Dimension Reduction
Part 1: Summarizing and Understanding Your Data

Aram Balagyozyan

Department of Operations and Information Management
Kania School of Management
The
University of Scranton

March 10, 2022

Outline

1. Introduction
2. Data Summaries
3. Correlation Analysis
4. Reducing the Number of Categories in Categorical Variables

Outline

1. Introduction
2. Data Summaries
3. Correlation Analysis
4. Reducing the Number of Categories in Categorical Variables

Housing Prices in Boston Revisited

- We return to the Boston Housing example introduced in the Data Visualization module. For your reference again, below is the variables definition:

Variable	Description
CRIM	Crime Rate
ZN	Percentage of residential land zoned for lots over 25,000 sq. ft.
INDUS	Percentage of land occupied by non-retail business
CHAS	Does tract bound Charles River (1 if yes and 0 if no)
NOX	Nitric oxide concentration (parts per 10 million)
RM	Average number of rooms per dwelling
AGE	Percentage of owner-occupied homes built prior to 1940
DIS	Weighted distance to five Boston employment centers
RAD	Index of accessibility to radial highways
TAX	Full-value property tax rate per $10,000
PTRATIO	Pupil-to-teacher ratio by town
LSTAT	Percentage of lower status of the population
MEDV	Median Value of homes in $1000s
CAT.MEDV	Median value of homes in tract is above $30,000 (1 if yes and 0 if no)

Housing Prices in Boston Revisited

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	MEDV	CAT.MEDV
0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	24.0	0
0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6	0
0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	34.7	1
0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	33.4	1
0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	36.2	1
0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	5.21	28.7	0
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	12.43	22.9	0
0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	19.15	27.1	0
0.21124	12.5	7.87	0	0.524	5.631	100.0	6.0821	5	311	15.2	29.93	16.5	0
0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	17.10	18.9	0

- The above table represents the top 10 rows of `housing.df`.
- The first row represents the first neighborhood, which had an average per capita crime rate of 0.006, 18% of the residential land zoned for lots over 25,000 ft^2, 2.31% of the land devoted to non-retail business, no border on the Charles River, and so on.

Some Preliminaries

- In data mining, one often encounters situations where there is a large number of variables in the database. In such situations, it is likely that subsets of variables are highly correlated with each other.
- Including highly correlated variables in a classification or prediction model, or including variables that are unrelated to the outcome of interest, can lead to overfitting, and accuracy and reliability can suffer.
- In the artificial intelligence literature, dimension reduction is often referred to as *factor selection* or *feature extraction*.

Some Preliminaries

- In this module, we will learn few methods of dealing with situations when the predictor variables are either highly correlated with each other or all together irrelevant for predicting the outcome variable. Those methods include:
    1. Incorporating domain knowledge to remove or combine categories.
    2. Using data summaries to detect information overlap between variables.
    3. Using data conversion techniques such as converting categorical variables into numerical variables
    4. Employing automated reduction techniques, such as principal components analysis (PCA).
    5. Finally, we will briefly mention data mining methods such as regression models and classification and regression trees, which can be used for removing redundant variables and for combining "similar" categories of categorical variables.

Incorporating the Domain Knowledge.

- Although data mining prefers automated methods over domain knowledge, it is important at the first step of data exploration to make sure that the variables measured are reasonable for the task at hand.
- The integration of expert knowledge through a discussion with the data provider (or user) will probably lead to important insights.
- Practical considerations include:
  a) Which variables are most important for the task at hand, and which are most likely to be useless?
  b) Which variables are likely to contain much error?
  c) Which variables will be available for measurement (and what will it cost to measure them) in the future if the analysis is repeated?
  d) Which variables can actually be measured before the outcome occurs? For example, if we want to predict the closing price of an ongoing online auction, we cannot use the number of bids as a predictor because this will not be known until the auction closes.

Outline

1. Introduction
2. Data Summaries
3. Correlation Analysis
4. Reducing the Number of Categories in Categorical Variables

Data Summaries

- The importance of getting familiar with data cannot be overstated. The better you understand the data, the better wil be the results from the modeling or mining process.
- Data visualization is an important part of this process. Numerical summaries of the data are also very helpful for data reduction. The information that they convey can assist in combining categories of a categorical variable, in choosing variables to remove, in assessing the level of information overlap between variables, and more.

