Cheat sheet

Data analytics workflow

From data to insights in 4 easy steps

1 Read in data

```
import pandas as pd

## Read data
df = pd.read_csv("example_data.csv")
```

2 Explore data

```
## Explore data
df.info()
df.head()
df.describe()
```

3 Clean & filter data

```
## Clean and filter data
# Remove rows with missing values
df = df.dropna()

# Filter data
mask = df.loc[:,"feature1"] > 10
df = df.loc[mask,:]
```

4 Analyze data

4.1 Descriptive analysis

```
import matplotlib.pyplot as plt

## Descriptive analysis
# Crosstabulation
cross_tab = pd.crossstab(df.loc[:,"feature2"], df.loc[:,"target"])

cross_tab

# Scatterplot
df.plot(kind="scatter", x="feature1", y="target",
alpha=0.5, markersize=0.2, title="Feature1 vs.
Target")
plt.show()
```

4.2 Predictive analysis

```
import statsmodels.formula.api as smf

## Predictive analysis
# Linear regression model model
= smf.ols(formula="target ~ feature1", data=df)

results = model.fit()
# Summary of the regression results
results.summary()

# Predictions for a range of values
results.predict(pd.DataFrame({"feature1": range(1000)}))
```

Read in data

> Read data in csv format as a **DataFrame**

```
df = pandas.read_csv("data_dir/csv_file.csv")
```

Import data in csv format

Arguments:

header=0	Determines which line as Header serves
sep=";"	Determines how values should be separated from each other
thousands="."	Defines which character as Thousands sign is used
decimal=","	Defines which character as Decimal symbol is used
na_values=["?","??"]	Defines the values as NaN values are read in
index_col="my_index_col"	Defines the my_index_col column as an index column
columns = ["col_A", "col_B"] Sets column names	

> Create a DataFrame manually

```
df = pandas.DataFrame(data=[[1,2],[3,4]], index =
["row_1", "row_2"], columns = ["col_A", "col_B"])
```

Creates DataFrame from nested lists

> Read DataFrame from SQL database

```
engine =
sqlalchemy.create_engine("sqlite:///employees.db")
```

Specifies the database query engine

```
connection = engine.connect()
```

Represents the connection to the database

```
df = pandas.read_sql("SELECT col_A, col_B FROM table", connection,
parse_dates=["date_col"])
```

Creates **DataFrame** and reads entries in the date_col column as a date column

```
connection.close()
```

Terminates the connection to the database

> SQL query

Overview of clauses:

SELECT	Specifies the column names that will be read
SELECT *	The entire table is read in
FROM	Specifies the table to be read
WHERE	Specifies a filter condition
=,<,>,<=,>=,<>	Comparison operators
AND, OR	Combines multiple conditions
LIMIT	Limited number of rows queried
JOIN...ON	Joins two tables

Order of clauses:

```
1) SELECT table_1.col_A, table_2.col_B
2) FROM table_1
3) JOIN table_2 ON table_1.col_A = table_2.col_C
4) WHERE table_2.col_C > 30
5) LIMIT 12
```

> Explore databases

```
df = pandas.read_sql("SELECT * FROM sqlite_master", connection)
```

Gets a table containing information about a database schema SQLite database

```
inspector = sqlalchemy.inspect(engine)
```

Creates Inspector object with which databases of various Types can be explored

Methods applied to the Inspector object:

.get_table_names()	Returns the names of all tables in the database
.get_columns("table_A")	Provides information about the columns of a table

> Web crawling

```
response = requests.get(url)
```

Requests information from the server of the website in question

Arguments:

params={"attribute": "device_type"}	Translates URL into correct one format
headers={"Authorization": "token"}	Authorization via tokens
auth	Authentication with username and password

response.status_code	Stores http status code
response.headers	Stores information about query
response.encoding	Specifies the encoding of the characters (e.g. UTF-8)
html_str = response.text	Contains the text content
response.json()	Returns data in JSON format as a dict object
time.sleep(1)	Interrupts execution of Code for 1 second
response.close()	Closes the response

