Feature	Numpy import numpy as np	R

Random number	# Uniform distribution
	np.random.rand(4)
	np.random.randint(1,5,2)
	np.random.randint(1,5,size=(2,3))
	np.random.uniform(1,5,(2,3))
	
	# seed
	np.random.seed(100)
	
	# Normal distribution
	np.random.randn(5)
		# Uniform distribution
		runif(4)
		sample(1:5, 2, replace = TRUE)
		matrix(sample(1:5, 6, replace = TRUE), 2, 3)
		matrix(runif(6, min = 1, max = 5), 2, 3)
		
		# seed
		set.seed(100)
		
		# Normal distribution
		rnorm(5)
		rnorm(5, mean = 10, sd = 2)

Create	# 1d
	np.array([1,2,3]) //or tuple
	# 2d
	np.array([[1,2,3],[4,5,6]])
	np.arange(1,7).reshape((2,3))
	
	#
	a=np.arange(1,10)
	np.arange(1,10,2.5)
	np.arange(1,11).reshape(2,5)
	np.linspace(1,10,6)
	np.zeros(4)
	np.zeros((2,3))
	np.ones((2,3))
	np.full((2,3),1)
	np.eye(3) #eye means I (identity)
		# 1d
		a <- c(1,2,3)
		# 2d
		a <- rbind(c(1,2,3), c(4,5,6))
		a <- matrix(1:6, nrow, ncol, byrow = T)
		a <- array(1:6, dim = c(2,3))
		#
		a=1:9
		seq(1,10,by=2.5)
		
		seq(1,10,length.out=6)
		rep(0,4)
		matrix(0,2,3)
		matrix(1,2,3)
		matrix(1,2,3)
		diag(3)

Update (add)	np.append(a,10) np.append(a,[20,30]) np.insert(a,1,40)	a <- c(a,10) a <- c(a,20,30) a <- append(a,40,after=1)

Update	a[0]=100 a[:2]=100 a.astype(np.float64)	a[1]<-100 a[1:2]<-100 as.double(a)
Query	a.ndim a.shape a.size a.dtype	length(dim(a)) dim(a) length(a) typeof(a)
Create boolean index	a>=60 (a>=60) & (a<=100) # a.any() a.all()	a >= 60 (a >= 60) & (a <= 100) # any(a) all(a)
Indexing	# 5 index types: # 1 normal index a[1] # second element/row # 2 slicing a[1:2] a[2:] a[:9:2] # 3 comma separated a[0,1] or a[:,2] # 4 boolean a[[True, False, True]] # 5 fancy a[[0,2]] ## pairwise index to select eles a[[0,2],[3,4]] ## scattered selection to select rows and cols a[np.ix_([0, 2], [3, 4])]	# # 1 a[2], a[2,] # 2 slicing a[2:3], a[2:3,] a[3:length(a)], a[3:nrow(a),] a[seq(1,length(a),2)], a[seq(1,nrow(a),by=2),] # 3 a[1,2], a[,3] # 4 a[c(TRUE, FALSE, TRUE)] # 5 a[c(1,3)] ## pairwise index a[cbind(c(1,3),c(4,5))] ## scattered selection a[c(1,3),c(4,5)]

Set operations	s={1,1,2,2,3} # np.unique([1,1,2,2,3]) vs, cnts=np.unique(a, return_counts=True) # b=np.isin(a, [3,4,6]) np.where(b) # find index a[b] # find values	unique(c(1,1,2,2,3)) # t <- table(c(1,1,2,2,3)) vs <- names(t) cnts <- as.vector(t) # b <- a %in% c(3,4,6) which(b) # find index a[b] # find values
Dictionary	d={‘a’:1, ‘b’:2, ‘c’:3} d[‘a’]	d=list(a=1,b=2,c=3) d[[‘a’]]

Statistics	a.min()
	a.max(axis=1)
	a.sum()
	a.mean()
	a.cumsum()
	a.argmax() # index of max ele
	a.argmax(axis=0) # column-wise
	a.argmax(axis=1) # row-wise
	
	# ufunc (Universal functions)
	np.sum(a)
	np.maximum(a,b)
	np.sum([a,b])
		min(a)
		apply(a,1,max)
		sum(a)
		mean(a), rowMeans(a)
		cumsum(as.vector(a))
		which.max(a) # index of max ele
		apply(a,2, which.max)
		apply(a,1, which.max)
		
		#
		sum(a)
		pmax(a,b)
		sum(c(a,b))

Operations	# +,-,*,/,**, % a**2 7%2	# +,-,*,/,^,%% a^2 7%%2

Conditional expression	x=[1,2,3,4] y=[6,7,8,9] condition=[True,False,True,True] np.where(condition, x,y) # np.where(a>10,10,0) np.where(a>10,10,a)	x<-c(1,2,3,4) y<-c(6,7,8,9) condition=[T,F,T,T] ifelse(condition, x, y) # ifelse(a > 10, 10, 0) ifelse(a > 10, 10, a)

if statement		condition <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
		
		l <- c()
		for (i in seq_along(condition)) {
		  if (condition[i]) {
		  l <- c(l, x[i])
		  } else {
		  l <- c(l, y[i])
		  }
		}

stacking/concatenation	**# 1d**
	a=np.array([1,1,1])
	b=np.array([2,2,2])
	## by row
	np.vstack((a,b)) # same as
	np.row_stack((a,b))
	
	## by column
	np.column_stack((a,b))
	
	## by concatenation
	np.hstack((a,b)) # same as
	np.concatenate((a,b))
	
	# 2d
	a=np.array([[1,1,1],[2,2,2]])
	b=np.array([[3,3,3],[4,4,4]])
	## by row
	np.vstack((a,b)) # same as
	np.concatenate((a,b),axis=0)
	
	## by column
	np.hstack([a,b]) # same as
	np.concatenate((a,b),axis=1)
		# 1d
		a <- c(1,1,1)
		b <- c(2,2,2)
		
		## by row
		rbind(a, b)
		
		## by column
		cbind(a, b)
		
		## by concatenation
		c(a, b)
		
		# 2d
		a <- matrix(c(1,1,1,2,2,2), nrow=2,
		byrow=TRUE)
		b <- matrix(c(3,3,3,4,4,4), nrow=2,
		byrow=TRUE)
		
		## by row
		rbind(a, b)
		
		## by column
		cbind(a, b)

split	# equal split (horizontal)
	np.hsplit(a,2)
	# or if array exact divisibility
	np.split(a,2,axis=1)
	
	# equal split (vertical)
	np.vsplit(a,2)
	# or if array exact divisibility
	np.split(a,2,axis=0)
	
	# not equal split
	np.array_split(a,3,axis=1)
		#
		p1 <- a[, 1:floor(ncol(a)/2)]
		p2 <- a[, (floor(ncol(a)/2)+1):ncol(a)]
		
		#
		p1 <- a[1:(nrow(a)/2),]
		p2 <- a[(nrow(a)/2+1):nrow(a),]
		
		#
		cols <- split(1:ncol(a), cut(1:ncol(a), 3,
		labels=FALSE))
		parts <- lapply(cols, function(idx) a[, idx,
		drop=FALSE])

Feature		Matplotlib import matplotlib.pyplot as plt	R base graphics + plotrix	R grid graphics (ggplot)
Show plot		plt.show() # need for pyCharm		
Line	simple line	x=[1,2,3,4,5] y=[5,7,5,8,15] plt.plot(x,y)	x=c(1,2,3,4,5) y=c(5,7,5,8,15) plot(x,y,type='l')	
	label	plt.plot(x,y,label='y')	no need	

	linewidth	plt.plot(x,y,linewidth=3)	plot(x,y,type='l',lwd=3)
			# if plot second line
			lines(x, y1, type="l")	

	marker, color and line style	plt.plot(x,y,'or--')	plot(x, y,
			type="b", # both points and lines
			pch=1, # circle marker
			col="red", # red color
			lty=2) # dashed line	

	legend	plt.legend()
		
		#
		plt.legend(loc=[1,0.02])
		# loc='bottom left'
			legend("topleft",
			legend=c("y", "y1"),
			
			col=c("blue","red"),
			lty=c(1,2))
			
			# R base not support 'loc'
			legend(x,y,…)	

	grid	plt.grid(True)	grid()	

style scheme	plt.style.available plt.style.use('fivethirtyeight')	No style scheme	
title	plt.title('rel x and y')	plot(x,y,main="rel x and y")	
axis labels	plt.xlable('X') plt.ylable('Y')	plot(x,y,xlab='X',ylab='Y')	
scale	plt.xscale('log') plt.yscale('log')	plot(x, y, log = "xy")	
Fill area	plt.plot(x, y) plt.fill_between(x, y) **# same as** plt.fill_between(x, y, y2=0) # **alpha** plt.fill_between(x, y, alpha=0.5) **# for each condition, set different colors** plt.plot(x, y) y2=y.mean() plt.fill_between(x, y, y2, where=(y>y2), interpolate=True) plt.fill_between(x, y, y2, where=(y<=y2), interpolate=True)	plot(x, y, type = "l") polygon(c(x, rev(x)), c(y, rep(0, length(y))), col = "blue") # plot(x, y, type = "l") polygon(c(x, rev(x)), c(y, rep(y_mean, length(y))), col = "blue") # for each condition, set different colors No good support	
step-like lines	plt.plot(x, drawstyle='steps-post')		

Bar	simple bar	x=[10,20,30,40,50] y1=[5,7,5,8,20] plt.bar(x,y1)	x <- c(10,20,30,40,50) y1 <- c(5,7,5,8,20) barplot(names.arg=x,y1)	
	width	plt.bar(x,y1,width=3)		
	label	plt.bar(x,y1,label='y1')		
	multiple bars	x=np.array([10,20,30,40 ,50]) y1=[5,7,5,8,20] y2=[6,7,9,3,6] width=3 plt.bar(x-w/ 2,y1,width=w) plt.bar(x+w/ 2,y2,width=w)	x <- c(10,20,30,40,50) y1 <- c(5,7,5,8,20) y2 <- c(6,7,9,3,6) mat <- rbind(y1, y2) barplot(mat, names.arg=x, beside=TRUE, col=c("skyblue","orange "), legend.text=c("y1","y2"))	
	horizontal bar	w=3 plt.barh(x,y1,height=w)	barplot(…, horiz=TRUE, …)	

	tick locations and labels	x=[10,20,30,40,60]
		y1=[5,7,5,8,20]
		
		height=0.4
		
		# convert x to indexes
		xi=np.arange(len(x))
		
		# yticks
		plt.yticks(ticks=xi,
		labels=x)		

	limits	plt.xlim(0,12) # set
		plt.xlim() # get
			plot(x, y, xlim = c(0, 12))
			# if plot already
			xlim(c(0, 12))
			# get
			par('xaxp') # X-AXis Parameters	

	text	plt.text(…) # see ax.text for syntax		
	annotate	plt.annotate(…) # see ax.annotate for syntax		
Pie	simple pie	slices=[60,20,10] labels=['60','20','10'] plt.pie(slices, labels=labels)	slices <- c(60,20,10) labels <- c("60","20","10") pie(slices, labels=labels)	
	wedge properties	plt.pie(…,wedgeprops={'edgecolor':'white'})	pie(…, border="white")	
	slice colors	colors=['#F51720','#FA26A0','#F8D210'] plt.pie(slices, labels=labels, colors=colors)	colors=c('#F51720','#FA26A0','#F8D210') pie(slices, labels=labels, col=colors)	
	explode	explode=[0,0,0.5] # the first two slices stay plt.pie(…, explode=explode)	# Not convenient for 2D, have to use plotrix for 3D and only explode all slices library(plotrix) pie3D(slices, labels=lbls, explode=0.2, col=colors)	