Summary Statistics

- Base R has several functions and facilities that summarize data
- `summary(df)` gives an overview of of the entire set of variables in data `df`.
- The problem with `summary()` is that it produces only few predetermined descriptors of data such as the minimum, median, mean, maximum, 1st and 3rd quartiles, as well as the number of missing values.
- What if you wanted to learn about say the standard deviation or other measures of the data?
- For each variable in your data, you could run the `sd()` or other descriptive stat function.
- However, this approach is too laborious and doesn't allow for a direct comparison of the descriptive statistics of different variables.
- The code on the following slide produces a set of descriptive statistics for all variables in `housing.df` dataset.

Summary Statistics

```r
# import the Boston Housing data into R and
# compute mean, standard dev., min, max, median, length,
# and missing values for all variables
library(readr)
housing.df<-read_csv("BostonHousing.csv")
data.frame(mean=sapply(housing.df, mean),
            sd=sapply(housing.df, sd),
            min=sapply(housing.df, min),
            max=sapply(housing.df, max),
            median=sapply(housing.df, median),
            length=sapply(housing.df, length),
            miss.val=sapply(housing.df,
          function(x) length(which(is.na(x)))
                            )
           )
```

Summary Statistics

	mean	sd	min	max	median	length	miss.val
CRIM	3.6135236	8.6015451	0.00632	88.9762	0.25651	506	0
ZN	11.3636364	23.3224530	0.00000	100.0000	0.00000	506	0
INDUS	11.1367787	6.8603529	0.46000	27.7400	9.69000	506	0
CHAS	0.0691700	0.2539940	0.00000	1.0000	0.00000	506	0
NOX	0.5546951	0.1158777	0.38500	0.8710	0.53800	506	0
RM	6.2846344	0.7026171	3.56100	8.7800	6.20850	506	0
AGE	68.5749012	28.1488614	2.90000	100.0000	77.50000	506	0
DIS	3.7950427	2.1057101	1.12960	12.1265	3.20745	506	0
RAD	9.5494071	8.7072594	1.00000	24.0000	5.00000	506	0
TAX	408.2371542	168.5371161	187.00000	711.0000	330.00000	506	0
PTRATIO	18.4555336	2.1649455	12.60000	22.0000	19.05000	506	0
LSTAT	12.6530632	7.1410615	1.73000	37.9700	11.36000	506	0
MEDV	22.5328063	9.1971041	5.00000	50.0000	21.20000	506	0
CAT.MEDV	0.1660079	0.3724560	0.00000	1.0000	0.00000	506	0

- Notice how different variables have very different ranges of values. Also, comparing the mean and median measures for each variable, we can see how many of them are skewed. These conclusions are in line with the implications of the graphical analysis that we conducted in the previous module.
- The following few slides explain how the code above works.

sapply()

- `sapply(x,fun)` is another member of the `apply` family of functions. It takes in a vector (or any other object) `x`, applies `fun` to each element of `x`, and returns a **vector** (or matrix) of results.

```
movies <- data.frame(name=c("SPYDERMAN","BATMAN",
                            "VERTIGO","CHINATOWN"),
                     genre=c("Marvel Comic",
                             "Superhero",
                             "Suspense",
                             "Film Noir"))
# convert all elements to lower case
sapply(movies, tolower)
```

```
##      name        genre
## [1,] "spyderman" "marvel comic"
## [2,] "batman"    "superhero"
## [3,] "vertigo"   "suspense"
## [4,] "chinatown" "film noir"
```

sapply()

- Notice that running `tolower(movies)` would result in nonsense.

```
tolower(movies)
```

```
## [1] "c(\"spyderman\", \"batman\", \"vertigo\", \"chinatow
## [2] "c(\"marvel comic\", \"superhero\", \"suspense\", \"f
```

- However, you could run `tolower()` for each column alone (that are vectors when taken alone).

```
tolower(movies$name)
```

```
## [1] "spyderman" "batman"    "vertigo"   "chinatown"
```

- `sapply(movies, tolower)` cycles through each column of data frame `housing.df` treating each column as a vector and produces a matrix of results converted into lowercase.
- Thus, for given `FUN`, `sapply(housing.df, FUN)` cycles through each column of `housing.df` and return the application of `FUN` to that column

