Gruppieren & Aggregieren

by Woche 10

Wie in den vorigen Kapiteln setzen wir zunächst wieder Pandas Optionen und importieren unseren AirBnB Datensatz.

```
import numpy as np
import pandas as pd

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 = pd.read_csv(csv_url, dtype={25: str})
```

Darüber hinaus und basierend auf dem Wissen aus den letzten Kapiteln formatieren wir aber den importierten Datensatz auch ein wenig und erzeugen uns dann auch hier wieder einen übersichtlichen Teildatensatz, um die Gruppierung und Aggregation zu üben.

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

# Lösche Spalte
df.drop(columns=['room type', 'service fee'], inplace=True)

# Wähle Zeilen und Spalten für Teildatensatz
df2 = df.loc[
  [1, 101, 2, 24262, 233, 493, 45, 619],
  ['neighbourhood', 'room_type', 'price', 'fee']
]
```

```
# Setze Index zurück
df2.reset_index(drop=True, inplace=True)

# dies wird erst weiter unten erklärt
df2['neighbourhood'] = df2['neighbourhood'].cat.remove_unused_categories()
df2['room_type']     =     df2['room_type'].cat.remove_unused_categories()

df2
```

	neighbourhood	room_type	price	fee
0	Midtown	Entire home/apt	142.0	28.0
1	Harlem	Private room	913.0	183.0
2	Harlem	Private room	620.0	124.0
3	Midtown	Entire home/apt	NaN	105.0
4	Midtown	Entire home/apt	588.0	118.0
5	Harlem	Shared room	67.0	13.0
6	Harlem	Entire home/apt	62.0	12.0
7	Midtown	Private room	578.0	116.0

In diesem Kapitel lernen wir, wie wir Daten gruppieren und aggregieren können. In gewisser Hinsicht erzeugen wir so erstmals in diesem Kurs Ergebnisse, die zumindest in bestimmten Projekten als vollwertiges Ergebnise/ vollwertige Analyse betrachtet werden.

Die `.groupby()` Methode

Wozu?

Wir wissen bereits, dass wir beispielsweise den Mittelwert einer Spalte berechnen können, indem wir die Methode `.mean()` auf eine Spalte anwenden.

```
df2['price'].mean()
```

```
np.float64(424.2857142857143)
```

Ebenfalls wissen wir, wie wir die Daten filtern können, um nur bestimmte Zeilen zu betrachten. Demnach könnten wir in unserem Beispieldatensatz `df2` manuell den Durchschnittspreis pro Nachbarschaft berechnen, indem wir die Preise der einzelnen Nachbarschaften filtern und dann den Durchschnitt berechnen.

```
filter1 = df2['neighbourhood'] == 'Midtown'
df2.loc[filter1, 'price'].mean()
```

```
np.float64(436.0)
```

```
filter2 = df2['neighbourhood'] == 'Harlem'
df2.loc[filter2, 'price'].mean()
```

```
np.float64(415.5)
```

Genau hier kommt die `.groupby()` Methode ins Spiel. Sie ermöglicht es uns, die Daten nach einer bestimmten Spalte zu gruppieren und dann eine Aggregationsfunktion auf sämtliche Gruppen anzuwenden. Wir bräuchten also lediglich diesen deutlich kürzeren Code zu schreiben - egal wie viele Nachbarschaften wir haben.

```
# Zwei Nachbarschaften in df2
df2.groupby('neighbourhood')['price'].mean()
```

```
neighbourhood
Harlem     415.5
Midtown    436.0
Name: price, dtype: float64
```

```
# 224 Nachbarschaften in df
df.groupby('neighbourhood')['price'].mean()
```

```
neighbourhood
Allerton           636.343750
Arden Heights      804.888889
Arrochar           625.764706
Arverne            652.125561
Astoria            639.035275
                         ...
Windsor Terrace    579.784848
Woodhaven          630.518325
Woodlawn           587.137931
Woodrow            709.333333
Woodside           634.588336
Name: price, Length: 224, dtype: float64
```

Der Vorteil der `.groupby()` Methode ist also offensichtlich und die Möglichkeiten sind vielfältig. Bevor wir aber andere Funktionen etc. ausprobieren, wollen wir nochmal kurz einen Schritt zurück gehen und uns klar machen, was die `.groupby()` Methode genau macht.

Wie?

Die `.groupby()` Methode erzeugt ein `GroupBy` Objekt, das wir uns als eine Art “Plan” vorstellen können, wie die Daten gruppiert werden sollen. Dieses Objekt ist also nicht mehr nur ein DataFrame, sondern eine spezielle Art von Objekt, das wir dann weiter bearbeiten können.

```
df2_pro_nb = df2.groupby('neighbourhood')
df2_pro_nb
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001E4CF5D82D0>
```

Das Objekt selbst “weiß” also dauerhaft, dass die Daten nach der Spalte `neighbourhood` gruppiert wurden. Egal welche Methode wir auf das Objekt anwenden, es wird immer die Gruppierung berücksichtigen.

```
# Anzahl Zeilen
df2_pro_nb.size()
```

```
neighbourhood
Harlem     4
Midtown    4
dtype: int64
```

```
# Höchster Preis
df2_pro_nb['price'].max()
```

```
neighbourhood
Harlem     913.0
Midtown    588.0
Name: price, dtype: float64
```

```
# Anzahl einzigartiger Werte
df2_pro_nb['room_type'].nunique()
```

```
neighbourhood
Harlem     3
Midtown    2
Name: room_type, dtype: int64
```

Aus einem gruppierten Objekt können wir auch die Daten von einzelnen Gruppen auswählen, indem wir die `.get_group()` Methode verwenden. Diese Methode erwartet den Namen der Gruppe als Argument und gibt dann die entsprechenden Zeilen zurück.

```
df2_pro_nb.get_group('Midtown')
```

```
  neighbourhood        room_type  price    fee
0       Midtown  Entire home/apt  142.0   28.0
3       Midtown  Entire home/apt    NaN  105.0
4       Midtown  Entire home/apt  588.0  118.0
7       Midtown     Private room  578.0  116.0
```

> **Hinweis zu `.cat.remove_unused_categories()` von oben**
>
> Zu Beginn des Kapitels wurde beim Erzeugen von `df2` die Methode `.cat.remove_unused_categories()` verwendet. Diese Methode entfernt Kategorien/Gruppierungsstufen, die in einem DataFrame nicht mehr vorkommen. Dies passiert nämlich nicht automatisch, nur weil wir einen DataFrame filtern. Um das zu verdeutlichen einfach mal den Code von Anfang des Kapitels bis inklusive `df2.groupby('neighbourhood')['price'].mean()` erneut ausführen, diesmal aber ohne die Zeilen mit `.cat.remove_unused_categories()`. Das Ergebnis sieht dann so aus:
>
> ```
> neighbourhood
> Allerton          NaN
> Arden Heights     NaN
> Arrochar          NaN
> Arverne           NaN
> Astoria           NaN
>                    ..
> Windsor Terrace   NaN
> Woodhaven         NaN
> Woodlawn          NaN
> Woodrow           NaN
> Woodside          NaN
> Name: price, Length: 224, dtype: float64
> ```
>
> Man kann es zwar nicht sehen, aber nur für die Nachbarschaften Harlem und Midtown gäbe es den entsprechenden Mittelwert, während für alle anderen 222 Nachbarschaften keine Daten vorhanden waren, sodass der Mittelwert `NaN` ist. Der Knackpunkt ist also, dass die Gruppierungsstufen in `df2` noch alle 224 Nachbarschaften enthalten, obwohl nur zwei davon Daten enthalten. Dies ist eine Besonderheit von Spalten mit dem Datentyp `category`, welche vor allem bei der Gruppierung und Aggregation auffällt.
>
> Anstatt die ungenutzen Kategorien eines DataFrames via `.cat.remove_unused_categories()` zu entfernen hat man übrigens auch die alternative Möglichkeit in der `.groupby()` Methode die Option `observed=True` zu setzen. Dies führt auch dazu, dass nur die Kategorien berücksichtigt werden, die tatsächlich in den Daten vorkommen.

Mehrere Werte-Spalten

Diese Berechnungen pro Gruppe können wir auch direkt für mehrere Spalten durchführen. Anstatt also pro Nachbarschaft nur den Durschnittspreis zu berechnen,

können wir auch direkt die durchschnittliche Servicegebühr mitberechnen. Dazu übergeben wir eben eine Liste von Spaltennamen an das gruppierende Objekt.