	shadow	plt.pie(…, shadow=True)		
	startangle	plt.pie(…, startangle=20)		
	auto display the percentage value on each wedge (slice)	plt.pie(…, autopct='%.2f%%')		
Stack plot	simple stack plot	days=[1,2,3,4,5] task1=[7,6,5,3,0] # 8 hours task2=[1,1,2,3,4] task3=[0,1,1,2,4] plt.stackplot(days,task1, task2, task3)	# use ggplot	
	colors	colors=['#F51720','#FA26A0','#F8D210'] plt.stackplot(…,colors=colors)		
	labels	labels=['t1','t2','t3'] plt.stackplot(…,labels=labels)		
Histogram	simple histogram	n=np.random.randint(1,10,20) plt.hist(n)	n <- sample(1:9, 20, replace=TRUE) hist(n)	
	bins	bins=[1,2,3,4,5,6,7,8,9] plt.hist(n,bins)	bins=c(1,2,3,4,5,6,7,8,9) hist(n, breaks = bins)	
	edge color	plt.hist(n,edgecolor='red')	hist(…, border='red')	
	scale: log	plt.hist(…,log='True')		

Vertical line		plt.axvline(val, color='red', label='Median age’,linewidth=3)
		
		# (x,y1) to (x,y2)
		ax.vlines(x, y1, y2, colors=‘red’)
		
		# or just plot a line (x=3)
		plt.plot([x]∗len(y), y, color="red")
			abline(v = val, col = "red", lwd = 3)
			
			legend("topright", legend="Median age", col="red", lwd=3)
			
			# plot a line
			x <- rep(3, length(y))
			plot(x, y, type="l", col=“red")
			
			#
			lines(rep(3, 100), seq(0, 25, length.out=100), col=“red")	

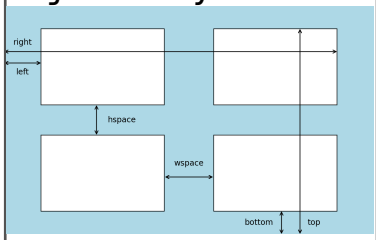
Horizontal line		plt.axhline(y=0, color=“red")
		
		# or
		plt.plot(x, 0∗x)
			abline(h=0, col=“blue")
			
			# or
			lines(x, 0∗x, col="red")	

Scatter plot	simple scatter	x=np.random.randint(0,10,10) y=np.random.randint(0,10,10) plt.scatter(x,y)	x <- sample(0:9, 10, replace=TRUE) y <- sample(0:9, 10, replace=TRUE) plot(x, y, pch=19)	
	size	plt.scatter(…, s=600) # sizes=np.random.randint(100,300,10) plt.scatter(…, s=sizes)	plot(…, cex=5) # 5 times # sizes=sizes <- sample(100:300, 10, replace=TRUE) plot(…, cex=sizes/100)	

	edgecolor, alpha, color	plt.scatter(…, edgecolor=‘black’,alpha=0.8,c=‘red’) # colors=np.arange(10,20) plt.scatter(…,c=colors)		
	linewidth	plt.scatter(…,linewidth=3)		
	marker	plt.scatter(…,marker=‘X’)		
	color bar	plt.scatter(…, c=colors, cmap=‘Reds') plt.colorbar(label='degrees')	# not support	

Point		plt.scatter(4, 5, color=“red”)
		# or
		plt.plot(4, 5, "ro")
			plot(4, 5, col="red", pch=19, cex=2)
			
			# if already plot, add point
			points(4, 5, col="red", pch=19, cex=2)	

Subplot		fig=plt.figure() ax1=fig.add_subplot(2,2,1) ax2=fig.add_subplot(222) a=[1,2,3,4] ax1.plot(a,'go–‘) # or fig, axes=plt.subplots(2,2) ax1=axes[0,0] ax1.plot(a,'go–')	arr <- c(1,2,3,4) par(mfrow = c(2,2)) # mfrow: multi-figure, row-wise # 1st subplot plot(arr) # 2nd subplot plot(arr)	
	Add padding	ax.margins(0.1, 0.1)	# auto padding	