Explore data

> Read data in csv format as a **DataFrame**

df.shape	Returns the number of rows and columns as a tuple
df.index	Returns the line names
df.colums	Returns the column names
series.dtype	Reflects the data type of the entries in the Series. If NaN values are included, the data type is object.
df.head(8)	Returns the first 8 lines of a DataFrame
df.tail(3)	Returns the last 3 lines of one DataFrame
df.dtypes	Specifies data type of the entries in des DataFrame
df.describe()	Outputs descriptive statistics for numeric columns
series.describe()	Gives descriptive statistics for Series out
df.memory_usage()	Gives disk space usage to everyone Column in bytes
series.memory_usage()	Indicates disk space usage Series in bytes

> Arithmetic with **pandas**

df.sum()	Sum of values per column
df.mean()	Average of values per Split
series.median()	Median of the series
series.quantile(0.25)	25th percentile of the series

Clean and filter data

> Show NaN values

```
df.isna()
```

Returns DataFrame indicating whether in df at the location NaN value is (True/False)

```
df.isna().sum()
```

Returns DataFrame , the number of NaN values for each column displays

Note: The Boolean value True is interpreted as 1 , False as 0. Therefore arithmetic operations are possible, e.g. df.isna().sum().

> Remove NaN values

```
df.dropna()
```

Returns DataFrame in which rows with NaN values have been removed (not *inplace)*

Arguments:

axis	Determines whether lines or Columns with NaN values are removed
how	Determines whether row or column is removed from DataFrame if at least one NaN value or all NaN values available
thesh	Requires minimum number of NaN values
subset	Rows or columns to consider

> Replace values

```
df.replace(to_replace={"col_A": {"no": 0, "yes": 1}})
```

Replaces all no or yes values in the col_A column with 0 or 1

> Sort data

```
df.sort_values(by="col_A", ascending=False)
```

Sorts the DataFrame in descending order by values in column col_A

```
df.sort_index()
```

Sorts the DataFrame by index values

> Categories

```
series.astype("category")
```

Converts the data types of the column to the data type **category** (not *inplace)*

```
series.cat.reorder_categories(["Quarter1","Quarter2", "Quarter3","Quarter4"])
```

Assigns a specific order to values with data type cat and outputs a Series

> Connecting **DataFrames**

```
df.merge(right=df2, how="inner", on="col_A")
```

Joins two **DataFrames** similar to the JOIN command in an SQL query

Descriptive analysis

> Groups with **df.groupby()**

```
df.groupby("col_A")
```

Groups the values of all columns of the DataFrame by values of the col_A column

```
df.groupby("col_A").agg(["mean", "std"])
```

Multiple metrics can be in a DataFrame
be summarized

```
df.groupby("col_A")["col_B"]
```

A column of the grouped DataFrame is selected and returned as
a Series

> Crosstabs with **pandas.crossstab()**

```
pandas.crossstab(index=series_1, columns=series_2)
```

Creates a frequency table

Arguments:

columns="count"	Returns simple frequency table
normalize="columns"	Normalizes values across columns
normalize="index"	Normalizes values across rows
values=series	Specifies values to aggregate by
aggfunc="sum"	Determines how values are aggregated should be

> Line charts

`series.plot()`	Draws a line chart of a **Series**
`df.plot()`	Draws line graphs of all numeric columns DataFrame

> Column charts

```
df.plot(kind="bar", xlim=[-10,10], ylim=[0,60])
```

Displays the numeric columns of the DataFrame as a column chart
The x-axis is represented by the index values of the columns.
Typically the DataFrame is a frequency table.

