R/Python Cheatsheet

Abishek Arunachalam

12 October 2019

1. Importing .csv file:

Getting data is the starting step for any analysis. Comma seperated value(csv) files are the most common format the data is stored.

- **R read.csv()** command from base R is used to import .csv files. The header=TRUE specifies the first line in the file is a column header.

```
mydata <- read.csv("Turtles.csv", header = TRUE)
```

- **Python read_csv()** method from the Pandas package is used to import .csv files in Python. The boolean value of 1 indicates the first row is a header.

```
import pandas as pd
data = pd.read_csv("Turtles.csv", header=1)
```

2. Row bind and column bind:

Row bind and column bind are commonly used for appending two data frames or records to a existing data frame.

- **R**

rbind() command is used to append rows and **cbind()** command to append columns.

```
# create two vectors x and y
x <- c(1,2,3,4,5)
y <- c(7,8,9,10,11)
# create a data frame from the two vectors
rand_num <- data.frame(x,y)

z <- c(7,16,27,40, 55)
# column bind vector z with the dataframe
rand_num <- cbind(rand_num, z)

k <- c(2,1,9)
# row bind vector k with the data frame
rand_num <- rbind(rand_num, k)
head(rand_num)
```

```
##   x  y  z
## 1 1  7  7
## 2 2  8 16
## 3 3  9 27
## 4 4 10 40
## 5 5 11 55
## 6 2  1  9
```

x and y are vectors and the **c()** command denotes the concatenation operation. A dataframe is created from these vectors using the **data.frame()** command. Then, **cbind()** is applied to add column z and **rbind()** is applied to add row k. In the example above, all vectors have integer values. If the elements are heterogenous, R changes the data type of the elements in the vector to make it homogeneous through a property called corecion.

- **Python**

Python uses **concat()** and **append()** methods from the Pandas packages to perform row bind and column bind operations similar to R.

```
import pandas as pd
# create two lists x and y
x = [1,2,3,4,5]
y = [7,8,9,10,11]

rand_num = {'x':x, 'y':y}
# create a data frame from the vectors
rand_num = pd.DataFrame(rand_num)

# create a new series
z = pd.Series([7,16,27,40,55])
# concatenate series with the dataframe columnwise
rand_num = pd.concat([rand_num, z.rename('z')], axis=1)

# create a series with index values
k = pd.Series([23,26,39], index=['x', 'y', 'z'])

# append series to the dataframe rowwise
rand_num = rand_num.append(k, ignore_index=True)
rand_num.head(6)
```

```
##      x   y   z
## 0    1   7   7
## 1    2   8  16
## 2    3   9  27
## 3    4  10  40
## 4    5  11  55
## 5   23  26  39
```

x and y are lists that are added to the dictionary rand_num and a data frame is created from the dictionary. The series z is added as a column to the dataframe using **concat()** method. axis = 1 argument specifies to concatenate columnwise.

append() method is used to add a single row to the dataframe. The ignore_index = True parameter tells the dataframe to ignore its index.

In Python, list datastructure can be a heterogenous collection of elements and series are a homogenous collection with indexes.

3. Filtering data:

It is used for conditionally filtering data to work on a subset and performing analysis. Also used during data cleaning phase to remove outliers.

- **R filter()** command from the dplyr package is used to conditionally filter records in a dataframe. Filters could be applied to both factor and quatitative variables with the help of logical operators. An example of filtering records in mtcars dataset:

```
library(dplyr)
mtcars %>%
  filter(cyl == 8, hp > 230)
```

```
##    mpg cyl disp  hp drat   wt  qsec vs am gear carb
## 1 14.3   8  360 245 3.21 3.57 15.84  0  0    3    4
## 2 13.3   8  350 245 3.73 3.84 15.41  0  0    3    4
## 3 15.8   8  351 264 4.22 3.17 14.50  0  1    5    4
## 4 15.0   8  301 335 3.54 3.57 14.60  0  1    5    8
```

The pipe symbol %>% is used to the perform multiple operations on the dataset in a single command while storing its state after each operation. The command above filters cars that have 8 cyclinders and horsepower greater than 230.

