

# Study Guide: Data Manipulation with Python

Afshine AMIDI and Shervine AMIDI

August 21, 2020

## 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	<code>os.chdir(path)</code>
	Get current working directory	<code>os.getcwd()</code>
	Join paths	<code>os.path.join(path_1, ..., path_n)</code>
Files	List files and folders in a directory	<code>os.listdir(path)</code>
	Check if path is a file / folder	<code>os.path.isfile(path)</code>
		<code>os.path.isdir(path)</code>
	Read / write csv file	<code>pd.read_csv(path_to_csv_file)</code>
		<code>df.to_csv(path_to_csv_file)</code>

□ **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	<code>df[col_list]</code>
	Remove unwanted columns	<code>df.drop(col_list, axis=1)</code>
	Look at $n$ first rows / last rows	<code>df.head(n)</code> / <code>df.tail(n)</code>
	Summary statistics of columns	<code>df.describe()</code>
Paths	Data types of columns	<code>df.dtypes</code> / <code>df.info()</code>
	Number of (rows, columns)	<code>df.shape</code>

□ **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 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	<code>==</code> / <code>!=</code>
	Inequalities	<code>&lt;</code> , <code>&lt;=</code> , <code>&gt;=</code> , <code>&gt;</code>
	And / or	<code>&amp;</code> / <code> </code>
Advanced	Check for missing value	<code>pd.isnull()</code>
	Belonging	<code>.isin([val_1, ..., val_n])</code>
	Pattern matching	<code>.str.contains('val')</code>

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

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

□ **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
$\sqrt{x}$	<code>np.sqrt(x)</code>
$\lfloor x \rfloor$	<code>np.floor(x)</code>
$\lceil x \rceil$	<code>np.ceil(x)</code>

□ **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	<code>'%Y' / '%y'</code>	With / without century	2020 / 20
Month	<code>'%B' / '%b' / '%m'</code>	Full / abbreviated / numerical	August / Aug / 8
Weekday	<code>'%A' / '%a'</code>	Full / abbreviated	Sunday / Sun
	<code>'%u' / '%w'</code>	Number (1-7) / Number (0-6)	7 / 0
Day	<code>'%d' / '%j'</code>	Of the month / of the year	09 / 222
Time	<code>'%H' / '%M'</code>	Hour / minute	09 / 40
Timezone	<code>'%Z' / '%z'</code>	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:


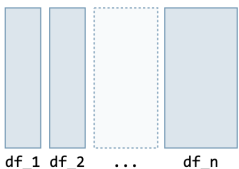
Case	Fields are equal	Fields are different
Command	<code>on='field'</code>	<code>left_on='field_1', right_on='field_2'</code>

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

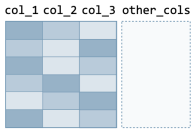
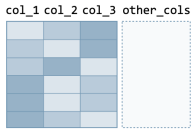
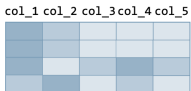
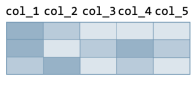
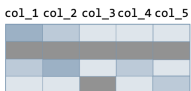
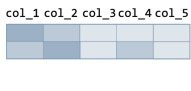
Join type	Option	Illustration
Inner join	<code>how='inner'</code>	
Left join	<code>how='left'</code>	
Right join	<code>how='right'</code>	
Full join	<code>how='outer'</code>	

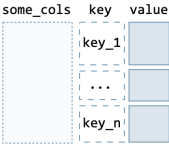

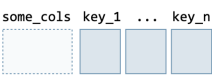
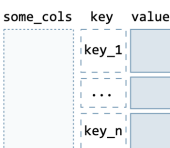
*Remark: a cross join can be done by joining on an undifferentiated column, typically done by creating a temporary column equal to 1.*

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

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

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

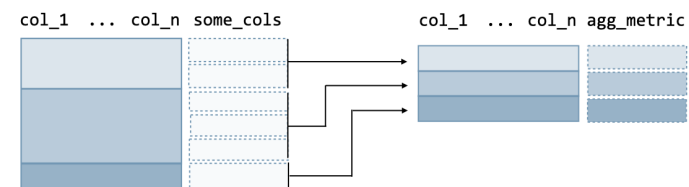
Action	Command	Illustration	
		Before	After
Sort with respect to columns	<code>df.sort_values(by=['col_1', ..., 'col_n'], ascending=True)</code>		
Dropping duplicates	<code>df.drop_duplicates()</code>		
Drop rows with at least a null value	<code>df.dropna()</code>		

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

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

## 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(lambda x: pd.Series({
        'agg_metric': some_aggregation(x)
    }))
)
```

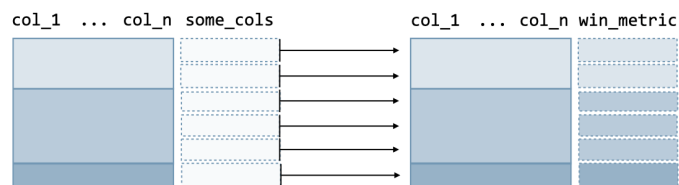
Join type	Command	Example
<code>x.rank(method='first')</code>	Ties are given different ranks	1, 2, 3, 4
<code>x.rank(method='min')</code>	Ties are given same rank and skip numbers	1, 2.5, 2.5, 4
<code>x.rank(method='dense')</code>	Ties are given same rank and do not skip numbers	1, 2, 2, 3

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

Command	Description
<code>x.shift(n)</code>	Takes the $n^{\text{th}}$ previous value of the column
<code>x.shift(-n)</code>	Takes the $n^{\text{th}}$ following value of the column

## Window functions

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



The Python command is as follows:

#### Python

```
(df
 .assign(win_metric = lambda x:
    x.groupby(['col_1', ..., 'col_n'])['col_1'].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: