) g23/de pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) g11/de pandas series, splitting strings on whitespace name_parts = name.str.split() g23/de pandas series, splitting with expand exp = trim['Time'].str.split(r': - ") g12/de pandas series, unstacking tot_wide = tot_amt.unstack('PGI') g16/de g16/de series, unstacking	pandas	series, applying a function to each element	name_clean = name_parts.apply(' '.join)	g23/demo.py
pandas series, contains RE or RE is_TT = trim['Days'].str.contains(r"Tu Th") g12/de pandas series, contains a plain string has_AM = trim['Time'].str.contains("AM") g12/de pandas series, contains an RE has_AMPM = trim['Time'].str.contains("AM.*PM") g12/de pandas series, converting strings to title case fixname = subset_view['NAME'].str.title() g16/de pandas series, converting to lower case fixname = subset_view['NAME'].str.title() g13/de pandas series, converting to lower case name = mod['name'].str.lower() g23/de pandas series, dropping rows using a list conus = states.drop(not_conus) g22/de pandas series, element-by-element or is_either = is_ca is_tx g16/de pandas series, filling missing values mod['comb_units'] = mod['comb_units'].fillna('feet') g23/de pandas series, renoving spaces units = units.str.strip() g23/de pandas series, replacing values using a dictionary units = units.replace(spellout) g23/de pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) g11/de pandas series, sorting by value low_to_high = normed['med_pers_inc'].sort_values() g09/de pandas series, splitting strings on whitespace name_parts = name.str.split(": - ") g12/de pandas series, splitting with expand exp = trim["Split'] = trim("Time"].str.split(r": - ") g12/de pandas series, summing reg_pop = by_reg['pop'].sum()/1e6 g11/de pandas series, unstacking tot_wide = tot_amt.unstack('PGI')	pandas	series, automatic alignment by index	$merged[`percent'] = 100 *merged[`pop']/div_pop$	g13/demo.py
pandas series, contains a plain string has_AM = trim['Time'].str.contains("AM") g12/de pandas series, contains an RE has_AMPM = trim['Time'].str.contains("AM.*PM") g12/de pandas series, converting strings to title case fixname = subset_view['NAME'].str.title() g16/de pandas series, converting to a list print(name_data['State'].to_list()) g13/de pandas series, converting to lower case name = mod['name'].str.lower() g23/de pandas series, dropping rows using a list conus = states.drop(not_conus) g22/de pandas series, element-by-element or is_either = is_ca is_tx g16/de pandas series, filling missing values mod['comb_units'] = mod['comb_units'].fillna('feet') g23/de pandas series, removing spaces units = units.str.strip() g23/de pandas series, replacing values using a dictionary units = units.replace(spellout) g23/de pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) g11/de pandas series, splitting strings on whitespace name_parts = name.str.split() g23/de pandas series, splitting via RE trim['Split'] = trim["Time"].str.split(r": - ") g12/de pandas series, summing reg_pop = by_reg['pop'].sum()/1e6 g11/de pandas series, unstacking tot_wide = tot_amt.unstack('PGI')	pandas	series, combining via where()	mod['comb_units'] = unit_part.where(unit_part!=" , mo	g23/demo.py
pandas series, contains an RE has_AMPM = trim['Time'].str.contains("AM.*PM") g12/de pandas series, converting strings to title case fixname = subset_view['NAME'].str.title() pandas series, converting to a list print(name_data['State'].to_list()) pandas series, converting to lower case name = mod['name'].str.lower() pandas series, dropping rows using a list conus = states.drop(not_conus) pandas series, element-by-element or is_either = is_ca is_tx pandas series, filling missing values pandas series, removing spaces units = units.str.strip() pandas series, replacing values using a dictionary units = units.str.strip() pandas series, retrieving an element print("\nFlorida's population:", pop['Florida']/1e6) pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) pandas series, sorting by value low_to_high = normed['med_pers_inc'].sort_values() pandas series, splitting strings on whitespace name_parts = name.str.split(": - ") pandas series, splitting with expand exp = trim["Time"].str.split(r": - ") pandas series, summing series, summing reg_pop = by_reg['pop'].sum()/le6 pandas series, unstacking pandas series, unstacking pandas series, unstacking pandas series, unstacking	pandas	series, contains RE or RE	$is_TT = trim['Days'].str.contains(r"Tu Th")$	g12/demo.py
pandas series, converting strings to title case fixname = subset_view['NAME'].str.title() pandas series, converting to a list print(name_data['State'].to_list()) pandas series, converting to lower case name = mod['name'].str.lower() pandas series, dropping rows using a list conus = states.drop(not_conus) pandas series, element-by-element or is_either = is_ca is_tx g16/de pandas series, filling missing values mod['comb_units'] = mod['comb_units'].fillna('feet') g23/de pandas series, removing spaces units = units.str.strip() g23/de pandas series, replacing values using a dictionary units = units.replace(spellout) pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) g11/de pandas series, sorting by value low_to_high = normed['med_pers_inc'].sort_values() g09/de pandas series, splitting strings on whitespace name_parts = name.str.split() pandas series, splitting with expand exp = trim["Time"].str.split(r": - ") g12/de pandas series, summing reg_pop = by_reg['pop'].sum()/1e6 pandas series, unstacking tot_wide = tot_amt.unstack('PGI')	pandas	series, contains a plain string	$has_AM = trim['Time'].str.contains("AM")$	g12/demo.py
pandas series, converting to a list print(name_data['State'].to_list()) g13/de pandas series, converting to lower case name = mod['name'].str.lower() g23/de pandas series, dropping rows using a list conus = states.drop(not_conus) g22/de pandas series, element-by-element or is_either = is_ca is_tx g16/de pandas series, filling missing values mod['comb_units'] = mod['comb_units'].fillna('feet') g23/de pandas series, removing spaces units = units.str.strip() g23/de pandas series, replacing values using a dictionary units = units.replace(spellout) g23/de pandas series, retrieving an element print("\nFlorida's population:", pop['Florida']/1e6) g09/de pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) g11/de pandas series, splitting via RE trim['Split'] = trim["Time"].str.split(r": - ") g12/de pandas series, splitting with expand exp = trim["Time"].str.split(r": - ", expand=True) g12/de pandas series, unstacking tot_wide = tot_amt.unstack('PGI')	pandas	series, contains an RE		g12/demo.py
pandas series, converting to lower case name = mod['name'].str.lower() pandas series, dropping rows using a list conus = states.drop(not_conus) pandas series, element-by-element or is_either = is_ca is_tx g16/de pandas series, filling missing values mod['comb_units'] = mod['comb_units'].fillna('feet') pandas series, removing spaces units = units.str.strip() pandas series, replacing values using a dictionary units = units.replace(spellout) pandas series, retrieving an element print("\nFlorida's population:", pop['Florida']/1e6) g09/de pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) pandas series, sorting by value low_to_high = normed['med_pers_inc'].sort_values() pandas series, splitting strings on whitespace name_parts = name.str.split() pandas series, splitting with expand exp = trim["Time"].str.split(r": - ") pandas series, summing reg_pop = by_reg['pop'].sum()/1e6 pandas series, unstacking tot_wide = tot_amt.unstack('PGI') g23/de	pandas	series, converting strings to title case	$fixname = subset_view['NAME'].str.title()$	${\sf g16/demo.py}$
pandas series, dropping rows using a list conus = states.drop(not_conus) pandas series, element-by-element or is_either = is_ca is_tx g16/de pandas series, filling missing values mod['comb_units'] = mod['comb_units'].fillna('feet') g23/de pandas series, removing spaces units = units.str.strip() g23/de pandas series, replacing values using a dictionary pandas series, retrieving an element print("\nFlorida's population:", pop['Florida']/1e6) g09/de pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) g11/de pandas series, sorting by value low_to_high = normed['med_pers_inc'].sort_values() g09/de pandas series, splitting strings on whitespace name_parts = name.str.split() g23/de pandas series, splitting via RE trim['Split'] = trim["Time"].str.split(r": - ") g12/de pandas series, summing reg_pop = by_reg['pop'].sum()/1e6 g11/de pandas series, unstacking tot_wide = tot_amt.unstack('PGI')	pandas	series, converting to a list	<pre>print(name_data['State'].to_list())</pre>	g13/demo.py
