Woche 05 Programmierung: Visualisierung

Skript

Erarbeitet von
Ann-Kathrin Selker

Lernziele

- rudimentäre Diagramme mit Plotly Express erstellen
- mithilfe eines gegebenen Handbuchs selbstständig Funktionen und Parameter recherchieren

Inhalt

Projekt „KI für alle"
HeiCAD – Heine Center for Artificial Intelligence and Data Science

CC BY

Datenvisualisierungen eignen sich unter anderem gut dafür, Trends in Daten zu erkennen oder Beziehungen zwischen Daten zu veranschaulichen. Doch wie können Datenvisualisierungen in Python erzeugt werden?

Einstieg

In diesem Video beschäftigen wir uns mit dem Visualisierungsmodul Plotly, insbesondere mit *Plotly express*. Der große Vorteil an Plotly Express ist, dass du bereits mit kleinen Befehlen gut lesbare und verständliche Grafiken erzeugen kannst. Daher ist dieses Modul für Anfänger gut geeignet. Plotly kann mit vielen verschiedenen Eingabeformaten umgehen, unter anderem auch mit Numpy-Arrays und sogenannten DataFrames aus dem Pandas-Modul. Pandas ist ein Modul, das auf Datenbereinigung und -analyse spezialisiert ist. Schau ruhig mal in das Pandas-Handbuch, ob du nützliche Befehle für deine Daten findest! Es gibt zum Beispiel viele Optionen zum Einlesen von Daten. Plotly hat bereits eingebaute Datensätze im DataFrames-Format, die wir im Folgenden benutzen werden. Für dieses Video benötigst du aber keinerlei Kenntnisse über das Pandas-Modul und das genaue Format der Daten ist hier irrelevant. Wir importieren zuerst das Modul plotly.express als px.

```
import plotly.express as px
```

Plotly

Wir betrachten den Datensatz "gapminder". Bei Gapminder handelt es sich um eine Stiftung, die mithilfe von zuverlässigen Daten über häufige Irrtümer aufklärt.

Quelle [1]

Der vorliegende Datensatz beinhaltet Länderdaten wie Name, Kontinent, Lebenserwartung usw.

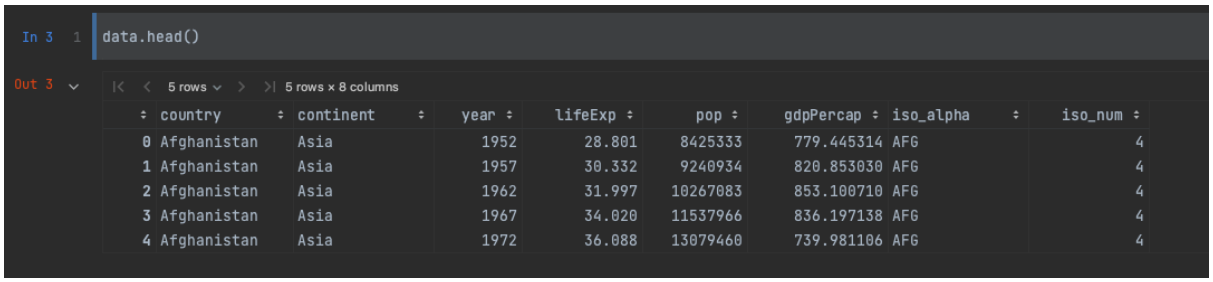
Wir laden die Daten mit dem folgenden Befehl:

```
data = px.data.gapminder()
```

Das *data =* gibt dabei an, dass wir unseren Datensatz in einer Variable namens *data* speichern, die wir im weiteren Verlauf benutzen können. Die Funktion *head* für DataFrames zeigt uns dann die ersten Zeilen unseres Datensatzes an.

```
data.head()
```

Auf diese Art und Weise können wir zum Beispiel sehen, welche Features unsere Daten haben und was einige Werte dieser Features sind. In unserem Fall, also wenn wir *data.head()* eingeben, wird uns dies angezeigt.



