Cheatography

Data Science (R) Cheat Sheet

by cmok1996 via cheatography.com/136331/cs/28443/

Variable Setting

Scalar	x <- 5	Set x = 5
Vectors	x <- c(1,2,-3,4,5)	Create a vector with 5 numeric elements
Named Vectors	x <- c("A" = 1, "B" = 2)	Row index of vectors are named
Matrix	x <- matrix-(c(1:12), byrow = T, nrow=3)	Create a matrix with 3 rows; fill each row first
Arrays	x <- array(-c(1:24), dim = c(4,3,2))	Matrix but with more than 2 dimensions - [row, col, ndim]; All elements must have same datatype
List	x <- list("N-ame1" = c(1,2,3), "-Name2" = c("A", "B", "-C", "D"))	Most unordered structure - Can contain vectors of different type and length
Data Frames	df <- data.f-rame(numeric_c-olumn = c(1,2,3), char_column = c("A", "B", "C"))	List but all vectors contains same length
Tibble	library(t-ibble); df <- tibble(x = runif(100), y=rnorm(n-=100))	Create a tibble with 100 observ-ations of x (uniform distri-bution) and y (normal distri-bution)

Variable Setting (cont)

Check type	class(x) ; typeof(x) ; is.blah(x)	Check the data type of variable
Coerce	as.blah(x); as_tibble(x)	Eg, as.character(x) coerce vector x into strings

Programming

Conc-atenate string-+varia-ble	x=5; cat("He-llo", x, sep=" ")	Returns "Hello 5"
Acce-ssing by index	df[row_id, col_id]; df[col_id]; x[1]	Returns a vector
Access by name	df$col-_name ; df[["co-l_n-ame"]]	Extracts the underlying component; Works for all named datatypes including lists
Subset datafr-ame	df[c(col_-name1, col_na-me2)]	Returns a dataframe
Insp-ect	str(x)	Structure of x
Length	nrow(df); ncol(df)	length(df) gives number of columns
Access List	list[[dim]]	Extract a single component from the list

Programming (cont)

If statem-ents	if (logic) {exp} else if (logic) {exp} else {exp}	If statements
Func-tions	func <- functi-on(args) {exp}	if ... in args, it accepts all arguments; useful if function is chained to another function
For loop	for (i in seq(le-ngth(x))) {exp}	for i in range(len(x)) equivalent
While loop	while (logic) {exp}	While loop
Memb-ership	x %in% c(...)	Returns a boolean value
Lambda Functi-ons	func <- functi-on(x) <func>	No {} is required, used together with apply/map

Programming (cont)

Iterate	map(x, func); map2(x,y,-func); map_dbl(x, func)	If vector is supplied, apply scalar_func to every element in vector x; If dataframe is applied, apply vector_func to every column; map_blah returns a vector instead of a list
Apply across columns	apply(df, margin=2, func)	Apply func across columns (margi-n=2); if margin=1, apply across rows
Shape of matrix	dim(X)	Prints the shape of matrix
Matrix multip-lication	X %*% Y	If X is a (m,n) matrix and Y is a (n,o) matrix; output returns a (m,o) matrix
Determi-nant	det(X)	Returns the determinant of matrix

Programming (cont)

Inverse	solve(X)	Returns the inverse of matrix X with same shape
Solving SLE	solve(X, y)	Equivalent to solve(X) %*% b; X is (m,n) matrix while y is (n,) vector

Inspect + Parse

Read file	library(readr); df <- read_d-elim(path, skip, col_na-mes=c(...), na="?"))	Supports read_csv, read_tsv; Parse "?" into NA values
Read excel	library(r-eadxl); df <- read_deli-m(path, skip, col_names-=c(...), na="?"))	Read excel files
Head	head(df, n)	See first n rows
Colu-mn/Row names	names(df); rownames(df)	Variable assign to overwrite column/row names
Desc-ribe	summary(df)	Basic descri-ptive statistics for each column, including count of missing values
Stru-cture	str(df)	Check column datatype

Inspect + Parse (cont)