pandas series, element-by-element or is_either = is_ca is_tx g16/de pandas series, filling missing values mod['comb_units'] = mod['comb_units'].fillna('feet') g23/de pandas series, removing spaces units = units.str.strip() g23/de pandas series, replacing values using a dictionary pandas series, retrieving an element print("\nFlorida's population:", pop['Florida']/1e6) g09/de pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) g11/de pandas series, sorting by value low_to_high = normed['med_pers_inc'].sort_values() g09/de pandas series, splitting strings on whitespace name_parts = name.str.split(": - ") g12/de pandas series, splitting via RE trim['Split'] = trim["Time"].str.split(r':: - ") g12/de pandas series, summing reg_pop = by_reg['pop'].sum()/1e6 g11/de pandas series, unstacking tot_wide = tot_amt.unstack('PGI')	pandas	series, converting to lower case	name = mod[`name'].str.lower()	g23/demo.py
pandas series, filling missing values mod['comb_units'] = mod['comb_units'].fillna('feet') g23/de pandas series, removing spaces units = units.str.strip() g23/de pandas series, replacing values using a dictionary units = units.replace(spellout) g23/de pandas series, retrieving an element print("\nFlorida's population:", pop['Florida']/1e6) g09/de pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) g11/de pandas series, sorting by value low_to_high = normed['med_pers_inc'].sort_values() g09/de pandas series, splitting strings on whitespace name_parts = name.str.split() g23/de pandas series, splitting via RE trim['Split'] = trim["Time"].str.split(r": - ") g12/de pandas series, splitting with expand exp = trim["Time"].str.split(r": - ", expand=True) g12/de pandas series, summing reg_pop = by_reg['pop'].sum()/1e6 g11/de pandas series, unstacking tot_wide = tot_amt.unstack('PGI')	pandas	series, dropping rows using a list	conus = states.drop(not_conus)	g22/demo.py
pandas series, removing spaces units = units.str.strip() g23/de pandas series, replacing values using a dictionary units = units.replace(spellout) g23/de pandas series, retrieving an element print("\nFlorida's population:", pop['Florida']/1e6) g09/de pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) g11/de pandas series, sorting by value low_to_high = normed['med_pers_inc'].sort_values() g09/de pandas series, splitting strings on whitespace name_parts = name.str.split() g23/de pandas series, splitting via RE trim['Split'] = trim["Time"].str.split(r": - ") g12/de pandas series, splitting with expand exp = trim["Time"].str.split(r": - ") g12/de pandas series, summing reg_pop = by_reg['pop'].sum()/1e6 g11/de pandas series, unstacking tot_wide = tot_amt.unstack('PGI')	pandas	series, element-by-element or	is_either = is_ca is_tx	g16/demo.py
pandas series, replacing values using a dictionary units = units.replace(spellout) pandas series, retrieving an element print("\nFlorida's population:", pop['Florida']/1e6) pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) pandas series, sorting by value low_to_high = normed['med_pers_inc'].sort_values() pandas series, splitting strings on whitespace name_parts = name.str.split() pandas series, splitting via RE trim['Split'] = trim["Time"].str.split(r": - ") pandas series, splitting with expand exp = trim["Time"].str.split(r": - ", expand=True) pandas series, summing reg_pop = by_reg['pop'].sum()/1e6 pandas series, unstacking tot_wide = tot_amt.unstack('PGI')	pandas	series, filling missing values		g23/demo.py
pandas series, retrieving an element print("\nFlorida's population:", pop['Florida']/1e6) g09/de pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) g11/de pandas series, sorting by value low_to_high = normed['med_pers_inc'].sort_values() g09/de pandas series, splitting strings on whitespace name_parts = name.str.split() g23/de pandas series, splitting via RE trim['Split'] = trim["Time"].str.split(r": - ") g12/de pandas series, splitting with expand exp = trim["Time"].str.split(r": - ", expand=True) g12/de pandas series, summing reg_pop = by_reg['pop'].sum()/1e6 g11/de pandas series, unstacking tot_wide = tot_amt.unstack('PGI') g16/de	pandas	series, removing spaces	units = units.str.strip()	g23/demo.py
pandas series, sort in decending order div_pop = div_pop.sort_values(ascending=False) g11/de pandas series, sorting by value low_to_high = normed['med_pers_inc'].sort_values() g09/de pandas series, splitting strings on whitespace name_parts = name.str.split() g23/de pandas series, splitting via RE trim['Split'] = trim["Time"].str.split(r": - ") g12/de pandas series, splitting with expand exp = trim["Time"].str.split(r": - ", expand=True) g12/de pandas series, summing reg_pop = by_reg['pop'].sum()/1e6 g11/de pandas series, unstacking tot_wide = tot_amt.unstack('PGI') g16/de	pandas	series, replacing values using a dictionary		g23/demo.py
