Mehr auf die x-Achse

by Woche 11

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

Nach all der Datenverarbeitung mit Pandas wollen wir uns zur Abwechslung mal wieder mit der Datenvisualisierung beschäftigen. Schon lange überfällig ist mittlerweile, dass wir auch mal mehr als eine Variable auf der x-Achse darstellen. Zur Erinnerung, bisher hatten wir beispielsweise in Kapitel 4.5 die Noten von zwei Personen in zwei separaten Darstellungen mit seaborn geplottet. Wahrscheinlich ist das den meisten von euch schon damals eigenartig vorgekommen, da es doch viel sinnvoller wäre, die Noten beider Personen in einer Darstellung zu vergleichen. Die Idee zu dem Zeitpunkt war, dass wir uns erstmal auf die Grundlagen konzentrieren und uns dann später um solche Feinheiten kümmern. Nun ist es soweit.

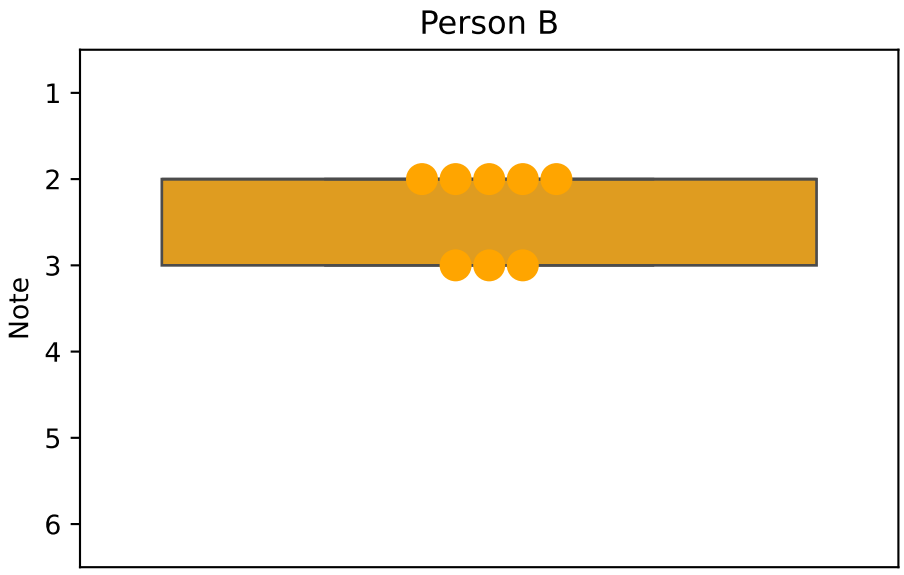
Noten pro Person

Rückblick Kapitel 4.5

Für den direkten Vergleich, wollen wir auch wirklich genau dieselben Daten verwenden, die wir schon in Kapitel 4.5 verwendet haben. Also laden wir die Daten nochmal ein und erzeugen sogar nochmal die separaten Abbildungen:

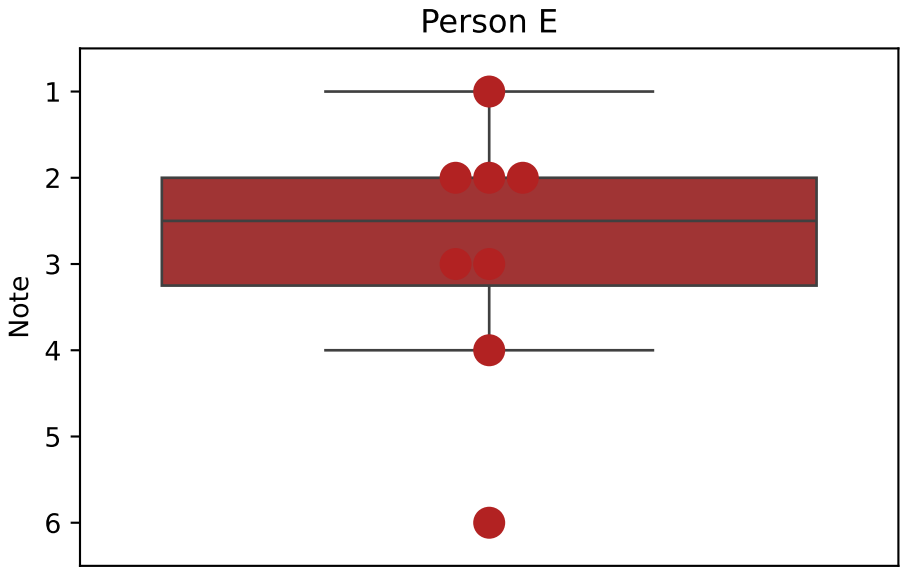
```
noten_B = np.array([2, 3, 3, 2, 2, 2, 3, 2])
pseudo_x = np.zeros(len(noten_B))

plt.figure()
plt.title('Person B')
sns.swarmplot(
  x=pseudo_x,
  y=noten_B,
  color='orange',
  size=12
)
sns.boxplot(
  x=pseudo_x,
  y=noten_B,
  color='orange'
```

```
)
plt.yticks(np.arange(1, 7))
plt.ylim(6.5, 0.5)
plt.ylabel('Note')
plt.xticks([])
plt.show()
```




```
noten_E = np.array([2, 3, 4, 2, 1, 2, 3, 6])
pseudo_x = np.zeros(len(noten_E))

plt.figure()
plt.title('Person E')
sns.swarmplot(
  x=pseudo_x,
  y=noten_E,
  color='firebrick',
  size=12
)
sns.boxplot(
  x=pseudo_x,
  y=noten_E,
  color='firebrick'
)
plt.yticks(np.arange(1, 7))
plt.ylim(6.5, 0.5)
plt.ylabel('Note')
```

```
plt.xticks([])
plt.show()
```



Beide Personen in einer Darstellung

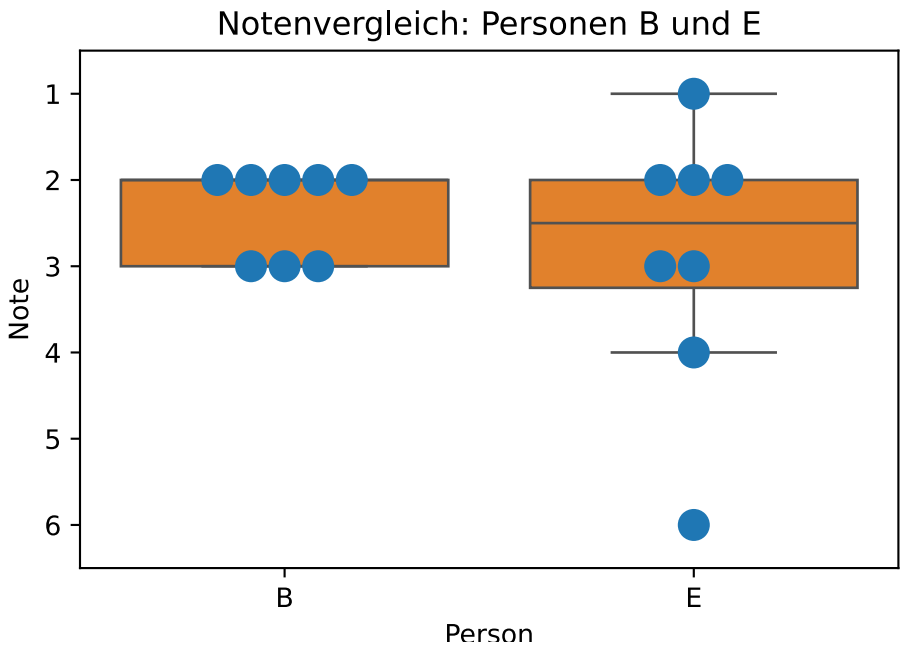
Wie gesagt sollten wir die Noten beider Personen in einer Darstellung vergleichen. Dafür müssen wir die Daten erstmal in einen einzigen DataFrame umwandeln. In der Praxis liegen Daten ja i.d.R. vor und müssen importiert werden. Hier können wir aber mal einen DataFrame direkt erstellen - entweder komplett manuell oder basierend auf den vorhandenen Arrays `noten_B` und `noten_E`.

```
# Option 1: DataFrame komplett manuell erstellen
df_noten = pd.DataFrame({
  'Person': ['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'E', 'E', 'E', 'E', 'E',
'E', 'E', 'E'],
  'Note': [2, 3, 3, 2, 2, 2, 3, 2, 2, 3, 4, 2, 1, 2, 3, 6]
})

# Option 2: DataFrame mithilfe der vorhandenen Arrays erstellen
df_noten = pd.DataFrame({
  'Person': ['B']*len(noten_B) + ['E']*len(noten_E),
  'Note': np.concatenate([noten_B, noten_E])
})

df_noten
```

```
   Person  Note
0       B     2
1       B     3
2       B     3
3       B     2
4       B     2
5       B     2
6       B     3
7       B     2
8       E     2
9       E     3
10      E     4
11      E     2
12      E     1
13      E     2
14      E     3
15      E     6
```

