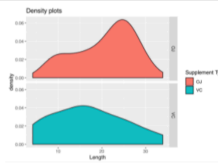
vector	an ordered gp of data of same type; generic vector has attributes	[1] is equivalent to [[1]] for atomic vectors basic=c(1,2,3) basic[5]=5 "1 2 3 NA 5" c takes input as vec elmts, creates a vector `c(1,2,3,4)+c(3,5) "4 7 6 9"`	atomic(logical,int,double,char,complex,raw) augmented: factors(built on top of int), dates and date times(numeric), data frames and tibbles(lists) Logical mode: vector can only have true or false; Requires less storage
list: rec vec can contain other lists	an ordered gp of any type	U1=c(203,204) U2=c(323,324,447) ms = list(U1=U1,U2=U2,Major="Statistics") content "323 324 447": ms[["U2"]] or ms$U2 list: ms[1] "204" ms[[c(1,2]] OR ms[[1]][2] OR ms[[1]][[2]] OR ms[1]$U1[2]	[[]] if you want to return multiple things in the list []: returns a list containing the vector; to get a collection of components name_list=c("mary","bob") mode(name_list) "character" return what is inside length(MS$U1) "2"
matrix	2D rectangular set of data of same type	mn[i,]: ith row mn[,j]: jth col mn[i,j]: ijth entry	X <- matrix(1:9,nrow=3,ncol=3) class(X) "matrix"; type of object structure mode(X) "numeric" attributes(X) $dim 3 3
array	can be greater than 2D	aperm(arr): permuting dimensions	typeof() give mode, tells what type of data is stored inside the array;

data frame	2D rec set of var(col) obs(row) any type	df_name$col: col	htru2_df <- read.csv(here("path/HTRU_2.csv"), header=FALSE)
			class(htru2_df) "data.frame"

tibble	Tibble is special data frame, designed to work with tidy data, consistent subsetting	tb[i]: tibble tb[[i]]: content carsdf[,1] %>% class(.) "numeric" carst[,1] %>% class(.) "tbl_df" "tbl" "data.frame"	read_csv class(htru2_tbl) "spec_tbl_df" "tbl_df" "tbl" "data.frame"
factor		R store factor var as numeric, store level as attributes;	

factor(data,levels=)	month_levels <- c("Jan", "Feb",) first need a list of valid levels y1 <- factor(x1, levels = month_levels) if omit the levels, alphabetical order
fct_lump	lump together least/most common factor levels into "other"
fct_reorder	fct_reorder(rincome, age); fac_reorder(code_lmp, n, .desc=true) was alphabetical order but order by desc freq bc n is count Write a line of code that will **create and assign the tibble** ontario_only used to create the plot in Figure 1. ontario_only <- can_pop10$ON %>% mutate(City=fct_reorder(City,Population))
fct_relevel	(<fct> or not, c("Monday",...number of levels that you want to move to the front of the line))

quantitative (numbers)	histogram: continuous, x has to be numbers boxplot: quantile, median -> the median of the OJ group is at the 75%ile of the VC group and the 25%ile of the OJ group is close to the median of the VC group. This shows the upward shift in central location more clearly than hist and density plot.	bar vs hist
bivariate quantitative	scatter with class info, SD, SD& skew [2 scatter plots, dose vs length -> the slope of the lines in the VC group is steeper, ie strength of association is stronger] 2D histogram 2D contour: The contours very much show the general location and spread of the three groups in relation to each other; similar to density 2D density(order matters for the layer) -> OJ supplement group has a more concentrated central location relative to the VC group, which has a very flat, spread out distribution	
qualitative (input as char)	mosaic: relative; class and freq, two qualitative var and measure the freq of intersection bar: discrete, categorical, x as qualitative var, if you care about count stacking bars: fct_infreq() to order levels in increasing freq	mosaic works for two characters variables, not work for int or dbl
interacting graphics	treemap: area of each square if relative to the proportion, bottom left to up right in desc order of size;	
ggplot	ggplot(data,aes(g=,x=,group=,col=,fill=category)); **fill=factor** ggplot(X,aes(x=x1,y=x2)) + geom () + ylim(c(-1,1)); ylim is the range of y axis ggplot(cleandata, aes(x=Age)) + geom_histogram(bins=25) facet_grid: forms a matrix of panels defined by row and column faceting variables; 2 discrete var facet_wrap: wraps a 1d sequence of panels into 2d; facet_wrap(~Month) return 12 graphs facet: take a discrete factor.. Add a grouping variable, split the data into two groups bquote: take the names inside the argument	**ggplot ignores the order in the table ->** have to do factor to main order in ggplot
	describe distribution: center, spread; left-skewed: right side is high	

