Python Cheet Sheet

Daten einlesen ¶

In [ ]:
```
# pandas importieren
import pandas as pd

# Daten einlesen - csv oder txt Datei
raw_data = pd.read_csv('C:/Users/xy/data.csv')

# Daten einlesen - xls, xlsx Datei
raw_data = pd.read_excel('C:/Users/xy/data.xlsx')
```

Erste Informationen über den Datensatz erhalten

In [ ]:
```
# Anzahl Zeilen und Spalten ausgeben
raw_data.shape
```

In [ ]:
```
# Anzahl Zeilen ausgeben
raw_data.shape[0]
```

In [ ]:
```
# Anzahl Spalten ausgeben
raw_data.shape[1]
```

In [ ]:
```
# Ersten 10 Zeilen ausgeben
raw_data.head(10)
```

In [ ]:
```
# Letzten 6 Zeilen ausgeben
raw_data.tail(6)
```

In [ ]:
```
# 4 Zufällige Zeilen ausgeben
raw_data.sample(4)
```

In [ ]:
```
# Ohne Argument werden bei head, tail und sample automatisch 5 Zeilen ausgegeben
raw_data.head()
```

In [ ]:
```
# Übersicht über Anzahl Werte, Mittelwert, Median, Standardabweichung, ...
raw_data.describe()
```

Indizieren / Einträge aus Datensatz auswählen

In [ ]:
```
# Generell: Achtung die Indizierung der Zeilen und Spalten beginnt jeweils mit 0
# Anmerkung hierzu: Einträge mit dem Zeilen- und Spaltenindex ist eher mühselig.
# Wenn möglich, geht man lieber über den Spaltennamen und das Datum, s. unten

# Mit Zeilenindex und Spaltenindex
data.iloc[0] # Gesamte 1. Zeile des Datensatzes auswählen (alle Spalten)
data.iloc[-1] # Letzte Zeile des Datensatzes auswählen (alle Spalten)
data.iloc[-2] # Vorletzte Zeile des Datensatzes auswählen (alle Spalten)
data.iloc[0:3] # Die ersten drei Zeilen des Datensatzes auswählen (alle Spalten, Achtung: Zeile mit Index 3 ist nicht dabei!)
data.iloc[[0,3,5]] # Zeilen mit Index 0, 3, 5 des Datensatzes auswählen
data.iloc[[0,3,4], 2] # jeweils Zeilen mit Index 0, 3 und 4 von Spalte mit Index 2 auswählen
data.iloc[[0,3,4], [2,8]] # jeweils Zeile 0, 3, und 4 von Spalten mit Indizes 2 und 8 auswählen
data.iloc[:, [3,4]] # alle Zeilen (: steht für alle) der Spalten mit Indizes 3 und 4 auswählen
data.iloc[2:5, [1,4,7]] # Zeilen mit Index 2, 3, 4 der Spalten mit Index 1, 4, und 7 auswählen
```

In [ ]:
```
# Mit Spaltenname
data['Energy'] # alle Zeilen der Spalte mit Namen 'Energy' auswählen
data['Energy'].loc[3] # 3. Zeile der Spalte mit Namen "Energy" auswählen
data['Energy'].loc[2:6] # 2. bis 5. Zeile der Spalte mit Namen "Energy" auswählen

# Achtung: hier braucht man zwei [[]]:
data[['Energy', 'Temperature']].loc[2:6] # jeweils die 2. bis 5. Zeile der Spalten "Energy" und "Temperature" auswählen
```

In [ ]:
```
# Wenn die Daten mit dem Zeitstempel indiziert sind
data.loc['2016'] # Daten des Jahres 2016
data.loc['2016':'2020'] # Daten der Jahre 2016 - 2020
data.loc['2016-01':'2016-02'] # Daten von Januar bis einschließlich Februar 2016
data.loc['2016-01-12':'2016-01-15'] # Zeitraum zwischen zwei Daten auswählen
```

In [ ]:
```
# Indizierung mit Spaltenname und Zeitstempel
#vorne Spaltenname(n) auswählen, hinten bei loc welcher Zeitbereich, z.B.
data['Energy'].loc['2016-01-12':'2016-01-15'] # Daten der Spalte "Energy" im Zeitraum zwischen den zwei angegeben Daten
data[['Solar', 'Wind']].loc['2016-01-12'] # Daten der Spalten "Solar" und "Wind" am 12.01.2016.
# Wichtig: wenn mehr als eine Spalte ausgewählt werden soll, werden die Spaltennamen in eine Liste geschrieben --> zusätzliche []
```

