# Data Manipulation with Python

## 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
```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' |

## Data prepr

❒ **Filtering** – ws according to some conditions as fol

Python
```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 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 |

❒ **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.

## Data frame transformation

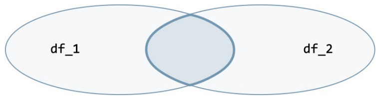
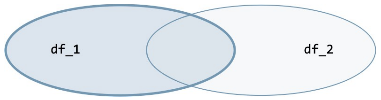
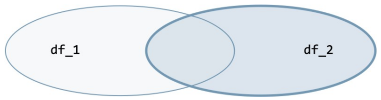
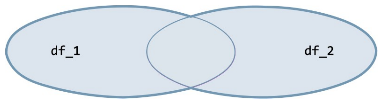
❒ **Merging data frames** – We can merge two data frames by a given field as follows:

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

where join_field indicates fields where the join needs to happen:

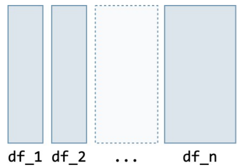
| Case | Fields are equal | Fields are different |
|---|---|---|
| Command | on='field' | left_on='field_1',right_on='field_2' |

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

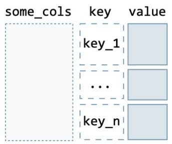
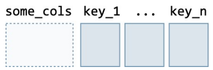
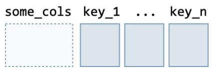
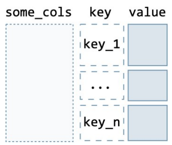
| Join type | Option | Illustration |
|---|---|---|
| Inner join | how='inner' |  |
| Left join | how='left' |  |
| Right join | how='right' |  |
| Full join | how='outer' |  |

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

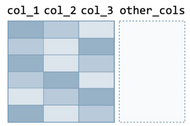
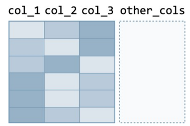
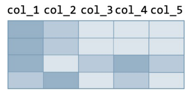
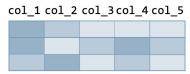
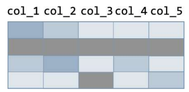
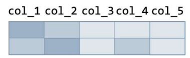
❒ **Concatenation** – The table below summarizes the different ways data frames can be con- catenated:

| 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:

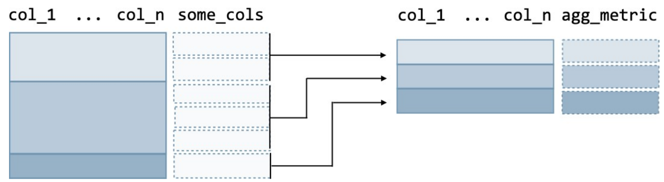
| Type | Command | Illustration | |
|---|---|---|---|
| | | Before | After |
| Long to wide | pd.pivot_table(<br>df,values='value',<br>index=some_cols,<br>columns='key',<br>aggfunc=np.sum<br>) |  |  |
| Wide to long | pd.melt( df,var_name='key',<br>value_name='value',<br>value_vars=[<br>'key_1', ...,'key_n'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

❒ **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(<br>by=['col_1', ...,'col_n'],<br>ascending=True<br>) |  |  |
| Dropping duplicates | df.drop_duplicates() |  |  |
| Drop rows with at least a null value | df.dropna() |  |  |

## 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:

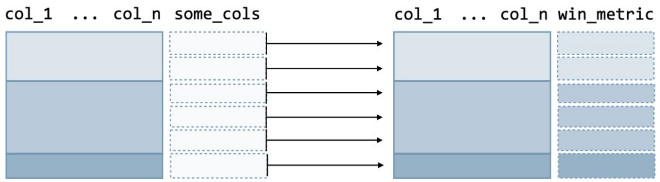
| Category | Action | Command |
|---|---|---|
| Properties | Count of observations | 'count' |
| Values | Sum of values of observations | 'sum' |
| | Max / min of values of observations | 'max' / 'min' |
| | Mean / median of values of observations | 'mean' / 'median' |
| | Standard deviation / variance 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)
  }))
)
```

## 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 $n^{\text{th}}$ previous value of the column |
| x.shift(-n) | Takes the $n^{\text{th}}$ following value of the column |