Semesterarbeit

Grundlagen Data Science

GrDS.MAS/CAS Web4B 2019.ZH1.HS19/20

Brigitte Burkhalter
Bruggmoosstrasse 32
5454 Bellikon
brigitte.burkhalter@bluewin.ch

Inhaltsverzeichnis

1 Einleitung

Die vorliegende Semesterarbeit hat zum Ziel, einen konkreten Datensatz mit einer Programmiersprache und dazugehörigen Bibliotheken zu analysieren und Visualisierungen zu erstellen. Im Vordergrund stehen die Auswahl und Anwendung von Bibliotheken und Funktionen.

Für die Datenanalyse wurde ein Datensatz mit biografischen Daten zu Kunstschaffenden aus dem SIKART Lexikon zur Kunst in der Schweiz ausgesucht. Als Programmiersprache wurde Python gewählt. Der Datensatz wurde bewusst ausgewählt, da er viele Textdaten enthält und diese für die Visualisierung zunächst aufbereitet werden müssen.

Der Code wurde in JupyterLab geschrieben, dem Nachfolger von Jupyter Notebook. Das Dateiformat ist jedoch dasselbe. Der Code wurde ausführlich kommentiert, einerseits zu Lernzwecken, andererseits, um die Analyse besser nachvollziehen zu können.

Ziel der Arbeit ist es, einen Überblick über die im Datensatz enthaltenen Informationen und über verschiedene Beziehungen zwischen den Daten zu erhalten. Die Analyse der Daten ist nicht abschliessend. Für einzelne Fragestellungen wurden verschiedene Funktionen und Plots erprobt, auch um Unterschiede zwischen den Visualisierungs-Bibliotheken sichtbar zu machen.

2 opendata.swiss, SIK-ISEA und SIKART

Der ausgewählte Datensatz «Biografische Daten zu Kunstschaffenden aus SIKART Lexikon zur Kunst in der Schweiz» vom Schweizerisches Institut für Kunstwissenschaft (SIK-ISEA) auf dem Portal opendata.swiss zur freien Nutzung zur Verfügung gestellt.

Das Portal opendata.swiss ist gemäss den Betreibern «das zentrale Portal für offene, d.h. frei verfügbare Daten der Schweizer Behörden» (BFS). Das Portal wurde vom Bundesarchiv entwickelt und obliegt nun der Leitung des Bundesamtes für Statistik. Das Portal umfasst unterschiedlichste Daten, unter vielen anderen auch zur Kultur.

Das Schweizerisches Institut für Kunstwissenschaft (SIK-ISEA) ist ein Kompetenzzentrum für Kunstwissenschaft und Kunsttechnologie mit Schwerpunkt auf historische und zeitgenössische Kunst in der Schweiz. SIK-ISEA betreibt das SIKART

Lexikon zur Kunst in der Schweiz (http://www.sikart.ch), welches als Online-Informationssystem allen Interessierten zur Verfügung steht (SIK-ISEA).

Der Datensatz aus SIKART umfasst rund 17'000 Einträge mit den biografischen Grunddaten zu aktuellen und historischen Kunstschaffenden, wobei Fotografinnen und Fotografen gesondert ausgewiesen werden. Zeitlich umspannen die Daten die Jahre 900 - Januar 2019. Bedauerlicherweise sind die Tätigkeitsbereiche, wie Malerei, Grafik, Schmuck, Installation, die eine differenzierte Auswertung erlaubt hätten, nicht im Datensatz enthalten («Biografische Daten»).

Die Eckwerte des Datensatzes werden weiter unten beschrieben. Zunächst soll kurz auf die verwendeten Bibliotheken eingegangen werden.

3 Verwendete Bibliotheken und Module

Der SIKART-Datensatz wurde mit der Programmiersprache Python bearbeitet. Python und die zu Verfügung stehenden Module und Bibliotheken sind gut geeignet für die Verarbeitung von Textdateien und deren Visualisierung. Die verwendeten Bibliotheken und Module werden im Folgenden kurz dargestellt. Die für die Auswertung benutzten Funktionen werden direkt bei der jeweiligen Datenanalyse besprochen.

3.1 Pandas

Die Bibliothek Pandas stellt für die Verarbeitung von wenig strukturierten Daten eine Fülle von Werkzeugen zur Verfügung («Pandas». Pandas bietet mit den *Series*- und *DataFrame*-Objekten die Möglichkeit, tabellarische Daten mit unterschiedlichen Datentypen oder fehlenden Daten auszuwerten. Diese sind im vorliegenden SIKART-Datensatz in grosser Menge vorhanden.

Ebenso bietet Pandas zahlreiche Methoden, um Daten auszuwählen, zu gruppieren, filtern oder kombinieren (VanderPlas 117-242). Diese Methoden sind bei den vorhandenen Textdaten besonders gefragt, da eine «direkte» Übergabe an einen Plot meist nicht möglich ist.

Weiter bietet Pandas Funktionen zum Umgang mit Zeit und Datumsangaben (Klein, "Numerisches Python"337-346). Obwohl im Datensatz exakte Geburts- und Sterbedaten ausgewiesen sind, wurden diese hier aus zeitlichen Gründen nicht berücksichtigt.

Pandas bietet ebenfalls eine Plot-Funktion *df.plot,* welche Matlotlib beziehungsweise pyplot benutzt (Soma). Visualisierungen mit Pandas lassen sich einfach und schnell realisieren und geben einen guten ersten Eindruck der Auswertung. Im Gegensatz dazu bietet Matplotlib mehr Möglichkeiten, Diagramme zu gestalten. Das Resultat ist in beiden Fällen ein Matplotlib-Objekt («Differences»).

3.2 Numpy

Die Bibliothek Numpy ist besonders geeignet für gut strukturierte, numerische Daten. Hier wird die Bibliothek im Wesentlichen benötigt, weil Pandas auf Numpy aufbaut. Ausserdem wurden einzelne Auswertungen mit Numpy durchgeführt.

3.3 Reguläre Ausdrücke (Modul re)

Reguläre Ausdrücke oder englisch regular expressions werden benutzt, um Texte oder Textstrings zu untersuchen. Sie erlauben es Texte oder Strings zu suchen, ersetzen, filtern, vergleichen oder zu teilen (splitten). Das Modul re ist eine Standardbibliothek von Python. Da es jedoch nicht zum Grundpaket gehört, muss es importiert werden (Klein, "Einführung" 387-413).

In Pandas gibt es einige Methoden, die reguläre Ausdrücken benutzen, wie zum Beispiel *match, findall, replace* oder *contain*. Reguläre Ausdrücke wurden hier verwendet, um Zeichen zu ersetzen.

3.4 Matplotlib

Die Bibliothek Matplotlib für Visualisierungen gilt zwar als etwas veraltet, doch sie eignet sich sehr gut als Grundlage, um mit weiteren Visualisierungs-Tools zu arbeiten. Ausserdem bietet sie eine Vielzahl an Anpassungsmöglichkeiten (VanderPlas 245-358).

Schwierigkeiten bereitete die Tatsache, dass Matplotlib zwei Schnittstellen anbietet, zum einen die zustandsorientierte pyplot-Schnittstelle, zum anderen die objektorientierte Schnittstelle. Bei der ersten wird mit plt.plot das Figure- und Axes-Objekt automatisch erzeugt. Bei der objektorientierten Schnittstelle werden Figure- und Axes-Objekt separat erstellt. Eine Figure hat eine oder mehrere Axes, welche jeweils einen Plot darstellen. Die Plot-Funktionen sind Methoden, die auf die Figure- und Axes-Objekte angewendet werden. Dadurch bietet diese Schnittstelle mehr Flexibilät bei umfassenderen Diagrammen («Matplotlib», Moffitt).

Für die Darstellung von Matplotlib-Diagrammen kann mit Stylesheets gearbeitet werden. Für die vorliegende Arbeit wurde ein Seaborn-Stil gewählt («Matplotlib Style»). Dieser wurde aufgerufen mit *sns.set* und *plt.style.use('seaborn-muted').* Muted steht für einen unbunten Stil. Mit der Funktion *%matplotlib inline* werden die Diagramme direkt im Notebook als statische Abbildung angezeigt.

3.5 Seaborn

Seaborn ist eine Bibliothek zur Visualisierung von Daten, die auf Matplotlib basiert («Seaborn»). Zwar bietet Seaborn eine Vielzahl an komplexeren und ansprechenderen Visualisierungen als Matplotlib oder Pandas, für eine vertiefte Einarbeitung fehlte jedoch die Zeit. Relativ neu in Seaborn ist die Möglichkeit mit *catplot* als Basis verschiedene Visualisierungen vorzunehmen (Cmdline). Die Art des Plots wird hier mit dem Parameter *kind* definiert.

4 Umfang und Inhalt des Datensatzes

Bevor eine eigentliche Auswertung des Datensatzes mit Visualisierungen stattfinden kann, muss die Datei gelesen und ein erster Überblick über die vorhandenen Dateb geschaffen werden.

4.1 Dateien mit Pandas lesen

Pandas bietet die Möglichkeit diverse Datenformate einfach zu lesen und zu schreiben. Der SIKART-Datensatz steht als CSV-Datei zur Verfügung und kann mit der Methode *read_csv* gelesen werden. Als Trennzeichen muss der Parameter «;» angegeben werden.

4.2 Erste Auswertung der vorhandenen Daten

Zunächst wurden mit den gängigen DataFrame-Attributen *dtypes, columns, head* und *tail* Inhalt und Art der Daten bestimmt.

Mithilfe des Attributes *dtypes* können die Datentypen der einzelnen Spalten ermittelt werden. Sie bestehen hier überwiegend aus dem Typ *object*, welcher für Strings oder gemischte numerische und nicht-numerische Werte benutzt wird.

Numerische Datentypen treten bei der Spalte Hauptnummer als *integer* sowie für Geburts- und Sterbejahre als *float* auf. Je nach Auswertung müssen die Daten in einen anderen Datentyp umgewandelt werden. Gerade bei den Geburts- und Sterbejahren würde der Datentyp *integer* für eine übliche Schreibweise der Jahre erwartet.

Das Attribut *columns* gibt die Spaltenbezeichnungen aus. Die Spaltenbezeichnungen sind wesentlich für die weitere Auswertung der Daten mittels Auswahl, Gruppierung oder Aggregation.

Der SIKART-Datensatz enthält die folgenden Spaltennamen:

> 'HAUPTNR', 'NUTZUNGSLIZENZ', 'BEREICH', 'TYPUS', 'NAME_VORNAME', 'NAME', 'VORNAME', 'NAMENSZUSATZ', 'GEBURTSJAHR', 'GEBURTSDATUM', 'GEBURTSORT', 'GEBURTSLAND', 'STERBEJAHR', 'STERBEORT', 'STERBELAND', 'STERBEDATUM', 'LEBENSDATEN', 'GND', 'HLS_ID', 'SIKART_LINK', 'BEARBEITUNGSTIEFE'.

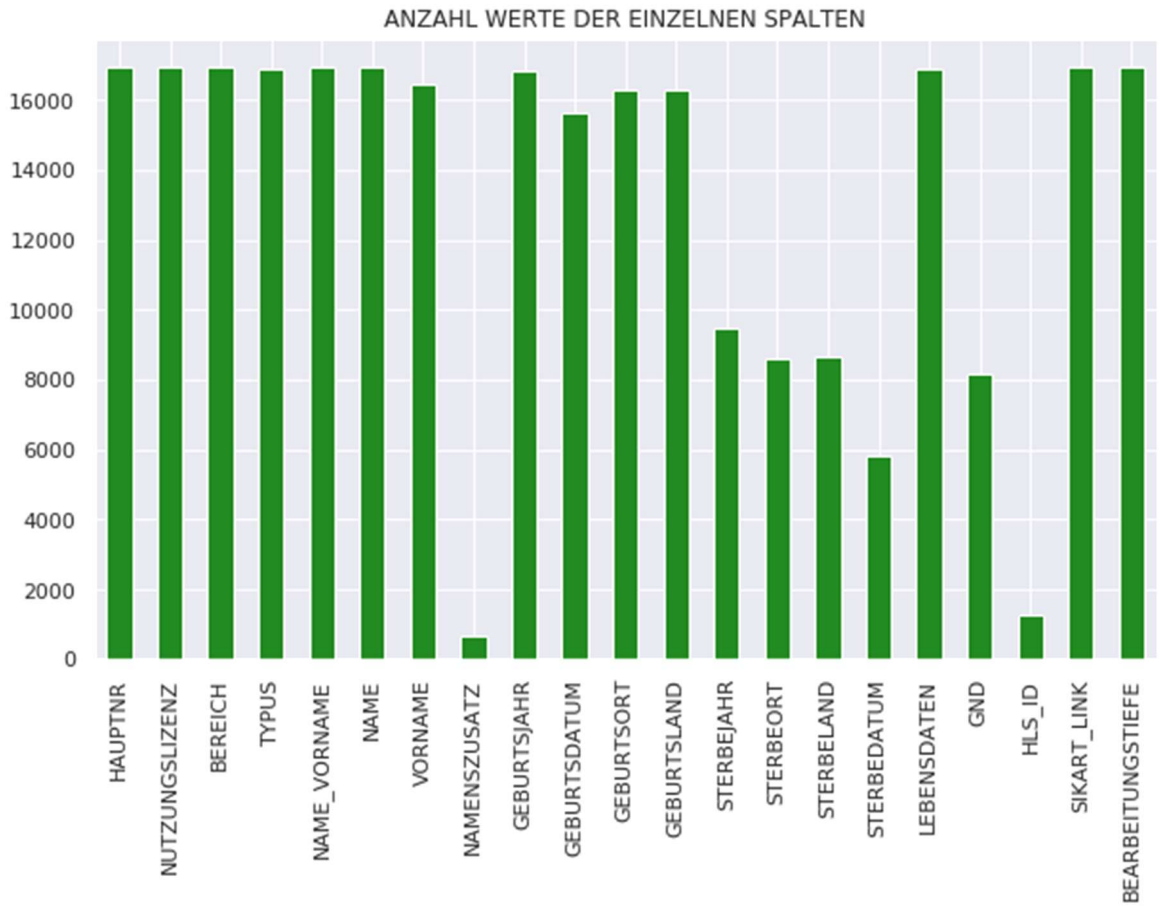
Die Grossschreibung wurde so belassen.

Die Attribute *df.head() und df.tail()* geben jeweils alle Spalten mit der gewünschten Anzahl Zeilen und den Werten in tabellarischer Form zurück. Standardmässig werden 5 Zeilen ausgegeben. Dadurch kann ein guter Eindruck vom Inhalt und der Form der Daten gewonnen werden.

4.3 Vorhandene Werte

Neben den Spaltennamen gibt die Anzahl Werte pro Spalte wichtige Hinweise darüber, welche Daten analysiert werden können und wo allenfalls Werte fehlen. Daher wurden mit Attribut *count* die Anzahl Werte pro Spalte (*axis=0*) gezählt. Damit nicht nur die numerischen Werte gezählt wurden, musste der Parameter *numeric_only* auf *False* gesetzt werden.

Für die Anzeige der Resultate wurde als erste Visualisierung mit Pandas ein Säulendiagramm erstellt. Dieses zeigt grosse Unterschiede, was die Vollständigkeit betrifft.



4.4 Fehlende Werte

Wie die erste Analyse der Daten gezeigt hat, fehlen zahlreiche Werte. Es sind im Gegenteil die wenigsten Spalten komplett. Pandas bietet verschiedene Möglichkeiten, mit fehlenden Werten umzugehen. So können fehlenden Werte gefiltert oder aufgefüllt werden (VanderPlas 141-144).

Die fehlenden Werte im SIKART-Datensatz werden mit *«*nan*»,* also «not a number» ausgegeben. Da es sich um biographische Daten und nicht um Messwerte irgend einer Art handelt, ist es nicht angebracht diese Werte mittels der Methode *fillna* aufzufüllen. Fehlende Werte wurden daher bei Bedarf mit der Methode *dropna* gefiltert.

Bei den fehlenden Werten zeigt sich neben dem Umgang mit Textdaten eine weitere Herausforderung was die Analyse und Visualisierung betrifft, da Parameter wie Spaltennamen nicht ohne weiteres übergeben werden können.

4.5 Daten, die nicht ausgewertet werden

Der SIKART-Datensatz bietet mit den Verweisen auf das Historische Lexikon der Schweiz (HLS) und die Gemeinsame Normdatei (GND) sowie dem SIKART-LINK interessante weiterführende Informationen. Die Frage, ob für die ausgewiesenen Kunstschaffenden diese Informationen vorhanden sind, bietet im vorliegenden Kontext jedoch keinen nennenswerten Mehrwert. Ebenso sind die Werte der Spalte NUTZUNGSLIZENZ für alle Instanzen dieselbe (CC-BY-NC-SA).

Die vier Spalten NUTZUNGSLIZENZ, HLS_ID, GND und SIKART_LINK wurden daher mit der Methode *DataFrame.drop* gelöscht, nicht zuletzt, um die Übersichtlichkeit bei der Ausgabe zu verbessern. Die Spalte HAUPTNR wurde hingegen belassen, um für die Analysen einen einfachen Bezugspunkt zu haben.

5 Programmierung und Visualisierung einzelner Fragestellungen

Mit der gewonnenen Übersicht und der Anpassung des DataFrame sind die Daten bereit für die Auswertung nach einzelnen Fragestellungen. Im Folgenden werden jeweils die verwendeten Funktionen und Visualisierungen beschrieben. Sofern gleiche Funktionen mehrfach verwendet wurden, werden diese nur einmal erläutert.

5.1 Häufigkeiten der Begriffe in den Spalten BEREICH und TYPUS

Die Spalten BEREICH und TYPUS eignen sich gut als Ausgangspunkt für Gruppierungen. Daher wurden die beiden Spalten mit der Methode *unique* hinsichtlich der verwendeten Begriffe untersucht.

Für die Spalte BEREICH wird ein Array mit den Begriffen ['Person', 'Gruppe', 'Familie'] ausgegeben.

Für die Spalte TYPUS wird ein etwas umfangreicheres Array mit den Berufsbezeichnungen ['Künstler', 'Künstlerin', 'Fotografin (Lexikon)', 'Fotograf (Lexikon)', 'Fotografin', 'Fotograf', nan, Kunsthistoriker'] ausgegeben. Der Zusatz «(Lexikon)» ist für die Analyse nicht relevant und wird mit einem regulären Ausdruck *replace* entfernt. Die Klammern () müssen in eckigen Klammern [] geschrieben werden, damit sie ersetzt werden.

Ein zentrales Element für die Analyse der Instanzen ist die DataFrame-Methode *groupby()*. Dabei wird ein Spaltenname als Schlüssel übergeben. Erst durch die nachfolgende Aggregation, z.B. mit den Methoden *count*, *sum*, *mean*, *max*, *min*,

filter, *apply*, *aggregate*) wird eine Berechnung ausgeführt. Weiter kann eine Spaltenindizierung vorgenommen werden. Hierfür wurde meist die Spalte HAUPTNR gewählt, wie das folgende Beispiel zeigt:

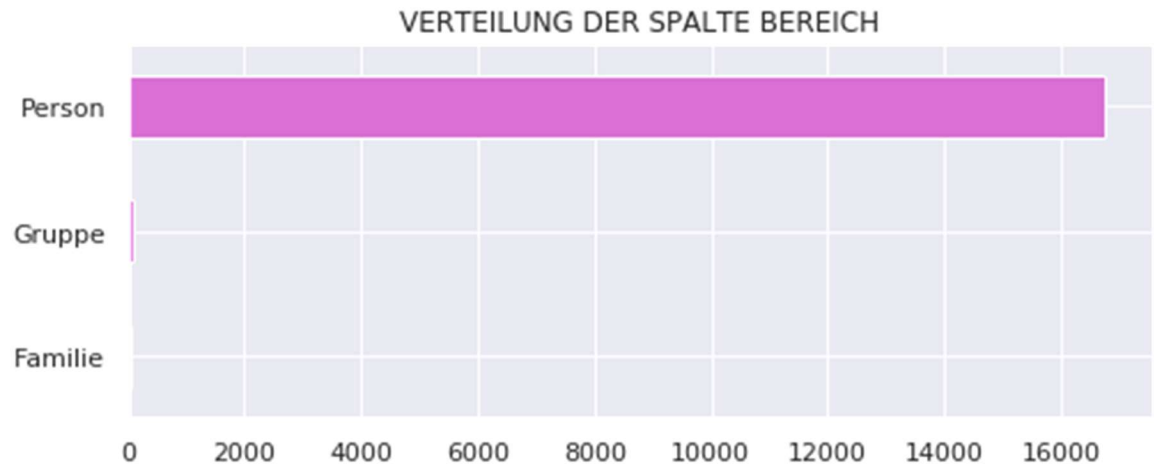
```
bereich = df.groupby("BEREICH")['HAUPTNR'].count()
```

DataFrames verfügen über eine Plot-Methode, weshalb ein Plot direkt anschliessend an die GroupBy-Methode ausgeführt werden könnte:

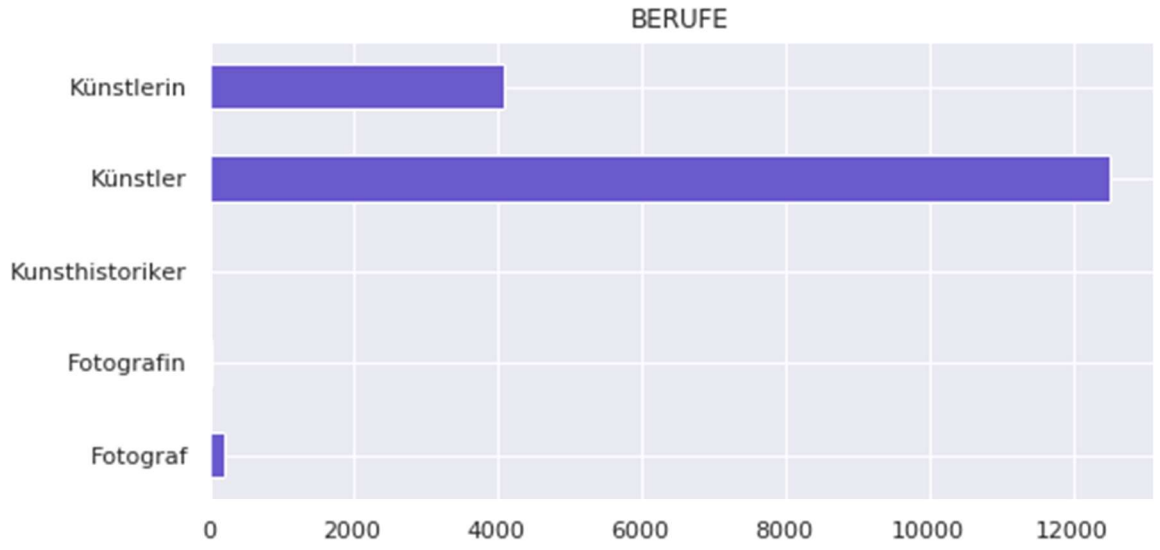
```
df.groupby("BEREICH")['HAUPTNR'].count().plot(kind='barh', color='orchid', title="...")
```

Aus Gründen der Übersichtlichkeit wurde jeweils eine Variable definiert und das Diagramm separat ausgeführt. Für die Visualisierungen wurden Pandas-Plots und Balkendiagramme gewählt. Angepasst wurden, Grösse, Farbe und Titel. Ebenso wurde die Beschriftung der y-Achse mit *plt.ylabel('')* entfernt.

Das Diagramm für die Spalte BEREICH zeigt, dass es sich beim überwiegenden Teil der Instanzen im Datensatz um Personen handelt, Gruppen und Familien sind vernachlässigbar.



Bei den Berufen überwiegen die Künstler deutlich. Bei den Fotografinnen und Fotografen handelt es sich um eine jüngere Berufsgattung, daher sind hier niedrige Zahlen zu erwarten.



5.2 Häufigste Namen

Die Frage nach der Häufigkeit von Namen ist beliebt. Sie eignet sich gut für eine Visualisierung und soll daher hier nicht fehlen.

Eine erste Auswertung mit absteigender Sortierung der Werte ergab mehr als 9000 unterschiedliche Namen:

```
name = df.groupby('NAME')['HAUPTNR'].count().sort_values(ascending=False)
```

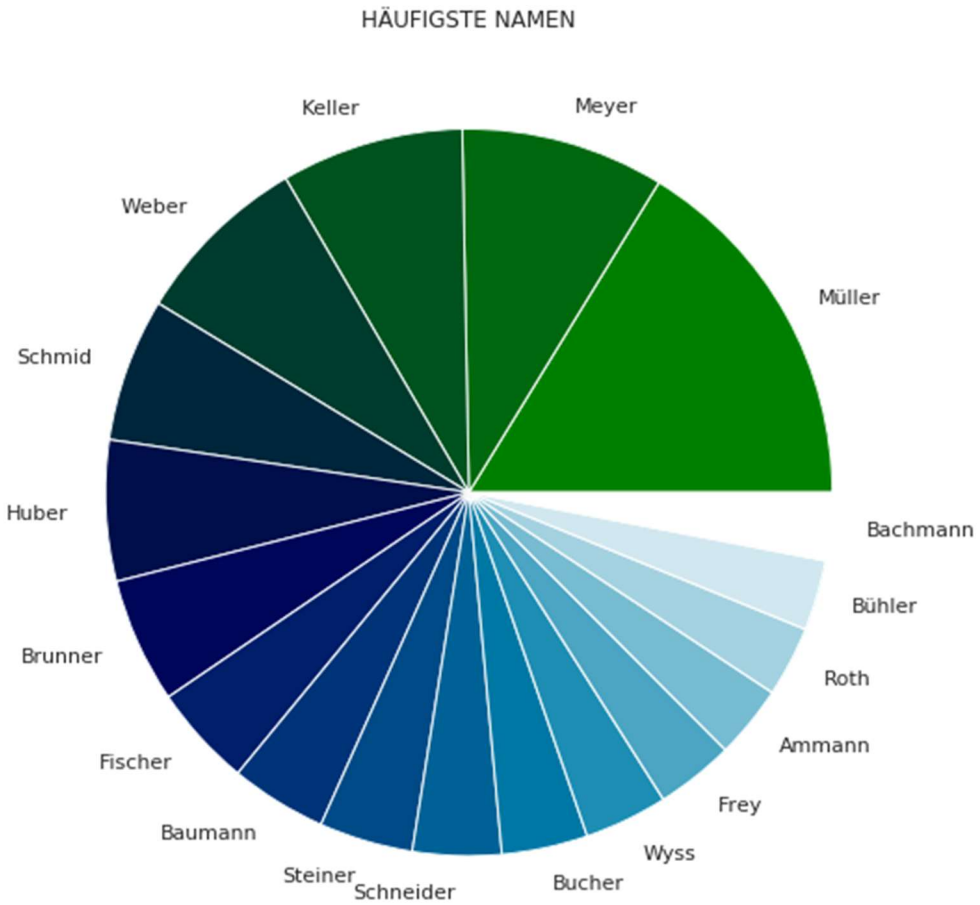
Dies sind deutlich zu viele für eine Darstellung mittels eines Diagramms, daher wurde zunächst mit verschiedenen booleschen Arrays die Verteilung der Häufigkeit untersucht:[1]

np.sum(name > 10)	126 Namen treten mehr als 10-mal auf
np.sum(name == 1)	6472 Namen treten genau einmal auf
np.sum((name > 1) & (name <= 10))	Die Verwendung der bitweisen Logikoperatoren von Python, zeigt, dass 2436 Namen 2- bis 10-mal auftreten

[1] JVDPL. S. 92-94)

Letztlich wurden mittels Maskierung (*topname = name[name > 25]*) diejenigen Namen ermittelt, die mehr als 25-mal *a*uftreten.