	Adjust layout
	 
	
		# auto adjust
		plt.tight_layout()
		
		# value 0 to 1.0
		(fraction of fig width
		or height)
		fig.subplots_adjust(left=0.1,
		bottom=0.5,right=0.2,top=1) # value 0.0 to 1.0
		
		# value 0 to 1.0
		(fraction of subplot
		width or height)
		fig.subplots_adjust(wspace=0.5,hspace=0.2)
			# auto adjust
			
			# 5 lines for bottom, 4
			lines for left, 3 lines
			for top, and 2 lines
			for right
			
			par(mar = c(5, 4, 3,
			2))	

	tick locations and labels for an axis	ax.set_xticks([0, 500,1000]) ax.set_xticklabels(list('abc'),rotation=45, fontsize='x-large')		
	axis title	ax.set_title('an example')		
	axis label	ax.set_xlabel('steps')		

	or use properties to set tick, ticklabel, title and label	properties={
		'title': 'an example',
		'xlabel':'steps',
		'xticks': [0,500,1000],
		
		‘xticklabels’:list(‘abc')
		}
		ax.set(**properties)
		ax.tick_params(axis='x', labelrotation=45, labelsize=‘x-large')		

	limits (scale)	ax.set_xlim([0,12]) ax.get_xlim()		
	size	fig.set_size_inches(10, 5) # or fig, ax = plt.subplots(figsize=(10,5)) # fig.get_size_inches()		

	set axis	plt.axis([xmin, xmax, ymin, ymax]) → manually set limits plt.axis("equal") → equal aspect ratio for x and y plt.axis("auto") → let Matplotlib decide automatically plt.axis("scaled") → equal aspect ratio and data fit nicely inside.		
	add axis	ax=fig.add_axes([0.5,0.1,0.5,0.4]) # left, bottom, width, height # values from 0 to 1 relative to the whole fig ax.plot(data)		

	global config	font_config={
		
		'family':'monospace',
		'weight':'bold',
		'size':10
		}
		text={
		'color':'red'
		}
		# runtime configuration (rc)
		plt.rc('font',**font_config)
		plt.rc('text',**text)
		# for font, text, lines, axes, xtick, ytick, legend, figure etc		

text	**# plt has a text()** function, equivalent to ax.text() ax.text(x, y, “Text”) # for x in range(10): ax.text(x,data[x],'%.2f ' % data[x], family=‘monospace',font size=10) # font properties ax.text(x, y, “Text", family=‘monospace’, fontsize=12, color="red", ha="center", va=“bottom”, rotation=45, bbox=dict(facecolor='ye llow', alpha=0.5))	text(x, y, "Text", col = "red", cex = 1.2, pos = 4) # text(data, labels = sprintf("%.2f", data), family = "mono", cex = 0.8, pos = 3)	
annotate	**# plt has annotate too** ax.annotate('hello', xy=(6,1), xytext=(6,5), arrowprops=dict(facecol or='red', headwidth=8, headlength=4, width=2), ha='left', va='top')	arrows(6, 1.5, 6, 1, col="red") # arrow from (6,1.5) to (6,1) text(6, 1.5, "hello")	

draw shapes	# 1 rect=plt.Rectangle((1,1),2,1,color='k') # 2 circle=plt.Circle((4,4),1,color='b') # 3 polygon=plt.Polygon([[1,4], [2,6], [3,4]],color='g',alpha=0.5) ax=plt.gca() # get curr axis ax.add_patch(rect) ax.add_patch(circle) ax.add_patch(polygon) # plt has no add_patch method	plot(0,0, type="n", xlim=c(0,10), ylim=c(0,10), asp=1) # 1: xleft, ybottom, xright, ytop rect(1, 1, 1+2, 1+1, col='red', border=NA) # 2 symbols(4, 4, circles=1, inches=FALSE, add=TRUE, bg=rgb(0,0,1,0.3)) # 3 polygon(c(1,2,3), c(4,6,4), col=rgb(0,1,0,0.5), border=NA)	

save fig to file	fig.savefig("plot.png")
	# svg, pdf etc
		# open device
		png("plot.png")
		#
		plot(…)
		# must close device
		dev.off()	