SECTION 1

Data retrieval with SQL

1.1 General concepts

❒ **Structured Query Language** – Structured Query Language, abbreviated as SQL, is a language that is largely used in the industry to query data from databases.

❒ **Query structure** – Queries are usually structured as follows:

```SQL
-- Select fields                        mandatory
SELECT
      col_1,
      col_2,
        ... ,
      col_n

-- Source of data                       mandatory
FROM table t

-- Gather info from other sources       optional
JOIN other_table ot
   ON (t.key=ot.key)

-- Conditions                           optional
WHERE some_condition(s)

-- Aggregating                          optional
GROUP BY column_group_list

-- Sorting values                       optional
ORDER BY column_order_list

-- Restricting aggregated values       optional
HAVING some_condition(s)

-- Limiting number of rows              optional
LIMIT some_value
```

Remark: the SELECT DISTINCT command can be used to ensure not having duplicate rows.

❒ **Condition** – A condition is of the following format:

```SQL
some_col some_operator some_col_or_value
```

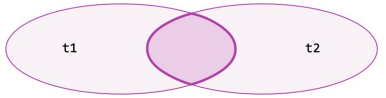
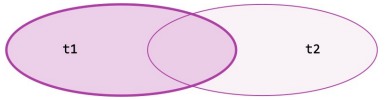
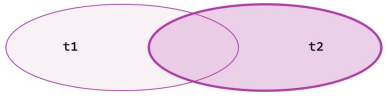
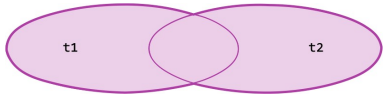
where some_operator can be among the following common operations:

Category	Operator	Command
General	Equality / non-equality	= / !=, <>
	Inequalities	>=, >, <, <=
	Belonging	IN(val_1, ..., val_n)
	And / or	AND / OR
	Check for missing value	ISNULL
	Between bounds	BETWEEN val_1 AND val_2
Strings	Pattern matching	LIKE '%val%'

❒ **Joins** – Two tables table_1 and table_2 can be joined in the following way:

```SQL
...

FROM table_1 t1 type_of_join
table_2 t2
   ON (t2.key=t1.key)

...
```

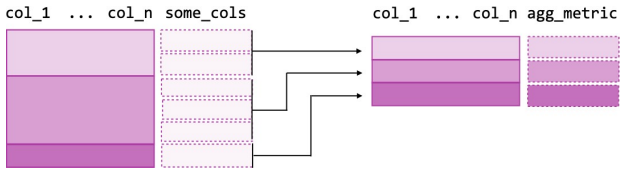
where the different type_of_join commands are summarized in the table below:

Type of join	Illustration
INNER JOIN	
LEFT JOIN	
RIGHT JOIN	
FULL JOIN	

Remark: joining every row of table 1 with every row of table 2 can be done with the CROSS JOIN command, and is commonly known as the cartesian product.

1.2 Aggregations

❒ **Grouping data** – Aggregate metrics are computed on grouped data in the following way:



The SQL command is as follows:

```sql
SELECT
    col_1, agg_function(col_2)
FROM table
GROUP BY col_1
```

❒ **Grouping sets** – The GROUPING SETS command is useful when there is a need to compute aggregations across different dimensions at a time. Below is an example of how all aggregations across two dimensions are computed:

```sql
SELECT
    col_1,
    col_2,
    agg_function(col_3)
FROM table
GROUP BY(
  GROUPING SETS
    (col_1),
    (col_2), (col_1,
    col_2)
)
```

❒ **Aggregation functions** – The table below summarizes the main aggregate functions that can be used in an aggregation query:

Category	Operation	Command
Values	Mean	AVG(col)
	Percentile	PERCENTILE_APPROX(col, p)
	Sum / # of instances	SUM(col) / COUNT(col)
	Max / min	MAX(col) / MIN(col)
	Variance / standard deviation	VAR(col) / STDEV(col)
Arrays	Concatenate into array	collect_list(col)

Remark: the median can be computed using the PERCENTILE_APPROX *function with pequal to* 0.5.

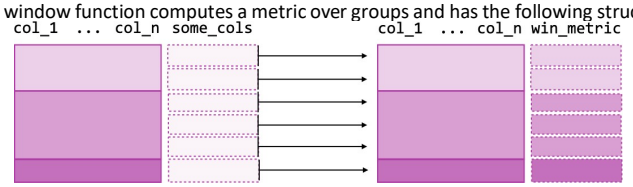
❒ **Filtering** – The table below highlights the differences between the WHERE and HAVING com- mands:

WHERE	HAVING
- Filter condition applies to individual rows - Statement placed right after FROM	- Filter condition applies to aggregates - Statement placed right after GROUP BY

Remark: if WHERE *and* HAVING *are both in the same query,* WHERE *will be executed first.*

1.3 Window functions

❒ **Definition** – A window function computes a metric over groups and has the following struc- ture:



The SQL command is as follows:

```sql
some_window_function() OVER(PARTITION BY some_col ORDER BY another_col)
```

Remark: window functions are only allowed in the SELECT *clause.*

❒ **Row numbering** – The table below summarizes the main commands that rank each row across specified groups, ordered by a specific column:

Command	Description	Example
ROW_NUMBER()	Ties are given different ranks	1, 2, 3, 4
RANK()	Ties are given same rank and skip numbers	1, 2, 2, 4
DENSE_RANK()	Ties are given same rank and don't skip numbers	1, 2, 2, 3

❒ **Values** – The following window functions allow to keep track of specific types of values with respect to the partition:

Command	Description
FIRST_VALUE(col)	Takes the first value of the column
LAST_VALUE(col)	Takes the last value of the column
LAG(col, n)	Takes the n^{th} previous value of the column
LEAD(col, n)	Takes the n^{th} following value of the column
NTH_VALUE(col, n)	Takes the n^{th} value of the column

1.4 Advanced functions

❒ **SQL tips** – In order to keep the query in a clear and concise format, the following tricks are often done:

Operation	Command	Description
Renaming columns	SELECT operation_on_column AS col_name	New column names shown in query results
Abbreviating tables	FROM table_1 t1	Abbreviation used within query for simplicity in notations
Simplifying group by	GROUP BY col_number_list	Specify column position in SELECT clause instead of whole column names
Limiting results	LIMIT n	Display only n rows

❒ **Sorting values** – The query results can be sorted along a given set of columns using the following command:

```
SQL
... [query] ... ORDER BY col_list
```

Remark: by default, the command sorts in ascending order. If we want to sort it in descending order, the DESC command needs to be used after the column.

❒ **Column types** – In order to ensure that a column or value is of one specific data type, the following command is used:

```
SQL
CAST(some_col_or_value AS data_type)
```

where data_type is one of the following:

Data type	Description	Example
INT	Integer	2
DOUBLE	Numerical value	2.0
STRING	String	'teddy bear'
VARCHAR		
DATE	Date	'2020-01-01'
TIMESTAMP	Timestamp	'2020-01-01 00:00:00.000'

Remark: if the column contains data of different types, the TRY_CAST() command will convert unknown types to NULL instead of throwing an error.