Für die Visualisierung wurde ein Kuchendiagram mit Pandas gewählt. Als Besonderheit wurde eine Colormap mit dem Matplotlib-Parameter *cmap* definiert. Wenig überraschend für ein Schweizer Lexikon, treten typische Schweizer Namen als häufigste Namen auf.



5.3 Häufigste Vornamen von Künstlerinnen

Als Variante zu den häufigsten Namen sollten auch die Vornamen von Künstlerinnen analysiert werden. Künstlerinnen haben keinen eigenen Spaltennamen, sondern werden unter TYPUS ausgewiesen. Daher musste mit *df_typus = df.set_index("TYPUS")* die Spalte TYPUS als Index gesetzt werden

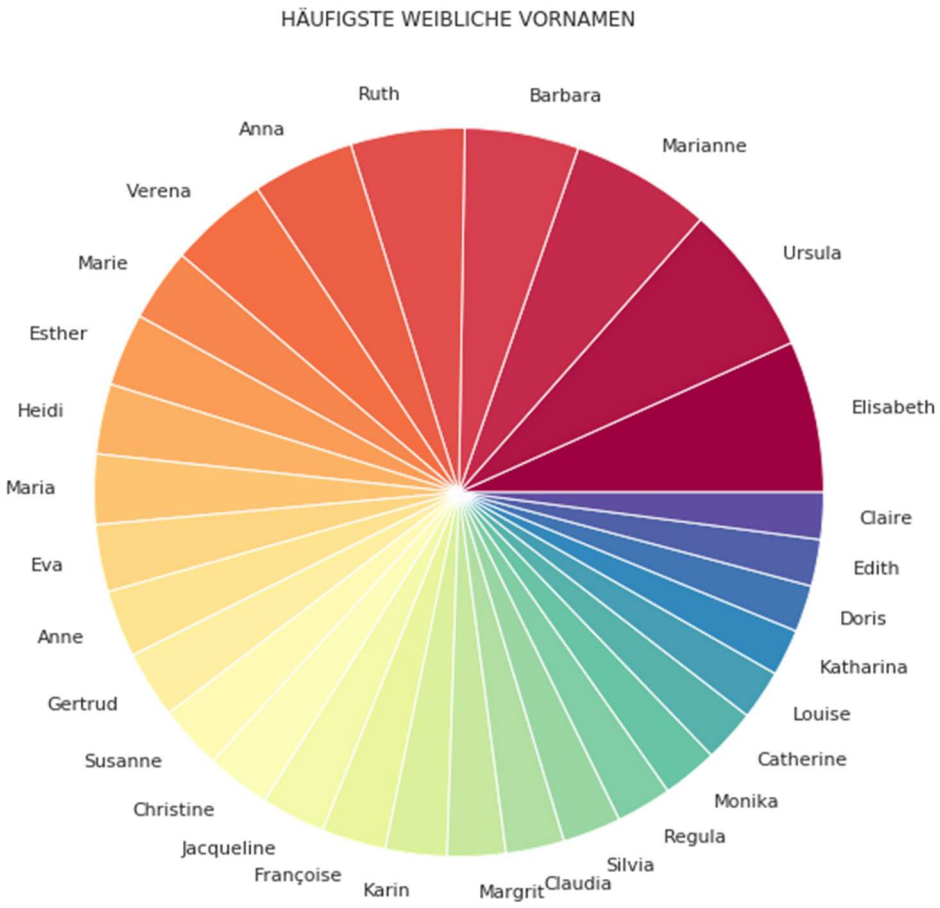
Für Indizierung und Slicing besitzt Pandas die praktischen Attribute *loc*, *iloc* und *ix*. *Loc* verwendet immer einen expliziten Index, d.h. der letzte Index wird

miteingeschlossen (VanderPlas 127-130). Hier wird *loc* für die Auswahl des Begriffes «Künstlerin» benutzt.

Damit ein DataFrame erhalten blieb, wurden die beiden Spalten VORNAME und HAUPTNR übergeben («Indexing and Selecting").

```
df_typus_weiblich = df_typus.loc[["Künstlerin"], ['VORNAME', 'HAUPTNR']]
wvorname = df_typus_weiblich.groupby('VORNAME')['HAUPTNR'].count().
sort_values(ascending=False)
```

Für die Visualisierung wurde wiederum ein Kuchendiagram mit Pandas gewählt, verändert wurden nur Grösse und Farbpalette. Inhaltlich überwiegen Vornamen aus der deutschsprachigen Schweiz.

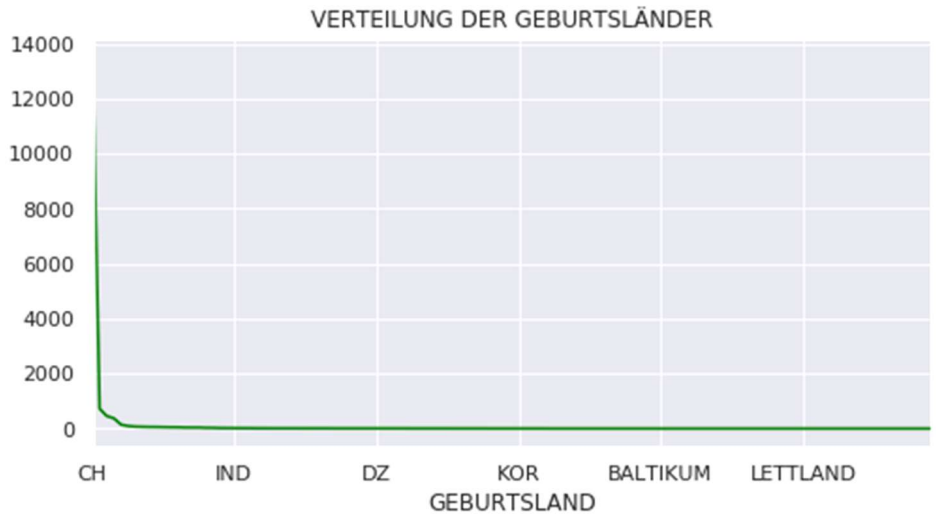


5.4 Zusammensetzung der Geburtsländer

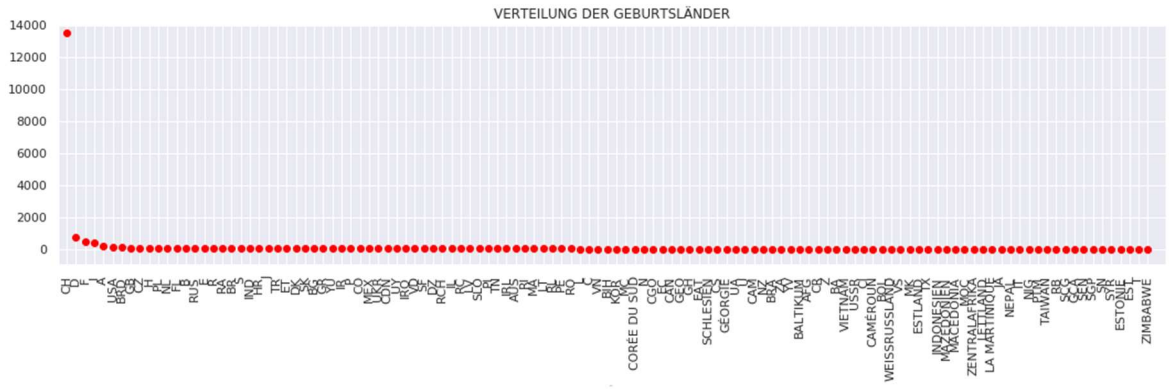
Im Vergleich zu den Sterbeländern sind für die Geburtsländer wesentlich mehr Daten vorhanden. Obwohl erwartet werden kann, dass die meisten Kunstschaffenden in der Schweiz geboren sind, sollten die Geburtsländer genauer analysiert werden.

Zunächst wurden sämtliche Begriffe ausgegeben. Diese bestehen meistens nur aus den Länderkürzeln und weisen auch einige veraltete Länderbezeichnungen aus. Für mehr als 13'500 Instanzen sind die Kunstschaffenden schweizerischer Herkunft, gefolgt von gut 723 aus Deutschland und 458 aus Frankreich. Durch diese ungleiche Verteilung der Werte liess sich eine sinnvolle Visualisierung nicht in einem Schritt realisieren, wie nachfolgend gezeigt wird.

Zunächst wurde ein einfacher Pandas-Plot als Liniendiagramm ausgegeben. Das Resultat ist nicht aussagekräftig. Auch konnte die Beschriftung unterhalb des Plots nicht entfernt werden.



Da es sich bei den Ländern um distinkte Einheiten handelt, wurde in einem nächsten Versuch ein Streudiagramm mit logarithmischer Darstellung der y-Achse realisiert. Dies brachte einige Verbesserungen. Die Beschriftungen der x-Achse wurde durch Rotation einigermassen lesbar, die Schriftgrösse und die ganze Grafik sind aber zu klein, wie nachfolgend zu sehen ist.



Als nächstes wurden daher Geburtsländer allein für Personen ermittelt, dies obwohl Familien und Gruppen nur selten vorkommen. Dazu wurde die Spalte BEREICH als Index gesetzt und mit dem Indexer *loc* die Personen ausgewählt

```
df_person = df_bereich.loc["Person"]
```

Anschliessend wurde eine Series ausgegeben, welche automatisch sortiert und NaN-Werte löscht.

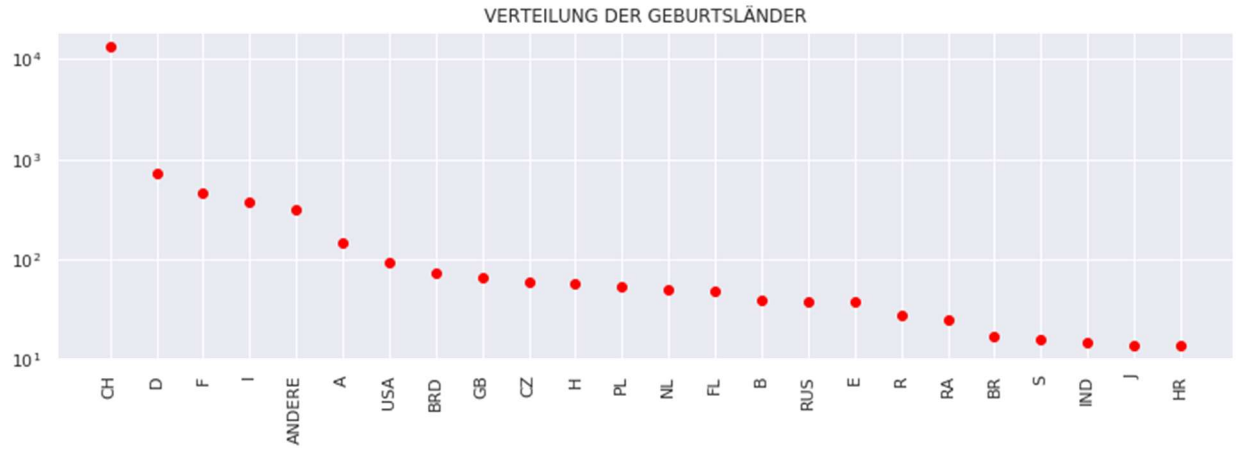
```
pers_land = df_person.GEBURTSLAND.value_counts()
```

Weiter wurden Geburtsländer, die weniger als 10-mal ausgewiesen werden, mittels Maskierung zu einer Gruppe «Andere» zusammengefasst, um die Verteilung der Werte einzuschränken. Dazu wurde ein Code-Beispiel aus dem Internet angepasst[2]. Anschliessend wurden die Werte neu sortiert, damit die Ländergruppe "Andere" nicht am Schluss angezeigt wird.

```
threshold = 10
mask = pers_land > threshold
tail_pers_land = pers_land.loc[~mask].sum()
pers_land = pers_land.loc[mask]
pers_land['ANDERE'] = tail_pers_land
pers_land = pers_land.sort_values(ascending=False)
```

Als Plot wurde wiederum ein Streudiagramm mit logarithmischer Darstellung der y-Achse gewählt. So wird immerhin ersichtlich, dass die nach der Schweiz häufigsten Geburtsländer alle an die Schweiz angrenzen. Eine Ausnahme bildet das Fürstentum Lichtenstein, das weniger oft vertreten ist.

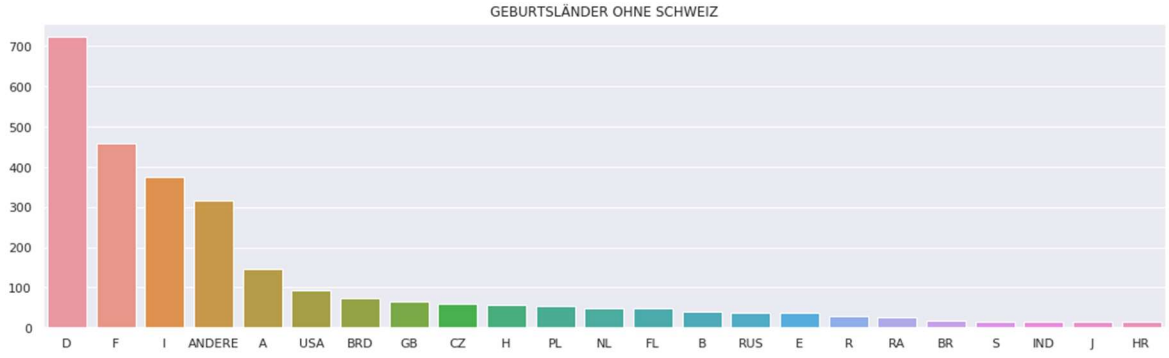
[2] https://stackoverflow.com/questions/37598665/how-to-plot-a-value-counts-in-pandas-that-has-a-huge-number-of-different-counts



Für eine letzte Visualisierung wurden schliesslich die Geburtsländer ohne Schweiz aber weiterhin mit der Gruppe «Andere» ausgewählt. Dazu wurden die Werte «CH» mit der Drop-Methode in Pandas entfernt.

```
ohne_ch = pers_land.drop(labels=['CH'])
```

Diese Visualisierung wurde als Säulendiagramm mit Seaborn erstellt und sieht farblich ansprechender aus, wenn auch die Textelemente insgesamt klein ausfallen. Die Beschriftungen wurden mit Matplotlib erstellt.



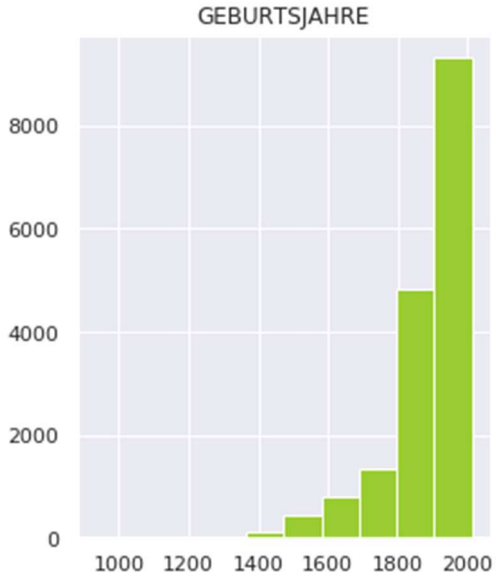
5.5 Geburts- und Sterbejahre

In einem weiteren Schritt wurden die Geburts- und Sterbejahre analysiert. Diese Daten wurden im DataFrame mit dem Datentyp *float* ausgegeben. Das heisst, hier mussten zunächst die fehlenden Werte entfernt und der Datentyp geändert werden. Dazu wurde jeweils ein Pandas-Series-Objekt erzeugt und mit der Methode *dropna* NaN-Werte entfernt. Anschliessend wurden mit den Methoden *astype* und *sort_values* die Jahre in Integers umgewandelt und absteigend sortiert.

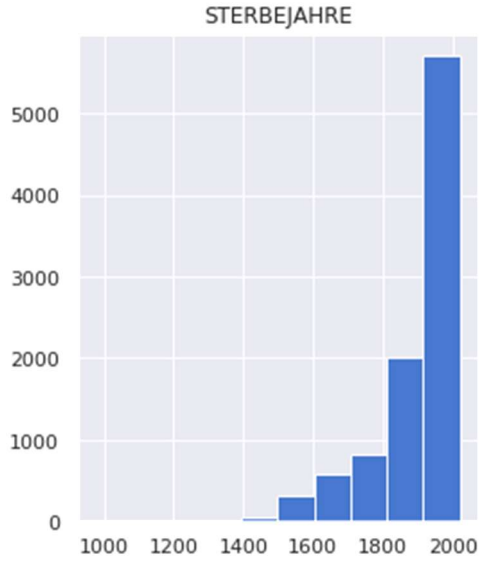
```
geburtsjahr = pd.Series(df['GEBURTSJAHR']).dropna()
geburtsjahr = geburtsjahr.astype('int').sort_values(ascending=False)
```

Auch bei diesen Plots wurden verschiedene Varianten ausprobiert. Daher sollen hier Visualisierung und Code nebeneinandergestellt werden.