Coerce	df$col %>% as.blah()	Convert vector typeof
Factors	parse_-factor-(col, levels = c("..."))	Factors are integer types where each category is serial-ized; One hot encoding is automatically applied for factors when modelling
Missing values	df %>% filter-(is.na-(col))	Find missing values
Implicit missing values	df %>% comple-te(col1, col2)	Eg, col1 = years, col2 = quarters. Ensure all year-q-uarter combination is captured
Fill Missing Values	df %>% fill(col, .direction = "up")	Direction = "up" -> Forward fill
Impute missing values	ifelse-(is.na(df-$col), ?, df$col)	Impute missing values
Drop duplic-ates	df %>% distinct(-col1)	Drop duplicates

Inspect + Parse (cont)

Convert to date	library(lubridate); as.Date(c("24/05/1996"), format = "%d/%m%Y") ; parse_date(vector)	Parse date vector; input vector is character
Create date	ymd(YYYYMMDD) ; ymd_hms("YYYY-MM-DD HH:MM:SS")	Create a datetime object
Extract date compo nents	year(dt)	Useful with mutate; month(), mday(), wday(), week(), hour(), minute()
Date intervals	(dt1 %--% dt2) %/% days(1)	Find number of days between dt1 and dt 2

Wrangle (dplyr)

Chain operator	df %>% func() %>% .$col_name	Equivalent to func(df), and then access col_name from the returned output
Tibble	df %>% as_tibble()	Works like dataframe with nice features
Access rows by index	df %>% slice(seq); df[seq,]	Python iloc equivalent

Wrangle (dplyr) (cont)

Index of True values	where(<logic>)	Python .index equivalent ; Saved indices could be used to access a vector
Slice on index	df %>% slice(where(<logic>)	Application of slicing
Rename	df %>% rename(col1 = newname)	Rename column names
Filter	df %>% filter(<logic> & <logic> | <logic>) ; df %>% filter(is.na(col1))	Filter based on logical function; Eg, col1 < 50 & col2 %in% c("A", "B")
Filter na values	df %>% filter(is.na(col1)) ; df %>% filter(!is.na(x))	Filter na values based on variable
Subset / Select	df %>% select(col1, col2 : col 4, -(remove_cols, ...), everything())	Select columns of dataframe; equivalent to df[c("col1", "col2", ...)]
Duplicates	df[duplicates(df$col1)]	Returns a df with duplicated values

Wrangle (dplyr) (cont)

Drop	df %>% select(-(-col1, ...))	- to indicate removal
Drop Duplicates	df %>% distinct(-col1, ..., .keep_all = T)	subset based on col inputs (optional); .keep_all = T to return all columns
Drop NA	df %>% drop_na(col1, ...)	Drop na values based on subset columns (optional)
Find missing value counts	sapply(as_tibble(df), function(x) sum(is.na(x)))	Find number of missing values per column
Fill missing values	df %>% fill(col, direction = "up"))	Direction controls whether forward or backward fill
Keep / Discard	df %>% keep(is.blah) ; df %>% discard(is.blah)	Keep/Discard based on a logical function
Sort	df %>% arrange(desc(col1))	Sort according to descending order of col1

Wrangle (dplyr) (cont)

Create new variable	df %>% mutate(new_col = ...)	Select all + Create new column; Use transmute() if only want to select new columns
Separate/Unite	df %>% separate(col = col1, into = c("newcol1", "newcol2" ...), sep = "?") ; df %>% unite(col1, col2, ..., sep = "?")	Create new columns based on split rule of character vector defined by sep argument ; Reverse for unite()
Group by + aggregate	df %>% group_by(col1) %>% summarise(new_col = agg_func(col2, na.rm=T))	Aggregate based on grouping - mean(), sum(), mean(), max(), n()
Value Counts	df %>% count(col1, sort = T)	Value counts
Nested dataframes	df %>% group_by(col1) %>% nest()	Create new column called "data" with a new dataframe of remaining columns for each cell

Wrangle (dplyr) (cont)

From wide to long	df %>% pivot_longer(c(col1, col2, ...), names_to = "newname", values_to = "newname", values_drop_na =T)	Transform a dataframe with columns as values to tidy dataframe; Lengthens the data by increasing number of rows and reducing number of columns
From long to wide	df %>% pivot_wider(names_from = "col1", values_from = "col2")	Inverse transformation of pivot_longer; Increases number of columns and reduces number of rows
Horizontal append	df1 %>% cbind(<named_vector> | <dataframe>)	Increase number of columns
Vertical append	df1 %>% rbind(df2)	Increase number of rows

Wrangle (dplyr) (cont)