In 3 1 `data.head()`

Out 3 — 5 rows × 8 columns

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	4
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	4
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	4
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	4
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	4

Es gibt 8 Spalten, unter anderem mit Land, Kontinent, Jahr, Lebenserwartung, Bevölkerung und Bruttoinlandsprodukt pro Kopf. Es ist voreingestellt, dass *head* 5 Zeilen anzeigt. Falls du mehr Zeilen einsehen möchtest, kannst du die entsprechende Anzahl in die Klammern schreiben, also z. B. *data.head(10).*

```
data.head(10)
```

Falls du dich nur für bestimmte feature-Werte interessierst, kannst du deine Daten auch mit dem *query*-Befehl einschränken. Vereinfacht gesagt benötigt *query* als Parameter einen Ausdruck, auf den es die Zeilen deiner Tabelle überprüfen soll. Dieser Ausdruck muss als String vorliegen, also von Anführungsstrichen umgeben sein. Zum Beispiel sorgt dieser Befehl dafür, dass nur die Daten für Deutschland angezeigt werden.

```
dataGermany = data.query("country == 'Germany'")
dataGermany.head(10)
```

Das Land wird im Feature *country* angegeben. Da es sich bei den Ländernamen um Strings handelt, darfst du die Anführungsstriche um *Germany* nicht vergessen. Mit dem doppelten Gleichheitszeichen kontrollierst du die Gleichheit von beiden Ausdrücken.

Falls du zum Beispiel nur Einträge seit 1990 betrachten willst, nimmst du diesen Befehl. Da es sich bei 1990 um eine Zahl handelt, wird sie nicht in Anführungsstrichen geschrieben.

```
dataSince1990 = data.query("year >= 1990")
dataSince1990.head(10)
```

Es ist außerdem möglich, mehrere mögliche Werte eines Features gleichzeitig zu betrachten. Dieser Befehl schränkt unsere Daten auf die Kontinente Europa und Afrika ein.

```
dataEuropeAfrica = data.query("continent in ['Europe', 'Africa']")
dataEuropeAfrica.head(10)
```

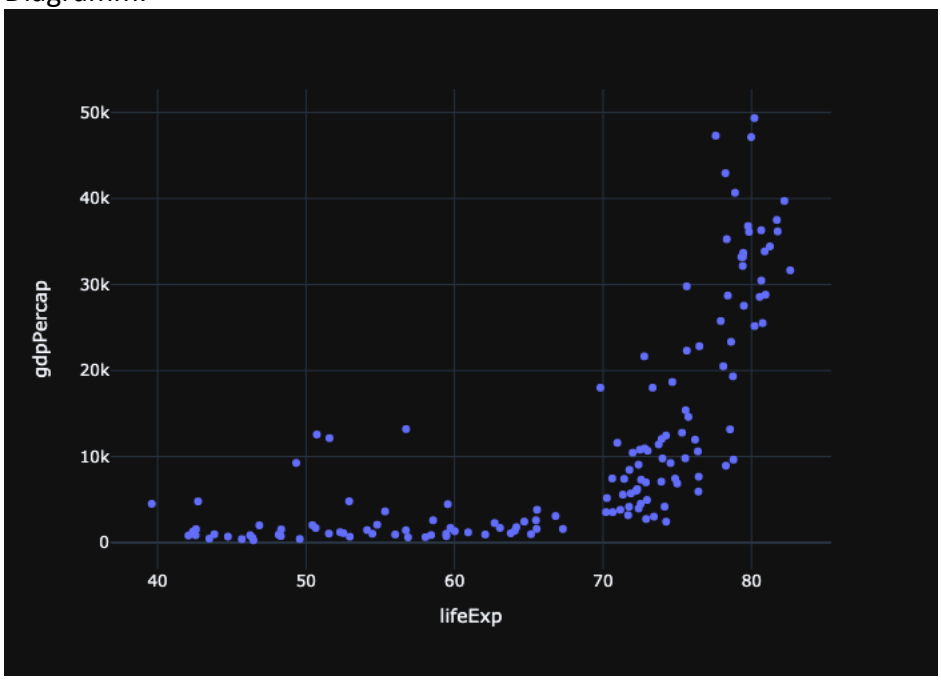
Streudiagramme

Fangen wir mit Streudiagrammen an, um uns zuerst einmal unsere Daten in grafischer Form anzeigen zu lassen. Für diese Diagramme benutzen wir den Befehl *scatter.* Wenn wir uns jetzt den Bezug zwischen Lebenserwartung und BIP der Länder ansehen wollen, können wir dafür diesen Befehl verwenden.

```
px.scatter(data.query("year == 2007"), x = 'lifeExp', y = 'gdpPercap')
```