FUNCTIONS: objects passed as arguments are copies, not original
functions: class of a function is function, body of a function is what's inside {}

```
p<-function(theta,x1,x2){
        1/(1+theta[1]*x1-theta[2]*x2)
}
L <- function(theta,x1,x2,y){
        -sum(y*log(p(theta,x1,x2))
}
```

	Write a line of code that will print the tibble row from can_pop10 which contains the data for Brampton. can_pop10$ON %>% filter(City=="Brampton") OR can_pop10$ON %>% slice(4) OR can_pop10$ON[[1]][4,]
filter	**pick observations by their values;** filter(cars,speed>24): output the rows that satisfies the condition
slice(htru2_tbl,1:4)	only 4 rows left
select(htru2 _tbl,Mean_IP,SD_IP)	only keep the selected columns, rows stay the same

arrange	select rows, small on top; reorder the rows
	ontario_only <- can_pop10$ON %>% arrange(desc(Population))
	ontario_only2 <- can_pop10$ON %>% mutate(City=fct_reorder(City,Population))

mutate	adds a new col/var
summarise	create a tibble, compute summary statistics; each function to one variable
summarise_all	apply these functions to all of the columns; give the output a different name; takes list **summarise_all(HTRU2,list(Avg=mean,Med=median))**
group_by	return new data set w group info; new row is the group

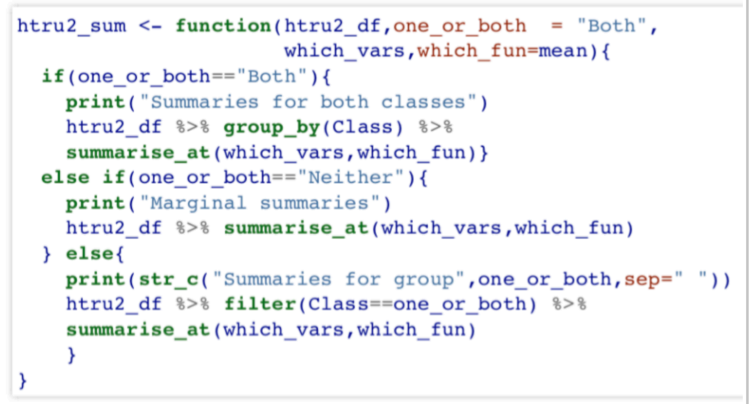
```
crime %>% group_by(DAY_OF_WEEK) %>% summarise(count=n()) %>%
  mutate(prop=count/sum(count))
```

```
# A tibble: 7 x 3
  DAY_OF_WEEK count  prop
  <chr>       <int> <dbl>
1 Friday      49758 0.152
2 Monday      46970 0.143
3 Saturday    45969 0.140
4 Sunday      41374 0.126
5 Thursday    47872 0.146
6 Tuesday     47726 0.146
7 Wednesday   48151 0.147
```

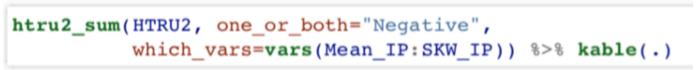
pivot_longer: put mul cols into one col; pivot_wider: put mul rows into one row	summary <- Diabetes %>% group_by(group) summarise_all(list(Avg=mean,Med=median)) %>% pivot_longer(cols=contains('_'), names_to = "Measure") %>% pivot_wider(id_cols=Measure, names_from=group) //group was rows, now change groups to columns, change the previous columns to the elements in Measure
%>%	pipe, reorder how you apply your function
str()	give list of var, mode, class of that object
glimpse()	$YEAR <dbl> 2018,2018,... $MONTH <dbl> 10,8,10,...
pull	return the vector you pulled on; crime %>% pull(OCCURRED_ON_DATE) %>% class(.) return "POSIXct" "POSIXt"
POSIXlt	If I pull a vector from a tibble, it returns an atomic vector
with()	a replacement of $; with(HTRU2,mean(Mean_IP))
bind_rows	my_results_tbl_0.1 %>% group_by(alpha) %>% summarise(Est=mean(VaR)) both_results_tbl <- bind_rows(my_results_tbl_0.1,my_results_tbl)