❒ **Column manipulation** – The main functions used to manipulate columns are described in the table below:

Category	Operation	Command
General	Take first non-NULL value	COALESCE(col_1, col_2, ..., col_n)
	Create a new column combining existing ones	CONCAT(col_1, ..., col_n)
Value	Round value to n decimals	ROUND(col, n)
String	Converts string column to lower / upper case	LOWER(col) / UPPER(col)
	Replace occurrences of old in col to new	REPLACE(col, old, new)
	Take the substring of col, with a given start and length	SUBSTR(col, start, length)
	Remove spaces from the left / right / both sides	LTRIM(col) / RTRIM(col)/ TRIM(col)
	Length of the string	LENGTH(col)
Date	Truncate at a given granularity (year, month, week)	DATE_TRUNC(time_dimension, col_date)
	Transform date	DATE_ADD(col_date, number_of_days)

❒ **Conditional column** – A column can take different values with respect to a particular set of conditions with the CASE WHEN command as follows:

```
SQL
CASE WHEN some_condition THEN some_value
          ...
     WHEN some_other_condition THEN some_other_value
     ELSE some_other_value_n END
```

❒ **Combining results** – The table below summarizes the main ways to combine results in queries:

Category	Command	Remarks
Union	UNION	Guarantees distinct rows
	UNION ALL	Potential newly-formed duplicates are kept
Intersection	INTERSECT	Keeps observations that are in all selected queries

❒ **Common table expression** – A common way of handling complex queries is to have tem- porary result sets coming from intermediary queries, which are called common table expressions (abbreviated CTE), that increase the readability of the overall query. It is done thanks to the WITH...AS... command as follows:

```
SQL
WITH cte_1 AS(
SELECT...
),
```

```
...

cte_nAS( SELECT...
)

SELECT...
FROM...
```

1.5 Table manipulation

❒ **Table creation** – The creation of a table is done asfollows:

SQL

```
CREATE[table_type]TABLE[creation_type]table_name( col_1
   data_type_1,
                ...              ,
      col_n data_type_n
)
[options];
```

where [table_type], [creation_type] and [options]are one of the following:

Category	Command	Description
Table type	Blank	Default table
	EXTERNALTABLE	External table
Creation type	Blank	Creates table and overwrites current one if it exists
	IFNOTEXISTS	Only creates table if it does not exist
Options	location'path_to_hdfs_folder'	Populate table with data from hdfs folder
	storedas data_format	Stores the table in a specific data format, e.g. parquet, orc or avro

❒ **Data insertion** – New data can either append or overwrite already existing data in a given table as follows:

SQL

```
WITH...                          -- optional
INSERT[insert_type] table_name   --mandatory
SELECT...;                       --mandatory
```

where [insert_type] is among the following:

Command	Description
OVERWRITE	Overwrites existing data
INTO	Appends to existing data

❒ **Dropping table** – Tables are dropped in the following way:

SQL

```
DROPTABLEtable_name;
```

❒ **View** – Instead of using a complicated query, the latter can be saved as a view which can then be used to get the data. A view is created with the following command:

SQL

```
CREATEVIEWview_nameAScomplicated_query;
```

Remark: a view does not create any physical table and is instead seen as a shortcut.

SECTION 2

Working with data with R

2.1 Data manipulation

2.1.1 Main concepts

❒ **File management** – The table below summarizes the useful commands to make sure the working directory is correctly set:

Category	Action	Command
Paths	Change directory to another path	setwd(path)
	Get current working directory	getwd()
	Join paths	file.path(path_1, ..., path_n)
Files	List files and folders in a given directory	list.files(path, include.dirs=TRUE)
	Check if path is a file / folder	file_test('-f', path)
		file_test('-d', path)
	Read / write csv file	read.csv(path_to_csv_file)
		write.csv(df, path_to_csv_file)

❒ **Chaining** – The symbol %>%, also called "pipe", enables to have chained operations and provides better legibility. Here are its different interpretations:

- f(arg_1, arg_2, ..., arg_n) is equivalent to arg_1%>%f(arg_2, arg_3, ..., arg_n) , and also to:
  - arg_1%>%f(., arg_2, ..., arg_n)
  - arg_2%>%f(arg_1, ., arg_3, ..., arg_n)
  - arg_n%>%f(arg_1, ..., arg_n-1, .)
- A common use of pipe is when a dataframe df gets first modified by some_operation_1, then some_operation_2, until some_operation_nin a sequential way. It is done as follows:

```R
# df gets some_operation_1, then some_operation_2, ..., # then
some_operation_n
df%>%
    some_operation_1%>%
    some_operation_2%>%
          ...          %>%
    some_operation_n
```

❒ **Exploring the data** – The table below summarizes the main functions used to get a complete overview of the data:

Category	Action	Command
Look at data	Select columns of interest	df%>%select(col_list)
	Remove unwanted columns	df%>%select(-col_list)
	Look at n first rows / last rows	df%>%head(n) / df%>%tail(n)
	Summary statistics of columns	df%>%summary()
Data types	Data types of columns	df%>%str()
	Number of rows / columns	df%>%NROW() / df%>%NCOL()

❒ **Data types** – The table below sums up the main data types that can be contained in columns:

Data type	Description	Example
character	String-related data	'teddy bear'
factor	String-related data that can be put in bucket, or ordered	'high'
numeric	Numerical data	24.0
int	Numeric data that are integer	24
Date	Dates	'2020-01-01'
POSIXct	Timestamps	'2020-01-01 00:01:00'

2.1.2 Data preprocessing

❒ **Filtering** – We can filter rows according to some conditions as follows:

```R
df%>%
  filter(some_col some_operation some_value_or_list_or_col)
```

where some_operation is one of the following:

Category	Operation	Command
Basic	Equality / non-equality	== / !=
	Inequalities	<, <=, >=, >
	And / or	& / |
Advanced	Check for missing value	is.na()
	Belonging	%in%(val_1, ..., val_n)
	Pattern matching	%like%'val'

Remark: we can filter columns with the select_if *command.*

❒ **Changing columns** – The table below summarizes the main column operations:

Action	Command
Add new columns on top of old ones	df%>%mutate(new_col=operation(other_cols))
Add new columns and discard old ones	df%>%transmute(new_col=operation(other_cols))
Modify several columns in-place	df%>%mutate_at(vars, funs)
Modify all columns in-place	df%>%mutate_all(funs)
Modify columns fitting a specific condition	df%>%mutate_if(condition, funs)
Unite columns	df%>%unite(new_merged_col, old_cols_list)
Separate columns	df%>%separate(col_to_separate, new_cols_list)

❒ **Conditional column** – A column can take different values with respect to a particular set of conditions with the case_when() command asfollows:

```R
case_when(condition_1 ~ value_1, # If condition_1 then value_1
          condition_2 ~ value_2, # If condition_2 then value_2
          ...
          TRUE ~ value_n)        # Otherwise, value_n
```

Remark: the ifelse(condition_if_true, value_true, value_other) *can be used and is easier to manipulate if there is only one condition.*

❒ **Mathematical operations** – The table below sums up the main mathematical operations that can be performed on columns:

Operation	Command
$\sqrt{x}$	sqrt(x)
$\lfloor x \rfloor$	floor(x)
$\lceil x \rceil$	ceiling(x)

❒ **Datetime conversion** – Fields containing datetime values can be stored in two different POSIXt datatypes:

Action	Command
Converts to datetime with seconds since origin	as.POSIXct(col, format)
Converts to datetime with attributes (e.g. time zone)	as.POSIXlt(col, format)

where formatis a string describing the structure of the field and using the commands summarized in the table below:

Category	Command	Description	Example
Year	'%Y' / '%y'	With / without century	2020 / 20
Month	'%B' / '%b'/ '%m'	Full / abbreviated / numerical	August / Aug/ 8
Weekday	'%A' / '%a'	Full / abbreviated	Sunday / Sun
	'%u' / '%w'	Number (1-7) / Number (0-6)	7 / 0
Day	'%d' / '%j'	Of the month / of the year	09 / 222
Time	'%H' / '%M'	Hour / minute	09 / 40
Timezone	'%Z' / '%z'	String / Number of hours from UTC	EST / -0400

Remark: data frames only accept datetime in POSIXct *format.*

❒ **Date properties** – In order to extract a date-related property from a datetime object, the following command is used:

```R
format(datetime_object, format)
```

where format follows the same convention as in the table above.

