Einführung in pandas: Datenanalyse mit Python

Till Frankenbach

Was ist pandas?

pandas ist eine Open-Source-Bibliothek für Datenmanipulation und -analyse in Python.

- Einfache Handhabung großer Datensätze
- Aufbau auf NumPy für hohe Performance
- Gute Integration mit anderen Data-Science-Bibliotheken (z. B. Matplotlib, Scikit-learn)

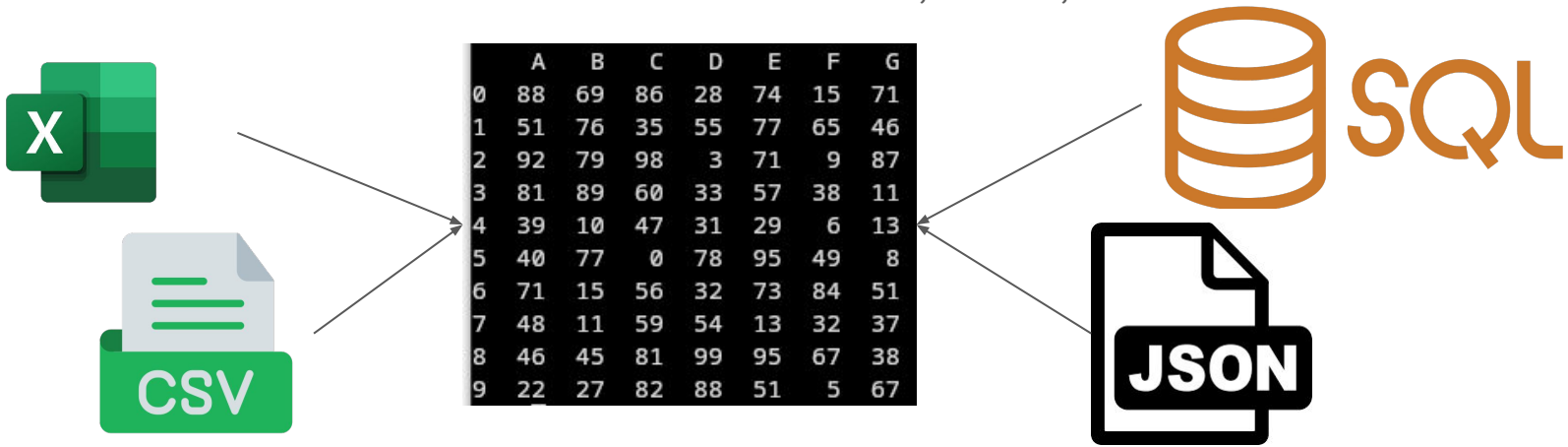
Warum pandas verwenden?

Vereinfacht Datenverarbeitung:

- Daten einfach bereinigen, transformieren und analysieren.

Flexibel & Leistungsstark:

- Unterstützt verschiedene Dateiformate wie CSV, Excel, SQL und JSON.



	A	B	C	D	E	F	G
0	88	69	86	28	74	15	71
1	51	76	35	55	77	65	46
2	92	79	98	3	71	9	87
3	81	89	60	33	57	38	11
4	39	10	47	31	29	6	13
5	40	77	0	78	95	49	8
6	71	15	56	32	73	84	51
7	48	11	59	54	13	32	37
8	46	45	81	99	95	67	38
9	22	27	82	88	51	5	67

Datenstrukturen in pandas

Series:
Eindimensional, ähnlich einer Liste oder einer Spalte.

DataFrame:
Zweidimensional, wie eine Tabelle in SQL oder ein Tabellenblatt.

```
s = pd.Series()
```

```
0    88
1    51
2    92
3    81
4    39
5    40
6    71
7    48
8    46
9    22
Name: A, dtype: int64
```

```
df = pd.DataFrame()
```

	A	B	C	D	E	F	G
0	88	69	86	28	74	15	71
1	51	76	35	55	77	65	46
2	92	79	98	3	71	9	87
3	81	89	60	33	57	38	11
4	39	10	47	31	29	6	13
5	40	77	0	78	95	49	8
6	71	15	56	32	73	84	51
7	48	11	59	54	13	32	37
8	46	45	81	99	95	67	38
9	22	27	82	88	51	5	67

Was ist eine Series?

