



Coach Market Woche 4

Robert Heise und Florian Edenhofner – Education4Industry GmbH

26.11.2021

University4Industry

Agenda

1. Getting Started with Jupyter Notebook and Python
2. Pandas Fundamentals

Getting Started with Jupyter Notebook and Python

Was ist Jupyter Notebook?

Jupyter Notebook ist eine interaktive Arbeitsumgebung für Data Science und keine Umgebung zur Softwareentwicklung.

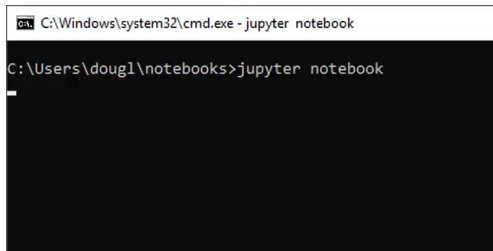
- Anwendung über Browser
- Extensions für Entwicklungsumgebungen (z.B. VS-Code) ermöglichen die Bearbeitung von Jupyter Notebook
- JupyterLab fügt weitere Funktionen hinzu

Unser heutiges Ziel:

- Verständnis der grundlegender Arbeitsweise und Anwendung in den Übungen

Jupyter Notebook: Starten

Jupyter Notebook vom Terminal/Anaconda Prompt aus starten:



```
C:\Windows\system32\cmd.exe - jupyter notebook  
C:\Users\dougl\notebooks>jupyter notebook  
_
```

Figure 1: Jupyter vom Terminal aus starten

Anschließend neues *.ipynb erstellen und benennen.

Jupyter Notebook: Unterschied Command Mode und Edit Mode

Im **Edit Mode** können Sie wie in einem normalen Texteditor die Zellen bearbeiten.

Im **Command Mode** können Sie das Notebook als Ganzes bearbeiten, aber nicht in den Inhalt einzelner Zellen.

Hilfe zu den Keyboard Shortcuts findet man mit der **Taste H** während man im Command Mode ist.

Esc : Wechsel von Edit Mode zu Command Mode

Ctrl+Enter : Führt Zelle aus

Shift+Enter : Führt Zelle aus + springt in nächste Zelle

- **Alt+Enter** : Führt Zelle aus + fügt neue Zelle darunter ein + Edit Mode
- **A**: Einfügen einer leeren Zelle über der ausgewählten (above)
- **B**: Einfügen einer leeren Zelle unter der ausgewählten (below)
- **D zweimal**: Löschen einer Zelle
- **Z** : Undo Löschen einer Zelle
- **C** : Zelle kopieren
- **X**: Zelle ausscheiden
- **V**: Zelle aus der Ablage unter der ausgewählten Zelle einfügen

Enter: Wechsel von Command Mode zu Edit Mode

Ctrl+Enter : Führt Zelle aus und springt in den Command Mode

Shift+Enter : Führt Zelle aus + springt in nächste Zelle und den Command Mode

- **Alt+Enter :** Führt Zelle aus + fügt neue Zelle darunter ein
- **Ctrl-X:** Ausschneiden des markierten Bereichs
- **Ctrl-C:** Kopieren markierten Bereichs
- **Ctrl-V:** Einfügen markierten Bereichs
- **Ctrl-Z:** Undo

Markdownzellen

Zellen können Markdown wiedergeben, um Text strukturiert darzustellen

- Taste M im Command Mode ändert Zelle zu Markdownzelle
- Taste X im Command Mode ändert Zelle zu Codezelle

Inline Help

- Inline help: z.B. `random.shuffle?` -> Signature und Docstring (Docstring ist eine Beschreibung in Modul, Klasse oder Funktion)

Magic Commands

- `%lsmagic` listet alle magic commands
- `%whos` zeigt Informationen zu Variablen
- `%whos?` Inline Help funktioniert auch mit magic commands

Pandas Fundamentals

Eine Bibliothek für Python für das Manipulieren und Analysieren strukturierter multidimensionaler Daten (Tabellen).