2.1.3 Data frame transformation

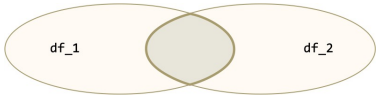
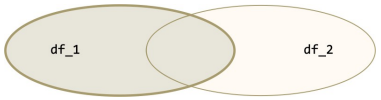
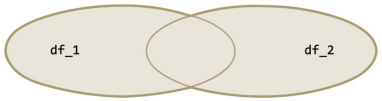
❒ **Merging data frames** – We can merge two data frames by a given field asfollows:

```R
merge(df_1, df_2, join_field, join_type)
```

where join_field indicates fields where the join needs to happen:

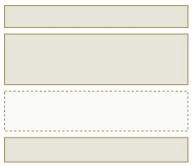
Case	Fields are equal	Different field names
Command	by='field'	by.x='field_1', by.y='field_2'

and where join_type indicates the join type, and is one of the following:

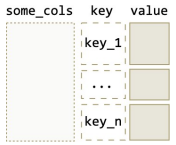
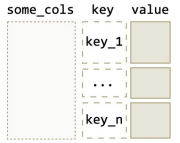
Join type	Option	Illustration
Inner join	default	
Left join	all.x=TRUE	
Right join	all.y=TRUE	
Full join	all=TRUE	

Remark: if the by *parameter is not specified, the merge will be a cross join.*

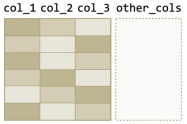
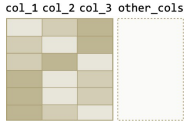
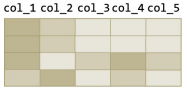
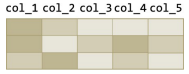
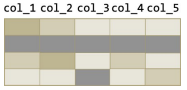
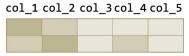
❒ **Concatenation** – The table below summarizes the different ways data frames can be con- catenated:

Type	Command	Illustration
Rows	rbind(df_1, ..., df_n)	
Columns	cbind(df_1, ..., df_n)	

❒ **Common transformations** – The common data frame transformations are summarized in the table below:

Type	Command	Illustration	
		Before	After
Long to wide	spread(df, key='key', value='value')		
Wide to long	gather(df, key='key' value='value', c(key_1, ..., key_n))		

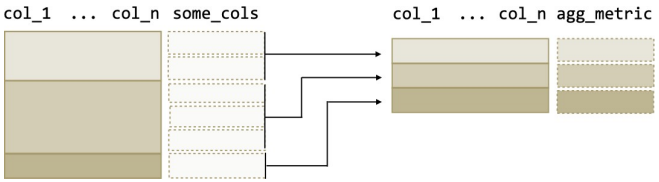
❒ **Row operations** – The following actions are used to make operations on rows of the data frame:

Action	Command	Illustration	
		Before	After
Sort with respect to columns	df%>% arrange(col_1, ..., col_n)		
Dropping duplicates	df%>%unique()		
Drop rows with at least a null value	df%>%na.omit()		

Remark: by default, the arrange *command sorts in ascending order. If we want to sort it in descending order, the - command needs to be used before a column.*

2.1.4 Aggregations

❒ **Grouping data** – Aggregate metrics are computed across groups asfollows:



The R command is as follows:

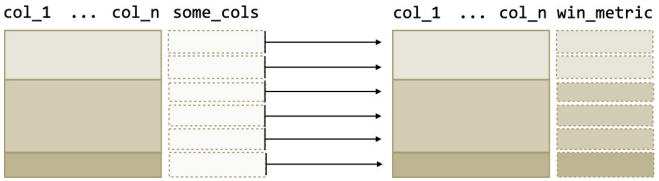
```R
df%>%                                          # Ungrouped data frame
  group_by(col_1, ..., col_n)%>%               # Group by some columns
  summarize(agg_metric=some_aggregation(some_cols))   # Aggregation step
```

❒ **Aggregate functions**– The table below summarizes the main aggregate functions that can be used in an aggregation query:

Category	Action	Command
Properties	Count of observations	n()
Values	Sum of values of observations	sum()
	Max / min of values of observations	max() / min()
	Mean / median of values of observations	mean() / median()
	Standard deviation / variance across observations	sd() / var()

2.1.5 Window functions

❒ **Definition** – A window function computes a metric over groups and has the following struc- ture:



The R command is as follows:

```R
df%>%                                          # Ungrouped data frame
  group_by(col_1, ..., col_n)%>%               # Group by some columns
  mutate(win_metric=window_function(col))      # Window function
```

Remark: applying a window function will not change the initial number of rows of the data frame.

❒ **Row numbering** – The table below summarizes the main commands that rank each row across specified groups, ordered by a specific field:

Join type	Command	Example
row_number(x)	Ties are given different ranks	1, 2, 3, 4
rank(x)	Ties are given same rank and skip numbers	1, 2.5, 2.5, 4
dense_rank(x)	Ties are given same rank and do not skip numbers	1, 2, 2, 3

❒ **Values** – The following window functions allow to keep track of specific types of values with respect to the group:

Command	Description
first(x)	Takes the first value of the column
last(x)	Takes the last value of the column
lag(x, n)	Takes the n^{th} previous value of the column
lead(x, n)	Takes the n^{th} following value of the column
nth(x, n)	Takes the n^{th} value of the column

2.2 Data visualization

2.2.1 General structure

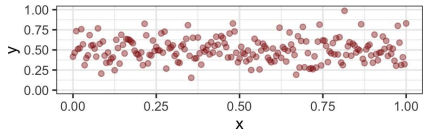
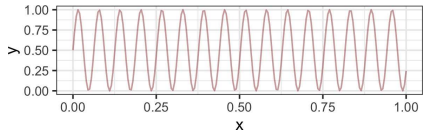
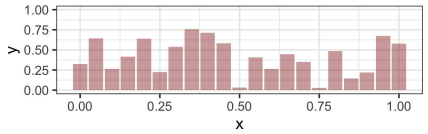
❒ **Overview**– The general structure of the code that is used to plot figures is as follows:

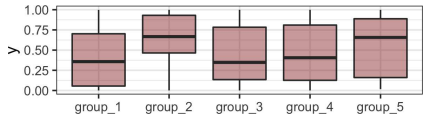
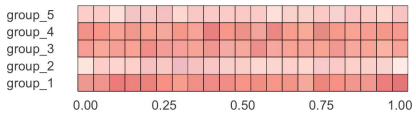
```R
ggplot(...)+                    # Initialization
  geom_function(...)+           # Main plot(s)
  facet_function(...)+ # Facets (optional) labs(...)+
                                # Legend (optional)
  scale_function(...)+ # Scales (optional)
  theme_function(...)           # Theme (optional)
```

We note the following points:

- The ggplot() layer is mandatory.
- When the data argument is specified inside the ggplot() function, it is used as default in the following layers that compose the plot command, unless otherwise specified.
- In order for features of a data frame to be used in a plot, they need to be specified inside the aes() function.