```
# Mit vorgruppiertem Objekt
df2_pro_nb[['price', 'fee']].mean()




#
```

```
                price    fee
neighbourhood
Harlem          415.5  83.00
Midtown         436.0  91.75
```

```
# Mit Method Chaining
(
  df2
  .groupby('neighbourhood')
  [['price', 'fee']]
  .mean()
)
```

```
                price    fee
neighbourhood
Harlem          415.5  83.00
Midtown         436.0  91.75
```

Mehrere Aggregationsfunktionen

Darüber hinaus ist es ebenfalls möglich, direkt mehrere Aggregationsfunktionen auf die Daten anzuwenden. Dazu übergeben wir eine Liste von Funktionen (also `mean()`, `min()` usw.) an die `.agg()` Methode. Es folgen zwei Möglichkeiten um pro Nachbarschaft für die Spalte `price` den Mittelwert, das Minimum und das Maximum zu berechnen. Die erste von beiden ist dem bisherigen vorgehen sehr ähnlich: Wir gruppieren, dann wählen wir die Spalte aus und wenden schließlich einfach die `.agg()` Methode statt wie bisher `.mean()` usw. an. Innerhalb der `.agg()` Methode übergeben wir dann eine Liste von Funktionsnamen als Strings. Die zweite Möglichkeit zeigt, dass man aber auch das auswählen der Spalte innerhalb der `.agg()` Methode machen kann. Man nutzt dann ein Dictionary, in dem die Spaltennamen als Keys und die Funktionsnamen als Values stehen.

```
(
  df2
  .groupby('neighbourhood')
  ['price']
  .agg(['mean', 'min', 'max'])
)
```

```
                mean    min    max
neighbourhood
Harlem         415.5   62.0  913.0
Midtown        436.0  142.0  588.0
```

```
(
  df2
  .groupby('neighbourhood')
  .agg({'price':['mean', 'min', 'max']})
)
```

```
               price
                mean    min    max
neighbourhood
Harlem         415.5   62.0  913.0
Midtown        436.0  142.0  588.0
```

Ein Vorteil der zweiten Methode ist, dass man individuelle Funktionsnamen für jede Spalte angeben kann.

```
(
  df2
  .groupby('neighbourhood')
  [['price', 'fee']]
  .agg(['min', 'max'])
)
```

```
               price          fee
                 min    max   min    max
neighbourhood
Harlem          62.0  913.0  12.0  183.0
Midtown        142.0  588.0  28.0  118.0
```

```
#
(
  df2
```

```
  .groupby('neighbourhood')
  .agg({'price':['mean', 'min', 'max'], 'fee':'mean'})
)
```

```
               price                  fee
                mean    min    max   mean
neighbourhood
Harlem         415.5   62.0  913.0  83.00
Midtown        436.0  142.0  588.0  91.75
```

Die `.describe()` Methode

In der Praxis ist es tatsächlich die Regel sich direkt mehrere Lage-/Streuungsmaße für die Daten ausgeben zu lassen. Anstatt sich eine Liste mit den gewünschten Funktionen zu überlegen, kann aber auch einfach die `.describe()` Methode verwendet werden. Diese Methode gibt standardmäßig die Anzahl der Werte, den Mittelwert, die Standardabweichung, das Minimum, das 25%-Quantil, das Median, das 75%-Quantil und das Maximum aus.

```
pd.set_option('display.max_columns', 8) # Damit alle Spalten angezeigt werden

df2_pro_nb['price'].describe() # alternativ:
df2_pro_nb.agg({'price':'describe'})

pd.set_option('display.max_columns', 4) # Zurücksetzen
```

```
               count   mean         std    min     25%    50%     75%    max
neighbourhood
Harlem           4.0  415.5  422.587663   62.0   65.75  343.5  693.25  913.0
Midtown          3.0  436.0  254.660558  142.0  360.00  578.0  583.00  588.0
```

Mehrere Grupperierungs-Spalten

Es können ebenso mehrere Spalten an die `.groupby()` Methode übergeben werden. In diesem Fall wird die Gruppierung nach den Kombinationen der Werte in den Spalten durchgeführt. Wir könnten also z.B. prüfen was die jeweils günstigste Unterkunft in einer Nachbarschaft und einem Zimmertyp ist.

```
(
  df2
  .groupby(['neighbourhood', 'room_type'])
  ['price']
```

```
  .min()
)
```

```
neighbourhood  room_type
Harlem         Entire home/apt     62.0
               Private room       620.0
               Shared room         67.0
Midtown        Entire home/apt    142.0
               Private room       578.0
               Shared room          NaN
Name: price, dtype: float64
```

```
(
  df2
  .groupby(['room_type', 'neighbourhood'])
  ['price']
  .min()
)
```

```
room_type        neighbourhood
Entire home/apt  Harlem            62.0
                 Midtown          142.0
Private room     Harlem           620.0
                 Midtown          578.0
Shared room      Harlem            67.0
                 Midtown            NaN
Name: price, dtype: float64
```

Weitere Details & Tipps

Es kann sinnvoll sein innerhalb von `groupby()` zusätzlich die Option `as_index=False` zu setzen. Dies führt dazu, dass die Gruppierungs-Spalten nicht als Multi-Index im Ergebnis auftauchen, sondern als normale Spalten.