Eine Series ist eine eindimensionale Datenstruktur mit beschrifteten Werten, ähnlich einer Liste.

pd.Series([1, 2, 3, 4])

```
>>> pd.Series([1, 2, 3, 4])
0    1
1    2
2    3
3    4
dtype: int64
```

Was ist ein DataFrame?

Ein DataFrame ist eine zweidimensionale Datenstruktur, die aus mehreren Series besteht, ähnlich wie eine Tabelle.

pd.DataFrame({'A': [1, 2], 'B': [3, 4]})

```
>>> pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
   A  B
0  1  3
1  2  4
```

Unterstützte Datenformate in pandas

CSV: Weit verbreitet für Datenaustausch

```
pd.read_csv('file.csv')
```

Excel: Integration von .xlsx und .xlsx-Dateien

```
pd.read_excel('file.xlsx')
```

SQL: Verbindung zu SQL-Datenbanken für Datenabfragen

```
pd.read_sql('SELECT * FROM table', connection)
```

JSON: Häufig genutzt für Webdaten und APIs

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

Daten erkunden mit pandas

Wichtige Methoden für den Überblick:

- .head(): Zeigt die ersten Zeilen, um einen schnellen Überblick zu bekommen.
- .info(): Zeigt die Struktur und den Datentyp jeder Spalte, nützlich für große Datenmengen.
- .describe(): Gibt statistische Kennzahlen für numerische Spalten aus.

Daten erkunden mit pandas: df.head()

Wichtige Methoden für den Überblick:

- .head(): Zeigt die ersten Zeilen, um einen schnellen Überblick zu bekommen.

```
>>> df = pd.DataFrame(np.random.randint(0,100,size=(100, 7)), columns=list('ABCDEFG'))
>>> df.head()
    A   B   C   D   E   F   G
0  13  11  49  26  10  27  13
1  16  52  40  37  54  96  55
2  11  86  99  71  95  43  74
3  16  83  15  33  71  38   9
4  35  82  98  73  27  81   3
```

Daten erkunden mit pandas

Wichtige Methoden für den Überblick:

- .info(): Zeigt die Struktur und den Datentyp jeder Spalte, nützlich für große Datenmengen.

```
>>> df = pd.DataFrame(np.random.randint(0,100,size=(100, 7)), columns=list('ABCDEFG'))
>>> df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       100 non-null    int64
 1   B       100 non-null    int64
 2   C       100 non-null    int64
 3   D       100 non-null    int64
 4   E       100 non-null    int64
 5   F       100 non-null    int64
 6   G       100 non-null    int64
dtypes: int64(7)
memory usage: 5.6 KB
```

Daten erkunden mit pandas

Wichtige Methoden für den Überblick:

- .describe(): Gibt statistische Kennzahlen für numerische Spalten aus.

```
>>> df = pd.DataFrame(np.random.randint(0,100,size=(100, 7)), columns=list('ABCDEFG'))
>>> df.describe()
                A           B           C  ...           E           F           G
count  100.000000  100.000000  100.000000  ...  100.000000  100.000000  100.000000
mean    54.030000   51.380000   49.400000  ...   48.980000   46.040000   49.210000
std     28.848421   31.606412   29.635155  ...   27.125217   29.967295   29.181527
min      0.000000    1.000000    0.000000  ...    1.000000    1.000000    0.000000
25%     31.750000   27.000000   24.000000  ...   27.500000   16.750000   23.750000
50%     59.500000   53.000000   53.000000  ...   45.000000   42.000000   50.000000
75%     78.500000   81.250000   72.500000  ...   71.250000   72.250000   74.000000
max     99.000000   99.000000   99.000000  ...   99.000000   98.000000   99.000000

[8 rows x 7 columns]
```

Indexierung und Auswahl von Daten in pandas

Was ist Indexierung?:

- Der Prozess, bestimmte Daten innerhalb eines DataFrames oder einer Series auszuwählen.