- **Python dataframe.loc()** in Python is the equivalent of **filter()** command in R. The **loc()** command locates a group of columns by name and filters with logical operators. The example below uses the same mtcars dataset to run Python code with help of **reticulate** package:

```
df = r.mtcars
df = df.loc[(df['cyl'] == 8) & (df['hp'] > 230)]
df.head(3)
```

```
##                 mpg  cyl   disp     hp  drat  ...   qsec   vs   am  gear  carb
## Duster 360     14.3  8.0  360.0  245.0  3.21  ...  15.84  0.0  0.0   3.0   4.0
## Camaro Z28     13.3  8.0  350.0  245.0  3.73  ...  15.41  0.0  0.0   3.0   4.0
## Ford Pantera L 15.8  8.0  351.0  264.0  4.22  ...  14.50  0.0  1.0   5.0   4.0
##
## [3 rows x 11 columns]
```

The code gives the same output as observed in R using filter. The conditional operations must be enclosed in parantheses as & operation has precedence over logical operators in Python.

4. Removing NA's:

- **R**

na.omit() is used to omit rows with NA's in the dataframe

```
df <- na.omit(airquality)
```

- **Python**

dropna() method is used to drop rows with Nan.

```
df = r.airquality
df.dropna(how='any', axis=0, inplace=True)
```

5. Select columns:

Selecting columns is used for data cleaning and data preparation to select a subset of columns to work with.

- **R**

select() command with the names of the columns as arguments is used for selecting the required columns.

```
mtcars %>%
  select(mpg, cyl, disp, hp)%>%
  head(3)
```

```
##                mpg cyl disp  hp
## Mazda RX4     21.0   6  160 110
## Mazda RX4 Wag 21.0   6  160 110
## Datsun 710    22.8   4  108  93
```

- **Python**

In Python, the columns can be directly selected with its names passed as a list.

```
r.mtcars[['mpg','cyl','disp','hp']].head(3)
```

```
##                 mpg  cyl   disp     hp
## Mazda RX4      21.0  6.0  160.0  110.0
## Mazda RX4 Wag  21.0  6.0  160.0  110.0
## Datsun 710     22.8  4.0  108.0   93.0
```

6. Group by and Mutate:

Used during feature engineering to create new variables.

- **R**

group_by() command is used to group records based on categories and **mutate()** method is used to create a new variable from existing variables while preserving the existing ones.

```
# group by gender and calculate bmi variable
df <- starwars %>%
  group_by(gender) %>%
  mutate(bmi = mass / ((height/100)^2))
head(df,3)
```

```
## # A tibble: 3 x 14
## # Groups:   gender [2]
##   name  height  mass hair_color skin_color eye_color birth_year gender
##   <chr>  <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
## 1 Luke~     172     77 blond      fair       blue              19 male
## 2 C-3PO     167     75 <NA>       gold       yellow           112 <NA>
## 3 R2-D2      96     32 <NA>       white, bl~ red               33 <NA>
## # ... with 6 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>, bmi <dbl>
```

The code above groups records based on gender and creates a new variable BMI from mass and height variables.

- **Python**

Python uses **groupby()** and **apply()** methods to calculate bmi from the variables. **concat()** method is used to concatenate the calculated variable as a new column to the dataframe.

```
import pandas as pd
df = r.starwars
# concat to calculate bmi and a
pd.concat([df, df.groupby('gender').apply(lambda x: (x.mass)/((x.height/100)**2)).reset_index(drop=True
```

```
##               name  height  ...                     starships          0
## 0  Luke Skywalker     172  ...  [X-wing, Imperial shuttle]  26.892323
## 1            C-3PO     167  ...                          []  34.722222
## 2            R2-D2      96  ...                          []  34.009990
##
## [3 rows x 14 columns]
```

7. Transposing data:

Transposing a dataframe is a common operation in the dataset preparation phase. The columns in the dataframe are converted to rows and viceversa.

- **R**

t() command is used to transpose the dataframe or the matrix in R.

```
# create a matrix
df <- as.data.frame(matrix(1:9, nrow = 3, ncol = 3))
# print matrix
print(df)
```

```
##   V1 V2 V3
## 1  1  4  7
## 2  2  5  8
## 3  3  6  9
```

```
# transpose of the matrix
print(t(df))
```

```
##     [,1] [,2] [,3]
## V1    1    2    3
## V2    4    5    6
## V3    7    8    9
```

- **Python**

transpose() method in Python is used to transpose the dataframe.

```
r.df.transpose()
```

```
##     0  1  2
## V1  1  2  3
## V2  4  5  6
## V3  7  8  9
```

