

Python VS R: Comparison Chart for Data Processing



Dis- advantages	Usability and Flexibility	<ul style="list-style-type: none">more natural for people with a software engineering backgroundeasy to code and debugindentation mattersany piece of functionality is always written the same way with Pythonflexible for creating something that has never been done before.requires users to install packages for data analysis, and these packages have greatly improved in recent years	<ul style="list-style-type: none">easier to learn if you have no coding experiencethe same piece of functionality can be written in several ways with Reasy to use complex functions in R, since all kinds of statistical tests and models are readily available and easily usedhandle basic data analysis without needing to install packages.
		<ul style="list-style-type: none">general-purpose programming languages are useful beyond just data analysisgreat for mathematical computation and learning how algorithms work	<ul style="list-style-type: none">best tool for making beautiful graphs and visualizationshas many functionalities for data analysis
		<ul style="list-style-type: none">doesn't have as many librariesvisualizations are more convoluted, and results are not as eye-pleasing or informative	<ul style="list-style-type: none">finding the right packages to use may be time consumingthere are many dependencies between R librariesnot as popular for deep learning and NLP
		<ul style="list-style-type: none">Pandas to easily manipulate dataSciPy and NumPy for scientific computingScikit-learn for machine learningMatplotlib and seaborn to make graphicsstatsmodels to explore data, estimate statistical models, and perform statistical tests and unit test	<ul style="list-style-type: none">dplyr, tidyr and data.table to easily manipulate datastringr to manipulate stringszoo to work with regular and irregular time seriesggplot2 to visualize datacaret for machine learning

import	
import library_name	library("package_name")
tips: (python) must put package name when calling functions (r) package_name::function is necessary only when the namespace collisions.	

read dataset	
df = pd.read_csv(file.csv/url)	data(df) df <- read.csv("file.csv")
tips: In r, some packages require us to load the data separately, while for others we can directly use the data	

get dataset information	
df.describe()	summary(df), str(df)

get row/column information	
df.columns, pd.index	colnames(df), rownames(df)
df.shape[1], df.shape[0] or len(df.columns), len(df.index)	ncol(df), nrow(df), dim(df),

create a dataframe	
df = pd.DataFrame({"val1", "val2"}, {"val3", "val4"}), columns = [col1, col2])	df <- data.frame (col1=c("val1", "val2"), col2=c("val3", "val4"))

change index names	
df.set_index("index column, e.g Columbia UNI", inplace = True)	rownames(df) <- df\$ "Columbia UNI"

change data type	
df["column"]=df.column.astype('float')	df\$col1 <- as.numeric(df\$col1)

slicing/subset	
df.loc[[col1,col2], [row1,row2]]; df.iloc(m:n, j:k)	df[c(row1,row2), c(col1,col2)], df[m:n,j:k]
tips: col1, row1, etc. are variable names. m, n, j, k are indices.	

merge and concat	
merge: pd.merge(df1, df2, on='key') concat: pd.concat([df1, df2...], join='outer')	merge: merge(df1, df2, by = 'key') concat: cbind(df1, df2), rbind(df1, df2)

group by	
df.groupby(by=["col"]).sum()	df %>% group_by(col) %>% summarize(Count = n())

filter/query	
df[boolean conditions] df.query(boolean conditions)	df[boolean conditions]; filter(df, boolean conditions)

sort	
df.sort_values(by=["col1", ascending=False])	df[order(df\$col1),]

handle NAs	
df.dropna()	na.omit(df)