CC BY

Der erste Parameter gibt den betrachteten Datensatz an. Der Übersichtlichkeit halber betrachten wir nur Daten aus dem Jahr 2007, schränken also wieder mit *query* unsere Daten ein. Auf der x-Achse ist die Lebenserwartung und auf der y-Achse das BIP aufgetragen, daher weisen wir diese mit *x=* und *y=* zu. Zur Erinnerung: Die x-Achse ist die horizontale und die y-Achse die vertikale Achse eines Koordinatensystems. Die Zuweisung funktioniert einfach über den entsprechenden Spaltennamen. Auch hier gilt: Die Spaltennamen sind Strings, also die Anführungsstriche nicht vergessen! Als Ausgabe erhalten wir dieses Diagramm.



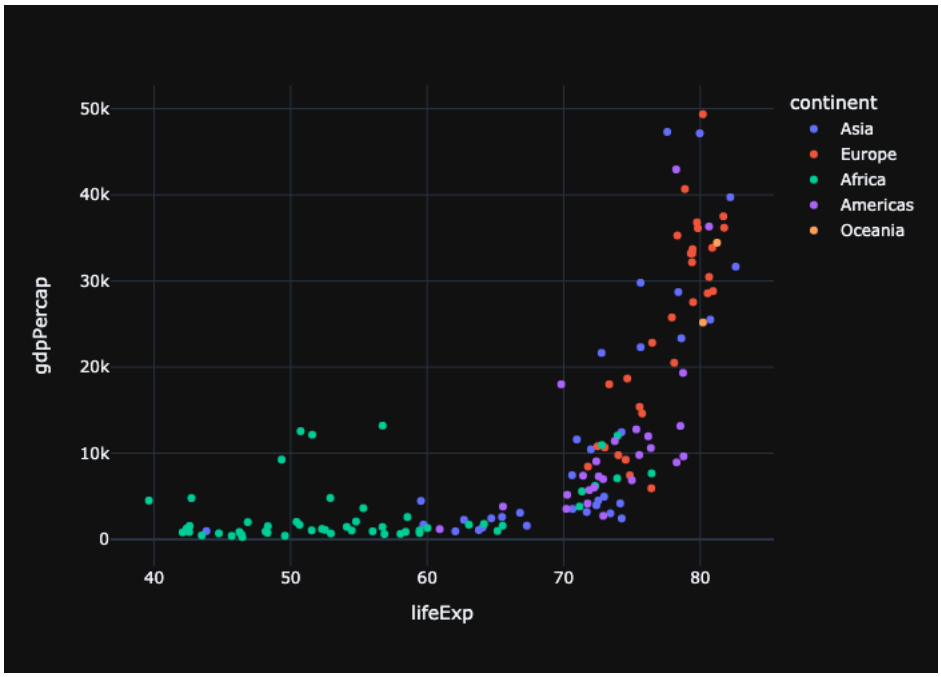
Einblendung Streudiagramm

Das Diagramm ist interaktiv. Du musst also nur mit der Maus über die Linie fahren, um die genauen Werte angezeigt zu bekommen. Doch was, wenn wir uns zusätzlich für den Vergleich zwischen den Kontinenten interessieren? Mit dem Parameter *color* können gleiche Feature-Werte gleich eingefärbt werden. In diesem Diagramm haben also alle Länder desselben Kontinents auch dieselbe Farbe.

```
px.scatter(data.query("year == 2007"), x = 'lifeExp', y = 'gdpPercap',
           color = 'continent')
```