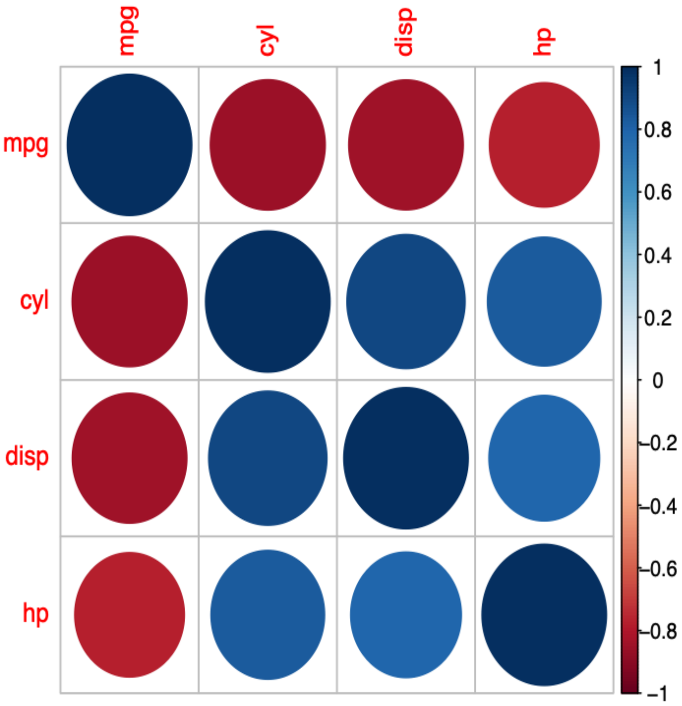
8. Correlation plot:

Correlation is used to examine relationships between pairs of variables. It shows the degree to which two variables vary together. Positive correlation between variables will have a value close to 1 and negative correlation have value close to -1. Correlation can only be calculated for quantitative variables. These plots are usually created during the exploratory data analysis phase to find predictors that could potentially be useful for predicting response and to study interaction among predictors.

- **R**

cor() command is used to calculate correlation values and then passed to **corrplot()** command to visualise. The darker shades of blue indicates higher positive correlation and darker shades of red indicate negative correlation.

```
library(corrplot)
df <- mtcars %>%
  select(mpg, cyl, disp, hp)
corrplot(cor(df))
```



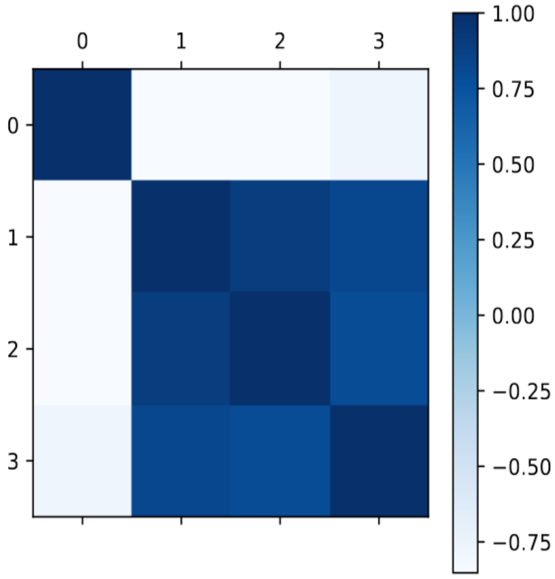
- **Python**

matshow() method from the matplotlib package is used for the correlation plot. Blue and white is used to contrast postive and negative correlation.

```
import matplotlib.pyplot as plt
plt.matshow(r.df.corr(), cmap='Blues')
plt.colorbar()
```

```
## <matplotlib.colorbar.Colorbar object at 0x12a9842b0>
```

```
plt.show()
```



9. Merging dataframes:

Merging dataframes is similar to performing join operations in SQL. The various join operations are perfomed with key columns common to both dataframes. It is used in the data preparation phase. Following examples shows the common left join operation.

