Pandas Cheat Sheet

https://pandas.pydata.org/

https://pypi.org/project/pandas/

Import

```
import pandas as pd
```

Series

Example inputs:

```
# list:
a = [1, 7, 2]

# dictionary:
kv = {"day1": 420, "day2": 380, "day3": 390}
```

Simple series, no labels:

```
myseries1 = pd.Series(a)

print(myseries1)
```

```
0    1
1    7
2    2
dtype: int64
```

Series with labels:

```
myseries2 = pd.Series(a, index=["x","y","z"])

print(myseries2)
```

```
x    1
y    7
z    2
dtype: int64
```

Key-value as series:

```
mykvseries = pd.Series(kv)
```

```
print(mykvseries)
```

```
day1    420
day2    380
day3    390
dtype: int64
```

Subset of key-value input:

```
mykvseries_filtered = pd.Series(kv, index = ["day1","day2"])

print(mykvseries_filtered)
```

```
day1    420
day2    380
dtype: int64
```

Dataframes

Input:

```
mydataset = {
    'cars': ["BMW", "Volvo", "Ford"],
    'passings': [3, 7, 2]
}
```

Load into a dataframe:

```
mydataframe = pd.DataFrame(mydataset)

print(mydataframe)
```

```
    cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2
```

Load from a File

CSV:

```
df = pd.read_csv('data.csv')
```

JSON:

```
df = pd.read_json('data.json')
```

Simple Analysis

First 10 rows:

```
print(df.head(10))
```

Last 5 rows (default):

```
print(df.tail())
```

Dataset info:

```
print(df.info())
```

The result tells us the following:

- Row count and column count
- The name of each column, with the data type
- How many non-null values there are present in each column

Clean Empty Cells

Drop empty cells, placing the results in a new dataframe:

```
new_df = df.dropna()
```

Drop empty cells, modifying the original dataframe:

```
df.dropna(inplace = True)
```

Replace empty cells with a default value (`130` in this example):

```
# WARNING: This affects all columns!

df.fillna(130, inplace = True)
```

Replace with a default value in a specific column:

```
df.fillna({"Calories": 130}, inplace=True)
```

Replace using the **mean**:

```
# Mean is the average value (the sum of all values divided by number of
values).
```

```
x = df["Calories"].mean()

df.fillna({"Calories": x}, inplace=True)
```

Replace using the **median**:

```
# Median is the value in the middle, after you have sorted all values
ascending.

x = df["Calories"].median()

df.fillna({"Calories": x}, inplace=True)
```

Replace using the **mode**:

```
# Mode is the value that appears most frequently.

x = df["Calories"].mode()[0]

df.fillna({"Calories": x}, inplace=True)
```

Clean Wrong Format

This example assumes that we have values that are not in a consistent format, but that can still be converted to a date:

```
df['Date'] = pd.to_datetime(df['Date'], format='mixed')
```

But, there may be some that can't be converted at all. They will end up with **NaT** (not a time) values. We can remove them with this:

```
df.dropna(subset=['Date'], inplace = True)
```

Clean Wrong Data

Sometimes, data is just wrong, e.g., typos.

For simple fixes, we can update the row directly:

```
# Assign a value of 45 to the Duration column in row 7:

df.loc[7, 'Duration'] = 45
```

For large data sets, use rules-based updating:

```
# For each row with a Duration value larger than 120, assign a new value of
```

```
120:

for x in df.index:
    if df.loc[x, "Duration"] > 120:
        df.loc[x, "Duration"] = 120
```

Remove bad rows altogether:

```
# For each row with a Duration value larger than 120, drop the row:

for x in df.index:
    if df.loc[x, "Duration"] > 120:
        df.drop(x, inplace = True)
```

Remove Duplicates

Find duplicates:

```
print(df.duplicated())
```

Remove them:

```
df.drop_duplicates(inplace = True)
```

Correlation

The `corr()` method calculates the relationship between each column in a data set. The closer to 1 a correlation value is, the more closely related the columns are.

A positive correlation means values are likely to move together, e.g., if one goes up, the other probably will too. A negative correlation shows the opposite, e.g., if one goes up, the other is likely to go down.

```
df.corr()
```

Example output:

	Duration	**Pulse**	**Maxpulse**	**Calories**
Duration	1.000000	-0.155408	0.009403	0.922717
Pulse	-0.155408	1.000000	0.786535	0.025121
Maxpulse	0.009403	0.786535	1.000000	0.203813
Calories	0.922717	0.025121	0.203813	1.000000

Plotting

Import `matplotlib`:

```
import matplotlib.pyplot as plt
```

Line plot (default):

```
df.plot()

plt.show()
```

Scatter plot:

```
# You can use .corr() to check for strong correlation and determine good
# argument candidates for a scatter plot.

df.corr()
```

```
df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')

plt.show()
```

Histogram:

```
df["Duration"].plot(kind = 'hist')
```

python