❒ **Basic plots** – The main basic plots are summarized in the table below:

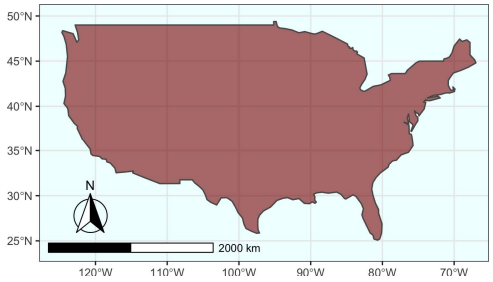
Type	Command	Illustration
Scatter plot	geom_point(x, y, params)	
Line plot	geom_line(x, y, params)	
Bar chart	geom_bar(x, y, params)	

Type	Command	Illustration
Box plot	geom_boxplot(x, y, params)	
Heatmap	geom_tile(x, y, params)	

where the possible parameters are summarized in the table below:

Command	Description	Use case
color	Color of a line / point / border	'red'
fill	Color of an area	'red'
size	Size of a line / point	4
shape	Shape of a point	4
linetype	Shape of a line	'dashed'
alpha	Transparency, between 0 and 1	0.3

❒ **Maps** – It is possible to plot maps based on geometrical shapes asfollows:



The following table summarizes the main commands used to plot maps:

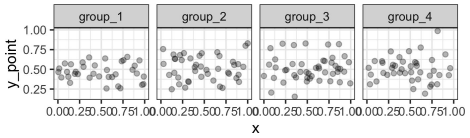
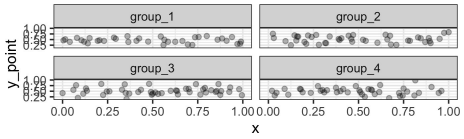
Category	Action	Command
Map	Draw polygon shapes from the geometry column	geom_sf(data)
Additional elements	Add and customize geographical directions	annotation_north_arrow(l)
	Add and customize distance scale	annotation_scale(l)
Range	Customize range of coordinates	coord_sf(xlim, ylim)

❒ **Animations** – Plotting animations can be made using the gganimate library. The following command gives the general structure of the code:

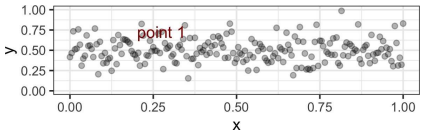
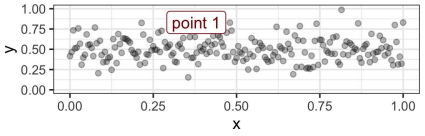
```R
# Main plot ggplot()+
   ...+
   transition_states(field, states_length)

# Generate and save animation
animate(plot, duration, fps, width, height, units, res, renderer) anim_save(filename)
```

2.2.2 Advanced features

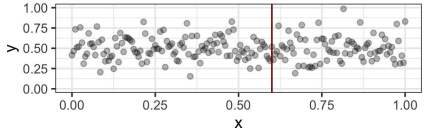
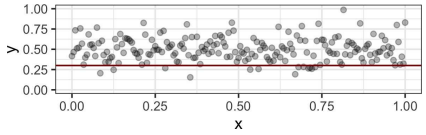
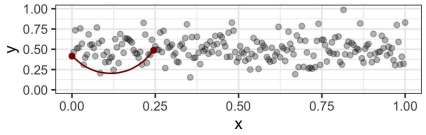
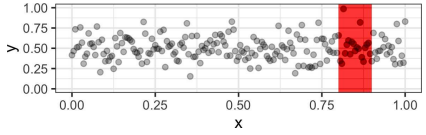
❒ **Facets** – It is possible to represent the data through multiple dimensions with facets using the following commands:

Type	Command	Illustration
Grid (1 or 2D)	facet_grid(row_var ~ column_var)	
Wrapped	facet_wrap(vars(x1, ..., xn), nrow, ncol)	

❒ **Text annotation** – Plots can have text annotations with the followingcommands:

Command	Illustration
geom_text(x, y, label, hjust, vjust)	
geom_label_repel(x, y, label, nudge_x, nudge_y)	

❒ **Additional elements** – We can add objects on the plot with the following commands:

Type	Command	Illustration
Line	geom_vline(xintercept, linetype)	
	geom_hline(yintercept, linetype)	
Curve	geom_curve(x, y, xend, yend)	
Rectangle	geom_rect(xmin, xmax, ymin, ymax)	

2.2.3 Last touch

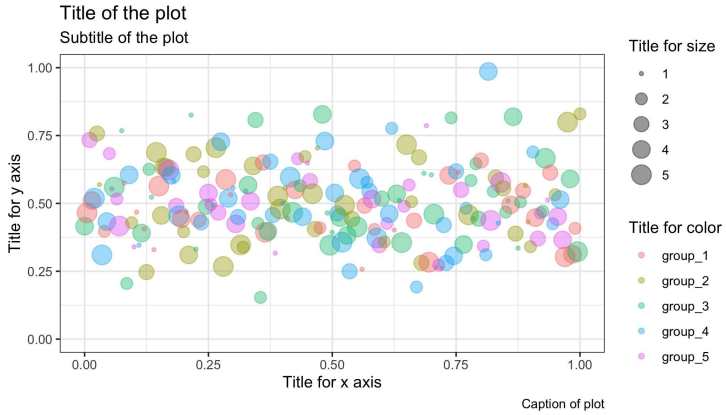
❒ **Legend** – The title of legends can be customized to the plot with the following command:

```R
plot+labs(params)
```

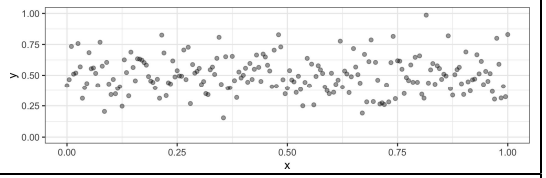
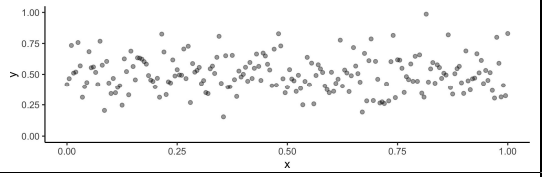
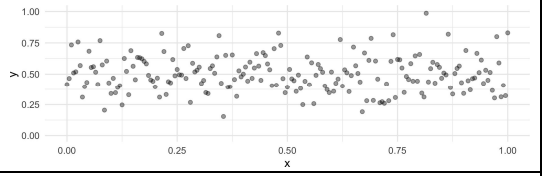
where the params are summarized below:

Element	Command
Title / subtitle of the plot	title='text' / subtitle='text'
Title of the x / y axis	x='text' / y='text'
Title of the size / color	size='text' / color='text'
Caption of the plot	caption='text'

This results in the following plot:



❒ **Plot appearance** – The appearance of a given plot can be set by adding the following command:

Type	Command	Illustration
Black and white	theme_bw()	
Classic	theme_classic()	
Minimal	theme_minimal()	
None	theme_void()	

In addition, theme() is able to adjust positions/fonts of elements of the legend.

Remark: in order to fix the same appearance parameters for all plots, the theme_set() *function can be used.*

❒ **Scales and axes** – Scales and axes can be changed with the followingcommands:

Category	Action	Command
Range	Specify range of x / y axis	xlim(xmin, xmax)
		ylim(ymin, ymax)
Nature	Display ticks in a customized manner	scale_x_continuous()
		scale_x_discrete()
		scale_x_date()
Magnitude	Transform axes	scale_x_log10()
		scale_x_reverse()
		scale_x_sqrt()

Remark: the scale_x() *functions are for the x axis. The same adjustments are available for the y axis with* scale_y() *functions.*

❒ **Double axes** – A plot can have more than one axis with the sec.axisoption within a given scale function scale_function(). It is done as follows:

```R
scale_function(sec.axis=sec_axis( ~ .))
```

❒ **Saving figure** – It is possible to save figures with predefined parameters regarding the scale, width and height of the output image with the following command:

```R
ggsave(plot, filename, scale, width, height)
```

SECTION 3

Working with data with Python

3.1 Data manipulation

3.1.1 Main concepts

❒ **File management** – The table below summarizes the useful commands to make sure the working directory is correctly set:

Category	Action	Command
Paths	Change directory to another path	os.chdir(path)
	Get current working directory	os.getcwd()
	Join paths	os.path.join(path_1, ..., path_n)
Files	List files and folders in a directory	os.listdir(path)
	Check if path is a file / folder	os.path.isfile(path)
		os.path.isdir(path)
	Read / write csv file	pd.read_csv(path_to_csv_file)
		df.to_csv(path_to_csv_file)

❒ **Chaining** – It is common to have successive methods applied to a data frame to improve readability and make the processing steps more concise. The method chaining is done as follows:

```
Python
# df gets some_operation_1, then some_operation_2, ..., then some_operation_n (df
.some_operation_1(params_1)
.some_operation_2(params_2)
.          ...
.some_operation_n(params_n))
```

❒ **Exploring the data** – The table below summarizes the main functions used to get a complete overview of the data:

Category	Action	Command
Look at data	Select columns of interest	df[col_list]
	Remove unwanted columns	df.drop(col_list,axis=1)
	Look at n first rows / last rows	df.head(n) / df.tail(n)
	Summary statistics of columns	df.describe()
Paths	Data types of columns	df.dtypes / df.info()
	Number of (rows, columns)	df.shape

❒ **Data types** – The table below sums up the main data types that can be contained in columns:

Data type	Description	Example
object	String-related data	'teddy bear'
float64	Numerical data	24.0
int64	Numeric data that are integer	24
datetime64	Timestamps	'2020-01-01 00:01:00'

3.1.2 Data preprocessing

❒ **Filtering** – We can filter rows according to some conditions as follows:

```
Python
df[df['some_col'] some_operation some_value_or_list_or_col]
```

where some_operation is one of the following:

Category	Operation	Command
Basic	Equality / non-equality	== / !=
	Inequalities	<, <=, >=, >
	And / or	& / |
Advanced	Check for missing value	pd.isnull()
	Belonging	.isin([val_1, ..., val_n])
	Pattern matching	.str.contains('val')

❒ **Changing columns** – The table below summarizes the main column operations:

Operation	Command
Add new columns on top of old ones	df.assign(new_col=lambda x:some_operation(x))
Rename columns	df.rename(columns={'current_col':'new_col_name'}) })
Unite columns	df['new_merged_col']=(df[old_cols_list].agg('-'.join,axis=1))

❒ **Conditional column** – A column can take different values with respect to a particular set of conditions with the np.select() command as follows:

```python
np.select(
    [condition_1, ..., condition_n], # If condition_1, ..., condition_n [value_1, ..., value_n],
                                     # Then value_1, ..., value_n respectively
    default=default_value            # Otherwise, default_value
)
```

Remark: the np.where(condition_if_true, value_true, value_other) *command can be used and is easier to manipulate if there is only one condition.*

❒ **Mathematical operations** – The table below sums up the main mathematical operations that can be performed on columns:

Operation	Command
$\sqrt{x}$	np.sqrt(x)
$\lfloor x \rfloor$	np.floor(x)
$\lceil x \rceil$	np.ceil(x)

❒ **Datetime conversion** – Fields containing datetime values are converted from string to date- time as follows:

```python
pd.to_datetime(col, format)
```

where formatis a string describing the structure of the field and using the commands summarized in the table below:

Category	Command	Description	Example
Year	'%Y' / '%y'	With / without century	2020 / 20
Month	'%B' / '%b'/ '%m'	Full / abbreviated / numerical	August / Aug/ 8
Weekday	'%A' / '%a'	Full / abbreviated	Sunday / Sun
	'%u' / '%w'	Number (1-7) / Number (0-6)	7 / 0
Day	'%d' / '%j'	Of the month / of the year	09 / 222
Time	'%H' / '%M'	Hour / minute	09 / 40
Timezone	'%Z' / '%z'	String / Number of hours from UTC	EST / -0400

❒ **Date properties** – In order to extract a date-related property from a datetime object, the following command is used:

```python
datetime_object.strftime(format)
```

where format follows the same convention as in the table above.

3.1.3 Data frame transformation

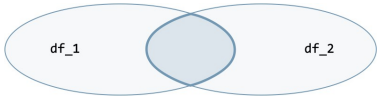
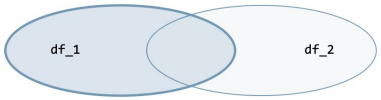
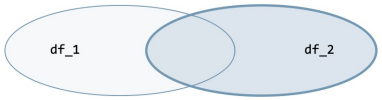
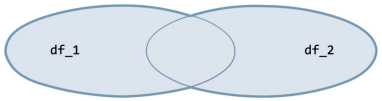
❒ **Merging data frames** – We can merge two data frames by a given field asfollows:

```python
df1.merge(df2, join_field, join_type)
```

where join_field indicates fields where the join needs to happen:

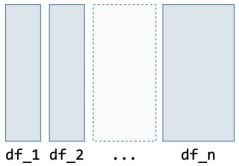
Case	Fields are equal	Fields are different
Command	on='field'	left_on='field_1',right_on='field_2'

and where join_type indicates the join type, and is one of the following:

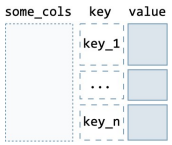
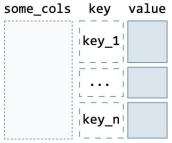
Join type	Option	Illustration
Inner join	how='inner'	
Left join	how='left'	
Right join	how='right'	
Full join	how='outer'	

Remark: a cross join can be done by joining on an undifferentiated column, typically done by creating a temporary column equal to 1.