Einblendung Streudiagramm

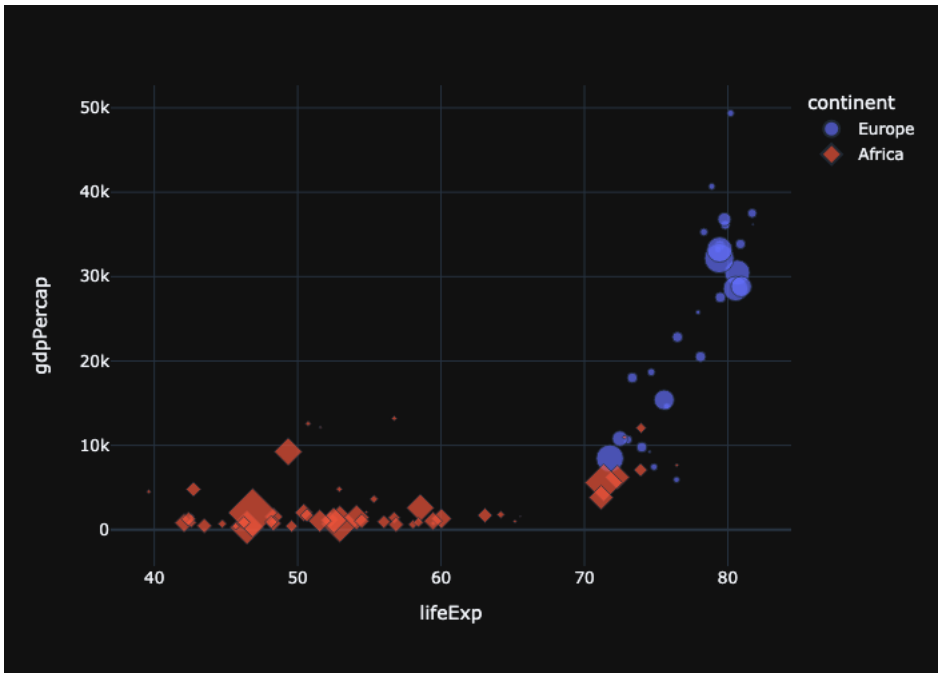
CC BY



Mit dem Parameter *symbol* kann das benutzte Symbol für die Punkte geändert werden, entweder für alle Punkte auf einmal oder in Abhängigkeit eines Features. Dasselbe gilt für den Parameter *size* für die Größe der Punkte. Damit das Diagramm nicht zu unübersichtlich wird, beschränken wir uns auf die Daten aus Europa und Afrika. Das Resultat sieht so aus.

```
px.scatter(dataEuropeAfrica.query("year == 2007"), x = 'lifeExp',
           y = 'gdpPercap', color = 'continent', symbol = 'continent',
           size = 'pop')
```

