|  |  |  |
| --- | --- | --- |
| **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) |

**Chaining**Successive operations can be chained in R using the %>% symbol in order to provide better legibility. It is done as follows:

# df gets some\_operationChainingSuccessive operations can be chained in R using the %>% symbol in order to provide better legibility. It is done as follows:

R

# df gets some\_operation\_1, then some\_operation\_2, ..., then some\_operation\_n

df %>%

some\_operation\_1(params\_1) %>%

some\_operation\_2(params\_1) %>%

... %>%

some\_operation\_n(params\_1)\_1, then some\_operation\_2, ..., then some\_operation\_n

df %>%

  some\_operation\_1(params\_1) %>%

  some\_operation\_2(params\_1) %>%

             ...             %>%

  some\_operation\_n(params\_1)

The corresponding Python command is as follows:

# 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** | **R command** | **Python command** |
| Look at data | df %>% select(col\_list) | df[col\_list] |
| df %>% select(-col\_list) | df.drop(col\_list, axis=1) |
| 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 |

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

|  |  |  |
| --- | --- | --- |
| **Category** | **R command** | **Python command** |
| Overview | df %>% filter(col operation val\_or\_col) | df[df['col'] operation val\_or\_col] |
| Operation | == / != | == / != |
| <, <=, >=, > | <, <=, >=, > |
| & / | | & / | |
| is.na() | pd.isnull() |
| %in% (val\_1, ..., val\_n) | .isin([val\_1, ..., val\_n]) |
| %like% 'val' | .str.contains('val') |

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

|  |  |  |
| --- | --- | --- |
| **Action** | **R command** | **Python command** |
| Add new columns on top of old ones | df %>%   mutate(new\_col = operation(other\_cols)) | df.assign(   new\_col=lambda x: some\_operation(x) ) |
| Unite columns | df  %>%   unite(new\_merged\_col, old\_cols\_list) | 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 following R command:

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

The corresponding Python command is as follows:

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

)

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

|  |  |  |
| --- | --- | --- |
| **Operation** | **R command** | **Python command** |
| √x | sqrt(x) | np.sqrt(x) |
| ⌊x⌋ | floor(x) | np.floor(x) |
| ⌈x⌉ | ceiling(x) | np.ceil(x) |

**Datetime conversion**Fields containing datetime values are converted from string to datetime as follows:

|  |  |  |
| --- | --- | --- |
| **Action** | **R Command** | **Python Command** |
| Converts string to datetime | as.POSIXct(col, format) | pd.to\_datetime(col, format) |

where format is 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 |

## Data frame transformation

**Merging data frames**We can merge two data frames by a given field as follows:

|  |  |  |
| --- | --- | --- |
| **Category** | **R command** | **Python command** |
| Overview | merge(df\_1, df\_2, join\_field, join\_type) | df\_1.merge(df\_2, join\_field, join\_type) |
| Join field | by = 'field' | on='field' |
| by.x = 'field\_1', by.y = 'field\_2' | left\_on='field\_1', right\_on='field\_2' |
| Join type | all.x = TRUE | how='left' |
| all.y = TRUE | how='right' |
| default | how='inner' |
| all = TRUE | how='outer' |

**Concatenation**The table below summarizes the different ways data frames can be concatenated:

|  |  |  |
| --- | --- | --- |
| **Type** | **R command** | **Python command** |
| Rows | rbind(df\_1, ..., df\_n) | pd.concat([df\_1, ..., df\_n], axis=0) |
| Columns | cbind(df\_1, ..., df\_n) | pd.concat([df\_1, ..., df\_n], axis=1) |

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

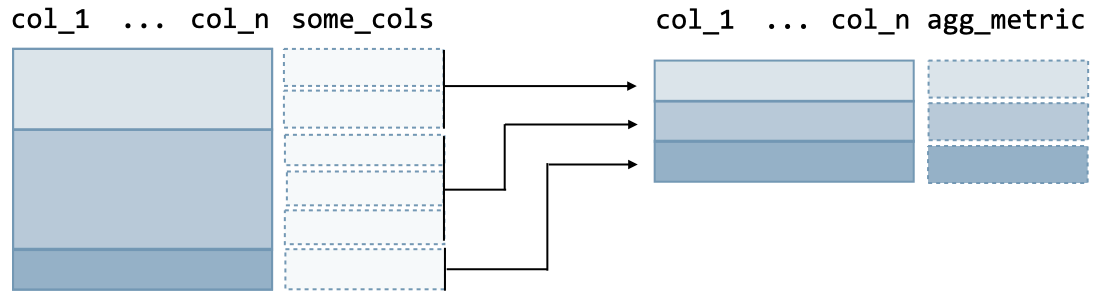
|  |  |  |
| --- | --- | --- |
| **Type** | **R command** | **Python command** |
| Long to wide | spread(   df, key = 'key\_name'   value = 'value\_name' ) | pd.pivot\_table(   df, columns='key\_name',   values='value\_name',   index=other\_cols,   aggfunc=np.sum ) |
| Wide to long | gather(   df, key = 'key\_name',   value = 'value\_name',   c(key\_1, ..., key\_n) ) | 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** | **R command** | **Python command** |
| Sort data frame | df %>% arrange(col\_list) | df.sort\_values(by=col\_list, ascending=True) |
| Drop duplicates | df %>% unique() | df.drop\_duplicates() |
| Drop rows with null values | df %>% na.omit() | df.dropna() |

