DataCraft

1 Pandas

DataCraft

- pandas ist das beliebteste und nützlichste Package zum Umgang mit Datentabellen. pandas basiert auf numpy

- Das zentrale Objekt in pandas ist der **DataFrame**, der einer Datentabelle entspricht. Jede Spalte ist eine **Serie** und hat einen festgelegten Datentyp.

- Die gebräuchliche Abkürzung für pandas ist pd

```
import pandas as pd
```

DataCraft

- Eine Serie lässt sich aus einer Liste erzeugen

```
s = pd.Series(["rot","grün","blau","gelb"])
```

- oder aus einem dictionary

```
d = {'Berlin':3645000, 'München':1472000, 'Hamburg':1841000,
     'Köln':1086000 }
staedte = pd.Series(d)
```

- Zugriff auf einzelne Elemente wie bei einer Liste

```
zahlen[3]
zahlen[0:3]
zahlen[990:]
```

- Die Zeilenidentifikation ist der sogenannte Index. Bei einem dictionary werden die Schlüssel zum Index

```
staedte["Hamburg"]
staedte[["Hamburg","München"]]
```

- NaN (not a number) steht für einen fehlenden Wert

```
s = ["Berlin","München","Hamburg","Köln","Bielefeld"]
pd.Series(staedte, index=s)
```

Ein DataFrame setzt sich aus Serien mit Namen zusammen

Name *object*	**Alter** *int64*	**Wohnort** *object*
Katja	32	Berlin
Nina	32	München
Sven	36	Hamburg
Matthias	31	Köln

DataCraft

- DataFrames lassen sich aus Listen erzeugen

```
df = pd.DataFrame(["rot","grün","blau","gelb"], columns=["Farbe"])
```

- oder aus dictionaries mit Listen als Werte

```
d = {"Name": ['Katja', 'Nina', 'Sven', 'Matthias'],
     "Alter": [32, 32, 36, 31],
     "Ort":['Berlin', 'München', 'Frankfurt', 'Köln']
    }
df = pd.DataFrame(d)
```

- Das Package seaborn bringt einige Beispiel-Datensätze mit:

```
import seaborn as sns
print(sns.get_dataset_names())
titanic = sns.load_dataset('titanic')
```

Zugriff auf Zeilen und Spalten

DataCraft

- Erste n Zeilen ausgeben. Defaultwert ist n=5

```
df.head()
```

- Zugriff auf die Spalten über den Spaltennamen

```
df.columns
df["Alter"]            # es geht auch df.Alter
df[["Name","Alter"]]
```

- Zugriff auf die Zeilen (und Spalten) per iloc (iloc = index location)

```
df.iloc[0]
df.iloc[0,0]
df.iloc[0,0:2]
```

Zugriff auf Zeilen und Spalten

DataCraft

- Oder Zugriff über loc per Index-Label

```
df.loc[:,"Alter"]
# Der Zeilenindex kann auch per Label angesprochen werden
df.index = df["Name"]
df.loc[["Nina","Sven"],"Ort"]
```

- loc versteht auch booleans

```
df.loc[df["Alter"]>=32]
df.loc[df['Name'].str[:1]=="M"]
df.loc[df["Alter"].isnull()]
```

- Zuweisung von Werten möglich

```
df.loc['Sven':,'Alter'] = 30
```

Spalten bearbeiten

DataCraft

- Eine Spalte hinzufügen

```
df['fuenf'] = 5
df['neue Spalte'] = 2*df['alte Spalte']
```

- Eine Spalte löschen

```
del df['fuenf']
```

- Spalten umbenennen

```
df.rename(columns = {'alt1':'neu1','alt2':'neu2'}, inplace = True)
df.columns = ['neu1', 'neu2', 'neu3', 'neu4']
```

- Die Funktion Rename kann auch Funktionen entgegennehmen

```
df.rename(columns= str.upper, inplace = True)
df.rename(columns=lambda x: f'T1_{x}', inplace = True)
```

Zeilen bearbeiten

DataCraft

- Zeilen anhand des Index löschen

```
df = df.drop(1)
df = df.drop([0,1,2,3])
```

- Funktioniert auch für Spalten mit dem Parameter axis=1

```
df.drop('fuenf', axis=1, inplace=True)
```