Warum ist es wichtig?:

- Ermöglicht das Extrahieren, Filtern und Bearbeiten von Daten.

Auswahl von Spalten und Zeilen

Spalten auswählen:

- Einzelne Spalte: df['Spalte'] oder df.Spaltname
- Mehrere Spalten: df[['Spalte1', 'Spalte2']]
- Mit .iloc[] (positionsbasiert) und .loc[] (labelbasiert).
  - df.iloc[0]
  - df.loc['a']

Auswahl von Spalten und Zeilen

Spalten auswählen:

- Einzelne Spalte: df['Spalte'] oder df.Spaltname

```
>>> df = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
>>> df
   A  B
0  1  3
1  2  4
>>> df["A"]
0    1
1    2
Name: A, dtype: int64
```

Auswahl von Spalten und Zeilen

Spalten auswählen:

- Mehrere Spalten: df[['Spalte1', 'Spalte2']]

```
Name: A, dtype: int64
>>> df
   A  B
0  1  3
1  2  4
>>> df[["B", "A"]]
   B  A
0  3  1
1  4  2
```

Auswahl von Spalten und Zeilen

Spalten auswählen:

- Mit .iloc[] (positionsbasiert) und .loc[] (labelbasiert).
  - df.iloc[0]
  - df.loc['a']

```
>>> df
   A  B
0  1  3
1  2  4
>>> df.iloc[0]
A    1
B    3
Name: 0, dtype: int64
```

```
>>> df.index = ["C", "D"]
>>> df
   A  B
C  1  3
D  2  4
>>> df.loc["C"]
A    1
B    3
Name: C, dtype: int64
```

Bedingte Auswahl mit booleschen Werten

Filtern mit Bedingungen: Verwenden von Vergleichsoperatoren wie ==, >, < etc.

```
>>> df = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
>>> df[df < 3]
   A   B
0  1 NaN
1  2 NaN
>>> df[df == 3]
    A    B
0 NaN  3.0
1 NaN  NaN
```

Daten bereinigen – Umgang mit fehlenden Werten

Methoden zur Bereinigung von Daten:

- .dropna(): Entfernt Zeilen oder Spalten mit fehlenden Werten.
- .fillna(value): Füllt fehlende Werte mit einem festgelegten Wert.

```
>>> df = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
>>> df.loc[1, "A"] = None
>>> df
     A  B
0  1.0  3
1  NaN  4
```

Daten bereinigen – Umgang mit fehlenden Werten

Methoden zur Bereinigung von Daten:

- .dropna(): Entfernt Zeilen oder Spalten mit fehlenden Werten.

```
>>> df
     A  B
0  1.0  3
1  NaN  4
>>> df.dropna()
     A  B
0  1.0  3
```

Daten bereinigen – Umgang mit fehlenden Werten

Methoden zur Bereinigung von Daten:

- .fillna(value): Füllt fehlende Werte mit einem festgelegten Wert.

```
>>> df
     A  B
0  1.0  3
1  NaN  4
>>> df.fillna(100)
       A  B
0    1.0  3
1  100.0  4
```

Zusammenführen und Verbinden von Daten in pandas

Was ist Merging und Joining?:

- Das Verbinden mehrerer DataFrames basierend auf gemeinsamen Werten oder Schlüsseln.

Warum ist es nützlich?:

- Ermöglicht die Integration verschiedener Datenquellen, was in der Datenanalyse und -verarbeitung oft erforderlich ist.