`function(x) length(which(is.na(x)))`

- Instead of feeding `sapply()` with a generic function (such as `mean`, `median`, etc.), you can provide your own function in the same way as the code that generates the summary statistics does.
- All you have to do is type `function(x)` and then what that the function must do.
- To understand what `length(which(is.na(x)))` does, consider the following code:

```
library(dplyr)
#generate a 5x1 vector and subs. 2 vals. w. NAs
source("random_NAs.R")
vec<- matrix(1:5,5,1) %>% random_NAs(2)
```

function(x) length(which(is.na(x)))

```r
is.na(vec)
```

```
##       [,1]
## [1,]  TRUE
## [2,] FALSE
## [3,] FALSE
## [4,]  TRUE
## [5,] FALSE
```

```r
which(is.na(vec))
```

```
## [1] 1 4
```

```r
length(which(is.na(vec)))
```

```
## [1] 2
```

- Thus, `length(which(is.na(x)))` returns the number of `NA`s in vector `x`.

Aggregation

- In modules dedicated to Getting and Cleaning Data and Data Visualization respectively, we saw how to use functions `table()` and `aggregate()`. Both are very useful for aggregating data, the former for obtaining counts and the latter for all sorts of summary statistics.
- Just as a refresher, examples of both are provided below.

```r
# Create room categories
housing.df$RM.bin<-.bincode(housing.df$RM, c(1:9))
# Obtain counts
table(housing.df$CAT.MEDV, housing.df$RM.bin)
```

```
##
##       3   4   5   6   7   8
##   0   2  13 156 244   6   1
##   1   0   1   1  25  45  12
```

- The first row is the count of neighborhoods with `CAT.MEDV=0` while the second row is the count of neighborhoods with `CAT.MEDV=1` falling in each `RM.bin` category (columns).

.bincode()

- .bincode(x, bins) of the base R package takes a numerical vector x and a numeric vector of two or more cut points, sorted in increasing order (passed on in bins), and returns a numerical vector of the same size as x representing what bin each member of x is falling.

```r
x<-c(0,1.2,2,3,3.5,7.1,10)
bn<-c(0:10)
.bincode(x,bn)
```

```
## [1] NA  2  2  3  4  8 10
```

- Thus, .bncode() creates a set of semi-closed intervals (0,1],(1,2],(2,3]...(9,10] and when a number in x falls in a given interval, it returns a label for that number being equal to the upper limit of the interval it falls into.

Pivot Table

- `aggregate()` of the base R package (covered in the Data Visualization module) can do more than providing you with simple counts, it can apply all kinds of aggregating functions (such as `mean`, `median`, etc.) to specific subgroups of data.

```
tbl<-aggregate(housing.df$MEDV, by=list
  (RM.bin=housing.df$RM.bin, CHAS=housing.df$CHAS),
  FUN=mean)
tbl
```

```
##    RM.bin CHAS        x
## 1       3    0 25.30000
## 2       4    0 15.40714
## 3       5    0 17.20000
## 4       6    0 21.76917
## 5       7    0 35.96444
## 6       8    0 45.70000
## 7       5    1 22.21818
## 8       6    1 25.91875
## 9       7    1 44.06667
## 10      8    1 35.95000
```

Pivot Table

- The above code computes the average of `MEDV` by `RM.bin` and `CHAS`
- In using aggregate() use the argument `by=` to define the list of aggregating variables, and `FUN=` to provide the aggregating function.
- The pivot table produced by the `aggregate()` function above doesn't have exactly the same shape as a pivot table that Excel would produce. To get to the same Excel shape, use the `cast()` function of the **reshape** package.

Pivot Table

```r
library(reshape)
cast(tbl, RM.bin ~ CHAS,
     margins = c("grand_row","grand_col"), mean)
```

```
##   RM.bin        0        1    (all)
## 1      3 25.30000      NaN 25.30000
## 2      4 15.40714      NaN 15.40714
## 3      5 17.20000 22.21818 19.70909
## 4      6 21.76917 25.91875 23.84396
## 5      7 35.96444 44.06667 40.01556
## 6      8 45.70000 35.95000 40.82500
## 7  (all) 26.89013 32.03840 28.94944
```

- The above is the same pivot table that was produced earlier using the `aggregate()` function, except it is in an Excel-like form with the horizontal and vertical margins representing the column and row averages respectively.
- Columns labels `0` and `1` stand for `CHAS=0` and `CHAS=1` respectively

cast()

