Open in app Get started

Published in Towards Data Science

Shiu-Tang Li Follow

Oct 9, 2019 · 4 min read · Listen

Save

Cheat sheet for Python dataframe ↔ R dataframe syntax conversions

A mini-guide for those who're familiar with data analysis using either Python or R and want to quickly learn the basics for the other language

Comments / suggestions are welcome.

Python ↔R basics

```
# Python ⇔ R: object types
type(a)  ⇔ class(a)   # "class" is better than "typeof"

# Python ⇔ R: variable assignment
a=5      ⇔ a<-5       # a=5 also works for R

# Python list ⇔ R vector:
a = [1,3,5,7]                ⇔  a <- c(1,3,5,7)
a = [i for i in range(3,9)]  ⇔  a <- c(3:9)

# Python 'for loop':
for val in [1,3,5]:
    print(val)

# R 'for loop':
for (val in c(1,3,5)){
    print(val)
}

# Python function:
def new_function(a, b=5):
    return a+b

# R function:
new_function <- function(a, b=5) {
    return (a+b)
}
```

Inspecting dataframe

```
# Python ⇔ R
df.head()       ⇔  head(df)
df.head(3)      ⇔  head(df,3)
df.describe()    ⇔  summary(df)      # similar, not exactly the same
NO EQUIVALENT    ⇔  str(df)
```

File I/O

```
# Python
import pandas as pd
df = pd.read_csv("input.csv",
                 sep    = ",",
                 header = 0)
df.to_csv("output.csv", index = False)

# R
df <- read.csv("input.csv",
               header = TRUE,
               na.strings=c("","NA"),
               sep = ",")
write.csv(df, "output.csv", row.names = FALSE)

# na.strings: make sure NAs are not read as empty strings
```

Create a new dataframe

```
# Python
import pandas as pd
df = pd.DataFrame(dict(col_a=['a','b','c'], col_b=[1,2,3]))

# R
col_a <- c('a','b','c')
col_b <- c(1,2,3)
df <- data.frame(col_a, col_b)
```

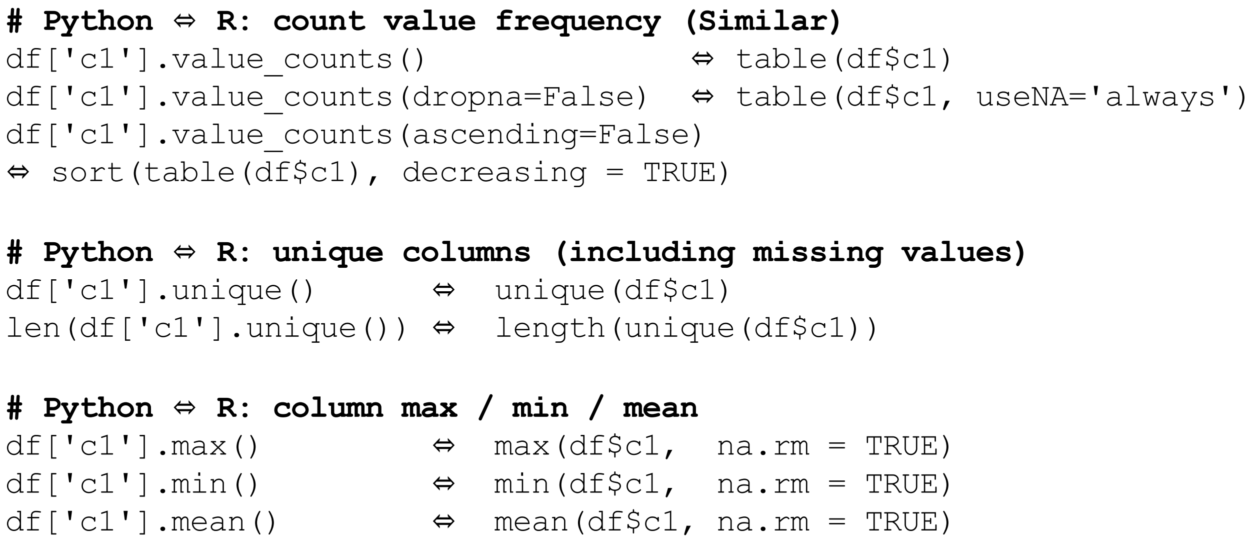
Column / row filtering

```
df[(df$column_1 > 3) &
   (is.na(df$column_2)), ]
OR
library(dplyr)
df %>% filter((column_1 > 3) & (is.na(column_2)))

# Python ⇔ R: column filtering (keep columns)
df[['c1', 'c2']] ⇔  df[c('c1', 'c2')]   # OR: df[,c('c1', 'c2')]

# Python ⇔ R(with dplyr): column filtering (drop columns)
df.drop(['c1', 'c2'], axis=1)  ⇔  df %>% select(-c('c1', 'c2'))

# Python ⇔ R: select columns by position
df.iloc[:,2:5]  ⇔  df[c(3:5)]           # Note the indexing

# Python: check if a column contains specific values
df[df['c1'].isin(['a','b'])]
OR
df.query('c1 in ("a", "b")')

# R: check if a column contains specific values
df[df$c1 %in% c('a', 'b'), ]
OR
library(dplyr)
df %>% filter(c1 %in% c('a', 'b'))
```