Mutating Joins	df1 %>% left_join(df2, by = "col1")	Supports left_join, right_join, inner_join, full_join ; Ensure that join is by primary key of left table ; by = c("col1" = "col2") if column names are not identical
Self joins	df1 %>% semi_join(df2, by="col")	df1 does not merge with df2, but instead only returns the existence of the match

Wrangle (dplyr) (cont)

Mapping row	sapply-(df$col, func) ; ifelse-(df$col > 0.5, NA, df$col)	Apply function iteration to each element; apply -> map; sapply -> map_blah; ~ lambda function. When map is used on a dataframe instead of a vector, it applies the function to each column
Mapping across columns	sapply(df, functi-on(x) ifelse-(x>0.5, NA, x);); df %>% map_df-(~ifel-se(.>0,5, NA, .))	Apply lambda across all columns

Wrangle (dplyr) (cont)

Crosstab / Contin-gency Table	xtabs(~ col1 + col2, data = df); table(-df$col1, df$col2)	Crosstab with col1 unique values as rows and col2-u-nique values as columns; Print value counts for (col1, col2) grouping, useful for discrete factor variables

Modelling (caret)

Trai-n/Test Split	inTrain <- sample.int(n-row(df), n) OR inTrain <- createDataPa-rtition(y = df$col, p=?, list=F); training <- df[inTrain,]; testing <- df[-in-Train,]	1. Create a vector of indices to split on; 2. Filter the vector based on the masked index vector
k Fold	createFolds(-y=df$col, k = ?, list=T, return-Train = T) #returnTrain = F returns Test set	Returns a list of k elements, each element contains each dataframe fold

Modelling (caret) (cont)

Boot-strap	createRes-ample(y=d-f$col, times=k, list=T)	Returns a list of k elements, each element contains a bootstrap sample (with replacement)
Time series Blocked CV	createTim-eSlices(y=df, initialWindo-w=20, horizon=10)	Returns a list of 2 lists - train & test. In each list, it contains the indices of each time-w-indow partition
Feature Plot	library(p-sych); pairs.pan-el(df, method = "pearson", hist.col = "-#00AFBB", density = T)	Correlation Scatter Plot with Density plots in the diagonals

Modelling (caret) (cont)

Feature Selection - Zero Covariates	nearZeroVar(df, saveMetrics = T)	Explanatory variables with small values of percentUnique and nzv==T should be removed as they have little variation
Train & Predict	train_model <- train(y ~ ., data, method, ..., trControl = trainControl(...), tuneGrid = expand.grid(...), preProcess = c(method...)); pred <- predict(train_model, newdata)	General pipeline for model training; Grid Search CV on k-folds with applied transformations for imputation/scaling/etc; predict() returns a vector
Scaling	preObj <- preProcess(df, method = c("center", "scale")); predict(preObj, df)	Normal standardization; Always fit using training data; Transform (predict) using trained fitted object
Scaling using baseR	m <- apply(df, margin=2, FUN=mean); s <- apply(df, margin=2, FUN=sd); scale <- (z,m,s)	Normal standardization

Modelling (caret) (cont)

Impute	preObj <- preProcess(df, method = c("medianImpute")); df$col <- predict(preObj, df)$col	Median imputation using fit and transform
One-hot encoding	dummies <- dummyVars(y ~ x, data); dummy_df <- predict(dummies, newdata = df)	Fit and transform using one-hot encoder
PCA - preProcess	preProc <- preProcess(df, method="pca", pcaComp=2); dfPC <- predict(preProc, df)	Fit and transform PCA on dataframe to reduce dimensions
Grid Search	..., tuneGrid = expand.grid(parameter1 = c(...), parameter 2 = c(...), ...)	Parameters as defined by ? method in train()
Cross Validation	..., trControl = trainControl("cv", number = 5)	To be used within the train function; Other methods include "repeatedcv", "oob" for RF, "LOOCV", etc

Modelling (caret) (cont)

CV Errors	train_model$results$RMSE/Accuracy	Average cross-validation errors for regression and classification tasks -> Each row represent a particular grid combination
Confusion Matrix	predict(train_model, newdata) %>% confusionMatrix(newdata$y)	confusionMatrix(pred, actual); pred as rows, actual as columns
Linear Regression	library(olsrr); lm_model <- lm(y~x1 + x2 + ..., data=df); summary(lm_model); predict(lm_model, newdata); plot(lm_model)	1. Fit the MLR onto training data; 2. Check coefficients and statistical significance; 3. Predict new data; 4.Diagnostic checks