- To understand the cast() function better recall that in the Data Visualization module we used melt(), another function of the **reshape** package. As a refresher, below is the melt() example that we used in the Data Vis. module.

```
champs
```

```
##      pullups pushups
## John       1       3
## Paul       2       4
```

```
melt(champs)
```

```
##     X1      X2 value
## 1 John pullups     1
## 2 Paul pullups     2
## 3 John pushups     3
## 4 Paul pushups     4
```

cast()

- In essence, cast() is the opposite of melt(), it can transform “molten” data back to its original format:

```
moltenchamps <- melt(champs)
cast(moltenchamps,X1 ~ X2)
```

```
##      X1 pullups pushups
## 1 John       1       3
## 2 Paul       2       4
```

- Thus, cast() transforms data supplied in a long “molten” format back to a normal, wide format.

Outline

1. Introduction
2. Data Summaries
3. Correlation Analysis
4. Reducing the Number of Categories in Categorical Variables

Correlated Predictor Variables (Features)

- One characteristic of a well-functioning predictive or classification model is parsimony, when few uncorrelated predictors together do a good job in predicting or classifying the outcome variable.
- There are at least a couple of reasons for this.
- One is that by adding additional variables we increase the complexity of the model exponentially. For example, on a 2-dimensional chess board (2 predictors), a chess piece can be on any of the 64 squares. If the game of chess had 3-dimensions (only one more dimension than traditional chess), a piece could be in any of the 512 cubes (8^3).
- Another case for parsimony is that by including more and more predictors into the model, we are likely to introduce the problem of *multicolinearity*, a situation when one or more predictors are correlated with each other.
- Under multicolinearity, when model contains two or more strongly correlated predictors, it gets "confused" and researchers fail to pinpoint which predictor is more potent.

Correlated Predictor Variables (Features)

- In datasets with a large number of variables (which are likely to serve as predictors), there is usually much overlap in the information covered by the set of variables.
- One simple way to find redundancies is to look at the correlation matrix or correlation heatmap.
- Pairs that have a very strong (positive or negative) correlation contain a lot of overlap in information and are good candidates for data reduction by removing one of the variables.
- Removing variables that are strongly correlated with others is useful for avoiding multicollinearity problems that can arise in various models.
- The PCA method that we will discuss at the end of this module is an effective and automatic way of dealing with correlated features.

Outline

1. Introduction
2. Data Summaries
3. Correlation Analysis
4. Reducing the Number of Categories in Categorical Variables

Reducing the Number of Categories in Categorical Variables

- When a categorical variable has several categories, and the variable is destined to be a predictor, many data mining algorithms will require converting the categorical variable into several dummy variables.
- For example, suppose your table contains 9 rows and 3 columns and one of the columns is a categorical variable with 4 categories:

rowid	semester	enrollment
1	Spring	23
2	Summer	89
3	Fall	53
4	Winter	10
5	Winter	6
6	Winter	54
7	Spring	94
8	Summer	11
9	Summer	3

Reducing the Number of Categories in Categorical Variable

- Certain if not most data mining algorithms may require that you convert this table into the following shape:

rowid	enrollment	semester_Fall	semester_Spring	semester_Summer
1	23	0	1	0
2	89	0	0	1
3	53	1	0	0
4	10	0	0	0
5	6	0	0	0
6	54	0	0	0
7	94	0	1	0
8	11	0	0	1
9	3	0	0	1

- Note that although, the `semester` variable has 4 categories, you need only 3 (i.e. 4-1) dummy variables.
- The problem is that if a categorical variable assumes not 4 but many categories, the modification above will lead to many new variables in your data set.