Zunächst wurden für Geburts- und Sterbejahre separate Histogramme als Pandas-Plots mit farblicher Unterscheidung ausgegeben.

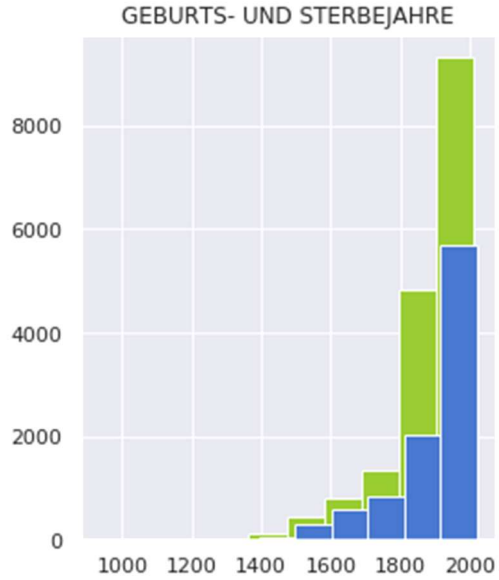



```
geburtsjahr.plot
(kind='hist', figsize=(4,5),
 title='GEBURTSJAHRE',
color='yellowgreen')
plt.ylabel('')
```



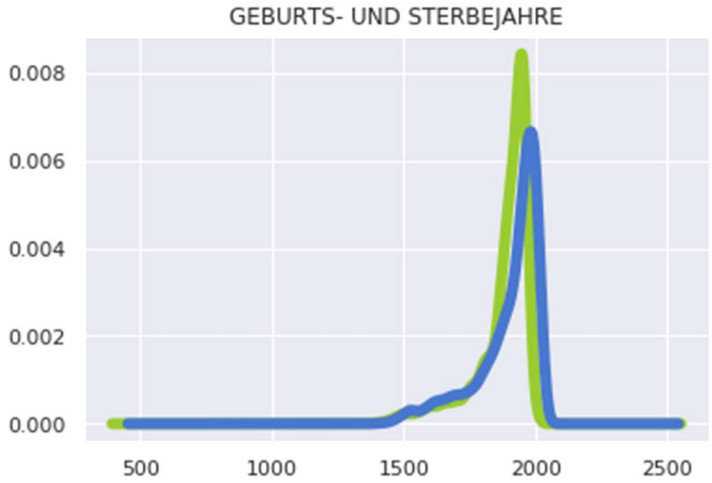

```
sterbejahr.plot
(kind='hist', figsize=(4,5),
 title='STERBEJAHRE')
plt.ylabel('')
```

Anschliessend wurden die beiden Histogramme übereinandergelegt. Dazu wurde die Matplotlib-Funktion *subplots* genutzt.



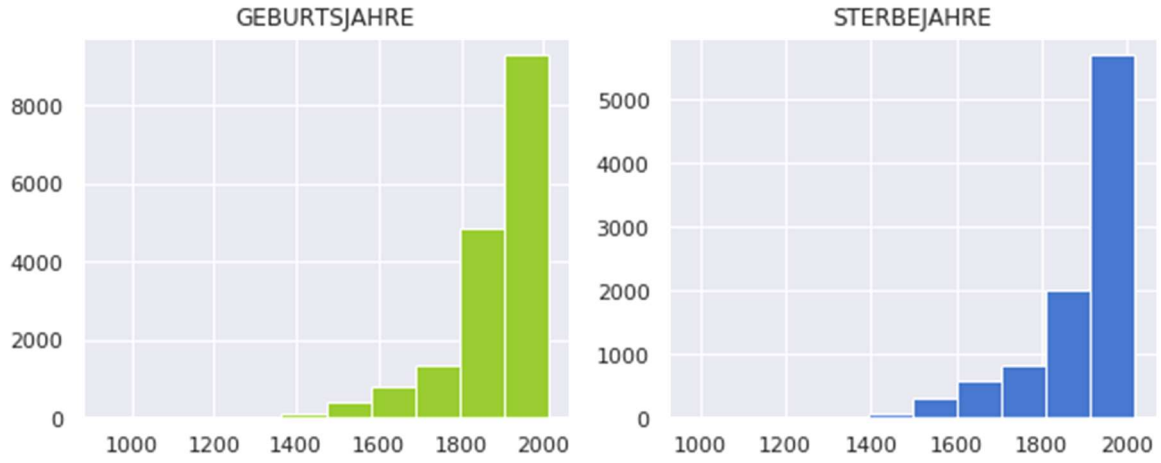

```
fig, ax = plt.subplots()
geburtsjahr.plot(kind='hist',
color='yellowgreen')
sterbejahr.plot(kind='hist',
figsize=(4,5),
title='GEBURTS- UND
STERBEJAHRE')
plt.ylabel('')
```

Als Spezialfall eines Histogrammes wurden die Geburts- und Sterbejahre auch als Density-Plot ausgegeben. Anstelle der Series wurden die originalen Spalten des DataFrame übergeben. Ausserdem wurde eine dicke Linienbreite gewählt.




```
fig, ax = plt.subplots()
df['GEBURTSJAHR'].
plot.density(figsize=(6,4),
title='GEBURTS- UND
STERBEJAHRE',
color='yellowgreen',
linewidth=(6))
df['STERBEJAHR'].
plot.density(linewidth=(6))
plt.ylabel('')
```

Zuletzt wurden die beiden Subplots mit Matplotlib nebeneinander angeordnet. Dazu wurde die objektorientierte Schnittstelle genutzt.



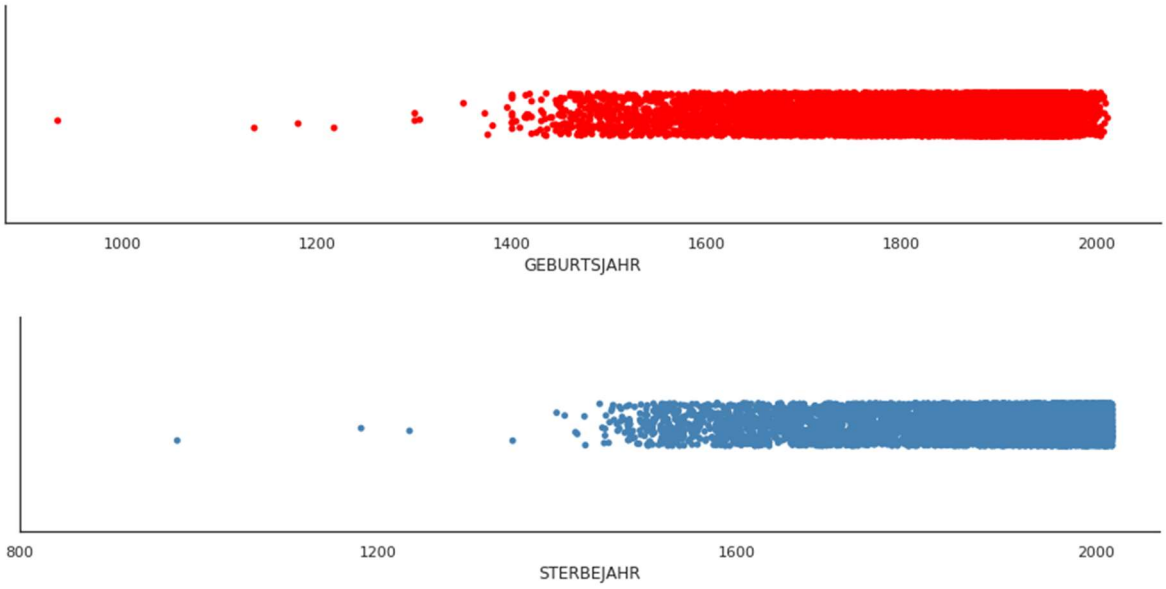
Der dazugehörige Code:

```
fig = plt.figure(figsize=(10,8))
sub1 = fig.add_subplot(221)
sub1.set_title('GEBURTSJAHRE')
geburtsjahr.plot(kind='hist', color='yellowgreen')
plt.ylabel('')

sub2 = fig.add_subplot(222)
sub2.set_title('STERBEJAHRE')
sterbejahr.plot(kind='hist')
plt.ylabel('')
```

Diese einfachen Grafiken lassen sich mit gut mit Pandas und Matplotlib realiseren und sind durch den Seaborn-Stil dennoch ansprechend.

Gleichwohl wurden die Geburts- und Sterbejahre auch noch als Seaborn-Visualisierung ausgegeben. Genutzt wurde die Catplot-Funktion mit dem Parameter *kind='strip',* so dass ein Punktediagramm ausgegeben wurde. Die Datenmenge ist dafür aber zu gross, bzw. deren Verteilung zu gering. Hier konnte wie auch bei den übrigen Seaborn-Plots keine Titel angegeben werden. Auch die Veränderung der Schriftgrösse erschliesst sich nicht so einfach.