Einblendung Streudiagramm

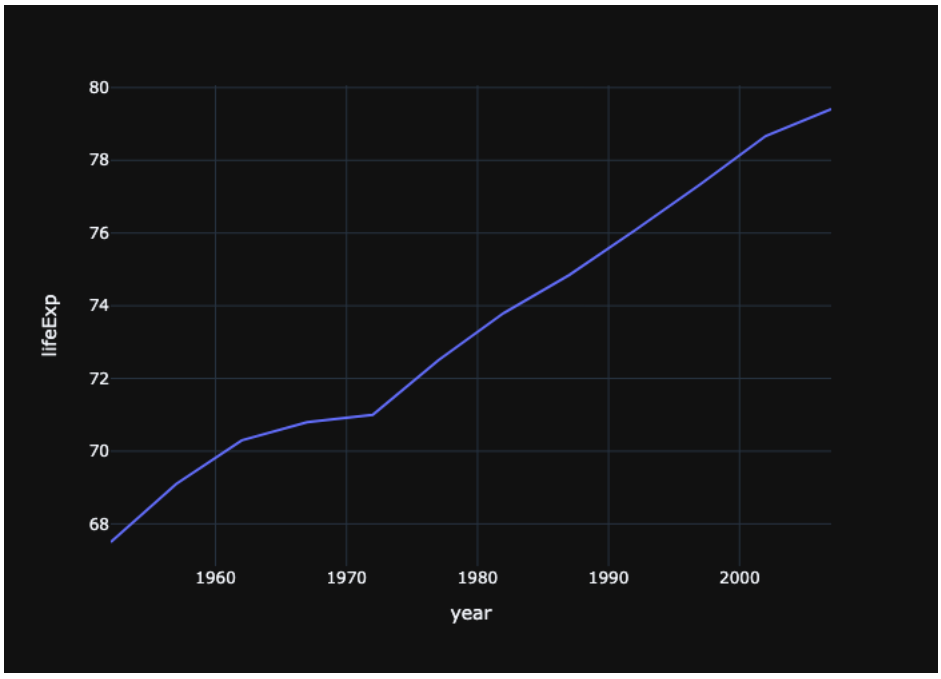


Scatterplots, wie viele andere Diagrammarten, gibt es auch in 3D. Der Befehl hierfür lautet *scatter_3d* und benötigt noch einen zusätzlichen Parameter z für die dritte Achse.

Liniendiagramme

Als nächstes schauen wir uns Liniendiagramme an. Für diese Diagramme gibt es den Befehl *line*. Betrachten wir zuerst die Entwicklung der Lebenserwartung in Deutschland. Dabei sollen auf der x-Achse die Jahre und auf der y-Achse die Lebenserwartung angezeigt werden. Der erste Parameter des *line*-Befehls ist wieder unser Datensatz, hier also der über Deutschland genannt *dataGermany*. Danach wird mit *x = 'year'* und *y = 'lifeExp'* angegeben, welche Spalten unseres Datensatzes auf welchen Achsen angezeigt werden sollen. Ein letztes Mal: Denk an die Anführungsstriche, da es sich hier um Spaltennamen handelt! Das Endergebnis sieht wie folgt aus:

```
px.line(dataGermany, x = 'year', y = 'lifeExp')
```

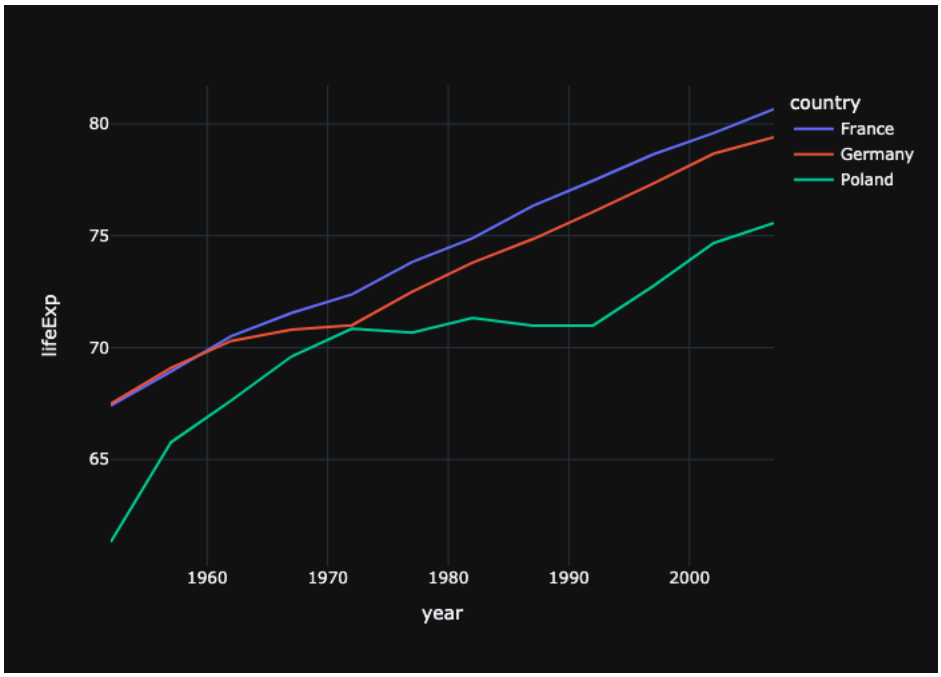


Einblendung Liniendiagramm

Wir können auch mehrere Linien in einem Diagramm anzeigen. Wenn wir die Lebenserwartungen von Deutschland, Frankreich und Polen vergleichen wollen, geht das z. B. folgendermaßen. Um die Länder auseinanderzuhalten, kann die Linienfarbe auch hier auf ein Feature gesetzt werden. Das bewirkt, dass alle Datenpunkte, die für dieses Feature einen gleichen Wert haben, im Diagramm auch die gleiche Farbe zugewiesen bekommen. Eine Legende mit der Farbzuordnung wird automatisch mit erstellt. Durch Klicken auf einen Eintrag in der Legende können einzelne Länder auch ausgeblendet werden.

