core dictionary, adding a new entry co po'] = 'CO' core dictionary, creating core dictionary, creating via comprehension fips_cols = {colst for col in fips_vars} gb5/demo,py core dictionary, looking up a value name = ny{name} con po'] = 'CO' con dictionary, making a list of dictionary, making a list of dictionary, making a list of wide point of dictionary, obtaining a list of keys name = ny{name} con point point	Module	Description	Example	Script
core dictionary, creating via comprehension fips_cols = {col:str for col in fips_vars} g05/demo.py g13/demo.py g05/demo.py g06/demo.py g06/de	core	dictionary, adding a new entry	co['po'] = 'CO'	g05/demo.py
core dictionary, looking up a value name = ny[name'] g05/demo.py g07/demo.py g07/demo.py g07/demo.py g07/demo.py g07/demo.py g05/demo.py g07/demo.py g05/demo.py g05/demo.py g06/demo.py g05/demo.py g06/demo.py g05/demo.py g05/demo.	core	dictionary, creating	co = {'name':'Colorado', 'capital':'Denver'}	g05/demo.py
core dictionary, looking up a value name = ny[name'] g05/demo.py g07/demo.py g07/demo.py g07/demo.py g07/demo.py g07/demo.py g05/demo.py g07/demo.py g05/demo.py g05/demo.py g06/demo.py g05/demo.py g06/demo.py g05/demo.py g05/demo.	core	dictionary, creating via comprehension	fips_cols = {col:str for col in fips_vars}	
core dictionary, obtaining a list of keys names = super_dict.keys() g05/demo.py core f-string, grouping with commas print(f'Total population: {tot_pop:,}') g11/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') g05/demo.py core file, output using print print("It was written during",year,file=fh) g02/demo.py core file, output using write print("Notuer:\n', join_o[_merge'].value_counts(), s g14/demo.py core file, reading one line at a time for line in fh: core function, calling with an optional argument core function, defining with optional argument core function, defining with optional argument core function, defining with optional argument core function, returning a result core list, appending an element a_list.append("four") core list, create via comprehension core list, create via comprehension list, extending with spaces print("a_list) for in a_list: g03/demo.py core list, generating a sequence b_list extending with spaces is sequence list, selecting elements 0 to 3 print("f'Total population: {tot_pop.}') g11/demo.py g02/demo.py g03/demo.py g02/demo.py g02/demo.py g02/demo.py g02/demo.py g02/demo.py g02/demo.py g03/demo.py g05/demo.py g05/demo.py g06/demo.py g06/demo.py g07/demo.py g06/demo.py	core	dictionary, looking up a value	name = ny['name']	g05/demo.py
core f-string, grouping with commas print(f'Total population: {tot_pop:,}') g11/demo.py core f-string, using a formatting string print(f'ToV of {payment} with T={year} and r={r} is \${p} g07/demo.py core file, opening for reading fh = open(states.csv') g05/demo.py core file, opening for writing fh = open(filename,"w') g02/demo.py core file, output using print print("It was written during",year,file=fh) 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 print("\nOuter:\n", join_o["_merge"].value_counts(), s g14/demo.py core file, reading one line at a time for line in fh: g05/demo.py core function, calling through a list of tuples for number,name in div_info: g04/demo.py core function, defining to function, defining with an optional argument core function, defining with optional argument core function, telling with optional argument core function, returning a result return values g06/demo.py core list, appending an element a_list.append("four") core list, create via comprehension cubes = [n**3 for n in a_list] 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) b_list = range(1.6) g03/demo.py core list, extending with spaces a_s_string = ".join(a_list) g03/demo.py core list, selecting element print(a_list(4)) print(a_list(4))	core	dictionary, making a list of	list1 = [co, ny]	g05/demo.py
core file, closing file, closing file, closing file, closing file, closing file, opening for reading file, opening for reading file, opening for writing file, opening file, opening for writing file, opening for writing file, opening file, opening file, opening file, opening file, opening file, opening for writing file, opening file, opening file, opening for writing file, opening file, opening for writing file, opening	core	dictionary, obtaining a list of keys	$names = super_dict.keys()$	g05/demo.py
core file, closing for reading fh.close() g02/demo.py core file, opening for reading fh = open('states.csv') g05/demo.py core file, opening for writing fh = open(filename,"w") g02/demo.py core file, output using print print("It was written during", year, file=fh) 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 print('\nOuter:\n', join_o['_merge'].value_counts(), s 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 function, calling sit for n in a_list: g05/demo.py core function, calling with an optional argument core function, defining def sumsq(values): g06/demo.py core function, defining with optional argument core function, returning a result return values g06/demo.py core list, create via comprehension cubes = [n**3 for n in a_list] g04/demo.py core list, creating a list is a_list = ["zero", "one", "two", "three"] g03/demo.py core list, determining length n = len(b_list) g04/demo.py core list, generating a sequence b_list = range(1,6) g04/demo.py core list, selecting an element print(a_list[0]) print(a_list[0]) g03/demo.py core list, selecting an element print(a_list[0]) print(a_list[0]) g03/demo.py core list, selecting an element print(a_list[0]) g03/demo.py core	core	f-string, grouping with commas		g11/demo.py
core file, opening for reading fh = open('states.csv') g05/demo.py core file, opening for writing fh = open(filename, 'w') g02/demo.py core file, output using print print(''the was written during", year, file=fh) 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 print('\nOuter:\n', join_o['_merge'].value_counts(), s 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 function, calling d1_ssq = sumsq(d1) g06/demo.py core function, defining with an optional argument core function, defining with optional argument core function, defining with optional argument def sample_function(payment,year,r=0.05): 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, creating a_list = ['zero', 'one", "two", "three"] 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) g03/demo.py core list, selecting an element print(a_list[:4])	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, opening for writing fh = open(filename, "w") g02/demo.py core file, output using print print("It was written during", year, file=fh) 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 print('\nOuter:\n', join_o['merge'].value_counts(), s 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 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 def sample_function(payment,year,r=0.05): 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, 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, selecting an element print(a_list['4]) g03/demo.py core list, selecting an element print(a_list['4])	core	file, closing		g02/demo.py