> Histograms

```
series.plot(kind="hist", bins=20, color="red")
```

Draws a histogram of a series

```
df.groupby("col_A")["col_B"].plot(kind="hist", alpha=0.7, legend=True)
```

Draws histograms for col_B (numeric) grouped by the
Values from col_A (categorical)

> Scatterplots

```
df.plot(kind="scatter", x="col_A", y="col_B", alpha=0.7, marker="<")
```

Draws a scatterplot

```
df.groupby("col_A").plot(kind="scatter", x="col_B", y="col_C")
```

Displays the grouped values as scatterplots

```
pandas.plotting.scatter_matrix(df)
```

Represents all numeric values in the DataFrame as a scatter matrix
(also: correlogram).

> Pie charts

```
df.plot(kind="pie", y="col_A")
```

Draws a pie chart

```
df.groupby("col_A").plot(kind="scatter", x="col_B", y="col_C")
```

Represents the values of col_A named after their index values as
Pie chart. Typically the DataFrame is named df
a frequency table.

> Boxplots

`df.boxplot(column="col_A", by="col_B")`	Draws boxplot
`ax.boxplot(series)`	Draws boxplot on the axes ax

> Visualize correlation

`df.corr()`	Calculates one Correlation matrix
`seaborn.pairplot(df)`	Draws a pair plot
`seaborn.heatmap(df.corr(), annot=True)`	Draws a heatmap

> Functionalities of **matplotlib**

```
fig, axs = matplotlib.pyplot.subplots()
```

Initiates Figure and one or more Axes for a figure as subplots. If a pandas plot is then created, the respective Axes must be specified as an ax argument. For example, ax=axs[0].

Arguments:

nrows=1	number of lines
ncols=1	Number of columns
figsize=(15,4)	Size/dimensions of the chart

`fig.suptitle("figure title")` Title of the entire figure

`fig.savefig("abb.png", bbox_inches="tight")`	Saves image in png format
`fig.tight_layout()`	Labeled without overlapping
`axs[0].axis("off")`	Removes axes

stackfuel

axs[0].set(ylabel="count")	Y-axis label
axs[0].xaxis.set_tick_params(label rotation=0)	Rotation of the label
axs[0].grid(False, **axis="x")**	Removes vertical grid lines
axs[0].vlines(x=100, ymin=0, ymax=1)	Inserts vertical line
axs[0].text(x=100, y=10, s="text")	Inserts text into figure
transform = ax[0].transAxes	Normalized Coordinate systems
axs[0].annotate(s="text", xy=[1,0], **xytext=[4,8])**	Adds text with arrow in Figure one

> seaborn functionalities

```
seaborn.scatterplot(x="col_A", y="col_B", data=df, ax=ax)
```

Draws scatterplot for columns col_A and col_B from DataFrame df

Arguments:

alpha=0.8	Sets transparency of points
hue="col_C"	Color the dots according to theirs values in col_C

```
seaborn.lineplot(data=series)
```

Draws line chart for Series

```
seaborn.regplot(x="col_A", y="col_B", data=df, fit_reg=True, ax=ax)
```

Draws scatterplot with regression line

```
seaborn.boxplot(x="col_A", y="col_B", data=df)
```

Draws boxplot for values from column col_B and categories from column col_A

Predictive analysis

> Regression models

```
model = statsmodels.formula.api.ols(formula="y~x", data=df)
```

Linear regression model with dependent variable y, independent variable x

```
model = statsmodels.formula.api.logit(formula="y~x", data=df)
```

Logistic regression model with dependent variable y, independent variable x

```
results = model.fit()
```

Fits the model to the data

```
results.summary()
```

Returns coefficients, descriptive and inferential statistics for the fitted model.
Coefficients of determination Adj. R-squared or Pseudo R-squ. indicate model quality.

```
y_hat = results.predict(pd.Series(range(1000))
```

Generates predictions to values between 0 and 999 and returns them as a Series

```
y_hat[y_hat>=0.5].index(0)
```

Specifies at which index position in the series the prediction first exceeds a value of 0.5. In positive coefficient logistic regression, this is the threshold that assigns a 50% probability to class 1.