```
(
  df2
  .groupby(['neighbourhood', 'room_type'], as_index=False)
  ['price']
  .min()
)
```

```
  neighbourhood        room_type  price
0        Harlem  Entire home/apt   62.0
```

```
1        Harlem     Private room  620.0
2        Harlem      Shared room   67.0
3       Midtown  Entire home/apt  142.0
4       Midtown     Private room  578.0
5       Midtown      Shared room    NaN
```

> **i** Hinweis zu Multi-Index
>
> Ein Multi-Index ist ein spezieller Index, bei dem die Index-Spalten mehrere Ebenen haben. In diesem Fall - also bevor wir `as_index=False` genutzt haben - hatten wir also zwei Ebenen, die Nachbarschaft und den Zimmertyp. In folgenden Kapiteln werden wir noch detailierter auf Multi-Indizes eingehen.

Direkt beim Erzeugen dieser Tabellen mit verschiedenen deskriptiven Statistiken können wir auch direkt die Spalten benennen anstatt die Standard/Funktionsnamen zu verwenden:

```
(
  df2
  .groupby('neighbourhood', as_index=False)
  .agg(
    Durchschnittspreis = ('price', 'mean'),
    Günstigster = ('price', 'min'),
    Teuerster = ('price', 'max')
  )
)
```

```
  neighbourhood  Durchschnittspreis  Günstigster  Teuerster
0        Harlem               415.5         62.0      913.0
1       Midtown               436.0        142.0      588.0
```

Es ist auch möglich andere Funktionen als die Standardfunktionen zu verwenden. Beispielsweise können wir den Median auch mit der Funktion `np.median` aus dem NumPy Modul berechnen.

```
# Option 1
(
  df2
  .groupby('neighbourhood')
  ['price']
  .agg(['mean', np.median])
)
```

```
                mean  median
neighbourhood
Harlem         415.5   343.5
Midtown        436.0   578.0
```

```
# Option 2

(
  df2
  .groupby('neighbourhood')
  .agg({'price':['mean', np.median]})
)
```

```
               price
                mean median
neighbourhood
Harlem         415.5  343.5
Midtown        436.0  578.0
```

Oft haben die Ergebnisse unnötig viele Nachkommastellen. Dies lässt sich mit der `.round()` Methode beheben, welche einfach im Method Chaining nach der Aggregation aufgerufen wird.

```
(
  df2
  .groupby('room_type')
  ['price']
  .agg(['mean', 'var'])

)
```

```
                       mean          var
room_type
Entire home/apt  264.000000  80332.000000
Private room     703.666667  33306.333333
Shared room       67.000000           NaN
```

```
(
  df2
  .groupby('room_type')
  ['price']
  .agg(['mean', 'var'])
```

```
  .round(1)
)
```

```
                  mean      var
room_type
Entire home/apt  264.0  80332.0
Private room     703.7  33306.3
Shared room       67.0      NaN
```

Weitere Ressourcen

- Pandas Crashkurs - Daten gruppieren mit GroupBy - Video 6/8 kompletter Kurs kostenlos deutsch/german
- 7 + 1 Pandas GROUP BY Tips – Great Tips & Tricks (Python Tutorial)

Übungen

Nutze den vollen Datensatz `df` um jeweils den Mittelwert, die Varianz und die Anzahl Beobachungen des Preises pro Zimmertyp zu berechnen (ggf. gerundet auf eine Nachkommastelle). Ergänze dann folgende Ergebnisse:

- _____ $ ist der Durchschnittspreis für ein Hotelzimmer. (Achtung, schreibe Punkt statt Komma als Dezimaltrennzeichen!)
- Der Zimmertyp ‘Entire home/apt’ hat mit _____ Beobachtungen die meisten Datenpunkte aller Zimmertypen im Datensatz.
- Die Preise des Zimmertyps ‘ _____ room’ schwanken/variieren am wenigsten.

Basierend auf `df2`: Berechne pro Kombination aus Nachbarschaft und Zimmertyp (immer gerundet auf keine Nachkommastelle) den Mittelwert für den Preis, sowie das Minimum und Maximum der Servicegebühr.

- (A) Geschafft