core file, output using print print("It was written during", year, file=fh) g02/demo.py core file, output using write fh.write("Where was this file was written?\n") g02/demo.py g02/demo.py g02/demo.py g05/demo.py g06/demo.py g06/demo.py g13/demo.py g06/demo.py g13/demo.py g06/demo.py g07/demo.py g06/demo.py g07/demo.py g06/demo.py g07/demo.py g07/demo.py g07/demo.py g08/demo.py g08/demo.	core	file, opening for reading	fh = open('states.csv')	g05/demo.py
core file, output using write file, write("Where was this file was written?\n") core file, print without adding spaces print('\nOuter:\n', join_o['_merge'].value_counts(), s g14/demo.py g05/demo.py core for, looping through a list for line in fh: core for, looping through a list for n in a_list: core function, calling core function, calling with an optional argument core function, defining def sumsq(values): core function, defining with optional argument core function, returning a result core list, appending an element core list, create via comprehension core list, determining length n = len(b_list) core list, generating a sequence list, generating a sequence core list, selecting an element list, selecting an element print(a_list[4]) core list, selecting an element print(a_list[4]) fh.write("Where was this file was written?\n") g02/demo.py g14/demo.py g05/demo.py g05/demo.py g06/demo.py g06/demo.py g06/demo.py g06/demo.py g06/demo.py g06/demo.py g06/demo.py g03/demo.py	core			g02/demo.py
core file, print without adding spaces file, reading one line at a time for line in fh: core for, looping through a list for n in a_list: core for, looping through a list of tuples for number,name in div_info: core function, calling d1_ssq = sumsq(d1) core function, defining with an optional argument core function, defining with optional argument core function, defining with optional argument core function, returning a result return values core list, appending an element a_list.append("four") core list, create via comprehension cubes = [n**3 for n in a_list] core list, determining length n = len(b_list) core list, generating a sequence b_list = range(1,6) core list, generating a sequence b_list = range(1,6) core list, selecting an element print(a_list[0]) core list, selecting elements 0 to 3	core	file, output using print	<pre>print("It was written during",year,file=fh)</pre>	g02/demo.py
core file, reading one line at a time for line in fh: core for, looping through a list for n in a_list: core for, looping through a list of tuples for number,name in div_info: core function, calling d1_ssq = sumsq(d1) core function, calling with an optional argument core function, defining with optional argument def sample_function(payment,year,r=0.07) core function, defining with optional argument def sample_function(payment,year,r=0.05): core function, returning a result return values core list, appending an element core list, create via comprehension core list, creating a_list=["revor","one","thvo","three"] core list, determining length n = len(b_list) core list, generating a sequence b_list = range(1,6) core list, selecting an element print(a_list[0]) core list, selecting an element print(a_list[1]) core list, selecting elements 0 to 3 print(a_list[1])	core	file, output using write		g02/demo.py
core for, looping through a list for n in a_list: g04/demo.py for, looping through a list of tuples for number,name in div_info: g13/demo.py core function, calling d1_ssq = sumsq(d1) sample_function(100, 10, r=0.07) g07/demo.py def sumsq(values): g06/demo.py def sumsq(values): g06/demo.py def sample_function(payment,year,r=0.05): g07/demo.py def sample_function(payment,year,r=0.05): g07/demo.py g06/demo.py def sample_function(payment,year,r=0.05): g07/demo.py g06/demo.py def sample_function(payment,year,r=0.05): g07/demo.py g06/demo.py g06/demo.py g06/demo.py g06/demo.py g06/demo.py g06/demo.py g03/demo.py g06/demo.py g03/demo.py g06/demo.py g03/demo.py g0	core	file, print without adding spaces	<pre>print('\nOuter:\n', join_o['_merge'].value_counts(), s</pre>	g14/demo.py
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	core	file, reading one line at a time	for line in fh:	g05/demo.py
core function, calling d1_ssq = sumsq(d1) g06/demo.py core function, calling with an optional argument core function, defining def sumsq(values): g06/demo.py core function, defining with optional argument core function, returning a result def sample_function(payment,year,r=0.05): 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, 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, selecting an element print(a_list[0]) g03/demo.py core list, selecting elements 0 to 3 print(a_list[4]) g03/demo.py g03/demo.py g03/demo.py core list, selecting elements 0 to 3	core	for, looping through a list	for n in a_list:	g04/demo.py
core function, calling with an optional argument core function, defining def sumsq(values): function, defining with optional argument core function, defining with optional argument def sample_function(payment,year,r=0.05): g07/demo.py def sample_function(payment,year,r=0.05): g03/demo.py demo.py def sample_function(payment,year,r=0.05): g03/demo.py demo.py	core	for, looping through a list of tuples	for number,name in div_info:	g13/demo.py
core function, defining def sumsq(values): core function, defining with optional argument core function, returning a result def sample_function(payment,year,r=0.05): core function, returning a result return values g06/demo.py core list, appending an element a_list.append("four") core list, create via comprehension cubes = [n**3 for n in a_list] core list, creating a_list = ["zero","one","two","three"] core list, determining length n = len(b_list) core list, extending with another list a_list.extend(a_more) core list, generating a sequence b_list = range(1,6) core list, joining with spaces a_string = " ".join(a_list) core list, selecting an element print(a_list[0]) core list, selecting elements 0 to 3 def sumsq(values): g07/demo.py g03/demo.py g04/demo.py g03/demo.py	core	function, calling	$d1_ssq = sumsq(d1)$	g06/demo.py