Um nun beide Personen in einer Darstellung zu vergleichen, müssen wir gar nicht so viel ändern. Die wichtigste Änderung ist, dass wir die Daten in `sns.swarmplot()` und `sns.boxplot()` nicht mehr direkt an `x=` und `y=` übergeben, sondern stattdessen den gesamten DataFrame an das bisher nicht genutzte Argument `data=` übergeben. Wenn das geschehen ist, können an `x=` und `y=` einfach die Spaltennamen als strings übergeben werden. Die x-Achse bekommt nun auch nicht länger Pseudo-Werte, sondern die Personenbezeichnungen. Diese lassen wir dementsprechend auch sichtbar anstatt sie wie vorher mit `plt.xticks([])` zu verstecken. Wenn wir für den Moment die Farben ignorieren, erhalten wir so schon die gewünschte Darstellung:

```
plt.figure()
plt.title('Notenvergleich: Personen B und E')
sns.swarmplot(
  x='Person',
  y='Note',
  data=df_noten,
  size=12
)
sns.boxplot(
  x='Person',
  y='Note',
  data=df_noten
)
plt.yticks(np.arange(1, 7))
plt.ylim(6.5, 0.5)
plt.ylabel('Note')
plt.show()
```

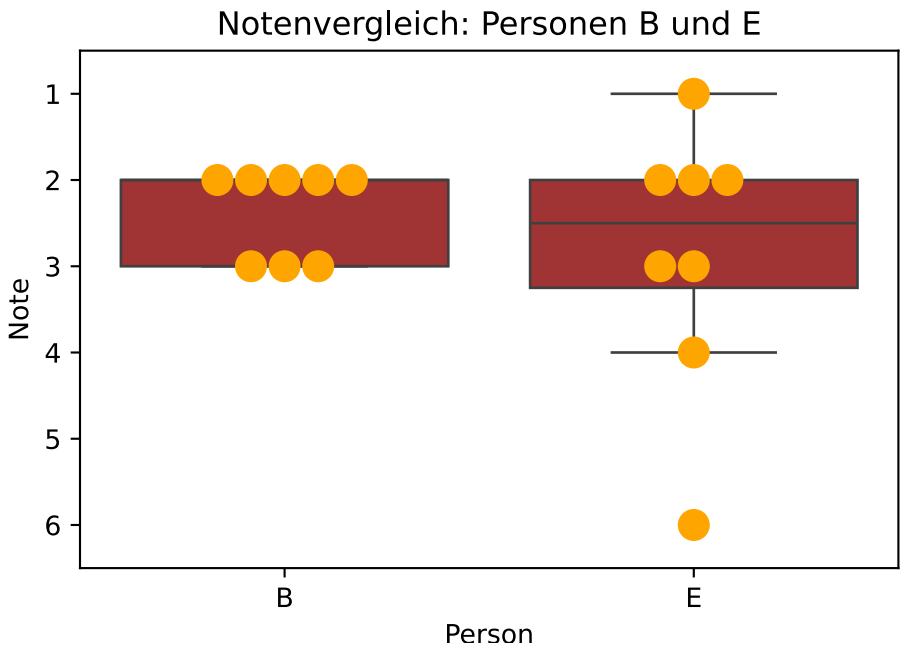


Es sei kurz angemerkt, dass wir prinzipiell die Farben je Person nicht brauchen, um die Noten zu vergleichen. Die Information, die die Farben liefern ist redundant, da die Personen ja bereits anhand der x-Achse unterschieden werden. Trotzdem kann es - wenn auch nur aus ästhetischen Gründen - sinnvoll sein, in solchen Fällen die Farben wie in den vorherigen, separaten Abbildungen beizubehalten.