Data Science (R) Cheat Sheet

by cmok1996 via cheatography.com/136331/cs/28443/

Modelling (caret) (cont)

Panel Models	library(plm); pdf <- pdata.frame(df, index = c("indiv", "time")) ; plm_model <- plm(y~x, model = c("within", "random", "fd",...); summary(plm_model); phtest(fe_model, re_model);	1. Create panel dataframe indicating the indices; 2.Fit FE/RE; 3. Check coefficients and statistical significance; 4. Conduct Hausman Test, if null is not rejected -> RE and FE are the same and hence should use RE as more efficient
Regularized Linear Models	lasso <- train(y ~., data, method = "glmnet", family?, trControl = ..., tuneGrid = expand.grid(alpha = c(...), lambda = c(...), preProcess = c("scale", "center")); lasso_coefs <- coef(lasso$finalModel, lasso$bestTune$lambda)	Elastic net model; alpha = 1 is lambda penalty, alpha = 0 is ridge, lambda is the penalty parameter; family = c("binomial", "multinomial") to turn into classification task

Modelling (caret) (cont)

Trees	mod_rf <- train(y~ ., data, method = "ranger", importance = "impurity", trControl = trainControl("oob"), tuneGrid = expand.grid(mtry = c(...), num.trees = c(...), min.node.size = c(...), splitrule = "gini")); varimp <- mod_rf$finalModel$variable.importance %>% sort(decreasing = T)	Train Random Forest Model using "ranger" method, and then inspect the variable importance depending on impurity
SVM	svm <- train(y ~., data, method = c("svmLinear", "svmPoly", "svmRadial"), trControl = ..., tuneGrid = expand.grid(C = c(...), degree = c(...) #for polynomial kernel, sigma=c(...) #for rbf, preProcess = c("scale", "center")); lasso_coefs <- coef(lasso$finalModel, lasso$bestTune$lambda)	Train a Support Vector Machine. Hyperparameters depend on C :regularization is inversely related, degree : Controls the kernel function; sigma : Aka gamma, defines how far the influence of a single training example reaches (higher values = closer reach = less linear/smooth decision boundary = overfit)

Modelling (caret) (cont)

K-means clustering	kmeans1 <- kmeans(normalizd_df, centers = k); normalized_df$cluster <- kmeans1$cluster; qplot(x, y, color = clusters, data=normalized_df)	K means clustering
Hierarchical clustering	distance <- dist(normalized_df) ; hc <- hclust(distance, method = c("single", "complete", "average"); plot(hc);	Hierarchical clustering on distance function

Plotting (ggplot)

General	ggplot(data) + geom_func(mapping = aes(x, y, ...), stat="...", position = "..", ...) + scale_func(...) + coord_func() + labs(...) + theme(legend.position = "..") + facet_wrap(~col, nrow = ?) + theme_bw()	ggplot -> Contains data, optional global mapping which can be overwritten by geom_func; geom_func -> graph aesthetic layer separated by + geom_func2(); scale_func -> control axes labels/ticks/scale/etc; labs(...) -> Title, label names, etc; facet -> plot separate plots side by side

Plotting (ggplot) (cont)

Graphical layers	geom_func(mapping = aes(x=col1, y=col2, color = col3, size = col4, group = col5, fill = col6, group = col7 ...), stat=c("identity", "count"), position = c("dodge", "stacked"), lwd = ?, pch = ?, cex = ?, alpha = ?)),)	Creates an aesthetic layer on the graph (scatterplot, barchart, boxplot, etc). The drawing is defined inside aes(...). Plot options are outside aes() where it applies a stats function, adjust the position of the plots wherever relevant, and other global options
Quick plot	qplot(x, y, data, geom = "...", stat = "...", main = "title", xlab = "...", ylab = "...")	Another way to generate plots

Plotting (ggplot) (cont)

Scale	... + scale_x_continuous(limits = range(df$col)) + scale_y_continuous(breaks = seq(x, y, by=z))	Predefined axes limits
Scale log axes	... + scale_x_log10(); scale_x_log2()	Log scale axis
Scale color	... + scale_color_manual(values = c(col1 = "red", col2 = "blue")	Used when geom_func(mapping = aes(color = col)) is available; Manually define the color for each class in the aesthetic mapping
Hide axes labels	... + scale_x_continuous(labels = NULL)	Turn off axes labels
Hide legend	... + theme(legend.position="none")	Hide auto-legends due to aes(color)