## Aggregations

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



The R command is as follows:

df %>%                                                 # Original ungrouped data frame

 group\_by(col\_1, ..., col\_n) %>%                       # Group by some columns

 summarize(agg\_metric = some\_aggregation(some\_cols))   # Aggregation step

and the Python command is as follows:

(df                                                    # Original ungrouped data frame

 .groupby(['col\_1', ..., 'col\_n'])                     # Group by some columns

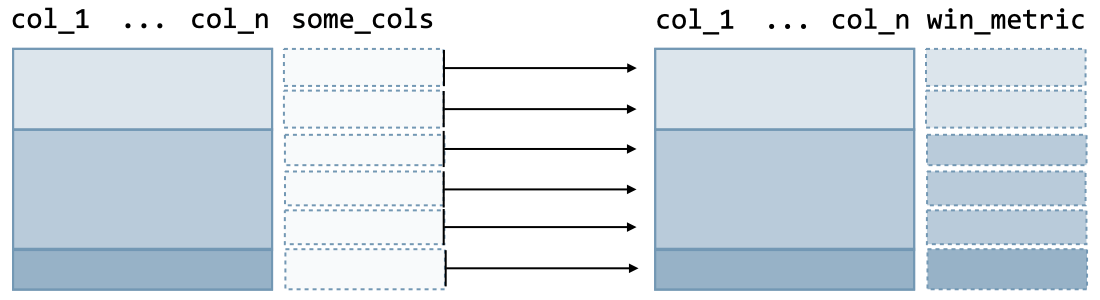
 .agg({'col': builtin\_agg})                            # Aggregation step

where builtin\_agg is among the following:

|  |  |  |
| --- | --- | --- |
| **Action** | **R Command** | **Python Command** |
| Count across observations | n() | 'count' |
| Sum across observations | sum() | 'sum' |
| Max / min of values of observations | max() / min() | 'max' / 'min' |
| Mean / median of values of observations | mean() / median() | 'mean' / 'median' |
| Standard deviation / variance across observations | sd() / var() | 'std' / 'var' |

## Window functions

**Definition**A window function computes a metric over groups and has the following structure:



The R command is as follows:

df %>%                                       # Original ungrouped data frame

 group\_by(col\_1, ..., col\_n) %>%             # Group by some columns

 mutate(win\_metric = window\_function(col))   # Window function

and the Python command is as follows:

(df

 .assign(win\_metric = lambda x:

           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:

|  |  |  |
| --- | --- | --- |
| **R Command** | **Python Command** | **Description** |
| row\_number(x) | x.rank(method='first') | Ties are given different ranks |
| rank(x) | x.rank(method='min') | Ties are given same rank and skip numbers |
| dense\_rank(x) | x.rank(method='dense') | Ties are given same rank and do not skip numbers |

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

|  |  |  |
| --- | --- | --- |
| **R Command** | **Python Command** | **Description** |
| lag(x, n) | x.shift(n) | Takes the nth previous value of the column |
| lead(x, n) | x.shift(-n) | Takes the nth following value of the column |

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

|  |  |  |
| --- | --- | --- |
| **Type** | **R command** | **Python command** |
| Scatter plot | geom\_point(   x, y, color, size, fill, alpha ) | sns.scatterplot(   x, y, hue, size ) |
| Line plot | geom\_line(   x, y, color, size, fill, alpha,   linetype ) | sns.lineplot(   x, y, hue, size ) |
| Bar chart  Histogram | geom\_bar(   x, y, color, size, fill, alpha ) | sns.barplot(   x, y, hue ) |
| Box plot | geom\_boxplot(   x, y, color ) | sns.boxplot(   x, y, hue ) |
| Heatmap | geom\_tile(   x, y, color, fill ) | sns.heatmap(   data, cmap, linecolor,   linewidth, cbar ) |

## Advanced features

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

|  |  |  |
| --- | --- | --- |
| **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,   color, fill, alpha ) |
| Text | geom\_text(   x, y, label,   hjust, vjust ) | ax.text(   x, y, s, color ) |

## Last touch

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

plot + labs(params)

where the params are summarized below:

|  |  |  |
| --- | --- | --- |
| **Element** | **R command** | **Python command** |
| Title / subtitle of the plot | title = 'text' | ax.set\_title('text', loc, pad) |
| subtitle = 'text' | plt.suptitle('text', x, y, size, ha) |
| Title of the x / y axis | x = 'text' | ax.set\_xlabel('text') |
| y = 'text' | ax.set\_ylabel('text') |
| Title of the size / color | size = 'text' / color = 'text' | ax.get\_legend\_handles\_labels() |
| Caption of the plot | caption = 'text' | ax.text('text', x, y, fontsize) |