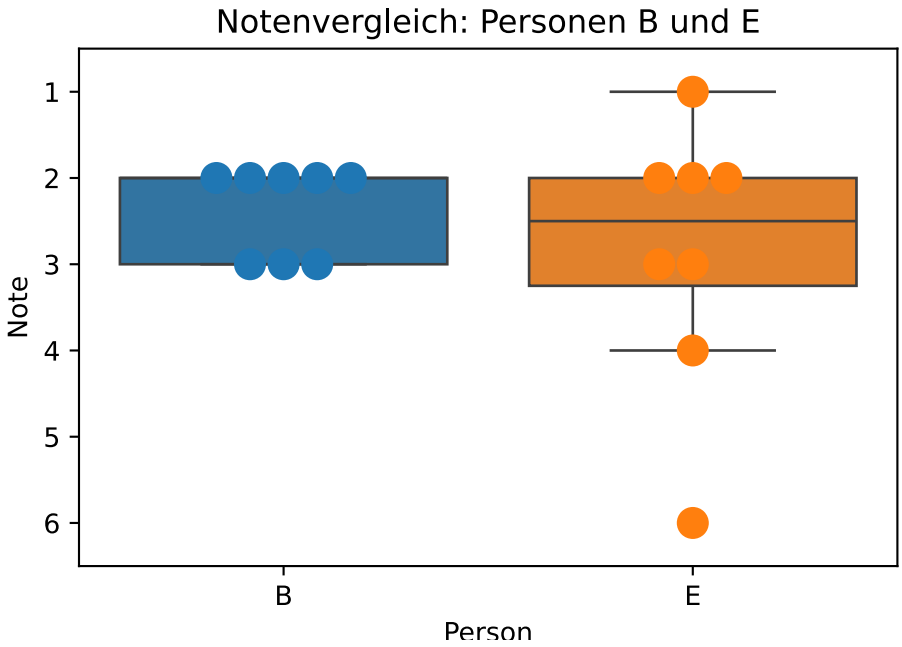
So wollen wir nun auch wieder dafür sorgen, dass sowohl Punkte als auch Boxplots von Person B orange und die von Person E rot dargestellt werden. Das können wir nicht einfach wieder über `color=` tun, da dies ja die Farbe für sämtliche Punkte und Boxplots festlegt. Stattdessen müssen wir die Farben pro Person festlegen. Das geht z.B. mit `hue=`, welches wie auch `x=` und `y=` Spaltennamen als strings erwartet.

```
plt.figure()
plt.title('Notenvergleich: Personen B und E')
sns.swarmplot(
  x='Person',
  y='Note',
  color='orange', # <-------
  data=df_noten,
  size=12
)
sns.boxplot(
  x='Person',
  y='Note',
  color='firebrick', # <-------
  data=df_noten
```

```
)
plt.yticks(np.arange(1, 7))
plt.ylim(6.5, 0.5)
plt.ylabel('Note')
plt.show()
```




```
plt.figure()
plt.title('Notenvergleich: Personen B und E')
sns.swarmplot(
  x='Person',
  y='Note',
  hue='Person', # <-------
  data=df_noten,
  size=12
)
sns.boxplot(
  x='Person',
  y='Note',
  hue='Person', # <-------
  data=df_noten
)
plt.yticks(np.arange(1, 7))
plt.ylim(6.5, 0.5)
plt.ylabel('Note')
plt.show()
```



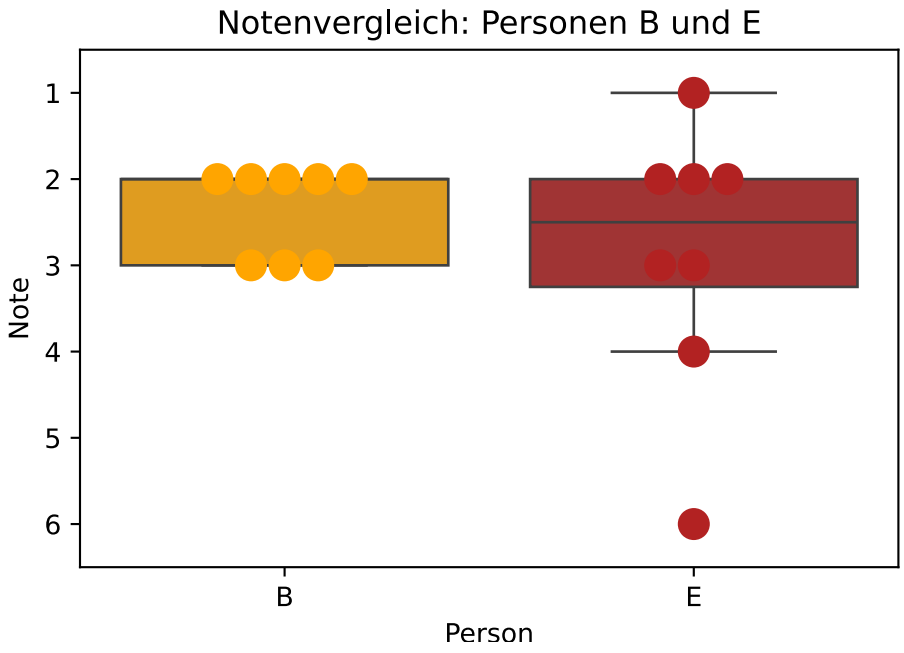
Wie man sieht, hat `hue=` also prinzipiell getan was wir wollten, allerdings wurden automatisch Standardfarben von seaborn/matplotlib verwendet. Wollen wir die Farben selbst bestimmen, müssen wir sie zusätzlich an das Argument `palette=` übergeben. Dieses Argument erwartet ein Dictionary, in dem wir den Personen die gewünschten Farben zuordnen. Es benötigt also die Farben als Werte und die Stufen als Schlüssel.

Schließlich hier noch ein Tipp: In speziell diesem Code fällt auf, dass wir viele Argumente doppelt - also an `sns.swarmplot()` und `sns.boxplot()` - übergeben. Das ist nicht nur redundant, sondern auch fehleranfällig. Spätestens wenn wir bestimmte Dinge sogar dreifach schreiben sollte uns das zu denken geben. Das DRY-Prinzip (Don't Repeat Yourself) besagt, dass wir Code möglichst nicht wiederholen sollten. Code ist leichter zu lesen, leichter zu warten und weniger fehleranfällig, wenn er sich nicht wiederholt. Hier könnten wir wie folgt die Argumente nur einmalig definieren und dann an beide Funktionen übergeben:

Mit doppelten Argumenten

```
farben_je_person = {
  'B': 'orange',
  'E': 'firebrick'
}

plt.figure()
plt.title('Notenvergleich: Personen B und E')
sns.swarmplot(
```

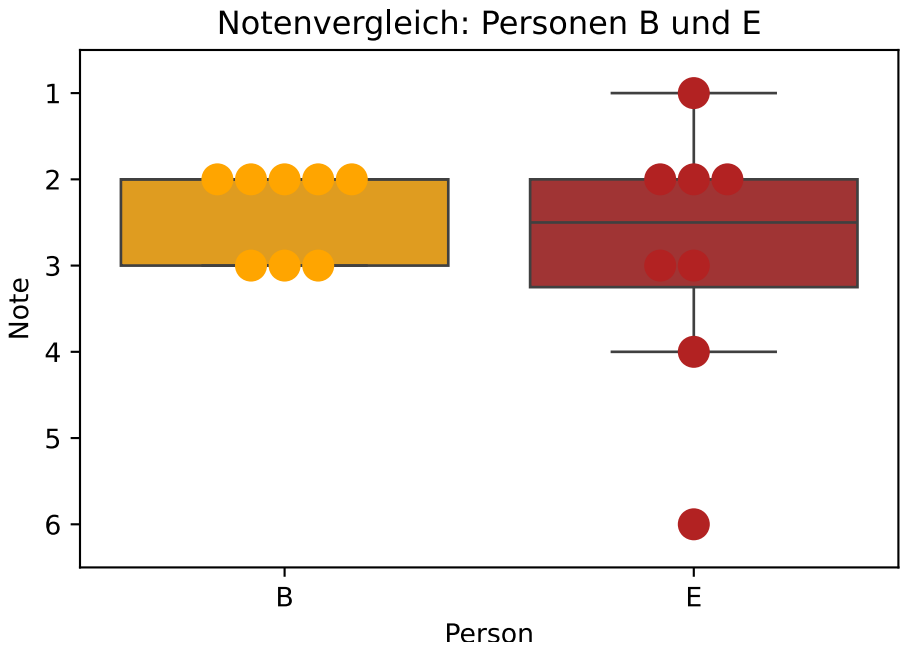
```
  x='Person',
  y='Note',
  hue='Person',
  palette=farben_je_person,
  data=df_noten,
  size=12
)
sns.boxplot(
  x='Person',
  y='Note',
  hue='Person',
  palette=farben_je_person,
  data=df_noten
)
plt.yticks(np.arange(1, 7))
plt.ylim(6.5, 0.5)
plt.ylabel('Note')
plt.show()
```



DRY-Prinzip

```
farben_je_person = {
  'B': 'orange',
  'E': 'firebrick'
}
```

```
plot_parameter = {
    'x': 'Person',
    'y': 'Note',
    'hue': 'Person',
    'palette': farben_je_person,
    'data': df_noten
}

plt.figure()
plt.title('Notenvergleich: Personen B und E')
sns.swarmplot(
  size=12,
  **plot_parameter
)
sns.boxplot(
  **plot_parameter
)
plt.yticks(np.arange(1, 7))
plt.ylim(6.5, 0.5)
plt.ylabel('Note')
plt.show()
```



9

AirBnB-Daten

Um noch mehr ausprobieren zu können, wollen wir zu den uns ebenfalls bekannten AirBnB-Daten zurückkehren. Mit diesen haben wir bereits viel gearbeitet, visualisiert haben wir sie aber noch fast gar nicht.

```
pd.set_option('display.max_columns', 4)
pd.set_option('display.max_rows', 6)
pd.set_option('display.max_colwidth', 20)

csv_url = 'https://github.com/SchmidtPaul/ExampleData/raw/main/airbnb_open_
data/Airbnb_Open_Data.csv'
df_airbnb = pd.read_csv(csv_url, dtype={25: str})
```

```
# Behalte nur bestimmte Nachbarschaften
df_airbnb = df_airbnb[df_airbnb['neighbourhood'].isin(['Harlem', 'East New
York'])]

# Behalte nur bestimmte Zimmertypen
df_airbnb = df_airbnb[df_airbnb['room type'].isin(['Entire home/apt', 'Private
room'])]

# Formatiere Spalten
df_airbnb = df_airbnb.assign(
  # Konvertiere zu category
  neighbourhood = df_airbnb['neighbourhood'].astype('category'),
  room_type     =     df_airbnb['room type'].astype('category'),
  # Für price: Erst $ und , entfernen, dann konvertieren
  price =       df_airbnb['price'].str.replace('$', '').str.replace(',',
'').astype(float)
)

# Behalte nur bestimmte Spalten
df_airbnb = df_airbnb.loc[:, ['neighbourhood', 'room_type', 'price']]

# Entferne Zeilen mit fehlenden Werten
df_airbnb.dropna(inplace=True)

# Behalte je Kombination aus Nachbarschaft und Zimmertyp nur die
# mit den 50 höchsten Preisen
df_airbnb = df_airbnb.sort_values('price', ascending=True)
df_airbnb = df_airbnb.groupby(['neighbourhood', 'room_type']).head(50)

# Setze Index zurück
df_airbnb.reset_index(drop=True, inplace=True)

df_airbnb
```

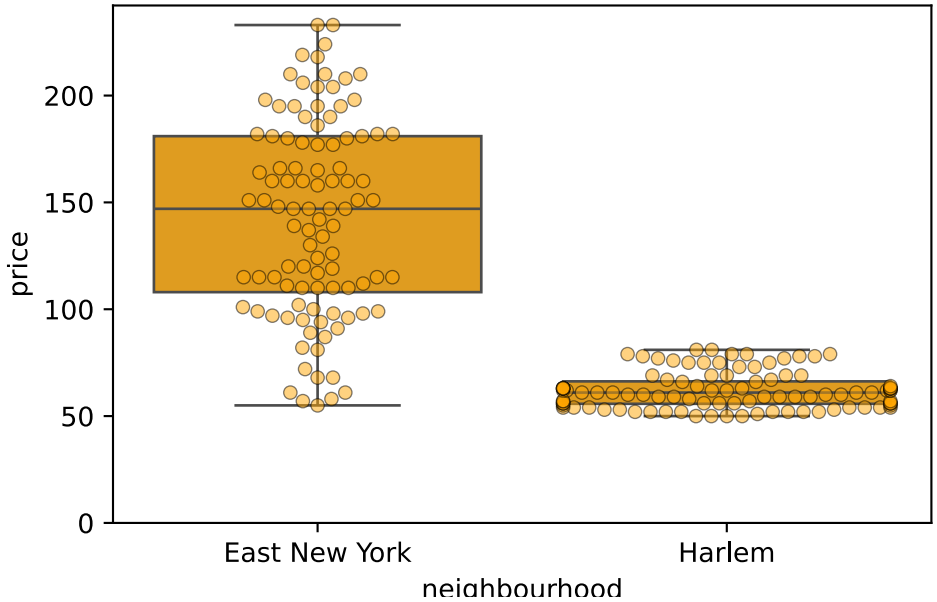
```
     neighbourhood        room_type  price
0           Harlem     Private room   50.0
1           Harlem     Private room   50.0
2           Harlem  Entire home/apt   50.0
..             ...              ...    ...
197  East New York  Entire home/apt  224.0
198  East New York  Entire home/apt  233.0
199  East New York  Entire home/apt  233.0

[200 rows x 3 columns]
```

Unser Teildatensatz enthält also die 50 günstigsten Angebote je Kombination aus Nachbarschaft und Zimmertyp für die Nachbarschaften Harlem und East New York, sowie die Zimmertypen Entire home/apt und Private room. Wir wollen primär die Preise zwischen den Nachbarschaften vergleichen und setzen deshalb die Nachbarschaften auf die x-Achse. Bis hierhin würde dies also in etwa zur selben Art von Plot führen, wie wir ihn bereits für die Noten von Personen B und E erstellt haben. Allerdings ändern wir noch einige Feinheiten beim Swarmplot, um die Daten besser darzustellen: Da es sich hier um deutlich mehr Datenpunkte handelt, müssen die Punkte kleiner sein - wir reduzieren also die `size=`. Zusätzlich setzen wir auch `alpha=0.5`, was die Punkte 50% transparent macht. So wird dafür gesorgt, dass die Punkte den dahinterliegenden Boxplot nicht zu sehr verdecken. Gleichzeitig muss aber sichergestellt werden, dass die Punkte auf dem gleichfarbigen Hintergrund des Boxplots gut sichtbar sind. Dafür setzen wir `edgecolor='black'` und `linewidth=0.5`, was die Punkte schwarz umrandet und die Linienstärke auf 0.5 setzt. Schließlich wollen wir noch die y-Achse bei 0 beginnen lassen, möchten das obere Limit der Achse aber nicht manuell festlegen und setzen deshalb `plt.ylim(0, None)`. So wird das obere Limit der y-Achse weiterhin automatisch von seaborn bestimmt.

Zusätzlich können wir in einer weiteren Darstellung noch das `hue=` Argument nutzen, um die Zimmertypen zu unterscheiden. Der Unterschied zu oben ist, dass wir diesmal nicht dieselben, sondern unterschiedliche Informationen an `x=` und `hue=` übergeben. Das führt dazu, dass es nun vier Boxplots und auch verschiedenfarbige Punkte gibt. Außerdem wird automatisch eine Legende hinzugefügt, die die Farben den Zimmertypen zuordnet. Interessanterweise ist dies oben nicht geschehen, war ja aber auch nicht nötig, da die Personen ja bereits anhand der x-Achse unterschieden wurden.

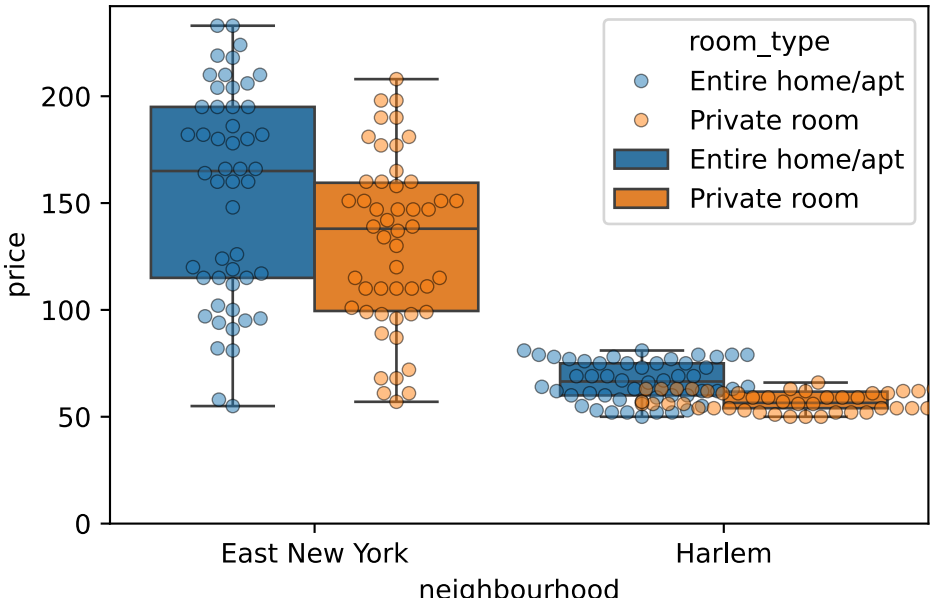
```
plot_parameter = {
    'y': 'price',
    'x': 'neighbourhood',
    'color': 'orange',  # <-----

    'data': df_airbnb
```

```
}

plt.figure()
sns.swarmplot(
  size=5,
  alpha=0.5,
  edgecolor='black',
  linewidth=0.5,
  **plot_parameter
)
sns.boxplot(
  **plot_parameter
)
plt.ylim(0, None)
plt.show()
```




```
plot_parameter = {
    'y': 'price',
    'x': 'neighbourhood',
    'hue': 'room_type', # <-----
    'dodge': True,      # <-----
    'data': df_airbnb
}

plt.figure()
sns.swarmplot(
```

```
  size=5,
  alpha=0.5,
  edgecolor='black',
  linewidth=0.5,
  **plot_parameter
)
sns.boxplot(
  **plot_parameter
)
plt.ylim(0, None)
plt.show()
```



Übungen

Nimm dir die zuletzt erzeugte Abbildung vor und probiere folgende Dinge aus und beobachte die Unterschiede:

- Setze `dodge` auf `False`
- Erhöhe die Größe der Punkte auf 12.
- Setze `alpha` auf 1 oder auf 0
- Setze `linewidth` auf 3
- (A) Geschafft

Suche dir eine dritte Nachbarschaft, sowie einen dritten Zimmertypen aus und erstelle den AirBnB DataFrame erneut aber eben so, dass diese ebenfalls enthalten sind. Betrachte die Abbildungen erneut.

- (A) Geschafft