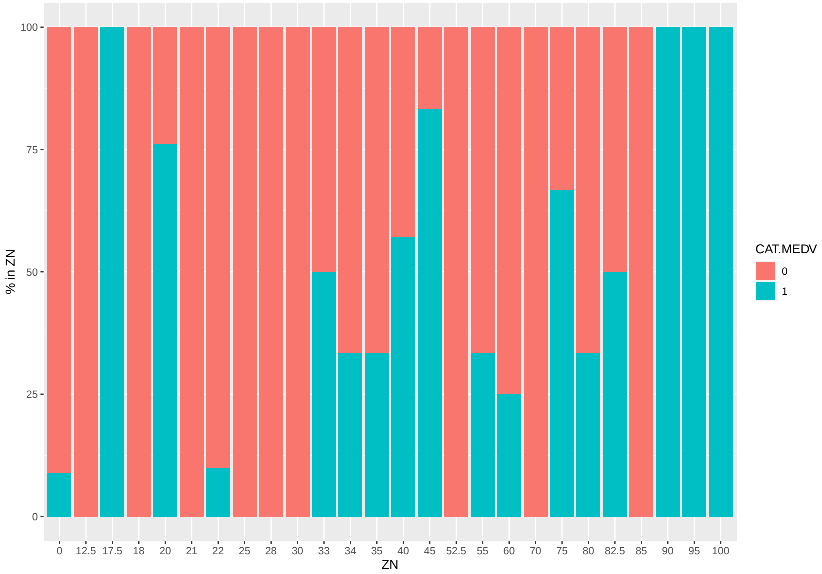
Reducing the Number of Categories in Categorical Variable

- One way of dealing with the issue is to combine close or similar categories. This may requires incorporating expert knowledge and common sense. Use only the categories that are most relevant to the analysis and label the rest as "other."
- Generally, categories that contain very few observations are good candidates for combining with other categories. Pivot tables are useful for this task.
- In classification tasks (with a categorical outcome variable), a pivot table broken down by the outcome classes can help identify categories that do not separate the classes. Those categories too are candidates for inclusion in the "other" category.
- As an example, suppose we are trying to predict `CAT.MEDV` in the Boston Housing data using `ZN` as a predictor. For this purpose, we are interested in what percentage of each `ZN` category consists of each `CAT.MEDV` category.

Reducing the Number of Categories in Categorical Variable

```r
library(dplyr)
library(ggplot2)
#Get the count of neignborhoods for each ZN & CAT.MEDV
tbl <- count(housing.df,ZN,CAT.MEDV)
#Get the count of neignborhoods in each ZN
agg <- count(housing.df,ZN)
names(agg)<-c("ZN","TotalInZn")
#(left_)join tbl and agg
agg <- left_join(tbl,agg, by="ZN")
#obtain percentages of each CAT.MEDV category
# in each ZN. More on left_join() below.
agg$PercentInZn <- agg$n / agg$TotalInZn*100
#plot
ggplot(agg) + geom_col(aes(x = factor(ZN),
                           y = PercentInZn,
                           fill=factor(CAT.MEDV))) +
    labs(x="ZN",y="% in ZN", fill='CAT.MEDV')
```

Reducing the Number of Categories in Categorical Variable



Reducing the Number of Categories in Categorical Variable

- We can see that the distribution of `CAT.MEDV` is identical for `ZN` = 17.5, 90, 95, and 100 (where all neighborhoods have `CAT.MEDV` = 1). These four categories can then be combined into a single category.
- Similarly, categories `ZN` = 12.5, 18, 25, 28, 30, 52.5, 70, and 85 can be combined.
- Further combinations are also possible based on similar bars.

`left_join()` function of the **dplyr** package

- The chunk of code above relied on the `left_join()` function of the **dplyr** package that belongs to the family of `join` functions.
- These functions are very handy when it comes to joining together two or more tables. Let's take a moment and go over few of these functions.
- `left_join(x,y,by)` returns all rows from `x`, and all columns from `x` and `y`. Rows in `x` with no match in `y` will have `NA` values in the new columns. If there are multiple matches between `x` and `y`, all combinations of the matches are returned.
- Consider the following simple example.

left_join() function of the **dplyr** package

```r
band_members # data from the dplyr package
```

```
## # A tibble: 3 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```r
band_instruments # data from the dplyr package
```

```
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

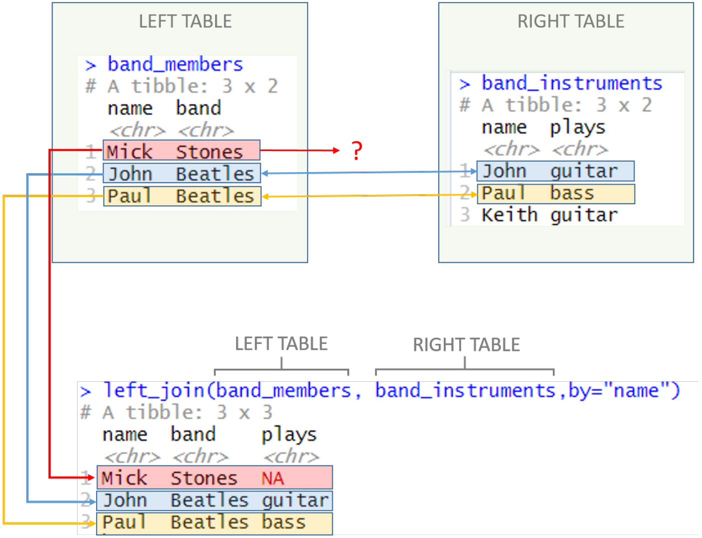
left_join() function of the **dplyr** package