CC BY

```
dataDeFrPo = data.query("country in ['Germany', 'France', 'Poland']")
px.line(dataDeFrPo, x = 'year', y= 'lifeExp', color = 'country')
```



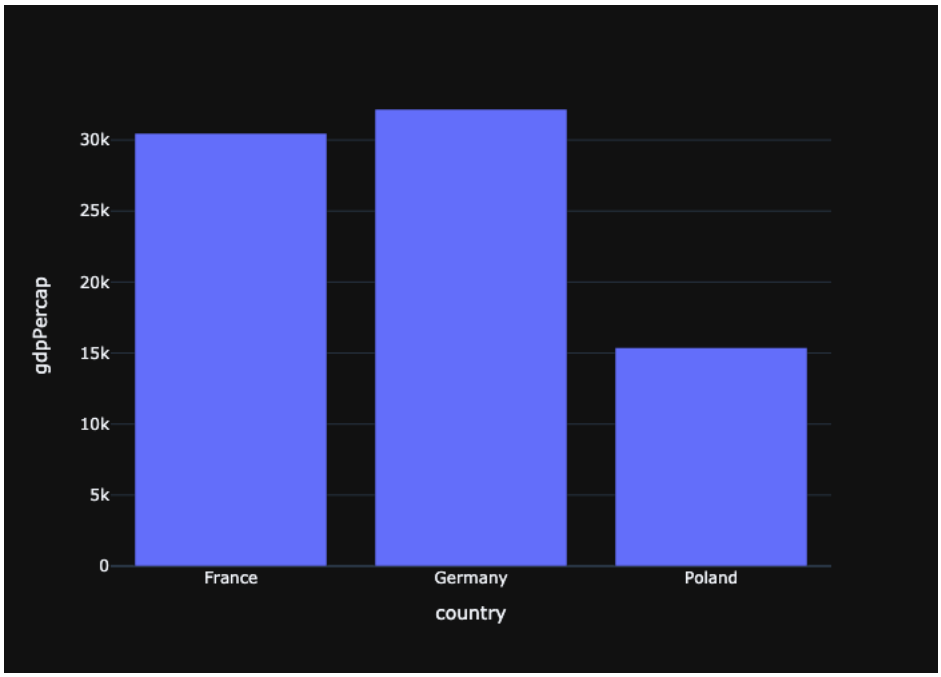
Einblendung Liniendiagramm

Balken- und Säulendiagramme

Als Nächstes beschäftigen wir uns mit Säulendiagrammen. In Plotly Express kannst du Säulendiagramme mit dem Befehl *bar* erstellen. Wenn wir das Bruttoinlandsprodukt des Jahres 2007 von Deutschland, Frankreich und Polen vergleichen wollen, nutzen wir dafür z. B. die folgende Zeile:

```
px.bar(dataDeFrPo.query("year == 2007"), x = 'country', y = 'gdpPercap')
```