Missing value handling / count

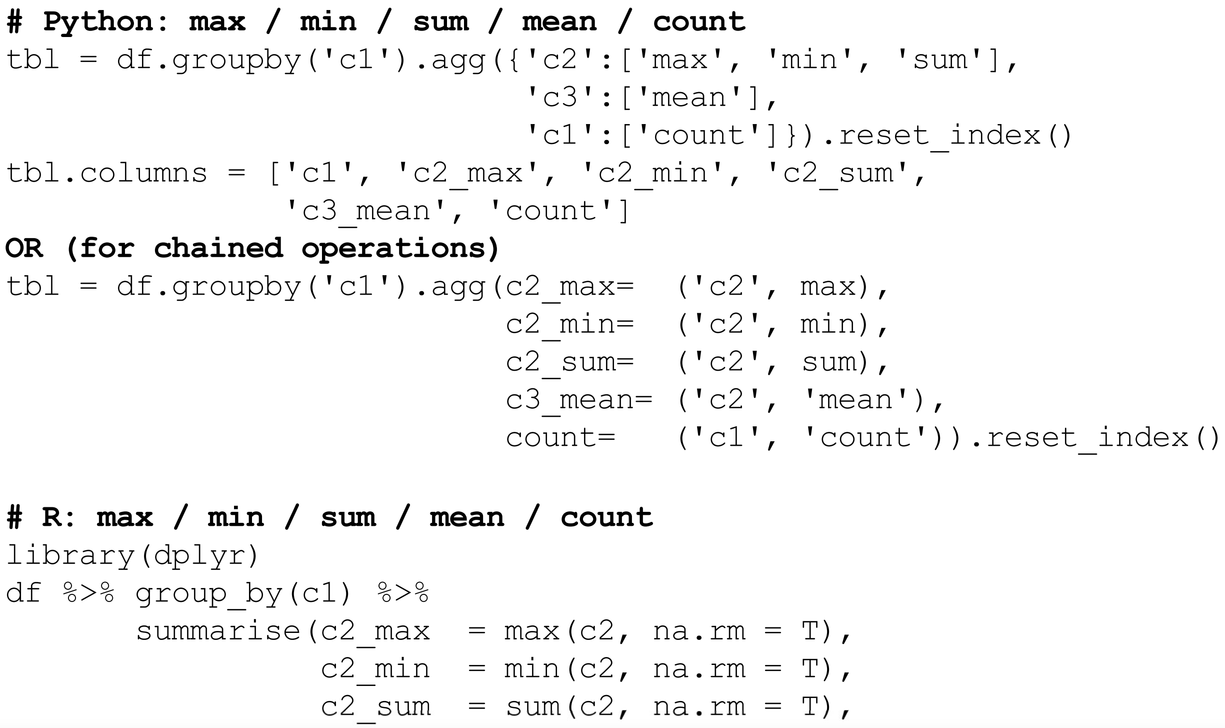
```
# Python: missing value imputation
df['c1'] = df['c1'].fillna(0)
OR
df.fillna(value={'c1': 0})

# R: missing value imputation
df$c1[is.na(df$c1)] <- 0
OR
df$c1 = ifelse(is.na(df$c1) == TRUE, 0, df$c1)
OR
library(dplyr)
library(tidyr)
df %>% mutate(c1 = replace_na(c1, 0))
```