core function, defining with optional argument core function, returning a result return values g07/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, selecting an element print(a_list[0]) g03/demo.py core list, selecting elements 0 to 3 print(a_list[:4])	core	function, calling with an optional argument	sample_function(100, 10, r=0.07)	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 $g03/demo.py$ core list, creating $g03/demo.py$ $g03/demo.py$ core list, determining length $g03/demo.py$ $g03/demo.py$ core list, extending with another list $g03/demo.py$ core list, generating a sequence $g03/demo.py$ $g03/demo.py$ core list, joining with spaces $g03/demo.py$ $g03/demo.py$ core list, selecting an element $g03/demo.py$ $g03/demo.py$ core list, selecting an element $g03/demo.py$	core	function, defining	def sumsq(values):	g06/demo.py
core list, appending an element a_list.append("four") g03/demo.py core list, create via comprehension cubes = $[n**3 \text{ for n in a_list}]$ g04/demo.py core list, creating a_list = $["zero", "one", "two", "three"]$ 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])	core	function, defining with optional argument	def sample_function(payment,year,r=0.05):	g07/demo.py
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	core	function, returning a result	return values	g06/demo.py
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	core	list, appending an element	a_list.append("four")	g03/demo.py
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	core	list, create via comprehension	cubes = $[n^{**3} \text{ for n in a_list}]$	g04/demo.py
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	core	list, creating	a_list = ["zero","one","two","three"]	g03/demo.py
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	core	list, determining length	$n = len(b_list)$	g03/demo.py
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	core	list, extending with another list	a_list.extend(a_more)	g03/demo.py
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	core	list, generating a sequence		g04/demo.py
core list, selecting elements 0 to 3 print(a_list[:4]) g03/demo.py	core		a_string = " ".join(a_list)	g03/demo.py
\cdot	core	list, selecting an element	$print(a_list[0])$	g03/demo.py
core list, selecting elements 1 to 2 $print(a_list[1:3])$ $g03/demo.py$	core	-	print(a_list[:4])	g03/demo.py
	core	list, selecting elements 1 to 2	print(a_list[1:3])	g03/demo.py

Module	Description	Example	Script
core	list, selecting elements 1 to the end	print(a_list[1:])	g03/demo.py
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	<pre>print(name_states - pop_states)</pre>	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))	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, matching partial string	<pre>is_gas = trim['DBA Name'].str.contains('XPRESS')</pre>	g27/demo.py
core	string, matching start	$is_big = trim['DBA Name'].str.startswith(store)$	g27/demo.py
core	string, splitting on a comma	parts = line.split(`,`)	g05/demo.py
core	string, splitting on whitespace	$b_{list} = b_{string.split}()$	g03/demo.py
core	string, stripping blank space	${\sf clean} = [{\sf item.strip}() \; {\sf for} \; {\sf item} \; {\sf in} \; {\sf parts}]$	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	$layers = fiona.listlayers(demo_file)$	g24/demo.py
geopandas	adding a heatmap legend	slices.plot('s_pop',edgecolor='yellow',linewidth=0.2,le	g26/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	computing areas	zips['z_area'] = zips.area	g26/demo.py
geopandas	construct a buffer	$near_water = water_dis.buffer(1600)$	g24/demo.py
geopandas	constructing centroids	centroids[`geometry'] = tracts.centroid	g27/demo.py

geopandas extracting geometry from a geodataframe importing the module import geopandas importing the module import geopandas importing the module import geopandas as gpd geopandas geopandas merging data onto a geodataframe conus = conus.merge(trim.on="STATEP"), how="left", valida	Module	Description	Example	Script
geopandas importing the module import geopandas as gpd g21/c geopandas join to nearest object served_by = centroids.sjoin_nearest(geo,how='left',valida g27/c geopandas obtaining coordinates print('Number of points:', len(wv_geo_exterior.coords) g22/c geopandas overlaying a layer using union slices = zips.overlay(county,how='union',keep_geom_type g26/c geopandas plotting a boundary syr.boundary.plot(color='gray',linewidth=1,ax=ax1) g21/c geopandas plotting a boundary syr.boundary.plot(color='gray',linewidth=1,ax=ax1) g21/c geopandas reading a file syr egod.read_file("tb_2016_3o_place-syracuse.zip") g21/c geopandas reading a shapefile states = gpd.read_file("tb_2016_3o_place-syracuse.zip") g22/c geopandas reading data in WKT format coords = gpd.GeoSeries.from_wkt(bigl'Georeference')) g27/c geopandas setting the color of a plot county.plot(color='tan',ax=ax1) g24/c geopandas setting transparency via alpha near_clip.plot(alpha=0.25,ax=ax1) g24/c geopandas spatial join, contains c_contains_z = county.sjoin(color='tan',ax=ax1) g24/c geopandas spatial join, coroses i_crosses_z = inter.sjoin(zips.how='right',predicate= g25/c geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate= g25/c geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate= g25/c geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate= g25/c geopandas spatial join, within z_within z_within_c = zips.sjoin(county,how='left',predicate= g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate= g25/c geopandas spatial join, within apper to a geodatabase const.to_file("conus.gpkg",layer="states") g22/c geopandas within a payer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/c geopandas within a payer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/c geopandas within a payer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/c geopandas conus garder garder garder garde	geopandas	drawing a heatmap	near_wv.plot("mil",cmap='Blues',legend=True,ax=ax)	g22/demo.py
geopandas importing the module import geopandas as gpd g21/c geopandas join to nearest object served_by = centroids.sjoin_nearest(geo.how='left',valida g27/c geopandas obtaining coordinates print('Number of points:', len(wv_geo.exterior.coords) g22/c geopandas overlaying a layer using union slices = zips.overlay(county,howv='union',keep_geom_type g26/c geopandas plotting a boundary syr.boundary.plot(color='gray',linewidth=1,ax=ax1) g21/c geopandas plotting a boundary syr.boundary.plot(color='gray',linewidth=1,ax=ax1) g21/c geopandas reading a file syr egol-add_file("tl_2016_36_place-syracuse.zip") g21/c geopandas reading data in WKT format coords = gpd.GeoSeries.from_wkt(bigl'Georefrence']) g22/c geopandas setting the color of a plot county.plot(color='tan',ax=ax1) g24/c geopandas setting transparency via alpha near_clip.plot(alpha=0.25,ax=ax1) g24/c geopandas spatial join, contains co-contains_z = county.sjoin(county,how='eff',predicate= g25/c geopandas spatial join, torcses i_crosses_z = inter.sjoin(zips.how='right',predicate= g25/c geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate= g25/c geopandas spatial join, within z_within z_within_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, within apper to a geodatabase const.to_file("conus.gpkg",layer="states") g22/c geopandas within apper to a geodatabase const.to_file("conus.gpkg",layer="states") g22/c geopandas within apper to a geodatabase const.to_file("conus.gpkg",layer="states") g22/c geopandas spatial join, within axes, adding a vertical line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/c matplotlib axe	geopandas	extracting geometry from a geodataframe	wv_geo = wv['geometry']	g22/demo.py