- Open-Source Bibliothek
- komplett in Python integriert
- großer Funktionsumfang

Unser heutiges Ziel:

- Kennenlernen der grundlegenden Datenobjekte
- Einstieg in grundlegende Funktionen

Pandas: Datenobjekte

```
1 import numpy as np
2 import pandas as pd
```

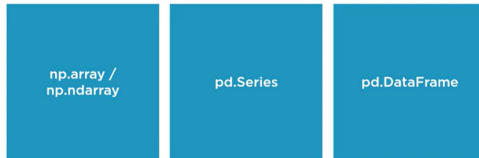


Figure 2: Verschiedene Datenobjekte in Pandas und Numpy

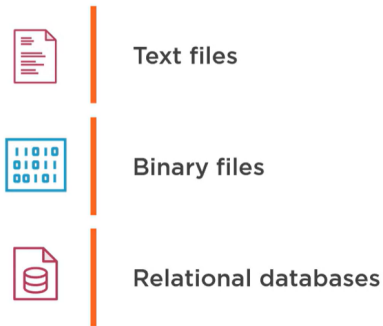


Figure 3: Übliche Datenquellen

Beispiel

Die ersten 5 Zeilen laden und als Index 'id' nutzen:

```
1 pd.read_csv(csv_path, nrows=5, index_col = 'id')
```

Series

```
1 s = pd.Series()           # Konstruktor erstellt leeres Objekt
2 s = pd.Series([2,13,3,6]) # Übergabe der Daten mittels List
```

Dataframe

```
1 df = pd.DataFrame()       # Konstruktor erstellt leeres Objekt
2 df = pd.DataFrame(        # Übergabe der Daten an den Konstruktor z.B. mittels Dict
3     {'col1': [2.3,5.1,7.2],
4     'col2': ['Montag', 'Dienstag', 'Mittwoch']
5     })
6 .
7 .
8 .
9 df = pd.read_csv(csv_path, nrows=5, index_col = 'id') # Lesen aus CSV-Datei
```

Auswahl von Spalten

```
1 df['col'] # Einzelner Spaltenname gibt eine Series zurück
2 df[['col']] # Liste von Spalten gibt einen Dataframe zurück
3 df[['col1', 'col2']] # mehrere Spalten werden zurückgegeben
```

Auswahl von Spalten und Zeilen mit der Methode 'loc'

Auswahl mittels der Labels der Spalten und Zeilen

```
1 df.loc['row1', 'col1'] # einzelnes Element
2 df.loc[['row1', 'row2'], ['col1', 'col2']] # mehrere Elemente
```

Auswahl von Spalten und Zeilen mit der Methode 'iloc'

Auswahl mittels der Reihenfolge der Spalten und Zeilen

```
1 df.iloc[4, 3] # einzelnes Element
2 df.iloc[[2,3,4], [0,1,2]] # mehrere Elemente
3 df.iloc[10:40, 2:5] # Slicing
```

```
1 df['col'] == 'name'      # Gibt eine Series mit Boolean Werten zurück

1 df[df['col'] == 'name'] # Gibt die Elemente df's zurück für welche die Bedingung wahr ist
2 df[(df['col1'] == 'name') | (df['col2'] == 'name2')] # ODER-Verbindung
```


Funktionen des Pakets Pandas

Beispiele:

```
1  pd.unique(Series)                # Ausgabe alle vorkommenden Werte in der Series
2  pd.isnull(DataFrame)            # Rückgabe eines Dataframes mit Boolean für Bedingung
3  pd.concat((DataFrame,DataFrame)) # Aneinanderhängen zweier Dataframes
4  pd.merge(df1,df2,on='col_name',how='right') # Verbund zweier Dataframes
5  ...
```

Methoden eines Dataframes/Series

Beispiele:

```
1  df['col'].value_counts()          # Häufigkeit der Werte einer Spalte
2  df.sort_values(by=['col1','col2']) # Sortieren
3  df['col'].mean()                 # Mittelswert der Spalte 'col'
4  df.append(df2)                   # Anhängen eines Dataframes
5  df.join(df2, how='inner')         # Verbund zweier Dataframes
6  ...
```

groupby-Objekt

```
1 df.groupby('col_group')
```

Lösung für diverse Problem mit Iteration über Groupby-Objekt und Funktion:

```
1 for name,group in df.groupby('col_group')  
2     .  
3     .  
4     .
```

Pandas Build-in für Gruppen

```
1 df.groupby('col_group')['col_value'].min() # Anwendung einer Build-In-Funktion
2 df.groupby('col_group')['col_value'].agg(np.min) # Aggregation mittels einer Numpy-Funktion
3 df.groupby('col_group')['col_value'].transform(func) # Transformation mittels einer Funktion
4 df.groupby('col_group').filter(lambda x: len(x.index) > 1) # Filtern einzelner Gruppen
```