pandas series, sorting by value low_to_high = normed['med_pers_inc'].sort_values() g09/de pandas series, splitting strings on whitespace name_parts = name.str.split() g23/de pandas series, splitting via RE trim['Split'] = trim["Time"].str.split(r": - ") g12/de pandas series, splitting with expand exp = trim["Time"].str.split(r": - ", expand=True) g12/de pandas series, summing reg_pop = by_reg['pop'].sum()/1e6 g11/de pandas series, unstacking tot_wide = tot_amt.unstack('PGI') g16/de	pandas	series, retrieving an element	print("\nFlorida's population:", pop['Florida']/1e6)	${ m g09/demo.py}$
pandas series, splitting strings on whitespace name_parts = name.str.split() g23/de pandas series, splitting via RE trim['Split'] = trim["Time"].str.split(r": - ") g12/de pandas series, splitting with expand exp = trim["Time"].str.split(r": - ", expand=True) g12/de pandas series, summing reg_pop = by_reg['pop'].sum()/1e6 g11/de pandas series, unstacking tot_wide = tot_amt.unstack('PGI') g16/de	pandas	series, sort in decending order	div_pop = div_pop.sort_values(ascending=False)	g11/demo.py
$\begin{array}{llllllllllllllllllllllllllllllllllll$	pandas	series, sorting by value	low_to_high = normed['med_pers_inc'].sort_values()	g09/demo.py
pandas series, splitting with expand $exp = trim["Time"].str.split(r": - ", expand=True)$ $g12/de$ pandas series, summing $reg_pop = by_reg['pop'].sum()/1e6$ $g11/de$ pandas series, unstacking $tot_wide = tot_amt.unstack('PGI')$ $g16/de$	pandas	series, splitting strings on whitespace		g23/demo.py
pandas series, summing $reg_pop = by_reg['pop'].sum()/1e6$ $g11/de$ pandas series, unstacking $tot_wide = tot_amt.unstack('PGI')$ $g16/de$	pandas	series, splitting via RE		g12/demo.py
pandas series, unstacking $tot_wide = tot_amt.unstack('PGI')$ $g16/de$	pandas	series, splitting with expand		g12/demo.py
	pandas			g11/demo.py
pandas series, using isin() fixed = flood['TAX_ID'].isin(dup_rec['TAX_ID']) g14/de	pandas	series, unstacking		g16/demo.py
	pandas	series, using isin()	$fixed = flood['TAX_ID'].isin(\ dup_rec['TAX_ID']\)$	g14/demo.py

Module	Description	Example	Script
pandas	series, using value_counts()	<pre>print('\nOuter:\n', join_o['_merge'].value_counts(), s</pre>	g14/demo.py
requests	calling the get() method	response = requests.get(api,payload)	g18/demo.py
requests	checking the URL	<pre>print('url:', response.url)</pre>	g18/demo.py
requests	checking the response text	<pre>print(response.text)</pre>	g18/demo.py
requests	checking the status code	<pre>print('status:', response.status_code)</pre>	g18/demo.py
requests	decoding a JSON response	rows = response.json()	g18/demo.py
requests	importing the module	import requests	g18/demo.py
scipy	calling newton's method	<pre>cr = opt.newton(find_cube_root,xinit,maxiter=20,args=[y</pre>	g07/demo.py
scipy	importing the module	import scipy.optimize as opt	g07/demo.py
seaborn	adding a title to a grid object	jg.fig.suptitle('Distribution of Hourly Load')	g17/demo.py
seaborn	barplot	sns.barplot(data=hourly,x='hour',y='usage',hue='month',	g17/demo.py
seaborn	basic violin plot	sns.violinplot(data=janjul,x="month",y="usage")	g17/demo.py
seaborn	boxenplot	sns.boxenplot(data=janjul,x="month",y="usage")	g17/demo.py
seaborn	calling tight_layout on a grid object	jg.fig.tight_layout()	g17/demo.py
seaborn	drawing a heatmapped grid	sns.heatmap(means,annot=True,fmt=".0f",cmap='Spectral',	g20/demo.py
seaborn	importing the module	import seaborn as sns	g17/demo.py
seaborn	joint distribution hex plot	jg = sns.jointplot(data=bymo,x=1,y=7,kind='hex')	g17/demo.py
seaborn	setting axis titles on a grid object	jg.set_axis_labels('January','July')	g17/demo.py
seaborn	setting the theme	sns.set_theme(style="white")	g17/demo.py
seaborn	split violin plot	sns.violinplot(data=eights,x="hour",y="usage",hue="mont	g17/demo.py
zipfile	importing the module	import zipfile	g15/demo.py
zipfile	opening a file in an archive	fh1 = archive.open('generators-oswego.csv')	g15/demo.py
zipfile	opening an archive	archive = zipfile.ZipFile('generators.zip')	g15/demo.py
zipfile	reading the list of files	<pre>print(archive.namelist())</pre>	g15/demo.py