rbind
same column, put the rows together
	x <- read.csv("data1.csv",header=T,sep=",")
	x3 <- rbind(x,x2)
	then columns stay the same, x3 has both rows for x and x2

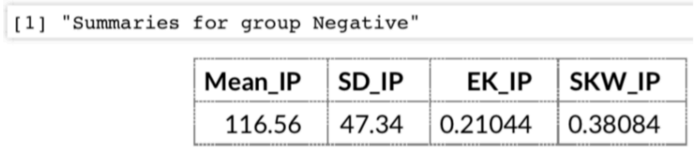
conditional control statements **#both is default value**



```
htru2_sum <- function(htru2_df,one_or_both  = "Both",
                      which_vars,which_fun=mean){
  if(one_or_both=="Both"){
    print("Summaries for both classes")
    htru2_df %>% group_by(Class) %>%
    summarise_at(which_vars,which_fun)}
  else if(one_or_both=="Neither"){
    print("Marginal summaries")
    htru2_df %>% summarise_at(which_vars,which_fun)
  } else{
    print(str_c("Summaries for group",one_or_both,sep=" "))
    htru2_df %>% filter(Class==one_or_both) %>%
    summarise_at(which_vars,which_fun)
    }
}
```



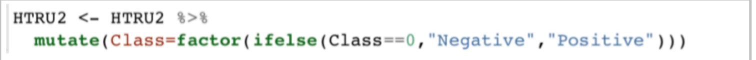
```
htru2_sum(HTRU2, one_or_both="Negative",
          which_vars=vars(Mean_IP:SKW_IP)) %>% kable(.)
```



```
[1] "Summaries for group Negative"
```

Mean_IP	SD_IP	EK_IP	SKW_IP
116.56	47.34	0.21044	0.38084

a <- c(1,2,3) ifelse(a%%2==0,"even","odd") "odd" "even" "odd"



```
HTRU2 <- HTRU2 %>%
  mutate(Class=factor(ifelse(Class==0,"Negative","Positive")))
```

simulation: to understand what would happen, if we assume distribution for certain variables, but interested for some function of those variables we are generating. We simulate different versions of reality to see what potential outcome will loook like over replicates.	Simulation is not actually random: you can control randomness (result will be be the same if all of the random things you do are the same in the same order)

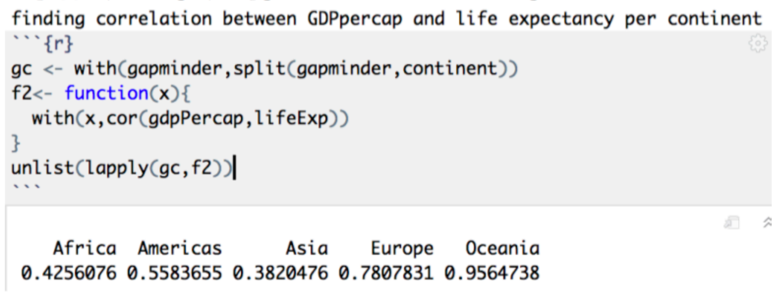
set.seed(19200)
J<-10
weights <- rep(50,J)
	rnorm(n,mean=,sd=)
	runif(n,mean=,max=)

one_VaR_sim <- function(reps= 1000,J,weights,sigma,alphas=c(0.95,0.99)){
VaR_alpha <- quantile(L_t_plus_1,alphas)
return(tibble(alpha=alpha,VaR=VaR_alpha))
}
	return value at risk for a bunch of replications
	
	generate 1000 replications, alpha quantiles = 99% value or risk under those hypothetical dist
	For one seed we do 1000 random replication of first block,

my_VaR_reps <- replicate(100,one_VaR_sim(reps= 100,J=J,weights=weights,sigma=sigma)) dim(my_VaR_rep) "2 100"	we see some variation in the values at risk, because the things we are generating are random; value at risk is sensitive to which thousands of replication we are looking at. We do replications to simulation the distribution of potential losses.