5.6 Lebensdauer von Personen

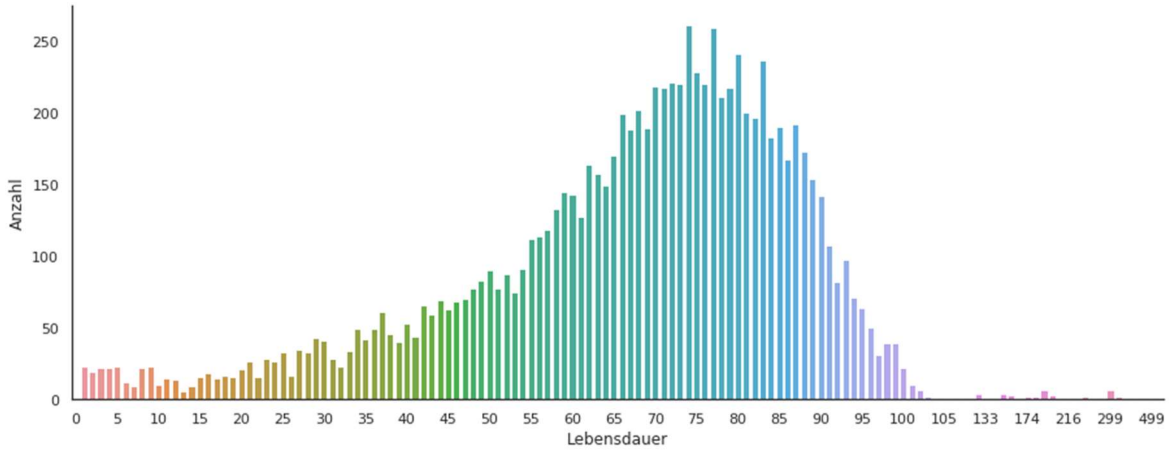
Nach der Analyse von Geburts- und Sterbejahren war es naheliegend, die Lebensdauer von Personen auszuweisen. Diese musste jedoch zuerst berechnet werden.

Bei einem DataFrame kann die gleiche Syntax wie bei einem Dictionary verwendet werden, das heisst, Spalten können zum Beispiel subtrahiert werden. Die Ausgabe des folgenden Codes brachte jedoch zum Teil unrealistische Lebensjahre von 0-1 oder 399-599 hervor.

```
df['LEBENSDAUER'] = df['STERBEJAHR'] - df['GEBURTSJAHR']
lebensdauer = pd.Series(df['LEBENSDAUER']).dropna()
lebensdauer = lebensdauer.astype('int').sort_values(ascending=False)
```

Daher wurden mit *df.loc['Wert']* einzelne Zeilen geprüft. In einem Fall handelte es sich bei der Instanz um eine Künstlerfamilie, im anderen um eine Person, deren Lebensdaten offensichtlich nicht bekannt sind. Eine Überprüfung des Datensatzes im SIKART-Lexikon zeigte, dass die Jahre angegeben wurden, in denen die Person nachweislich ein Kunstwerk geschaffen hat.

Die folgende Seaborn-Visualisierung zeigt die Lebensdauer für Personen, Familien und Gruppen. Familien können eine «Lebensdauer» von mehreren Jahrhunderten haben, die einzelnen Kunstschaffenden erfreuten sich mehrheitlich eines langen Lebens. Der Plot wurde als Catplot mit *kind=count* erzeugt.



Zwar wurde versucht, zumindest die Instanzen Familien und Gruppen für diese Auswertung aus dem DataFrame zu filtern, indem ein neuer DataFrame *df_person* erstelt wurde, wie die folgenden Zeilen zeigen.

```
df_bereich = df.set_index("BEREICH")
df_person = df_bereich.loc[["Person"], :]
df_person['LEBENSDAUER'] = df_person['STERBEJAHR'] - df_person['GEBURTS-
JAHR']

leben = pd.Series(df_person['LEBENSDAUER']).dropna()
leben = leben.astype(int)
```

Allerdings gelangt es nicht, die Jahre als Plot auszugeben. Die Ursache dafür ist noch zu ermitteln.

5.7 Geburtsjahre in Jahrzehnten

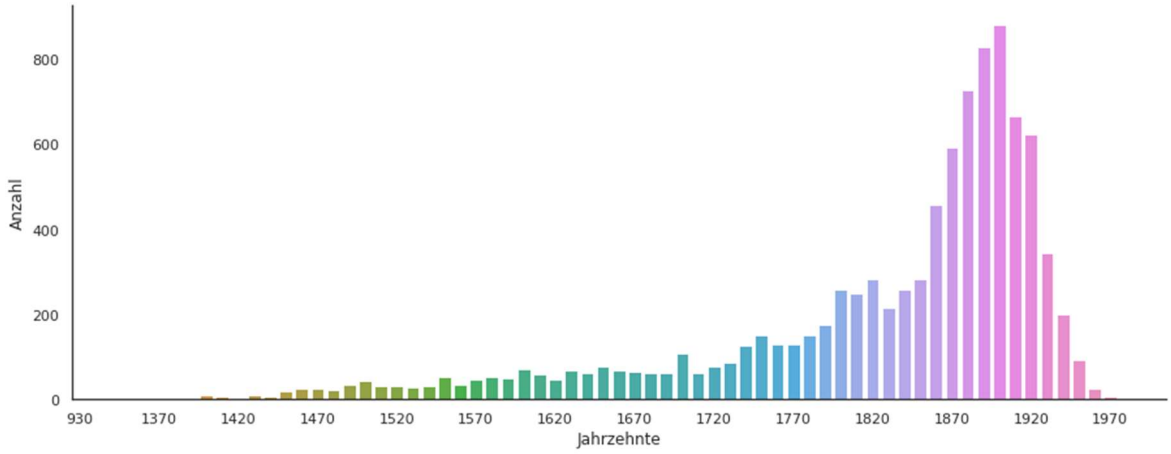
Wie bei der vorherigen Analyse der Lebensdauer wurde hier dem DataFrame eine weitere Spalte hinzugefügt, welche die Geburtsjahre in Jahrzehnten ausweist. Hierzu wurde eine Berechnung mit dem so genannten *floor division* Operator // vorgenommen und die Jahre in den Datentyp Integer umgewandelt.

```
df['JAHRZEHNT'] = 10 * (df['GEBURTSJAHR'] // 10)
```

Fehlende Werte wurden mit *dropna* gefiltert, wobei der Filter explizit nur für die Spalte «Jahrzehnt» angewendet wurde. Dazu kann der Parameter *subset* benutzt werden.

```
df = df.dropna(subset=['JAHRZEHNT'])
df['JAHRZEHNT'] = df['JAHRZEHNT'].astype(int)
```

Ziel war es, die Anzahl Geburtsjahre zu reduzieren und diese als Ausgangslage für weitere Analysen zu nutzen. Zur Kontrolle wurde wiederum ein Säulendiagramm mit Seaborn erstellt.



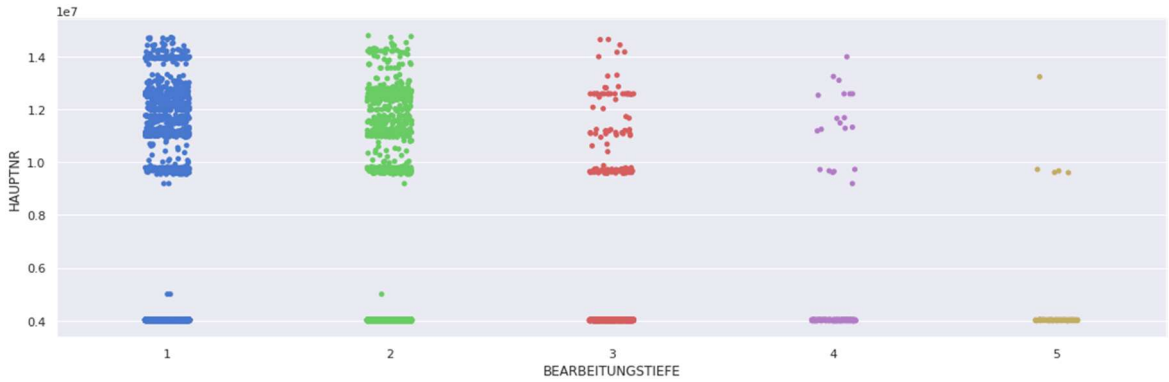
5.8 Bearbeitungstiefe

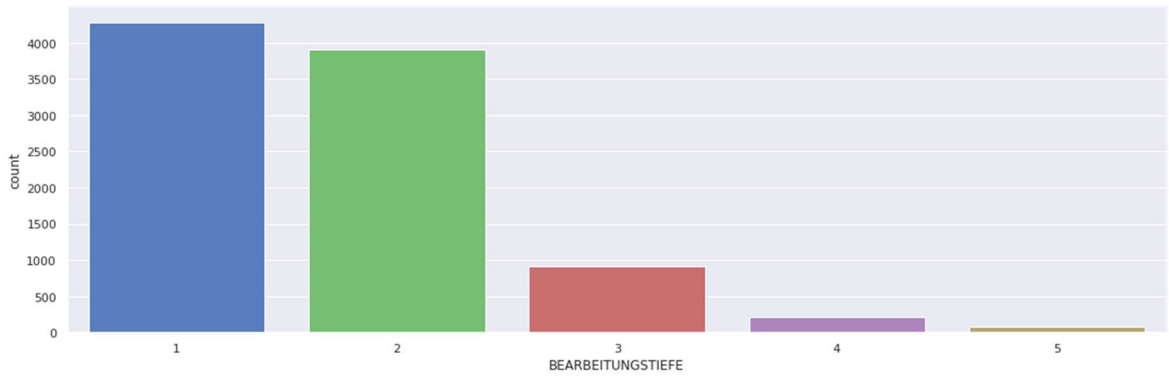
Das letzte Visualisierungskapitel befasste sich mit der Frage der Qualität oder Verlässlichkeit der erfassten Daten. Im SIKART-Datensatz wird eine Spalte mit einer so genannten BEARBEITUNGSTIEFE geführt. Die Bearbeitungstiefe wird mit 1-5 Sternchen * angegeben. Es ist nicht bekannt, nach welchen Kriterien die Sternchen vergeben werden.

