Python VS R: Comparison Chart for Data Processing

Usability and Flexibility	• more natural for people with a software engineering background • easy to code and debug • indentation matters • any piece of functionality is always written the same way with Python • flexible 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	• easier to learn if you have no coding experience • the same piece of functionality can be written in several ways with R • easy to use complex functions in R, since all kinds of statistical tests and models are readily available and easily used • handle basic data analysis without needing to install packages.
Advantages	• general-purpose programming languages are useful beyond just data analysis • great for mathematical computation and learning how algorithms work	• best tool for making beautiful graphs and visualizations • has many functionalities for data analysis
Dis-advantages	• doesn't have as may libraries • visualizations are more convoluted, and results are not as eye-pleasing or informative	• finding the right packages to use may be time consuming • there are many dependencies between R libraries • not as popular for deep learning and NLP
Popular Libraries and Packages	• Pandas to easily manipulate data • SciPy and NumPy for scientific computing • Scikit-learn for machine learning • Matplotlib and seaborn to make graphics • statsmodels to explore data, estimate statistical models, and perform statistical tests and unit test	• dplyr,tidyr and data.table to easily manipulate data • stringr to manipulate strings • zoo to work with regular and irregular time series • ggplot2 to visualize data • caret 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)

Created by Yu Liu yl3738@columbia.edu and Dingwen Xie dx2186@columbia.edu • Updated: 2020-10

References: https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis ; https://shiring.github.io/r_vs_python/2017/01/22/R_vs_Py_post