```
left_join(band_members, band_instruments,by="name")
```

```
## # A tibble: 3 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
```

`left_join()` function of the **dplyr** package

- Below is the breakdown of what `left_join()` does:



```
> band_members
# A tibble: 3 x 2
  name  band
  <chr> <chr>
1 Mick  Stones
2 John  Beatles
3 Paul  Beatles
```

```
> band_instruments
# A tibble: 3 x 2
  name  plays
  <chr> <chr>
1 John  guitar
2 Paul  bass
3 Keith guitar
```

```
> left_join(band_members, band_instruments,by="name")
# A tibble: 3 x 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  NA
2 John  Beatles guitar
3 Paul  Beatles bass
```

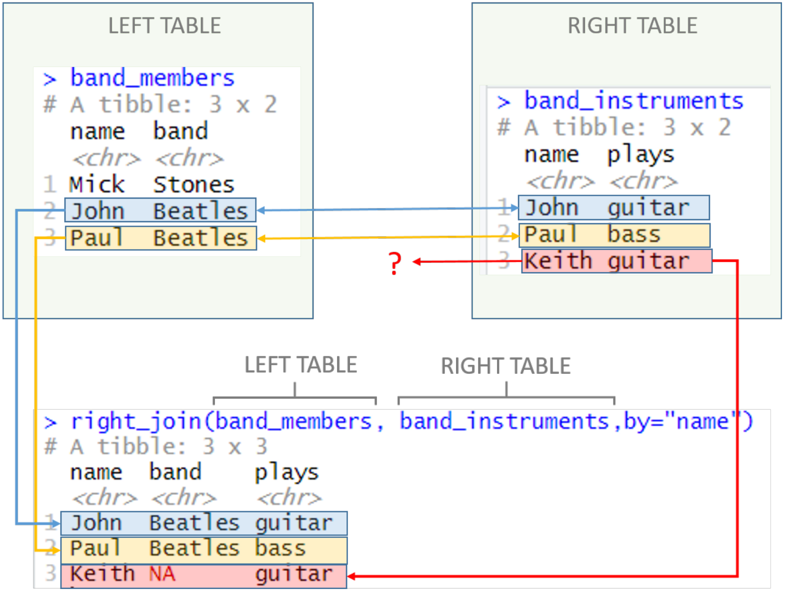
right_join() function of the **dplyr** package

- right_join(x,y,by) returns all rows from y, and all columns from x and y. Rows in y with no match in x will have NA values in the new columns. If there are multiple matches between x and y, all combinations of the matches are returned.

```r
right_join(band_members, band_instruments,by="name")
```

```
## # A tibble: 3 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 John  Beatles guitar
## 2 Paul  Beatles bass
## 3 Keith <NA>    guitar
```

`right_join()` function of the **dplyr** package



```
> band_members
# A tibble: 3 x 2
  name  band
  <chr> <chr>
1 Mick  Stones
2 John  Beatles
3 Paul  Beatles
```

```
> band_instruments
# A tibble: 3 x 2
  name  plays
  <chr> <chr>
1 John  guitar
2 Paul  bass
3 Keith guitar
```

```
> right_join(band_members, band_instruments,by="name")
# A tibble: 3 x 3
  name  band    plays
  <chr> <chr>   <chr>
1 John  Beatles guitar
2 Paul  Beatles bass
3 Keith NA      guitar
```

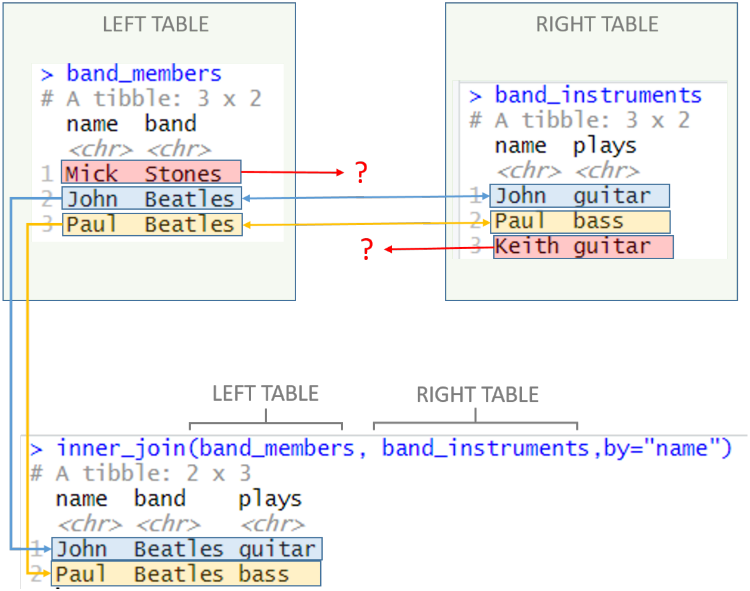
inner_join() function of the **dplyr** package

- inner_join(x,y,by) returns all rows from x where there are matching values in y, and all columns from x and y. If there are multiple matches between x and y, all combination of the matches are returned.

```
inner_join(band_members, band_instruments,by="name")
```

```
## # A tibble: 2 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 John  Beatles guitar
## 2 Paul  Beatles bass
```

`inner_join()` function of the **dplyr** package



```
> band_members
# A tibble: 3 x 2
  name  band
  <chr> <chr>
1 Mick  Stones
2 John  Beatles
3 Paul  Beatles
```

```
> band_instruments
# A tibble: 3 x 2
  name  plays
  <chr> <chr>
1 John  guitar
2 Paul  bass
3 Keith guitar
```

```
> inner_join(band_members, band_instruments,by="name")
# A tibble: 2 x 3
  name  band    plays
  <chr> <chr>   <chr>
1 John  Beatles guitar
2 Paul  Beatles bass
```

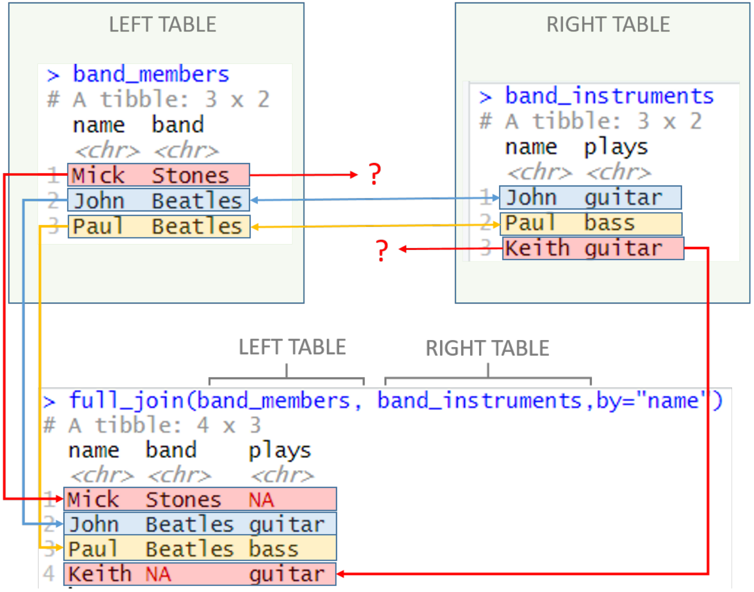
full_join() function of the **dplyr** package

- full_join(x,y,by) returns all rows and all columns from both x and y. Where there are not matching values, returns NA for the one missing.

```
full_join(band_members, band_instruments,by="name")
```

```
## # A tibble: 4 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
## 4 Keith <NA>    guitar
```

full_join() function of the **dplyr** package



```
> band_members
# A tibble: 3 x 2
  name  band
  <chr> <chr>
1 Mick  Stones
2 John  Beatles
3 Paul  Beatles
```

```
> band_instruments
# A tibble: 3 x 2
  name  plays
  <chr> <chr>
1 John  guitar
2 Paul  bass
3 Keith guitar
```

```
> full_join(band_members, band_instruments,by="name")
# A tibble: 4 x 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  NA
2 John  Beatles guitar
3 Paul  Beatles bass
4 Keith NA      guitar
```