Die Sternchen in der Spalte Bearbeitungstiefe wurden zunächst mit einem regulären Ausdruck durch numerische Werte (Integers) ersetzt.

```
df['BEARBEITUNGSTIEFE'] = df['BEARBEITUNGSTIEFE'].dropna().
replace("*****", int(5)).replace("****", int(4)).replace("***", int(3)).
replace("**", int(2)).replace("*", int(1))
```

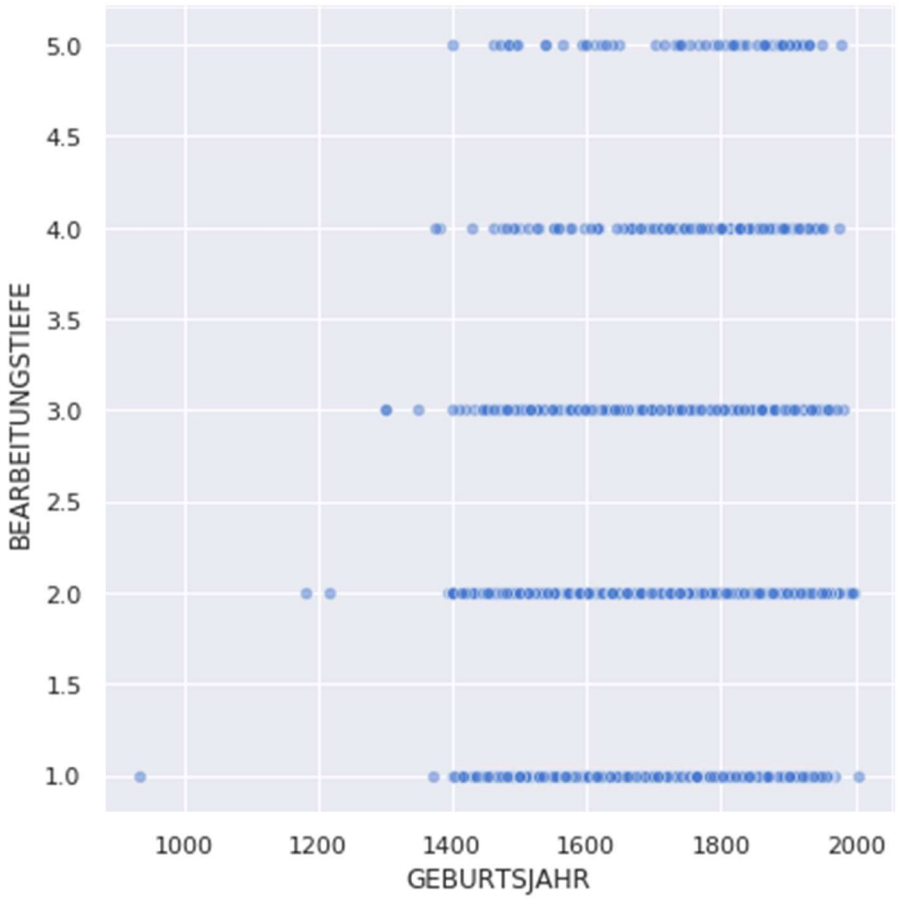
Bei den beiden folgenden Visualisierung ging es vor allem um das Ausprobieren und weniger um die Frage, welcher Plot wann geeignet ist. Im ersten Beispiel wurde ein Punktdiagramm (Stripchart) ausgegeben, im zweiten ein Balkendiagramm.





Das Balkendiagramm liefert die klarere und verständlichere Abbildung. Es wird gut sichtbar, dass die meisten Instanzen eine Bearbeitungstiefe von 1 oder 2 aufweisen. Nur ein kleiner Teil der Datensätze weist die Bearbeitungstiefe 3, 4 oder 5 auf.

In einem letzten Schritt wurde die Bearbeitungstiefe in Beziehung zu den Geburtsjahren gesetzt. Dazu wurde der *relplot* von Seaborn verwendet. Die Visualisierung zeigt, dass die Bearbeitungstiefe der SIKART-Daten nicht durch das Geburtsjahr bestimmt wird. Sowohl für historisch weiter zurückliegende Kunstschaffende wie auch für zeitgenössische Kunstschaffende ist die Bearbeitungstiefe recht gleichmässig verteilt wie die nachfolgende Abbildung zeigt.


BEARBEITUNGSTIEFE
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1000
1200
1400
1600
1800
2000
GEBURTSJAHR

6 Schlusswort

Insgesamt stellte der SIKART-Datensatz eine gute Möglichkeit dar, den Umgang mit wenig strukturierten, mehrheitlich textuellen Daten zu üben. Ein Schwerpunkt lag denn auch auf der Auswertung der Daten mit Pandas. Zum Beispiel durch Gruppierung und Indizierung, dem Filtern von fehlenden Daten oder durch das Erstellen von neuen Spalten.

Das Ziel, einen Überblick über die Informationen im Datensatz zu erhalten, konnte grösstenteils erreicht werden. Ausgelassen werden mussten die Geburts- und Sterbeorte sowie die exakten Lebensdaten. Jedoch wurde eine Basis geschaffen, auf der die Beziehungen zwischen den Daten stärker herausgearbeitet werden kann.

Für die vertiefte Erarbeitung der Plot-Bibliotheken, insbesondere für die Einarbeitung in Seaborn fehlte die Zeit. Ebenso wurden statistische Fragen nicht berücksichtigt. Möchte man den Fokus stärker auf die Visualisierungen legen, wäre ein Datensatz mit überwiegend numerischen Daten als Ausgangslage geeigneter.

7 Literaturverzeichnis

BFS. *Portal opendata.swiss*, https://opendata.swiss/de/about/. Zugriff 30.03.2020. *Biografische Daten zu Kunstschaffenden aus SIKART Lexikon zur Kunst in der Schweiz*, https://opendata.swiss/de/dataset/kuenstlernamen-aus-sikart-lexikon-zur-kunst-in-der-schweiz. Zugriff 30.03.2020

Cmdline. *Catplot Python Seaborn: One Function To Rule All Plots With Categorical Variables*, https://cmdlinetips.com/2019/03/catplot-in-seaborn-python. Zugriff 19.04.2020.

Differences between bar plots in Matplotlib and pandas, https://stackoverflow.com/questions/57677559/differences-between-bar-plots-in-matplotlib-and-pandas. Zugriff 10.03.2020.

Indexing and Selecting Data with Pandas, https://www.geeksforgeeks.org/indexing-and-selecting-data-with-pandas. Zugriff 19.04.2020.

Klein, Bernd. *Numerisches Python: Arbeiten Mit NumPy, Matplotlib Und Pandas*. Hanser, 2019.

Klein, Bernd. *Einführung in Python 3: Für Ein- Und Umsteiger*. 3., überarbeitete Auflage, Hanser, 2018.

Matplotlib - Object-oriented Interface, https://www.tutorialspoint.com/matplotlib/matplotlib_object_oriented_interface.htm. Zugriff 16.4.2020.
SIK-ISEA, https://www.sik-isea.ch/de-ch. Zugriff 30.3.2020.

Matplotlib Style Gallery, https://tonysyu.github.io/raw_content/matplotlib-style-gallery/gallery.html. Zugriff 15.4.2020

Pandas, https://pandas.pydata.org/ Zugriff 15.4.2020

Seaborn: statistical data visualization, https://seaborn.pydata.org/ Zugriff 20.4.2020

Soma, Jonathan. *Understand Df.plot in Pandas,* http://jonathansoma.com/lede/algorithms-2017/classes/fuzziness-matplotlib/understand-df-plot-in-pandas/. Zugriff 15.4.2020

VanderPlas, Jake. Data Science Mit Python: Das Handbuch Für Den Einsatz Von *IPython, Jupyter, NumPy, Pandas, Matplotlib, Scikit-Learn*. 1. Auflage., Mitp, 2018.

Visualization with Python, https://matplotlib.org/index.html. Zugriff 19.04.2020.

Moffitt, Chris. *Effectively Using Matplotlib,* https://pbpython.com/effective-matplotlib.html. Zugriff 15.4.2020.