sappy(x, function)
x=List, vector,df
output vector,matrix
	Apply a function to all the elements of the input;
	correlations_sapply <- sapply(gap_list_country,cor_for_lapply)

lapply(x, function) x=List, vector or df output list; Apply a function to all the elements of the input gc is a list of 5 tibbles splits creates a list	MS[lapply(MS,length) < 4] gives the vector that has length <4 (code below)



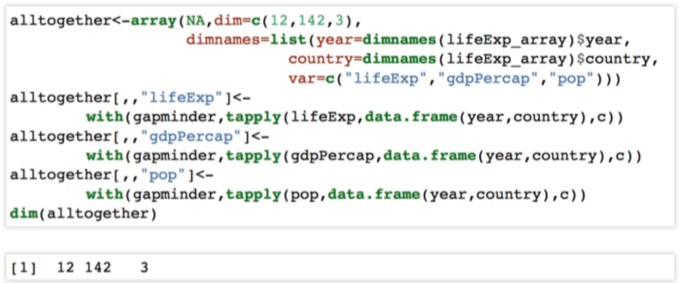
finding correlation between GDPpercap and life expectancy per continent

~~~
```{r}
gc <- with(gapminder,split(gapminder,continent))
f2<- function(x){
  with(x,cor(gdpPercap,lifeExp))
}
unlist(lapply(gc,f2))
```

    Africa  Americas      Asia    Europe   Oceania
 0.4256076 0.5583655 0.3820476 0.7807831 0.9564738
~~~

apply(x, MARGIN, FUN)
x=df or matrix
output vector, list, array
	Apply a function to the rows or columns or both
	myresults<-apply(aperm(my_VaR_reps,c(2,1)),2,unlist)

tapply (can be used as baseR group_by) input: can input arrays that are not consistent in size, or tibble output array? Apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.	(code below)



```
alltogether<-array(NA,dim=c(12,142,3),
                   dimnames=list(year=dimnames(lifeExp_array)$year,
                                 country=dimnames(lifeExp_array)$country,
                                 var=c("lifeExp","gdpPercap","pop")))
alltogether[,,"lifeExp"]<-
       with(gapminder,tapply(lifeExp,data.frame(year,country),c))
alltogether[,,"gdpPercap"]<-
       with(gapminder,tapply(gdpPercap,data.frame(year,country),c))
alltogether[,,"pop"]<-
       with(gapminder,tapply(pop,data.frame(year,country),c))
dim(alltogether)
```

```
[1]  12 142   3
```

want to find mean lifeExp per continent per year
lifeExp_matrix<-with(gapminder,tapply(lifeExp, data.frame(year,continent), c))

```
      continent
year   Africa Americas   Asia Europe Oceania
  1952 39.136   53.280 46.314 64.409  69.255
  1957 41.266   55.960 49.319 66.703  70.295
  1962 43.319   58.399 51.563 68.539  71.085
```

map or map_dbl(list or atomic vector,function)	transform their input by applying a function to each element and returning a vector the same length as the input; map = tidy lapply map returns a list; map_dbl returns the collapsed version of a certain type

map_dfr(list or atomic vector,function)	map_dfr(cars,sqfunction) returns a df with each element squared
	var_combs<-combn(names(HTRU2[,-9]),2) *## -9 excludes the 9th column, the Class variable*
	<u>Finally, produce the same tibble as in part (d), only using the var_combs matrix above and map_dfr(.). *Hint:* convert var_combs to a data.frame or tibble first.</u>
	map_test<-function(varnames){
	x1<-HTRU2[,varnames[1]] x2<-HTRU2[,varnames[2]] Class<-HTRU2$Class result<-optim(par=c(0,0,0),fn=L_theta,
	x1=x1,x2=x2, y=Class) return(tibble(Var1=varnames[1], Var2=varnames[2],
	Losses=result$value))
	}
	as_tibble(var_combs) %>% map_dfr(map_test) %>% arrange(Losses) %>% kable()