Notes for the BAN400 Exam

Table of contents

1 Functions

1.1 Basic functions

Function	Package	Description
mean()	base	Calculates the mean of a vector of numbers
median()	base	Calculates the median of a vector of numbers
sd()	base	Calculates the standard deviation of a vector of numbers
var()	base	Calculates the variance of a vector of numbers
sum()	base	Calculates the sum of a vector of numbers
c()	base	Creates vector
length()	base	The number of elements in a vector or list
ncol()	base	Number of columns of data frame or matrix
nrow()	base	Number of rows of data frame or matrix
min()	base	The smallest value in a set
max()	base	The largest value in a set

1.2 Math

Function	Package	Description
sqrt()	base	Calculates square root of number or vector of numbers
abs()	base	Calculates absolute value of number or vector of numbers
sign()	base	Returns the sign of x
round()	base	Rounds x to n decimal places
ceiling()	base	Rounds up to the nearest integer
floor()	base	Rounds down to the nearest integer
cumsum()	base	Cumulative sum
cor()	base	Correlation

1.3 Reading data

Function	Package	Description
read_delim()	readr	Read file with columns separated by any delimiter
read_csv()	readr	Read csv-file (comma separated values)
read_csv2()	readr	Uses semicolons as separators and keeps commas
read_excel()	readxl	Read data from excel files
read_fwf()	readr	Reads fixed width data
read_tsv()	readr	Reads tab separated values

1.4 Data wrangling

Function	Package	Description
head()	base	Returns the first few rows of a data frame or vector
tail()	base	Returns the first few rows of a data frame or vector
filter()	dplyr	Returns elements that satisfy conditions
select()	dplyr	Choose specific columns from a data frame
arrange()	dplyr	Sorts rows of a data frame by specified columns
sort()	base	Sorts a vector in ascending or descending order
mutate()	dplyr	Adds or modifies columns in a data frame
transmutate()	dplyr	Creates a new data frame containing only the specified computations
summarise()	dplyr	Summary statistics for columns in a data frame (typically used with grouped data)
group_by()	dplyr	Group data by one or more variables
ungroup()	dplyr	Ungroup data such that subsequent operations to apply to the entire dataset
left_join()	dplyr	Returns all values from the first data frame with all columns and values from the second data frame where there is a match
inner_join()	dplyr	Joins two data frames by keeping only records that match in both data sets
right_join()	dplyr	Returns all values from the second data frame with matching columns and values from the first data frame where there is a match
full_join()	dplyr	Returns all values and columns from both data frames and filling in NA where there is no match
semi_join()	dplyr	Filters the first data frame keeping only rows with matching keys in the second data frame
anti_join()	dplyr	Filters the first data frame to keep only rows with no match in the second data frame

1.5 Machine learning

Function	Package	Description
logistic_reg()	tidymodels	Specifies a logistic regression model
set_engine()	tidymodels	Specifies the computational engine for a model
set_mode()	tidymodels	Sets the mode (e.g. classification or regression) for a model
fit()	tidymodels	Fits the model to data
nearest_neighbor()	tidymodels	Specifies a k-nearest neighbors model
tune()	tidymodels	Marks a parameter for tuning in a model
finalize_workflow()	tidymodels	Finalizes the workflow with specific parameters
workflow()	tidymodels	Creates a workflow object
add_model()	tidymodels	Adds a model to a workflow
add_recipe()	tidymodels	Adds a recipe to a workflow
tune_grid()	tidymodels	Tunes hyperparameters across a grid of values
select_best()	tidymodels	Selects the best tuning parameter combination based on a metric

1.6 Many models

Function	Package	Description
add_predictions()	modelr	Adds model predictions to a data frame
add_residuals()	modelr	Adds residuals from a model to a data frame
group_by()	dplyr	Groups data by one or more variables
ungroup()	dplyr	Removes grouping structure from a data frame
nest()	tidyr	Creates a nested data frame by collapsing rows into list-columns
unnest()	tidyr	Expands list-columns back into regular columns
select()	dplyr	Selects specific columns from a data frame
pull()	dplyr	Extracts a single column as a vector
pluck()	purrr	Extracts an element from a list or vector by index or name
map()	purrr	Applies a function to each element of a list or vector
map2()	purrr	Applies a function to pairs of elements from two lists
glance()	broom	Generates a summary of model diagnostics in a tidy format

1.7 Parsing

Function	Package	Description
class()	base	Returns the class of an object
typeof()	base	Determines the type or storage mode of any object
mode()	base	Indicates the mode of storage
as.numeric()	base	Converts data to numeric format
as.character()	base	Converts data to character format
as.factor()	base	Converts data to factor format
gsub()	base	Find and replace
reshape()	base	Reshape datasets between wide and long formats
pivot_longer()	tidyr	Convert wide data to long format
pivot_wider()	tidyr	Convert long data to wide format.
na.omit()	base	Remove missing values
is.na()	base	Check for missing values
ymd()	lubridate	Parse dates
hms()	lubridate	Parse times
parse_datetime()	readr	Parse datetimes
separate()	tidyr	Split one column into multiple columns

1.8 Selecting

Function	Package	Description
any_of()	dplyr	Selects columns that match any of the given names in a vector
all_of()	dplyr	Selects columns that match all the given names in a vector
starts_with()	dplyr	Selects columns whose names start with a specified prefix
ends_with()	dplyr	Selects columns whose names end with a specified suffix
contains()	dplyr	Selects columns whose names contain a specified string
matches()	dplyr	Selects columns whose names match a specified regular expression
select()	dplyr	Selects specified columns from a data frame
slice_min()	dplyr	Selects rows with the smallest values of a variable
slice_max()	dplyr	Selects rows with the largest values of a variable

2 Topics

2.1 Filtering

The symbol | works as an "or" operator, meaning it will return items that satisfy either one or both of the conditions before and after the operator.

```
filter(flights, dest == "IAH" | dest == "HOU")
```

The & is an "and" operator, meaning both conditions need to hold. This is useful in conjunction with the |-operator, since otherwise you can simply just add filters after each other like this:

```
filter(!is.na(flight2), !is.na(flight1))
```

Another operator %in% returns TRUE if an item exists inside a vector:

```
flights %>%
  filter(dest %in% c("IAH", "HOU")
```

To create the opposite logic, i.e. "not in c()", you can use !:

```
flights %>%
  filter(!(dest %in% c("IAH", "HOU")))
```

2.2 Selecting

Some basic selection functions from dplyr are starts_with, ends_with and contains.

```
select(flights, starts_with("dep_"), starts_with("arr_"))
```

2.2.1 Regex (regular expressions)

The function matches uses *regular expressions*. These can be used to match precise patterns.

```
select(flights, matches("^(dep|arr)_(time|delay)$"))
```

Some common Regex meta characters are:

- . Matches any character except newline.
- ^ Matches the beginning of a string.
- $ Matches the end of a string.
- [] Matches any one of the characters inside the brackets.
- | Logical OR.
- * Matches 0 or more repetitions of the preceding character.
- + Matches 1 or more repetitions.
- ? Matches 0 or 1 repetition (optional match).
- {n,m} Matches between n and m repetitions.

2.3 Loops and iterations

2.3.1 Standard for-loop

```
for(i in 1:n) {
  ... do something with i...
}
```

Note that we can iterate over any type of vector, not just numbers, and we can give the iteration variable any name we want. In the example above it is i.

2.3.2 While loop

Repeat until a certain condition is met. For example

```
i <- 1
while(i < 10) {
  print(i)
  i <- i + 1
}
```

2.4 Math

The Integer Division Operator %/% performs integer division. It divides two numbers and returns the whole number part of the quotient, discarding any remainder. Example:

```
10 %/% 3  # Returns 3 (quotient without remainder)
```

The Modulo Operator %% operator returns the remainder from the division of two numbers. Example:

```
10 %% 3  # Returns 1 (remainder)
```

2.5 Plotting

We use `ggplot2` as the standard package for plotting, and the main function is `ggplot`. We supply a data frame to the first argument and an aesthetic mapping to the second argument. We add *layers* of plotting components using the plus sign. A simple example:

```
ggplot(df, aes(x = x_variable, y = y_variable, colour = grouping_variable)) +
  geom_point()
```

Many types of layers may contain other data sets via the `data` argument and/or updated aesthetic mappings via the `mapping` argument. Data and mappings are typically inherited from the layer above if not specified in a new layer. There are many types of functions for making further adjustments to labels, titles, axes and other properties. A more complete example may look like this:

```
ggplot(df, aes(x = x_variable, y = y_variable)) +
  geom_point() +
  geom_point(mapping = aes(x = new_x_variable, y = new_y_variable, colour = grouping_var),
             data = new_df) +
  xlab("X label") +
  ylab("Y label") +
  ggtitle("Title of plot") +
  theme_minimal()
```

2.5.1 Statistics

Most geoms come in pairs with complementary statistics arguments that are almost always used in concert. These functions can be used to retrieve the data that is used to generate the plot. For example, for these geoms:

```
geom_smooth()
geom_dotplot()
geom_point()
geom_bar()
```

We have these `stat` functions respectively:

```
stat_smooth()
stat_bindot()
stat_qq()
stat_count()
```

The corresponding stat functions can be found by reading the documentation with for example `?geom_smooth`.

2.6 Reading data

Sometimes data contains the delimiter. In that case, use quote argument to escape:

```
read_csv("x,y\n1,'a,b'", quote="'")
```

2.7 Parsing data

There are various parsing functions from `lubdridate` library, such as `ymd()`. These functions can be used to convert objects into time and date formats.

2.7.1 Dates and time

Examples:

```
d1 <- "January 1, 2010"
d2 <- "2015-Mar-07"
d3 <- "06-Jun-2017"
d4 <- c("August 19 (2015)", "July 1 (2015)")
d5 <- "12/30/14" # Dec 30, 2014
t1 <- "1705"
t2 <- "11:15:10.12 PM"
```

Can be parsed respectively:

```
mdy(d1)     #> [1] "2010-01-01"
ymd(d2)     #> [1] "2015-03-07"
dmy(d3)     #> [1] "2017-06-06"
mdy(gsub("\\(", "", gsub("\\)", "", d4)))  #> [1] "2015-08-19" "2015-07-01"
mdy(d5)     #> [1] "2014-12-30"
hm(t1)      #> [1] "17:05:00"
hms(t2)     #> [1] "23:15:10.12"
```

2.7.2 Pivoting

The library `tidyr` provides various ways to pivot data. An example of a pivot is from

```
preg <- tribble(
  ~pregnant, ~male, ~female,
  "yes", NA, 10,
  "no", 20, 12
)
```

To a longer data format:

```
preg_tidy2 <- preg %>%
  pivot_longer(c(male, female), names_to = "sex", values_to = "count", values_drop_na = TRUE)
```

2.7.3 Splitting columns

You can split a column into multiple columns using `separate()` and `extract()` from `tidyr`:

```
tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>%
  separate(x, c("one", "two", "three"))
```

In the above case, the second item has four “values” `"d,e,f,g"`. In this case, the `g` is dropped. There are various arguments in the functions to determine how to deal with extra or missing items like this.

2.7.4 Basic find and replace

You can perform a basic find-and-replace operation on a vector like this:

```
y[y == "find"] <- "replace"
```