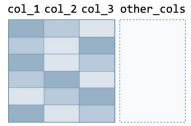
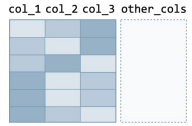
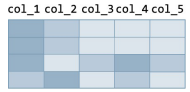
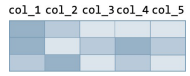
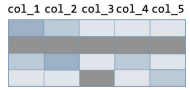
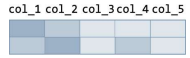
❒ **Concatenation** – The table below summarizes the different ways data frames can be con- catenated:

Type	Command	Illustration
Rows	pd.concat([df_1, ..., df_n],axis=0)	
Columns	pd.concat([df_1, ..., df_n],axis=1)	

❒ **Common transformations** – The common data frame transformations are summarized in the table below:

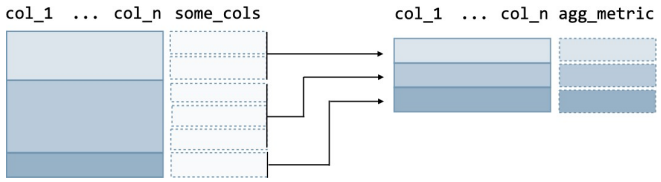
Type	Command	Illustration	
		Before	After
Long to wide	pd.pivot_table(df,values='value', index=some_cols, columns='key', aggfunc=np.sum)		
Wide to long	pd.melt(df,var_name='key', value_name='value', value_vars=['key_1', ...,'key_n'],id_vars=some_cols)		

❒ **Row operations** – The following actions are used to make operations on rows of the data frame:

Action	Command	Illustration	
		Before	After
Sort with respect to columns	df.sort_values(by=['col_1', ...,'col_n'], ascending=True)		
Dropping duplicates	df.drop_duplicates()		
Drop rows with at least a null value	df.dropna()		

3.1.4 Aggregations

❒ **Grouping data** – A data frame can be aggregated with respect to given columns as follows:



The Python command is as follows:

```Python
(df
.groupby(['col_1', ...,'col_n'])
.agg({'col': builtin_agg})
```

where builtin_agg is among the following:

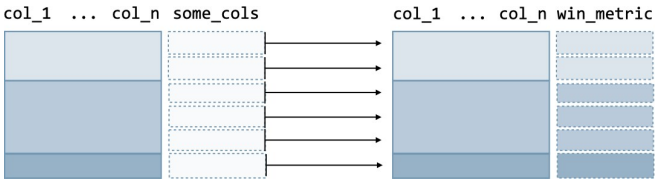
Category	Action	Command
Properties	Count of observations	'count'
Values	Sum of values of observations	'sum'
	Max / min of values of observations	'max' / 'min'
	Mean / median of values of observations	'mean' / 'median'
	Standard deviation / variance across observations	'std' / 'var'

❒ **Custom aggregations**– It is possible to perform customized aggregations by using lambda functions as follows:

```python
df_agg=(
    df
    .groupby(['col_1', ...,'col_n'])
    .apply(lambdax:pd.Series({
        'agg_metric':some_aggregation(x)
    }))
)
```

3.1.5 Window functions

❒ **Definition** – A window function computes a metric over groups and has the following struc- ture:



The Python command is as follows:

```python
(df
.assign(win_metric=lambdax:
                x.groupby(['col_1', ...,'col_n'])['col'].window_function(params))
```

Remark: applying a window function will not change the initial number of rows of the data frame.

❒ **Row numbering** – The table below summarizes the main commands that rank each row across specified groups, ordered by a specific field:

Join type	Command	Example
x.rank(method='first')	Ties are given different ranks	1, 2, 3, 4
x.rank(method='min')	Ties are given same rank and skip numbers	1, 2.5, 2.5, 4
x.rank(method='dense')	Ties are given same rank and do not skip numbers	1, 2, 2, 3

❒ **Values** – The following window functions allow to keep track of specific types of values with respect to the group:

Command	Description
x.shift(n)	Takes the n^{th} previous value of the column
x.shift(-n)	Takes the n^{th} following value of the column

3.2 Data visualization

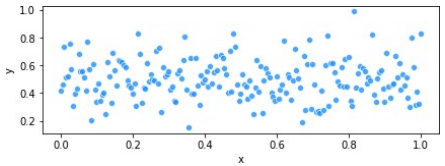
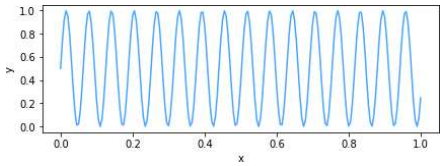
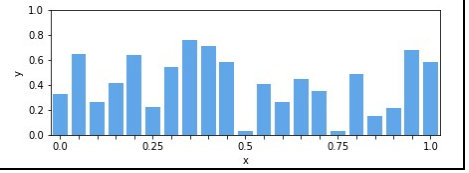
3.2.1 General structure

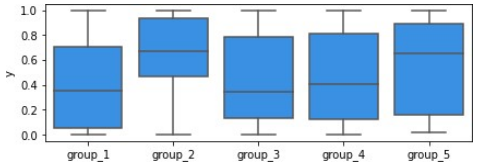
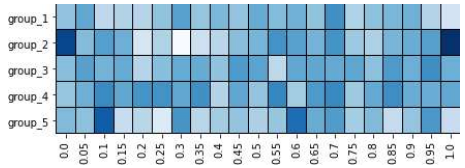
❒ **Overview**– The general structure of the code that is used to plot figures is as follows:

```python
# Plot
f, ax=plt.subplots(...) ax=sns...

# Legend plt.title()
plt.xlabel() plt.ylabel()
```

We note that the plt.subplots() command enables to specify the figure size.

❒ **Basic plots** – The main basic plots are summarized in the tablebelow:

Type	Command	Illustration
Scatter plot	sns.scatterplot(x, y, params)	
Line plot	sns.lineplot(x, y, params)	
Bar chart	sns.barplot(x, y, params)	

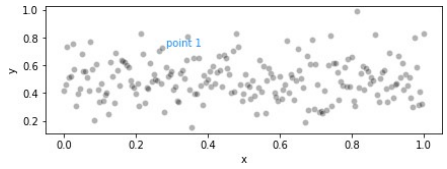
Type	Command	Illustration
Box plot	sns.boxplot(x, y, params)	
Heatmap	sns.heatmap(data, params)	

where the meaning of parameters are summarized in the table below:

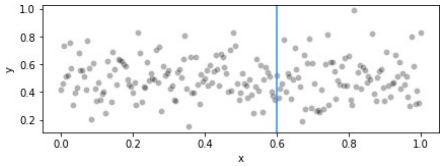
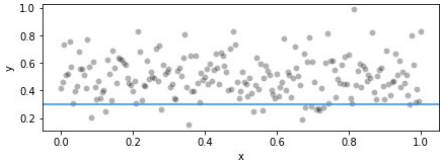
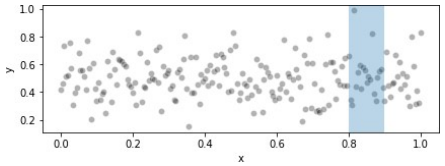
Command	Description	Use case
hue	Color of a line / point / border	'red'
fill	Color of an area	'red'
size	Size of a line / point	4
linetype	Shape of a line	'dashed'
alpha	Transparency, between 0 and 1	0.3

3.2.2 Advanced features

❒ **Text annotation** – Plots can have text annotations with the followingcommands:

Type	Command	Illustration
Text	ax.text(x, y, s, color)	

❒ **Additional elements** – We can add objects on the plot with the followingcommands:

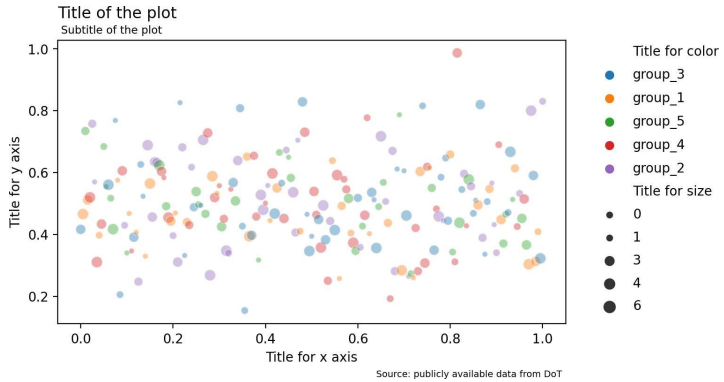
Type	Command	Illustration
Line	ax.axvline(x, ymin, ymax, color, linewidth, linestyle)	
	ax.axhline(y, xmin, xmax, color, linewidth, linestyle)	
Rectangle	ax.axvspan(xmin, xmax, ymin, ymax, color, fill, alpha)	

3.2.3 Last touch

❒ **Legend** – The title of legends can be customized to the plot with the commands summarized below:

Element	Command
Title / subtitle of the plot	ax.set_title('text', loc, pad)
	plt.suptitle('text', x, y, size, ha)
Title of the x / y axis	ax.set_xlabel('text') / ax.set_ylabel('text')
Title of the size / color	ax.get_legend_handles_labels()
Caption of the plot	ax.text('text', x, y, fontsize)

This results in the following plot:



❒ **Double axes** – A plot can have more than one axis with the plt.twinx() command. It is done as follows:

```python
ax2=plt.twinx()
```

❒ **Figure saving** – There are two main steps to save a plot:

- Specifying the width and height of the plot when declaring the figure:

```python
f, ax=plt.subplots(1,figsize=(width, height))
```

- Saving the figure itself:

```python
f.savefig(fname)
```

SECTION 4

Engineering productivity tips with Git, Bash and Vim

4.1 Working in groups with Git

4.1.1 Overview

❒ **Overview** – Git is a version control system (VCS) that tracks changes of different files in a given repository. In particular, it is useful for:

- keeping track of file versions
- working in parallel thanks to the concept of branches
- backing up files to a remote server

4.1.2 Main commands

❒ **Getting started** – The table below summarizes the commands used to start a new project, depending on whether or not the repository already exists:

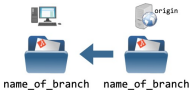
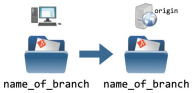
Case	Action	Command	Illustration
No existing repository	Initialize repository from local folder	gitinit	
Repository already exists	Copy repository from remote to local	gitclone git_address	

❒ **File check-in** – We can track modifications made in the repository, done by either modifying, adding or deleting a file, through the following steps:

Step	Command	Illustration
1. Add modified, new, or deleted file to staging area	gitadd file	
2. Save snapshot along with descriptive message	gitcommit-m'description'	

Remark 1: gitadd . *will have all modified files to the staging area.*
Remark 2: files that we do not want to track can be listed in the .gitignore *file.*

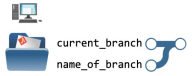
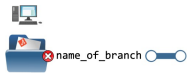
❒ **Sync with remote** – The following commands enable changes to be synchronized between remote and local machines:

Action	Command	Illustration
Fetch most recent changes from remote branch	gitpull name_of_branch	
Push latest local changes to remote branch	gitpush name_of_branch	

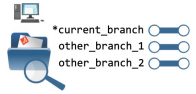
❒ **Parallel workstreams** – In order to make changes that do not interfere with the current branch, we can create another branch name_of_branch asfollows:

```
Bash
gitcheckout-bname_of_new_branch      # Create and checkout to that branch
```

Depending on whether we want to incorporate or discard the branch, we have the following commands:

Action	Command	Illustration
Merge with initial branch	gitmerge initial_branch	
Remove branch	gitbranch-Dname_of_branch	

❒ **Tracking status** – We can check previous changes made to the repository with the following commands:

Action	Command	Illustration
Check status of modified file(s)	gitstatus	
View last commits	gitlog --oneline	
Compare changes made between two commits	gitdiff commit_1 commit_2	
View list of local branches	gitbranch	

❒ **Canceling changes** – Canceling changes is done differently depending on the situation that we are in. The table below sums up the most common cases:

Case	Action	Command	Illustration
Unstaged	Revert file to last commit	gitcheckout -- file	
Staged	Remove file from staging area	gitreset HEAD file	
Committed	Go back to a previous commit	gitreset --hard prev_commit	

4.1.3 Project structure

❒ **Structure of folders** – It is important to keep a consistent and logical structure of the project. One example is as follows:

```
Terminal
my_project/ analysis/
            graph/
            notebook/
    data/
```

```
        query/ raw/
        processed/
    modeling/
        method/
        tests
    README.md
```

4.2 Working with Bash

❒ **Basic terminal commands** – The table below sums up the most useful terminal commands:

Category	Action	Command
Exploration	Display list of files (including hidden ones)	ls(-a)
	Show current directory	pwd
	Show content of file	catpath_to_file
	Show statistics of file (lines/words/characters)	wcpath_to_file
File management	Make new folder	mkdirfolder_name
	Change directory to folder	cdpath_to_folder
	Create new empty file	touchfilename
	Copy-paste file (folder) from origin to destination	scp(-R) origin destination
	Move file/folder from origin to destination	mvorigin destination
	Remove file (folder)	rm(-R) path
Compression	Compress folder into file	tar-czvfcomp_folder.tar.gz folder
	Uncompress file	tar-xzvfcomp_folder.tar.gz
Miscellaneous	Display message	echo"message"
	Overwrite / append file with output	output>file.txt / output>> file.txt
	Execute command with elevated privileges	sudocommand
	Connect to a remote machine	sshremote_machine_address

❒ **Chaining** – It is a concept that improves readability by chaining operations with the pipe | operator. The most common examples are summed up in the table below:

Action	Command
Count number of files in a folder	lspath_to_folder|wc-l
Count number of lines in file	catpath_to_file|wc-l
Show last n commands executed	history|tail-n

❒ **Advanced search** – The find command allows the search of specific files and manipulate them if necessary. The general structure of the command is as follows:

```Bash
findpath_to_folder/. [conditions] [actions]
```

The possible conditions and actions are summarized in the table below:

Category	Action	Command
Conditions	Certain names, regex accepted	-name'certain_name'
	Certain file types (d/f for directory/file)	-typecertain_type
	Certain file sizes (c/k/M/G for B/kB/MB/GB)	-sizefile_size
	Opposite of a given condition	-not[condition]
Actions	Delete selected files	-delete
	Print selected files	-print

Remark: the flags above can be combined to make a multi-condition search.