Statistics for a single column



```
# Python ⇔ R: count value frequency (Similar)
df['c1'].value_counts()               ⇔ table(df$c1)
df['c1'].value_counts(dropna=False)   ⇔ table(df$c1, useNA='always')
df['c1'].value_counts(ascending=False)
⇔ sort(table(df$c1), decreasing = TRUE)

# Python ⇔ R: unique columns (including missing values)
df['c1'].unique()      ⇔  unique(df$c1)
len(df['c1'].unique()) ⇔  length(unique(df$c1))

# Python ⇔ R: column max / min / mean
df['c1'].max()         ⇔  max(df$c1,  na.rm = TRUE)
df['c1'].min()         ⇔  min(df$c1,  na.rm = TRUE)
df['c1'].mean()        ⇔  mean(df$c1, na.rm = TRUE)
```

grouping and aggregations



```
# Python: max / min / sum / mean / count
tbl = df.groupby('c1').agg({'c2':['max', 'min', 'sum'],
                            'c3':['mean'],
                            'c1':['count']}).reset_index()
tbl.columns = ['c1', 'c2_max', 'c2_min', 'c2_sum',
               'c3_mean', 'count']
OR (for chained operations)
tbl = df.groupby('c1').agg(c2_max=  ('c2', max),
                           c2_min=  ('c2', min),
                           c2_sum=  ('c2', sum),
                           c3_mean= ('c2', 'mean'),
                           count=   ('c1', 'count')).reset_index()

# R: max / min / sum / mean / count
library(dplyr)
df %>% group_by(c1) %>%
       summarise(c2_max  = max(c2, na.rm = T),
                 c2_min  = min(c2, na.rm = T),
                 c2_sum  = sum(c2, na.rm = T),
                        .reset_index()\
                        .rename(columns={'c2':'c2_cnt_distinct'})

# R: count distinct
library(dplyr)
tbl <- df %>% group_by(c1)
          %>% summarise(c2_cnt_distinct = n_distinct(c2))
```

creating new columns / altering existing columns

```
# Python: rename columns
df.rename(columns={'old_col': 'new_col'})

# R: rename columns
library(dplyr)
df %>% rename(new_col = old_col)

# Python: value mapping
df['Sex'] = df['Sex'].map({'male':0, 'female':1})

# R: value mapping
library(dplyr)
df$Sex <- mapvalues(df$Sex,
          from=c('male', 'female'),
          to=c(0,1))

# Python ⇔ R: change data type
df['c1'] = df['c1'].astype(str)    ⇔  df$c1 <- as.character(df$c1)
df['c1'] = df['c1'].astype(int)    ⇔  df$c1 <- as.integer(df$c1)
df['c1'] = df['c1'].astype(float)  ⇔  df$c1 <- as.numeric(df$c1)
```

Updating column values by row filters

```
# Python ⇔ R:
df.loc[df['c1']=='A', 'c2'] = 99  ⇔  df[df$c1=='A', 'c2'] <- 99
```

```
merged_df1 = pd.merge(df1, df2, on='c1', how='inner')
merged_df2 = pd.merge(df1, df2, on='c1', how='left')
OR (for chained operations)
merged_df1 = df1.merge(df2, on='c1', how='inner')
merged_df2 = df1.merge(df2, on='c1', how='left')

# R: inner join / left join
merged_df1 <- merge(x=df1,y=df2,by='c1')
merged_df2 <- merge(x=df1,y=df2,by='c1',all.x=TRUE)
OR
library(dplyr)
merged_df1 <- inner_join(x=df1,y=df2,by='c1')
merged_df2 <- left_join(x=df1,y=df2,by='c1')

# Python: sorting
df.sort_values(by=['c1','c2'], ascending = [True, False])

# R: sorting
library(dplyr)
df %>% arrange(c1, desc(c2))
```

Concatenation / sampling

```
# Python (import pandas as pd) ⇔ R: concatenation
pd.concat([df1, df2, df3])      ⇔ rbind(df1, df2, df3)
pd.concat([df1, df2], axis=1)   ⇔ cbind(df1, df2)

# Python random sample
df.sample(n=3, random_state=42)

# R random sample
set.seed(42)
sample_n(df, 3)
```

An example of chained operations

```
# Python: chained operations with '.'
```

```
  .query('c3 > 10')

# R: chained operations with '%>%'
library(dplyr)
library(tidyr)
df %>% select(-c('c1')) %>%
       arrange(desc(c2)) %>%
       mutate(c3 = c1*3 + 2) %>%
       mutate(c2 = replace_na(c2, 0),
              c4 = replace_na(c4, -99)) %>%
       rename(TOT = total) %>%
       filter(c3 > 10)
```

381 10