geopandas merging data onto a geodataframe conus = conus.merge(trim.on="STATEFP",how="left",valida g22/c geopandas obtaining coordinates print ("Number of points:", len(ww_geo.exterior.coords) g22/c geopandas overlaying a layer using union slices = zips. soverlay(county,how='union', keep_geom_type g26/c geopandas plot with categorical coloring sel.plot("NAME",cmap="Dark2",ax=ax1) g22/c geopandas plotting a boundary syr.boundary.plot(color="gray",linewidth=1,ax=ax1) g21/c geopandas project a layer county = county.to_crs(eps==utm18n) g24/c geopandas reading a file syr=gpd.read_file("tl_2016_36_place-syracuse.zip") g21/c geopandas reading data in WKT format coords = gpd.feoSeoferis.from_wkt(big("Georeference")) g27/c geopandas setting the color of a plot county.plot(color="tan",ax=ax1) g24/c geopandas setting transparency via alpha near_clip.plot(alpha=0.25,ax=ax1) g24/c geopandas spatial join, cortains c_contains_z = county.sjoin(zips,how="right",predicate= g25/c geopandas spatial join, intersects z_incrosses_z = inter.sjoin(zips,how="right",predicate= g25/c geopandas spatial join, touches spatial join, within z_coverlaps_c = zips.sjoin(county,how=left",predicate= g25/c geopandas spatial join, within z_coverlaps_c = zips.sjoin(county,how=left",predicate= g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how=left",predicate= g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how=left",predicate= g25/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/c geopandas axes, adding a vertical line ax21.axhline(medians['inc'], c='r', ls='-', lw=1) g12/c matplotlib axes, labeling the X axis ax2.axvline(medians['inc'], c='r', ls='-', lw=1) g12/c matplotlib axes, labeling the Y axis ax1.axis('off') geo.plot(color='blue',markersize=1,ax=ax1) g27/c changing marker size	geopandas			g21/demo.py
geopandas overlaying a layer using union slices = zips.overlay(county,how='indri,ypedicate=' g22/c geopandas overlaying a layer using union slices = zips.overlay(county,how='indri,ypedicate=' g25/c geopandas plot with categorical coloring sel.plot('NAME',cmap='Dark2',ax=ax1) g21/c geopandas project a layer county = county,to_crs(epsg=utm18n) g24/c geopandas reading a file syr = gpd.read_file("cb_2019_as_exyracuse.zip") g21/c geopandas reading a shapefile states = gpd.read_file("cb_2019_us_state_500k.zip") g22/c geopandas reading data in WKT format coords = gpd.GeoSeries.from_wkt(big!' Georeference!) g27/c geopandas setting the color of a plot county.plot(color='tan',ax=ax1) g24/c geopandas setting transparency via alpha near_clip.plot(alpha=0.25,ax=ax1) g24/c geopandas spatial join, contains	geopandas	join to nearest object	served_by = centroids.sjoin_nearest(geo,how='left',dist	g27/demo.py
geopandas overlaying a layer using union slices = zips.overlay(county,how='indri,ypedicate=' g22/c geopandas overlaying a layer using union slices = zips.overlay(county,how='indri,ypedicate=' g25/c geopandas plot with categorical coloring sel.plot('NAME',cmap='Dark2',ax=ax1) g21/c geopandas project a layer county = county,to_crs(epsg=utm18n) g24/c geopandas reading a file syr = gpd.read_file("cb_2019_as_exyracuse.zip") g21/c geopandas reading a shapefile states = gpd.read_file("cb_2019_us_state_500k.zip") g22/c geopandas reading data in WKT format coords = gpd.GeoSeries.from_wkt(big!' Georeference!) g27/c geopandas setting the color of a plot county.plot(color='tan',ax=ax1) g24/c geopandas setting transparency via alpha near_clip.plot(alpha=0.25,ax=ax1) g24/c geopandas spatial join, contains	geopandas	merging data onto a geodataframe	conus = conus.merge(trim,on='STATEFP',how='left',valida	g22/demo.py
geopandas plot with categorical coloring sel.plot('NAME',cmap='Dark2', xa=xa1) g22/c geopandas plot with categorical coloring sel.plot('NAME',cmap='Dark2', xa=xa1) g22/c geopandas plot with categorical coloring sel.plot('NAME',cmap='Dark2', xa=xa1) g21/c geopandas project a layer county = county.to_crs(epsg=utm18n) g21/c geopandas reading a file syr = gpd.read_file("tl_2016_36_place-syracuse.zip") g21/c geopandas reading data in WKT format coords = gpd.read_file("tl_2016_36_place-syracuse.zip") g22/c geopandas reading data in WKT format coords = gpd.GeoSeries.from_wkt(big' Georeference') g27/c geopandas setting the color of a plot county.plot(color='tan',ax=ax1) g24/c geopandas setting transparency via alpha near_clip.plot(alpha=0.05,ax=ax1) g24/c geopandas spatial join, crosses i_crosses_z = inter.sjoin(zips,how='right',predicate= g25/c geopandas spatial join, corsses i_crosses_z = inter.sjoin(county,how='left',predicate= g25/c geopandas spatial join, overlaps z_overlaps_c = zips.sjoin(county,how='left',predicate= g25/c geopandas spatial join, vorlaps z_overlaps_c = zips.sjoin(county,how='left',predicate= g25/c geopandas spatial join, vorlaps z_overlaps_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join potental line average to a geodatabase conus.to.gle("conus.gpkg",layer="states") g25/c geopandas writing a layer to a geodatabase conus.to.gle("conus.gpkg",layer="states") g25/c geopandas spatial poin, vorlaps setting if rows touch a geometry touches_w = conus.touches(w_geo) g22/c geopandas writing a layer to a geodatabase conus.to.gle("conus.gpkg",layer="states") g25/c geopandas gatericate in average pating pating avertical line average pating avertical line average pating avertical line average pating avertical line average pating average pating average pating pating fire pating pating p	geopandas	obtaining coordinates		g22/demo.py
geopandas plotting a boundary syr.boundary.plot(color='gray',linewidth=1,ax=ax1) g21/c geopandas project a layer county = county.to_crs(epsg=utm18n) g24/c geopandas reading a file syr = gpd.read_file("tl_2016_36_place-syracuse.zip") g221/c geopandas reading a shapefile states = gpd.read_file("tl_2016_36_place-syracuse.zip") g221/c geopandas reading data in WKT format coords = gpd.GeoSeries.from_wkt(big['Georeference']) g27/c geopandas setting the color of a plot county.plot(color='tan',ax=ax1) g24/c geopandas setting transparency via alpha near_clip.plot(alpha=0.25,ax=ax1) g24/c geopandas spatial join, contains c_contains_z = county.sjoin(zips,how='right',predicate= g25/c geopandas spatial join, crosses i_crosses_z = inter.sjoin(zips,how='right',predicate= g25/c geopandas spatial join, overlaps z_overlaps_c = zips.sjoin(county,how='left',predicate= g25/c geopandas spatial join, voerlaps z_overlaps_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas testing if rows touch a geometry touches_ww = conus.touches(wv_geo) g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/c gaopandas yaexial join axes, adding a vertical line ax21.axvline(medians['etr'], c='r', ls='-', lw=1) g12/c matplotlib axes, adding a vertical line ax21.axvline(medians['etr'], c='r', ls='-', lw=1) g12/c matplotlib axes, labeling the X axis ax1.set_ylabel('Millions') g11/c matplotlib axes, turning off the label ax.set_ylabel('Millions') g26/c matplotlib axis, turning off the label ax.set_ylabel('None) g26/c matplotlib axis, turning off the label ax.set_ylabel('None) g26/c g26/c matplotlib calcal calcal calcal calcal ca	geopandas	overlaying a layer using union	slices = zips.overlay(county,how='union',keep_geom_type	g26/demo.py