❒ **Changing permissions** – The following command enables to change the permissions of a given file (or folder):

```Bash
chmod(-R) three_digits file
```

with three_digitsbeing a combination of three digits, where:

- the first digit is about the owner associated to thefile
- the second digit is about the group associated to the file
- the third digit is anyone irrespective of their relation to the file Each

digit is one of (0, 4, 5, 6, 7), and has the followingmeaning:

Representation	Binary	Digit	Explanation
---	000	0	No permission
r--	100	4	Only read permission
r-x	101	5	Both read and execution permissions
rw-	110	6	Both read and write permissions
rwx	111	7	Read, write and execution permissions

For instance, giving read, write, execution permissions to everyone for a given_file is done by running the following command:

```bash
chmod 777 given_file
```

Remark: in order to change ownership of a file to a given user and group, we use the command `chown user:group file .`

❒ **Terminal shortcuts** – The table below summarizes the main shortcuts when working with the terminal:

Action	Command
Search previous commands	Ctrl + r
Go to beginning / end of line	Ctrl + a / Ctrl + e
Remove everything after the cursor	Ctrl + k
Clear line	Ctrl + u
Clear terminal window	Ctrl + l

4.3 Automating tasks

❒ **Create aliases** – Shortcuts can be added to the ~/.bash_profilefile by adding the following code:

```bash
shortcut="command"
```

❒ **Bash scripts** – Bash scripts are files whose file name ends with .sh and where the fileitself is structured as follows:

```bash
#!/bin/bash

...[bash script] ...
```

❒ **Crontabs** – By letting the day of the month vary between 1-31 and the day of the week vary between 0-6 (Sunday-Saturday), a crontab is of the followingformat:

```
Terminal
 *       *       *       *       *
minute  hour    day     mont    day
                                of week
```

❒ **tmux** – Terminal multiplexing, often known as tmux, is a way of running tasks in the back- ground and in parallel. The table below summarizes the main commands:

Category	Action	Command
Session management	Open a new / last existing session	tmux / tmuxattach
	Leave current session	tmuxdetach
	List all open sessions	tmuxls
	Remove session_name	tmuxkill-session-tsession_name
Window management	Open / close a window	Cmd + b + c / Cmd + b + x
	Move to n^{th} window	Ctrl + b + n

4.4 Mastering editors

❒ **Vim** – Vim is a popular terminal editor enabling quick and easy file editing, which is partic- ularly useful when connected to a server. The main commands to have in mind are summarized in the table below:

Category	Action	Command
File handling	Go to beginning / end of line	0 / $
	Go to first / last line / i^{th} line	gg / G / i G
	Go to previous / next word	b / w
	Exit file with / without saving changes	:wq / :q!
Text editing	Copy line n line(s), where $n \in \mathbb{N}$	nyy
	Insert n line(s) previously copied	p
Searching	Search for expression containing name_of_pattern	/name_of_pattern
	Next / previous occurrence of name_of_pattern	n / N
Replacing	Replace old with new expressions with confirmation for each change	:%s/old/new/gc

❒ **Jupyter notebook** – Editing code in an interactive way is easily done through Jupyter notebooks. The main commands to have in mind are summarized in the table below:

Category	Action	Command
Cell transformation	Transform selected cell to text / code	Click on cell + m / y
	Delete selected cell	Click on cell + dd
	Add new cell below / above selected cell	Click on cell + b / a

SECTION A

Conversion between R and Python: data manipulation

A.1 Main concepts

❒ **File management** – The table below summarizes the useful commands to make sure the working directory is correctly set:

Category	R Command	Python Command
Paths	setwd(path)	os.chdir(path)
	getwd()	os.getcwd()
	file.path(path_1, ..., path_n)	os.path.join(path_1, ..., path_n)
Files	list.files(path, include.dirs=TRUE)	os.listdir(path)
	file_test('-f', path)	os.path.isfile(path)
	file_test('-d', path)	os.path.isdir(path)
	read.csv(path_to_csv_file)	pd.read_csv(path_to_csv_file)
	write.csv(df, path_to_csv_file)	df.to_csv(path_to_csv_file)

❒ **Exploring the data** – The table below summarizes the main functions used to get a complete overview of the data:

Category	R Command	Python Command
Look at data	df%>%select(col_list)	df[col_list]
	df%>%head(n) / df%>%tail(n)	df.head(n) / df.tail(n)
	df%>%summary()	df.describe()
Data types	df%>%str()	df.dtypes / df.info()
	df%>%NROW() / df%>%NCOL()	df.shape

❒ **Data types** – The table below sums up the main data types that can be contained in columns:

R Data type	Python Data type	Description
character	object	String-related data
factor		String-related data that can be put in bucket, or ordered
numeric	float64	Numerical data
int	int64	Numeric data that are integer
POSIXct	datetime64	Timestamps

A.2 Data preprocessing

❒ **Filtering** – We can filter rows according to some conditions as follows:

```
R
df%>%
  filter(some_col some_operation some_value_or_list_or_col)
```

where some_operation is one of the following:

Category	R Command	Python Command
Basic	== / !=	== / !=
	<, <=, >=, >	<, <=, >=, >
	& / |	& / |
Advanced	is.na()	pd.isnull()
	%in%(val_1, ..., val_n)	.isin([val_1, ..., val_n])
	%like%'val'	.str.contains('val')

❒ **Mathematical operations** – The table below sums up the main mathematical operations that can be performed on columns:

Operation	R Command	Python Command
$\sqrt{x}$	sqrt(x)	np.sqrt(x)
$\lfloor x \rfloor$	floor(x)	np.floor(x)
$\lceil x \rceil$	ceiling(x)	np.ceil(x)

A.3 Data frame transformation

❒ **Common transformations** – The common data frame transformations are summarized in the table below:

Category	R Command	Python Command
Concatenation	rbind(df_1, ..., df_n)	pd.concat([df_1, ..., df_n],axis=0)
	cbind(df_1, ..., df_n)	pd.concat([df_1, ..., df_n],axis=1)
Dimension change	spread(df, key, value)	pd.pivot_table(df,values='some_values', index='some_index', columns='some_column', aggfunc=np.sum)
	gather(df, key, value)	pd.melt(df,id_vars='variable', value_vars='other_variable')

SECTION B

Conversion between R and Python: data visualization

B.1 General structure

❒ **Basic plots** – The main basic plots are summarized in the tablebelow:

Type	R Command	Python Command
Scatter plot	geom_point(x, y, params)	sns.scatterplot(x, y, params)
Line plot	geom_line(x, y, params)	sns.lineplot(x, y, params)
Bar chart	geom_bar(x, y, params)	sns.barplot(x, y, params)
Box plot	geom_boxplot(x, y, params)	sns.boxplot(x, y, params)
Heatmap	geom_tile(x, y, params)	sns.heatmap(x, y, params)

where the meaning of parameters are summarized in the table below:

Command	Description	Use case
color / hue	Color of a line / point / border	'red'
fill	Color of an area	'red'
size	Size of a line / point	4
linetype	Shape of a line	'dashed'
alpha	Transparency, between 0 and 1	0.3

B.2 Advanced features

❒ **Additional elements** – We can add objects on the plot with the followingcommands:

Type	R Command	Python Command
Line	geom_vline(xintercept, linetype)	ax.axvline(x, ymin, ymax, color, linewidth, linestyle)
	geom_hline(yintercept, linetype)	ax.axhline(y, xmin, xmax, color, linewidth, linestyle)
Rectangle	geom_rect(xmin, xmax, ymin, ymax)	ax.axvspan(xmin, xmax, ymin, ymax)
Text	geom_text(x, y, label, hjust, vjust)	ax.text(x, y, s, color)