- Werte anhand von boolscher Bedingung ändern

```
df.loc[df["Alter"]>=32,'Ort'] = 'Berlin'
```

Broadcasting

- Wird ein Wert von einer Serie bzw. DataFrame abgezogen, wird der Wert von jedem Eintrag abgezogen

```
s1 = pd.Series(rng.standard_normal(10)) – 5
```

- Auch für DataFrames möglich, braucht man aber selten. Bei Verrechnung von Series mit DataFrame wählen, ob zeilen- oder spaltenweise

- Sortieren nach dem Index (zeilen- oder spaltenweise).

```
df.sort_index()
```

- Sortieren nach Werten einer/mehrerer Spalte(n)

```
df.sort_values(by='Wert2')
df.sort_values(by=['Wert2','Wert3'], ascending=[False, True])
```

- Ausgabe des sortieren DataFrames. Änderungen mit Zuweisung oder Inplace-Parameter

- Index der minimalen/maximalen Ausprägung mit **idxmin()** und **idxmax()**

```
df.idxmin()
df.idxmax()
```

- Zeile, bei denen eine Spalte den maximalen Wert enthält

```
df.loc[df['Wert1'].idxmax()]
```

- **argmin()** und **argmax()** liefern die Position (Zeilen-/Spaltennummer)

Eindeutige Werte und Duplikate

DataCraft

- Mit **isin()** wird für jeden Wert einer Serie überprüft, ob er in einer Liste ist

```
s.isin(['a','b'])
penguins[penguins['species'].isin(['Adelie'])]
```

- **unique()** gibt die eindeutigen Werte aus

```
import seaborn as sns
penguins = sns.load_dataset("penguins")
penguins["species"].unique()
```

- **duplicated()** gibt eine boolean Serie zurück, ob die Zeile schon vorhanden war. Mit **drop_duplicates()** können die Duplikate entfernt werden

- Mit **all()** wird geprüft, ob alle Elemente einer boolschen Serie True sind. Nur dann wird True zurückgegeben, andernfalls False

```
titanic["who"].isin(["child","woman"]).all()
```

- **any()** prüft, ob es mindestens ein Element einer boolschen Serie gibt, welches den Wert True hat

```
(titanic["age"] > 75).any()
```

Kategorien und Ersetzungen

DataCraft

- **value_counts()** gibt die Anzahl der eindeutigen Werte aus

- **cut()** sortiert Werte in Intervalle

```
pd.cut(penguins["body_mass_g"], bins=3)
```

- Werte können mit **replace()** ersetzt werden. Mehrere Ersetzungen mit Listen oder Dictionaries

```
df.replace('alter Wert','neuer Wert')
df.replace(['alt1','alt2'],'neu')
df.replace({'alt1':'neu1','alt2':'neu2'}, inplace=True)
```

Apply

DataCraft

- Anwenden einer Funktion spaltenweise oder zeilenweise

```
df.apply(max)
df.apply(max, axis=1)
```

- Kombination mit eigener Funktion

```
df.apply(meine_funktion)
df.apply(meine_funktion, axis=1)
```

- Für viele Aggregat-Funktionen (sum, max, …) nicht nötig, da diese einen axis-Parameter mitbringen

- Rückgabe von mehreren Elementen möglich

```
def f(x):
    return pd.Series([x.min(), x.max()], index=['min', 'max'])
df.apply(f, axis=1)
```

Map und ApplyMap

DataCraft

- **map** ermöglicht das Anwenden einer Funktion auf jedes Element einer Serie

```
f = lambda x: f'{x:.2f}'
df['Wert1'].map(format)
```

- **applymap** wendet eine Funktion auf jedes Element eines DataFrames an

```
df.applymap(lambda x: f'{x:.2f}')
```

- Zufällige Stichprobe mit **sample()**. Der Parameter n gibt die Größe an. Mit dem Parameter replace=True kann eine Zeile mehrfach ausgewählt werden (Ziehen mit Zurücklegen)

```
train_rate = 0.8
train_n = round(train_rate * coffee.shape[0])
train = coffee.sample(n = train_n)
test = coffee[~pd.Series(coffee.index).isin(pd.Series(train.index))]
print(train.shape)
print(test.shape)
```