Mit dem Zeitstempel indizieren

In [ ]:
```
# Schritt 1 - Spalte mit dem Datum, z.B. namens 'Date', in den Datentyp datetime umwandeln
data['Date'] = pd.to_datetime(data['Date'])

# Schritt 2 - Kontrollieren, ob es geklappt hat
print(data.dtypes)

# Schritt 3 - Zeitstempel als Index setzen
data.set_index('Date', inplace=True)

# Schritt 4 - erste Zeilen ausgeben lassen um zu überprüfen, ob es geklappt hat
data.head()
```

Spalten neu benennen

In [ ]:
```
df = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
df.rename(columns={"A": "Spalte 1", "B": "Spalte 2"}, inplace=True)
```

Umgang mit NaN

In [ ]:
```
# Überprüfen, wie viele NaN jeweils in den Spalten vorhanden sind
raw_data.isna().sum()
```

In [ ]:
```
# Überprüfen, wie viele NaN insgesamt vorhanden sind
raw_data.isna().sum().sum()
```

In [ ]:
```
# Die Zeilen anzeigen, in denen sich NaN befinden
raw_data[raw_data.isna().any(axis=1)]
```

In [ ]:
```
# Alle Zeilen entfernen, in denen mindestens 1x NaN vorkommt
data = raw_data.copy() # Kopie der Rohdaten anfertigen
data.dropna(inplace=True)
```

In [ ]:
```
# Oder fehlende Werte ersetzen durch lineare Interpolation
data = raw_data.copy() # Kopie der Rohdaten anfertigen
data.interpolate(inplace=True)
```

In [ ]:
```
# Oder fehlende Werte ersetzen durch quadratische Interpolation
data = raw_data.copy() # Kopie der Rohdaten anfertigen
data.interpolate(method='polynomial', order=2, inplace=True)
```

In [ ]:
```
# Oder fehlende Werte ersetzen durch kubische Interpolation
data = raw_data.copy() # Kopie der Rohdaten anfertigen
data.interpolate(method='polynomial', order=3, inplace=True)
```

In [ ]:
```
# Oder fehlende Werte ersetzen durch filling forward
data = raw_data.copy() # Kopie der Rohdaten anfertigen
data.interpolate(method='pad', limit=2, inplace=True)
# wegen limit=2 werden maximal 2 fehlende Werte nacheinander ersetzt
```

Resampling: Frequenz von Zeitreihen ändern

In [ ]:
```
# Downsampling (ursprüngliche Frequenz ist höher)
data_rs = data.resample('15min').mean() # Auf 15min Frequenz bringen, indem jeweils der Mittelwert über 15min gebildet wird
data_rs = data.resample('H', label='right').sum() # Auf stündliche Frequenz bringen, indem Summe über eine Stunde gebildet wird
data_rs = data.resample('H').first() # Auf stündliche Frequenz bringen; jeweils der Zeitstempel zur vollen Stunde wird verwendet
```

In [ ]:
```
# Upsampling (ursprüngliche Frequenz ist niedriger)
data.resample('1min').pad() # Auf 1min Frequenz bringen, fehlende Werte über filling forward generieren
data.resample('1min').interpolate() #Auf 1min Frequenz bringen, fehlende Werte über lineare Interpolation generieren
```

Die Argumente von resample geben die gewünschte Frequenz für das Resamplen an. Hier die wichtigsten:

minütlich: 'T'; beliebige Anzahl an Minuten: z.B. '4min'; stündlich: 'H'; täglich: 'D'; monatlich: 'M'; jährlich: 'Y'

Plotten

In [ ]:
```
# matplotlib importieren
import matplotlib.pyplot as plt

# für schönere Plots zusätzlich seaborn importieren
import seaborn as sns
# Benutze seaborn style defaults und setze die default Größe für die Plots
sns.set(rc={'figure.figsize':(18, 4)})

# einfacher Linienplot für Zeitreihen
figure = data.plot() # Linienplot ohne Marker
figure = data.plot(marker='o') # Linienplot mit Marker

# Boxplot
sns.boxplot(x='Month', y='Consumption', data=raw_data) #für x und y die gewünschten Spaltennamen eingeben

# Scatterplot
sns.scatterplot(x='Temperature', y='Consumption', data=raw_data) #für x und y die gewünschten Spaltennamen eingeben
```

Daten als csv-Datei speichern

In [ ]:
```
data.to_csv('out.csv')

# komprimiert
compression_opts = dict(method='zip',archive_name='out.csv')
data.to_csv('out.zip', compression=compression_opts)
```