geopandas project a layer county = county.to_crs(epsg=utm18n) geopandas reading a file syr = gpd.read_file("tt_2016_36_place-syracuse.zip") geopandas reading a shapefile states = gpd.read_file("tb_2019_us_state_500k.zip") geopandas reading data in WKT format coords = gpd.GeoSeries.from_wkt(bigl'Georeference']) geopandas setting the color of a plot county.plot(color='tan',ax=ax1) geopandas setting transparency via alpha near_clip.plot(alpha=0.25,ax=ax1) geopandas spatial join, contains c_contains_z = county.sjoin(zips,how='right',predicate= g25/c geopandas spatial join, cortains geopandas spatial join, intersects geopandas spatial join, oreses i_crosses_z = inter.sjoin(zips,how='right',predicate= g25/c geopandas spatial join, overlaps spatial join, overlaps z_overlaps_c = zips.sjoin(county,how='left',predicate= g25/c geopandas spatial join, touches geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate='vio g25/c geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") geopandas axes, adding a horizontal line ax21.axhline(medians['tirc'], c='r', ls='-', lw=1) matplotlib axes, labeling the X axis ax2.set_xlabel('Millions') matplotlib axes, labeling the Y axis ax1.axis('off') matplotlib axis, turning off ax1.axis('off') matplotlib axis, turning off ax1.axis('off') matplotlib axis, turning off ax1.axis('off') geoplot(color='blue',markersize=1,ax=ax1)	geopandas	plot with categorical coloring	sel.plot('NAME',cmap='Dark2',ax=ax1)	g22/demo.py
geopandas project a layer county = county.to_crs(epsg=utm18n) g24/c geopandas reading a file syr = gpd.read_file("tt_2016_36_place-syracuse.zip") g21/c geopandas reading a shapefile states = gpd.read_file("tb_2019_us_state_500k.zip") g22/c geopandas reading data in WKT format coords = gpd.GeoSeries.from_wkt(bigl'Georeference']) g27/c geopandas setting the color of a plot county.plot(color='tan',ax=ax1) g24/c geopandas setting transparency via alpha near_clip.plot(alpha=0.25,ax=ax1) g24/c geopandas spatial join, contains c_contains_z = county.sjoin(zips,how='right',predicate= g25/c geopandas spatial join, cortains c_contains_z = county.sjoin(zips,how='right',predicate= g25/c geopandas spatial join, intersects z_inter.sjoin(zips,how='right',predicate= g25/c geopandas spatial join, overlaps z_overlaps_c = zips.sjoin(county,how='left',predicate= g25/c geopandas spatial join, touches z_overlaps_c = zips.sjoin(county,how='left',predicate= g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate=-'vio g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate=-'vio g25/c geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/c geopandas patial join axes, adding a vertical line ax21.axhline(medians['inc'], c='r', ls='-', lw=1) g12/c matplotlib axes, labeling the X axis ax2.set_xlabel('Millions') g11/c matplotlib axes, labeling the Y axis ax1.set_ylabel('Millions') g11/c matplotlib axes, turning off ax1.axis('off') geo.plot(color='blue',markersize=1,ax=ax1) g26/c matplotlib changing marker size geo.plot(color='blue',markersize=1,ax=ax1)	geopandas	plotting a boundary	syr.boundary.plot(color='gray',linewidth=1,ax=ax1)	g21/demo.py
geopandas reading a shapefile states = gpd.read_file("cb_2019_us_state_500k.zip") g22/c geopandas reading data in WKT format coords = gpd.GeoSeries.from_wkt(big['Georeference']) g27/c geopandas setting the color of a plot county.plot(color='tan',ax=ax1) g24/c geopandas setting transparency via alpha near_clip.plot(alpha=0.25,ax=ax1) g24/c geopandas spatial join, contains c_contains_c county.sjoin(zips,how='right',predicate= g25/c geopandas spatial join, intersects i_crosses_z = inter.sjoin(zips,how='right',predicate='c g25/c geopandas spatial join, overlaps z_overlaps_c = zips.sjoin(county,how='left',predicate='c g25/c geopandas spatial join, touches z_overlaps_c = zips.sjoin(county,how='left',predicate='c g25/c geopandas spatial join, vouches z_touch_c = zips.sjoin(county,how='left',predicate='county.g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate='tou g25/c geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/c geopandas conus.to_file("conus.gpkg",layer="states") g25/c geopandas gaxe, aldeling the X axis ax21.axxlline(medians['etr'], c='r', ls='-', lw=1) g12/c axes, labeling the X axis ax2.set_xlabel('Millions') axes, labeling the Y axis ax1.set_ylabel('Millions') axis, turning off ax1.axis('off') g26/c matplotlib axis, turning off ax1.axis('off') g26/c matplotlib changing marker size geo.plot(color='blue',markersize=1,ax=ax1)	geopandas	project a layer		g24/demo.py
geopandas geopandas setting the color of a plot county, plot(color='tan', ax=ax1) g24/c geopandas setting transparency via alpha near_clip.plot(alpha=0.25, ax=ax1) g24/c geopandas spatial join, contains c_contains_z = county, sjoin(zips, how='right', predicate= g25/c geopandas spatial join, intersects i_crosses_z = inter.sjoin(zips, how='right', predicate= g25/c geopandas spatial join, intersects z_intersect_c = zips.sjoin(county, how='left', predicate= g25/c geopandas spatial join, verlaps z_overlaps_c = zips.sjoin(county, how='left', predicate= g25/c geopandas spatial join, verlaps z_overlaps_c = zips.sjoin(county, how='left', predicate=' g25/c geopandas spatial join, within z_within_c = zips.sjoin(county, how='left', predicate=' g25/c geopandas spatial join, within z_within_c = zips.sjoin(county, how='left', predicate=' g25/c geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg", layer="states") g22/c geopandas using to print an object nicely print(json.dumps(list1, indent=4)) g05/c goopandas axes, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/c goopandas goopandas axes, adding a vertical line ax21.axvline(medians['etr'], c='r', ls='-', lw=1) g12/c goopandas g	geopandas	reading a file	syr = gpd.read_file("tl_2016_36_place-syracuse.zip")	g21/demo.py
geopandas geopandas setting the color of a plot county, plot(color='tan', ax=ax1) g24/c geopandas setting transparency via alpha near_clip.plot(alpha=0.25, ax=ax1) g24/c geopandas spatial join, contains c_contains_z = county, sjoin(zips, how='right', predicate= g25/c geopandas spatial join, intersects i_crosses_z = inter.sjoin(zips, how='right', predicate= g25/c geopandas spatial join, intersects z_intersect_c = zips.sjoin(county, how='left', predicate= g25/c geopandas spatial join, verlaps z_overlaps_c = zips.sjoin(county, how='left', predicate= g25/c geopandas spatial join, verlaps z_overlaps_c = zips.sjoin(county, how='left', predicate=' g25/c geopandas spatial join, within z_within_c = zips.sjoin(county, how='left', predicate=' g25/c geopandas spatial join, within z_within_c = zips.sjoin(county, how='left', predicate=' g25/c geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg", layer="states") g22/c geopandas using to print an object nicely print(json.dumps(list1, indent=4)) g05/c goopandas axes, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/c goopandas goopandas axes, adding a vertical line ax21.axvline(medians['etr'], c='r', ls='-', lw=1) g12/c goopandas g	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(alpha=0.25,ax=ax1) g24/c geopandas spatial join, contains c_contains_z = county.sjoin(zips,how='right',predicate= g25/c geopandas spatial join, intersects i_crosses_z = inter.sjoin(zips,how='right',predicate='c g25/c geopandas spatial join, overlaps z_noverlaps_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/c geopandas writing a layer to a geodatabase conus.to_fle("conus.gpkg",layer="states") g25/c geopandas writing a layer to a geodatabase conus.to_fle("conus.gpkg",layer="states") g25/c g22/c geopandas writing a layer to a geodatabase ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/c gataplotlib axes, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/c gataplotlib axes, labeling the X axis ax2.set_xlabel('Millions') g11/c gataplotlib axes, labeling the Y axis gataplotlib axes, turning off the label ax.set_ylabel(None) g13/c gataplotlib axis, turning off sax1.axis('off') g26/c gataplotlib changing marker size geo.plot(color='blue',markersize=1,ax=ax1)	geopandas	reading data in WKT format		g27/demo.py
geopandas spatial join, contains c_contains_z = county.sjoin(zips,how='right',predicate= g25/c geopandas spatial join, crosses i_crosses_z = inter.sjoin(zips,how='right',predicate='c g25/c geopandas spatial join, intersects z_intersect_c = zips.sjoin(county,how='left',predicate= g25/c geopandas spatial join, touches z_overlaps_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate='wi g25/c geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/c json importing the module import json g05/c json using to print an object nicely print(json.dumps(list1,indent=4)) g05/c matplotlib axes, adding a horizontal line ax21.axhline(medians['ter'], c='r', ls='-', lw=1) g12/c matplotlib axes, labeling the X axis ax21.axvline(medians['inc'], c='r', ls='-', lw=1) g12/c matplotlib axes, labeling the X axis ax2.set_xlabel('Millions') g11/c matplotlib axes, turning off the label ax.set_ylabel(Mone) ax1.axet_ylabel(Mone) ax1.axet_ylabel('Millions') ax1.axet_ylabel('Mone) ax1.axet_ylabel('Mone) ax1.axet_ylabel('Mone) ax1.axet_ylabel('Millions') ax1.axet_ylabel('Millions'	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/d geopandas spatial join, intersects z_intersect_c = zips.sjoin(county,how='left',predicate=' g25/d geopandas spatial join, overlaps z_overlaps_c = zips.sjoin(county,how='left',predicate=' g25/d geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate='tou g25/d geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate='wi g25/d geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/d geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/d geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/d	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/c geopandas spatial join, overlaps z_overlaps_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate='tou g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate='wi g25/c geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g25/c json importing the module import json g05/c json using to print an object nicely print(json.dumps(list1,indent=4)) g05/c matplotlib axes, adding a horizontal line ax21.axyline(medians['etr'], c='r', ls='-', lw=1) g12/c matplotlib axes, labeling the X axis ax2.sax_xlabel('Millions') g11/c matplotlib axes, labeling the Y axis ax1.set_ylabel('Mone) g13/c matplotlib axis, turning off ax1.axis('off') g26/c matplotlib changing marker size geo.plot(color='blue',markersize=1,ax=ax1) g25/c g26/c g27/c	geopandas	spatial join, contains	<pre>c_contains_z = county.sjoin(zips,how='right',predicate=</pre>	g25/demo.py
geopandas spatial join, overlaps z_overlaps_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate=' g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate='wi g25/c geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/c geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/c geopandas vriting a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/c geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/c geopandas vriting a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/c geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/c geopandas vriting a layer to a geodatabase conus.touches(wv_geo) g22/c geopandas vriting a layer to a geodatabase conus.touches(wv_geo) g22/c geopandas vriting a layer to a geodatabase conus.touches(wv_geo) g22/c geopandas vriting a layer to a geodatabase conus.touches(wv_geo) g22/c geopandas vriting a layer to a geodatabase conus.touches(wv_geo) g22/c geopandas vriting a layer to a geodatabase conus.touches(wv_geo) g22/c geopandas vriting a layer to a geodatabase conus.touches(wv_geo) g22/c geopandas vriting a layer to a geodatabase conus.touches(wv_geo) g22/c geopandas vriting a layer to a geodatabase conus.touches(wv_geo) g22/c geopandas vriting a layer to a geodatabase conus.touches(wv_geo) g22/c geopandas vriting a layer to a geodatabase conus.touches(wv_geo) g22	geopandas	spatial join, crosses	i_crosses_z = inter.sjoin(zips,how='right',predicate='c	g25/demo.py