Zusammenführung mit pd.merge()

Syntax: pd.merge(left, right, on='Schlüssel', how='Typ')

```
>>> df1
   ID     Name
0   1    Alice
1   2      Bob
2   3  Charlie
>>> df2
   ID  Alter
0   1     25
1   2     30
2   4     35
>>> pd.merge(df1, df2, on='ID', how='inner')
   ID   Name  Alter
0   1  Alice     25
1   2    Bob     30
```

Grouping

Was bedeutet Gruppieren?:

- Gruppieren von Daten nach bestimmten Kategorien und Zusammenfassen der Daten mit statistischen Methoden.

Warum ist es wichtig?:

- Nützlich für Analysen, bei denen Daten zusammengefasst werden müssen, wie z.B. Durchschnittswerte pro Kategorie.

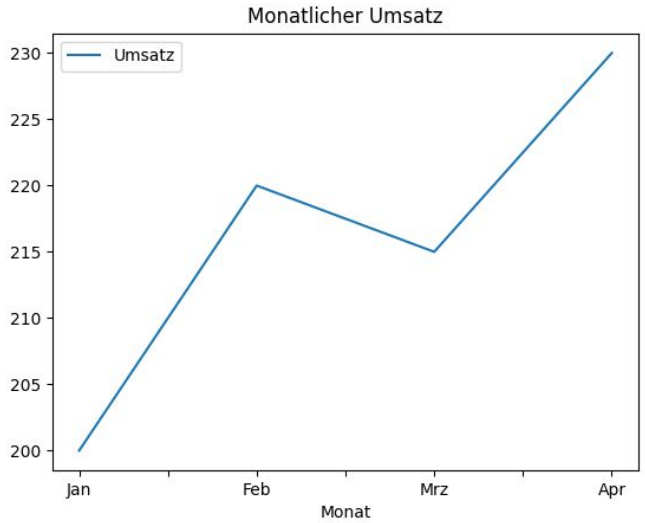
Gruppierung mit .groupby()

Syntax: df.groupby('Spalte')

```
>>> df
     Stadt Produkt  Verkäufe  Jahr
0   Berlin       A       100  2023
1   Berlin       B       150  2023
2  Hamburg       A       200  2023
3  Hamburg       B       120  2023
4  München       A       130  2023
5  München       B        90  2023
6   Berlin       A       110  2024
7  Hamburg       B       220  2024
8  München       A       160  2024
9   Berlin       B       140  2024
>>> gruppiert = df.groupby('Stadt')
>>> gruppiert.sum()
        Produkt  Verkäufe  Jahr
Stadt
Berlin     ABAB       500  8094
Hamburg     ABB       540  6070
München     ABA       380  6070
```

Datenvisualisierung

import matplotlib.pyplot as plt




```
>>> df
  Monat  Umsatz
0   Jan     200
1   Feb     220
2   Mrz     215
3   Apr     230
>>> df.plot(x="Monat", y="Umsatz", kind="line")
<Axes: xlabel='Monat'>
>>> plt.title("Monatlicher Umsatz")
Text(0.5, 1.0, 'Monatlicher Umsatz')
>>> plt.show()
```

Übung

Data:
https://raw.githubusercontent.com/mwaskom/seaborn-data/master/flights.csv

Online-Compiler:
https://www.w3schools.com/python/pandas/pandas_compiler.asp

a) Finden Sie die durchschnittliche und maximale Passagierzahl
b) Ermitteln Sie den Durchschnitt der Passagierzahlen pro Monat
c) Wie viele Monate waren die Passagierzahlen über 150

a) Finden Sie die durchschnittliche und maximale Passagierzahl

```
print(df.describe())
```

b) Ermitteln Sie den Durchschnitt der Passagierzahlen pro Monat

```
monthly_avg = df.groupby('month')['passengers'].mean()

print("Durchschnittliche Passagierzahlen pro Monat:")

print(monthly_avg)
```

c) Wie viele Monate waren die Passagierzahlen über 150

```
df[df['passengers'] > 150].shape[0]
```