- **R**

df1 contains student details and df2 contains their score in Maths. The two dataframes are merged with the common column StudentId. all.x = TRUE parameter performs a left join operation holding all records in the left dataframe and merging matching records in the right dataframe.

```
# create two dataframes
df1 <- data.frame(StudentId = c(1:5), Name = c('Arjun', 'Peter', 'John', 'Guo', 'Kent'))
df2 <- data.frame(StudentId = c(1:4), Maths = c(89, 95, 78, 92))

# perform left join with student id
merge(x = df1, y = df2, by = "StudentId", all.x = TRUE)
```

```
##   StudentId  Name Maths
## 1         1 Arjun    89
## 2         2 Peter    95
## 3         3  John    78
## 4         4   Guo    92
## 5         5  Kent    NA
```

- **Python**

Python uses pandas to merge two dataframes. The **on** argument tells to merge the dataframes on StudentIn column and **how** tells it to perform a left join.

```
import pandas as pd

# create datframe from lists
data1 = [[1,'Arjun'], [2,'Peter'], [3,'John'], [4,'Guo'], [5,'Kent']]
df1 = pd.DataFrame(data1, columns = ['StudentId', 'Name'])

data2 = [[1,89], [2,95], [3,78], [4,92]]
df2 = pd.DataFrame(data2, columns = ['StudentId', 'Maths'])

# left join dataframes
pd.merge(df1, df2, on='StudentId', how='left')
```

```
##    StudentId   Name  Maths
## 0          1  Arjun   89.0
## 1          2  Peter   95.0
## 2          3   John   78.0
## 3          4    Guo   92.0
## 4          5   Kent    NaN
```

10. Linear regression:

Linear regression is the simplest and most commonly used regression model. It tries to find a linear line through a cloud of points in p dimensional space(where p is number of predictors) so as to reduce the distance between the line and each point in the space.The following examples show running a linear model in R and Python, instead of focusing on validating model assumptions or prediction.

- **R**

lm() command to used to run the linear model in R. The data is split into train-set (80%) and test-set (20%) and predictions are made on the test set after training the model on train set. mpg is the response variable and cyl,disp,hp and wt are the predictors. The summary shows wt variable is a significant predictor as the p-value is less than 0.05.The R-squared value indicates the model explains 83.7% variance in the data.

```
df <- mtcars %>%
  select(mpg, cyl, disp, hp, wt)

# calculate 80% split cut off
cutoff = round(nrow(df)*0.8)

# split data into train and test set with the cut-oof
train = df[0:cutoff,]
test = df[cutoff:nrow(df),]

# linear model is trained on the train set
fit = lm(formula = mpg~.,data = train)
# summary of the model
summary(fit)
```

```
##
## Call:
## lm(formula = mpg ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1290 -1.5450 -0.4795  0.9901  5.7191
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 40.94837    3.57431  11.456  1.7e-10 ***
## cyl         -1.30888    0.79950  -1.637    0.117
## disp         0.01287    0.01451   0.887    0.385
## hp          -0.02917    0.02415  -1.208    0.241
## wt          -3.57016    1.28061  -2.788    0.011 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.695 on 21 degrees of freedom
## Multiple R-squared:  0.8371, Adjusted R-squared:  0.8061
## F-statistic: 26.98 on 4 and 21 DF,  p-value: 5.19e-08
```

Predictions are made on the test set with **predict()** command.

```
predict(fit, test)
```

```
##      Fiat X1-9  Porsche 914-2   Lotus Europa Ford Pantera L   Ferrari Dino
##       27.89636       26.96686       28.23927       15.97745       19.96768
##  Maserati Bora     Volvo 142E
##       11.83482       24.16594
```

- **Python**

fit() method in the LinearRegression module of scikit-learn package is used to run regression model. **score()** provides the R-squared value.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# select columns
df = r.mtcars[['mpg','cyl','disp','hp','wt']]

# create a predictors and response variables
X = df.drop('mpg',axis=1)
y = df.mpg

# 80% train split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# using linear regression object call fit
reg = LinearRegression().fit(X_train, y_train)
# adjusted-rsquared value
reg.score(X_train, y_train)
```

```
## 0.8420221898887312
```

The predictions for the test set is given by the **predict()** method.

```
# predicted mpg values
reg.predict(X_test)
```

```
## array([16.66946485, 24.49659915, 19.61123724, 15.52848882, 28.51626782,
##        15.69855219, 21.71106503])
```

11. Getting data from API:

As data scientist, it is common to source datasets from open data websites during the data collection phase.

- **R**

R uses the httr package that contains the HTTP(Hypertext transfer protocol) methods to hit the API's and gain access to the datasets. The status code 200 denotes the GET method was sucessful in retrieving the data.

```
library(httr)
r <- GET("http://httpbin.org/get")
r$status_code
```

```
## [1] 200
```

- **Python**

Python has all its HTTP methods in the requests package. Calling the **get()** method with requests object returns the status code 200 indicating success.

```
import requests
requests.get('https://api.github.com')
```

```
## <Response [200]>
```