geopandas spatial join, touches z_touch_c = zips.sjoin(county,how='left',predicate='tou g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate='wi g25/c geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/c geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/c json importing the module import json g05/c json using to print an object nicely print(json.dumps(list1,indent=4)) g05/c matplotlib axes, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/c matplotlib axes, labeling the X axis ax2.set_xlabel('Millions') g11/c matplotlib axes, labeling the Y axis ax1.set_ylabel('Millions') g11/c matplotlib axes, turning off ax1.axis('off') g26/c matplotlib axis, turning off changing marker size geo.plot(color='blue',markersize=1,ax=ax1) g25/c geopandas spatial join, touches z_tou g25/c geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate='tou g25/c geopandas desting in county,how='left',predicate='tou g25/c geopandas vertical join, within z_within_c = zips.sjoin(county,how='left',predicate='tou g25/c geopandas vertical join, within z_within_c = zips.sjoin(county,how='left',predicate='tou g25/c geopandas vertical join, within z_within_c = zips.sjoin(county,how='left',predicate='tou g25/c geopandas vertical join, within z_within_c = zips.sjoin(county,how='left',predicate='vi g25/c geopandas vertical join, within z_within_c = zips.sjoin(county,how='left',predicate='vio g25/c geopandas vertical join, within z_within_c = zips.sjoin(conty,how='left',predicate='vio g25/c geopandas vertical join, within z_within_c = zips.sjoin(conty,home) = g25/c geop	geopandas	spatial join, intersects	$z_{intersect_c} = z_{ips.sjoin(county,how='left',predicate=}$	g25/demo.py
geopandas spatial join, within z_within_c = zips.sjoin(county,how='left',predicate='wi g25/d geopandas testing if rows touch a geometry touches_wv = conus.touches(wv_geo) g22/d geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/d geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") g22/d g2	geopandas	spatial join, overlaps	$z_overlaps_c = zips.sjoin(county,how='left',predicate='$	g25/demo.py
geopandas testing if rows touch a geometry geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") json importing the module import json g05/c g22/c goopandas using to print an object nicely print(json.dumps(list1,indent=4)) matplotlib axes, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/c goopandas g22/c goopandas goopandas goopandas g22/c goopandas goopandas goopandas g22/c goopandas goopandas g22/c goopandas goopandas g22/c g22/c goopandas g22/c g22/c goopandas g22/c goopandas g22/c g22/c goopandas g22/c g22/c goopandas g22/c g22/c g22/c goopandas g22/c g	geopandas	spatial join, touches	$z_touch_c = zips.sjoin(county,how='left',predicate='tou$	g25/demo.py
geopandas writing a layer to a geodatabase conus.to_file("conus.gpkg",layer="states") geopandas writing a layer to a geodatabase conus.gpkg",layer="states") geopandas writing a layer to a geodatabase conus.gpkg",layer="states") geopandas valuer="states") geopandas vertical line import json geopandas vertical line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) geopandas geophot(color='blue',markersize=1,ax=ax1) geopandas geopandas geopandas geophot(color='blue',markersize=1,ax=ax1)	geopandas	spatial join, within	<pre>z_within_c = zips.sjoin(county,how='left',predicate='wi</pre>	g25/demo.py
json importing the module import json g05/c gosho using to print an object nicely print(json.dumps(list1,indent=4)) matplotlib axes, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/c	geopandas	testing if rows touch a geometry	touches_wv = conus.touches(wv_geo)	g22/demo.py
json using to print an object nicely print(json.dumps(list1,indent=4)) matplotlib axes, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) ax21.axvline(medians['inc'], c	geopandas	writing a layer to a geodatabase	conus.to_file("conus.gpkg",layer="states")	g22/demo.py
matplotlib axes, adding a horizontal line ax21.axhline(medians['etr'], c='r', ls='-', lw=1) g12/d matplotlib axes, adding a vertical line ax21.axvline(medians['inc'], c='r', ls='-', lw=1) g12/d matplotlib axes, labeling the X axis ax2.set_xlabel('Millions') g11/d matplotlib axes, labeling the Y axis ax1.set_ylabel("Millions") g11/d ax.set_ylabel(None) g13/d matplotlib axis, turning off the label ax.set_ylabel(None) g13/d matplotlib changing marker size geo.plot(color='blue',markersize=1,ax=ax1) g27/d	json	importing the module	import json	g05/demo.py
matplotlibaxes, adding a vertical lineax21.axvline(medians['inc'], c='r', ls='-', lw=1)g12/dmatplotlibaxes, labeling the X axisax2.set_xlabel('Millions')g11/cmatplotlibaxes, labeling the Y axisax1.set_ylabel("Millions")g11/cmatplotlibaxes, turning off the labelax.set_ylabel(None)g13/cmatplotlibaxis, turning offax1.axis('off')g26/cmatplotlibchanging marker sizegeo.plot(color='blue',markersize=1,ax=ax1)g27/c	json	using to print an object nicely	<pre>print(json.dumps(list1,indent=4))</pre>	g05/demo.py
matplotlibaxes, labeling the X axisax2.set_xlabel('Millions')g11/cmatplotlibaxes, labeling the Y axisax1.set_ylabel("Millions")g11/cmatplotlibaxes, turning off the labelax.set_ylabel(None)g13/cmatplotlibaxis, turning offax1.axis('off')g26/cmatplotlibchanging marker sizegeo.plot(color='blue',markersize=1,ax=ax1)g27/c	matplotlib	axes, adding a horizontal line	ax21.axhline(medians['etr'], c='r', ls='-', lw=1)	g12/demo.py
matplotlibaxes, labeling the Y axisax1.set_ylabel("Millions")g11/cmatplotlibaxes, turning off the labelax.set_ylabel(None)g13/cmatplotlibaxis, turning offax1.axis('off')g26/cmatplotlibchanging marker sizegeo.plot(color='blue',markersize=1,ax=ax1)g27/c	matplotlib	axes, adding a vertical line	ax21.axvline(medians['inc'], c='r', ls='-', lw=1)	g12/demo.py
matplotlibaxes, turning off the labelax.set_ylabel(None)g13/cmatplotlibaxis, turning offax1.axis('off')g26/cmatplotlibchanging marker sizegeo.plot(color='blue',markersize=1,ax=ax1)g27/c	matplotlib	axes, labeling the X axis	ax2.set_xlabel('Millions')	g11/demo.py
matplotlibaxes, turning off the labelax.set_ylabel(None)g13/cmatplotlibaxis, turning offax1.axis('off')g26/cmatplotlibchanging marker sizegeo.plot(color='blue',markersize=1,ax=ax1)g27/c	matplotlib	axes, labeling the Y axis	ax1.set_ylabel("Millions")	g11/demo.py
matplotlib changing marker size geo.plot(color='blue',markersize=1,ax=ax1) g27/c	matplotlib	axes, turning off the label	ax.set_ylabel(None)	g13/demo.py
matplotlib changing marker size geo.plot(color='blue',markersize=1,ax=ax1) g27/c	matplotlib	axis, turning off	ax1.axis('off')	g26/demo.py
	matplotlib		geo.plot(color='blue',markersize=1,ax=ax1)	g27/demo.py
matplotlib colors, xkcd palette syr.plot(color= xkcd:lightblue ,ax=ax1) g21/c	matplotlib	colors, xkcd palette	syr.plot(color='xkcd:lightblue',ax=ax1)	g21/demo.py
	matplotlib		fig2.suptitle('Pooled Data')	g12/demo.py
matplotlib figure, four panel grid fig3, $axs = plt.subplots(2,2,sharex=True,sharey=True)$ $g12/c$	matplotlib	figure, four panel grid	fig3, axs = plt.subplots(2,2,sharex = True,sharey = True)	g12/demo.py

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

Module	Description	Example	Script
pandas	dataframe, reading zipped pickle format	sample2 = pd.read_pickle('sample_pkl.zip')	g16/demo.py
pandas	dataframe, resetting the index	hourly = hourly.reset_index()	g17/demo.py
pandas	dataframe, right 1:1 merge	$join_r = parcels.merge(flood, how='right', on="TAX_ID",$	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' ")	${ m g10/demo.py}$
pandas	dataframe, set index keeping the column	$states = states.set_index('STUSPS',drop=False)$	g22/demo.py