Plotting (ggplot) (cont)

Modify axis, legend and plot labels	... + labs(title, subtitle, caption, x, y); xlab("..."); ylab("..."); ggtitle("...")	Modify axis, legend and plot labels
Facet	... + facet_wrap(~col, nrow = ?); facet_grid(col1~col2)	facet_grid -> 2d facet
Scatter plot	ggplot(data) + geom_point(mapping = aes(x, y), position = "jitter")	Create a scatterplot with y against x; position = "jitter" prevents overplotting
Line chart	ggplot(data) + geom_line(mapping = aes(x, y))	Connect the points using a line; Useful to plot residuals or time-plots

Plotting (ggplot) (cont)

Regression Line	ggplot(data) + geom_smooth(mapping = aes(x,y, color), method = "lm", se=F)	Draw a best fit line; se to include standard errors
Histogram	ggplot(data) + geom_histogram(mapping = aes(x = col1), binwidth = ?) + coord_cartesian(ylim = c(a,b))	Plot histogram of col1; zoom in using coord_cartesian()
Plot Regression line + Confidence Interval	predslm <- predict(lm_model, interval = "confidence"); combined_data <- cbind(data, predslm) ; ggplot(combined_data, aes(x, y)) + geom_point() + geom_ribbon(aes(ymin = lwr, ymax = upr, group=cyl, color=NULL), alpha=0.15) + geom_smooth(aes(y=fit), se=F, method='lm')	geom_ribbon plots out area, useful to plot confidence interval as controlled by alpha
Bar chart	ggplot(data) + geom_bar(mapping = aes(x, fill), stat="count", position = c("dodge", "stack"))	Each bar represents the count of unique values

Plotting (ggplot) (cont)

Identity bar chart	ggplot(data) + geom_bar(mapping = aes(x,y), stat="identity", position = c("dodge", "stack"))	Used for plotting a summarized dataframe where y=count
Boxplot	ggplot(data) + geom_boxplot(mapping = aes(x,y))	Boxplot where x is a categorical variable and y is continuous variable
Boxplot for continuous variables	ggplot(data) + geom_boxplot(mapping = aes(x,y, group = cut_width(x, ?))	cut_width(col, dbl) argument to bin the continuous variable into categories

Plotting (ggplot) (cont)

Count plots	ggplot(data) + geom_hex(mapping = aes(x, y))	To visualize relationship between 2 continuous variables and to reduce overplotting, bin each section by hexagons and its fill depends on the count of observations in the bin
Heatmaps	ggplot(data) + geom_tile(mapping = aes(x,y, fill))	Works on summarized dataframe; inputs x and y are categorical variables while the intensity is adjusted using the fill parameter

Plotting (ggplot) (cont)

Correlation heatmaps	library(reshape); cor_melt <- df %>% keep(is.numeric) %>% cor() %>% melt(); ggplot(cor_melt) + geom_tile(aes(Var1, Var2, fill=value))	1. Create correlation matrix with all the variables in the dataframe; 2. Melt the dataframe to only contain the columns ("Var1", "Var2", "value"); Plot the melted correlation dataframe using geom_tile()
Correlation Scatter Plot	library(GGally); df %>% keep(is.numeric) %>% ggpairs()	Scatter plot + kernel density at diagonals + pearson correlation

Plotting (ggplot) (cont)

Annotate at corner	abel <- mpg %>% summarise(col = ..., label = "....."); ... + geom_text(aes(label = label), data=label, vjust="top", hjust = "right")	1. Create a 1-row dataframe with the same columns with any value + label column for the text; 2.Apply geom_text() to annotate
Annotate	extra_df <- df %>% group_by(col_color) %>% summarise(col = median(col), ...); ggplot(df) + geom_point(aes(x,y,color = col_color) + ggrepel::geom_label_repel(aes(label = col_color), data = extra_df, size = ?)	Annotation is just another geom_func layer; Annotate the color "legend" based on label argument (x,y) input are the coordinates of the plot

Plotting (ggplot) (cont)

Saving plots	ggsave("filename.pdf", width = ?, height = ?, device = "pdf")	Save the latest ggplot