pandas	dataframe, shape attribute	<pre>print('number of rows, columns:', conus.shape)</pre>	g22/demo.py
pandas	dataframe, sorting by a column	<pre>county = county.sort_values('pop')</pre>	${ m g10/demo.py}$
pandas	dataframe, sorting by index	$summary = summary.sort_index(ascending=False)$	g15/demo.py
pandas	dataframe, summing a boolean	<pre>print('\nduplicate parcels:', dups.sum())</pre>	g14/demo.py
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	<pre>print(county.xs('04',level='state'))</pre>	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	$ymd = pd.to_datetime(sample['TRANSACTION_DT'], format=$	${\sf g16/demo.py}$
pandas	datetime, extracting day attribute	recs['day'] = date.dt.day	${\sf g14/demo.py}$
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)	g16/demo.py
pandas	general, displaying all rows	pd.set_option('display.max_rows', None)	g09/demo.py
pandas	general, importing the module	import pandas as pd	g09/demo.py
pandas	general, using copy_on_write mode	$pd.options.mode.copy_on_write = True$	${\sf g16/demo.py}$
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	cumulative_inc = group_by_state['pop'].cumsum()	g10/demo.py
pandas	groupby, descriptive statistics	inc_stats = group_by_state['pop'].describe()	g10/demo.py
pandas	groupby, iterating over groups	for t,g in group_by_state:	g10/demo.py
pandas	groupby, median of each group	pop_med = group_by_state['pop'].median()	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()	g10/demo.py
, pandas	groupby, return number within group	seqnum = group_by_state.cumcount()	g10/demo.py
•	5		5 , 13

pandas pandas pandas pandas pandas	groupby, return rank within group groupby, select first records groupby, select largest values groupby, select last records groupby, size of each group groupby, sum of each group	<pre>rank_age = group_by_state['pop'].rank() first2 = group_by_state.head(2) largest = group_by_state['pop'].nlargest(2) last2 = group_by_state.tail(2) num_rows = group_by_state.size() state = county.groupby('state')['pop'].sum()</pre>	g10/demo.py g10/demo.py g10/demo.py g10/demo.py g10/demo.py g10/demo.py
pandas pandas pandas	groupby, select largest values groupby, select last records groupby, size of each group groupby, sum of each group	<pre>largest = group_by_state['pop'].nlargest(2) last2 = group_by_state.tail(2) num_rows = group_by_state.size()</pre>	g10/demo.py g10/demo.py g10/demo.py
pandas pandas	groupby, select last records groupby, size of each group groupby, sum of each group	last2 = group_by_state.tail(2) num_rows = group_by_state.size()	g10/demo.py g10/demo.py
pandas	groupby, size of each group groupby, sum of each group	last2 = group_by_state.tail(2) num_rows = group_by_state.size()	g10/demo.py
•	groupby, sum of each group		- ,
		state = county.groupby('state')['pop'].sum()	g10/demo ny
pandas			gro/ 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>	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	print(low_to_high.iloc[-5:])	${ m g09/demo.py}$
pandas	index, setting to a column	$states = raw_states.set_index('name')$	g09/demo.py
pandas	plotting, bar plot	reg_pop.plot.bar(title='Population',ax=ax1)	g11/demo.py
pandas	plotting, histogram	$hh_data['etr'].plot.hist(ax=ax1,bins=20,title='Distribu$	g12/demo.py
pandas	plotting, horizontal bar plot	$div_pop.plot.barh(title='Population',ax=ax2)$	g11/demo.py
pandas	plotting, scatter colored by 3rd var	$tidy_data.plot.scatter(ax=ax4,x=`Income',y=`ETR',c='typ$	g12/demo.py
pandas	plotting, scatter plot	$hh_data.plot.scatter(ax=ax21,x='inc',y='etr',title='ETR$	g12/demo.py
pandas	plotting, turning off legend	sel.plot.barh(x='Name',y='percent',ax=ax,legend=None)	g13/demo.py
pandas	reading, csv data	raw_states = pd.read_csv('state-data.csv')	g09/demo.py
pandas	reading, from an open file handle	${\sf gen_oswego} = {\sf pd.read_csv(fh1)}$	g15/demo.py
pandas	reading, setting index column	$state_data = pd.read_csv(`state-data.csv',index_col='na$	g11/demo.py
pandas	reading, using dtype dictionary	county = pd.read_csv('county_pop.csv',dtype=fips)	g10/demo.py
pandas	series, RE at start	$is_LD = trim['Number'].str.contains(r"1 2")$	g12/demo.py
pandas	series, applying a function to each element	name_clean = name_parts.apply(' '.join)	g23/demo.py
pandas	series, automatic alignment by index	$merged[`percent'] = 100 *merged[`pop']/div_pop$	g13/demo.py
pandas	series, combining via where()	mod['comb_units'] = unit_part.where(unit_part!=" , mo	g23/demo.py
pandas	series, contains RE or RE	$is_TT = trim['Days'].str.contains(r"Tu Th")$	g12/demo.py
pandas	series, contains a plain string	$has_AM = trim['Time'].str.contains("AM")$	g12/demo.py
pandas	series, contains an RE	$has_AMPM = trim['Time'].str.contains(``AM.*PM'')$	g12/demo.py
pandas	series, converting strings to title case	$fixname = subset_view['NAME'].str.title()$	g16/demo.py
pandas	series, converting to a list	<pre>print(name_data['State'].to_list())</pre>	g13/demo.py
pandas	series, converting to lower case	name = mod['name'].str.lower()	g23/demo.py

pandas pandas pandas	series, dropping rows using a list series, element-by-element or	conus = states.drop(not_conus)	g22/demo.py
pandas	series, element-by-element or		gzz/ demo.py
		is_either = is_ca is_tx	g16/demo.py
	series, filling missing values	mod['comb_units'] = mod['comb_units'].fillna('feet')	g23/demo.py
pandas	series, removing spaces	units = units.str.strip()	g23/demo.py
pandas	series, replacing values using a dictionary	units = units.replace(spellout)	g23/demo.py
pandas	series, retrieving an element	<pre>print("\nFlorida's population:", pop['Florida']/1e6)</pre>	g09/demo.py
pandas	series, sort in decending order	div_pop = div_pop.sort_values(ascending=False)	g11/demo.py
pandas	series, sorting by value	low_to_high = normed['med_pers_inc'].sort_values()	g09/demo.py
pandas	series, splitting strings on whitespace	name_parts = name.str.split()	g23/demo.py
pandas	series, splitting via RE	trim['Split'] = trim["Time"].str.split(r": - ")	g12/demo.py
pandas	series, splitting with expand	exp = trim["Time"].str.split(r": - ", expand=True)	g12/demo.py
pandas	series, summing	$reg_pop = by_reg['pop'].sum()/1e6$	g11/demo.py
pandas	series, unstacking	$tot_wide = tot_amt.unstack('PGI')$	g16/demo.py
pandas	series, using isin()	fixed = flood['TAX_ID'].isin(dup_rec['TAX_ID'])	g14/demo.py
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	print('url:', response.url)	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	cr = opt.newton(find_cube_root,xinit,maxiter=20,args=[y	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	hue='month',palette='deep',ax=ax1)	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	<pre>jg = sns.jointplot(data=bymo,x=1,y=7,kind='hex')</pre>	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	hue="month",palette='deep',split=True)	g17/demo.py

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