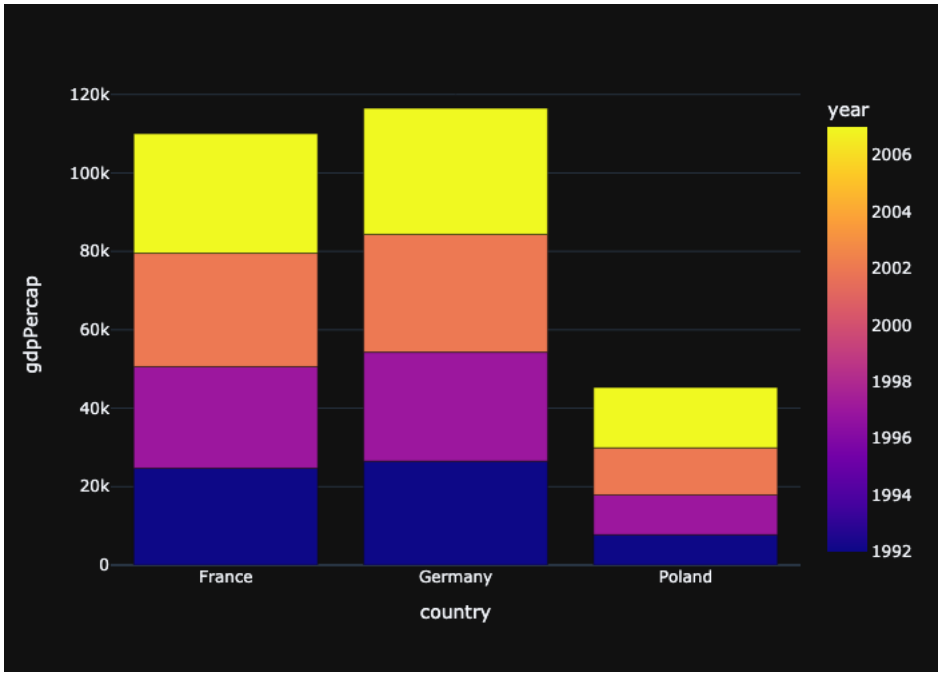

CC BY



Einblendung Säulendiagramm

Wir können die Werte auch stapeln, also übereinander darstellen, indem wir mehr Jahre zulassen. Hier bietet es sich an, die Jahre farblich voneinander zu trennen.

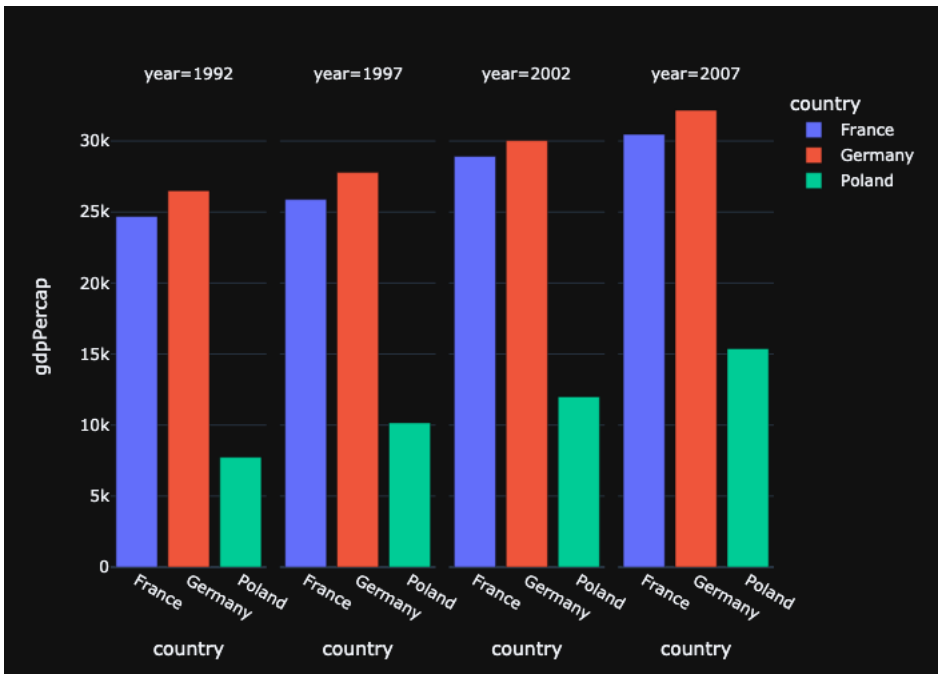
```
px.bar(dataDeFrPo.query("year >= 1990"), x = 'country', y = 'gdpPercap',
             color = 'year')
```



Einblendung Säulendiagramm

Wie du sehen kannst, gibt der höchste Punkt das addierte BIP aller dargestellten Jahre an. Allerdings eignet sich diese Darstellung in diesem Zusammenhang nicht wirklich, um Vergleiche der einzelnen Jahre zwischen diesen Ländern anzustellen. Der Parameter *facet_col* ermöglicht es, die Balken auch nach Jahren getrennt anzugeben. Das Resultat siehst du hier.

```
px.bar(dataDeFrPo.query("year > 1990"), x = 'country', y = 'gdpPercap',
             color = 'year', facet_col = 'year')
```

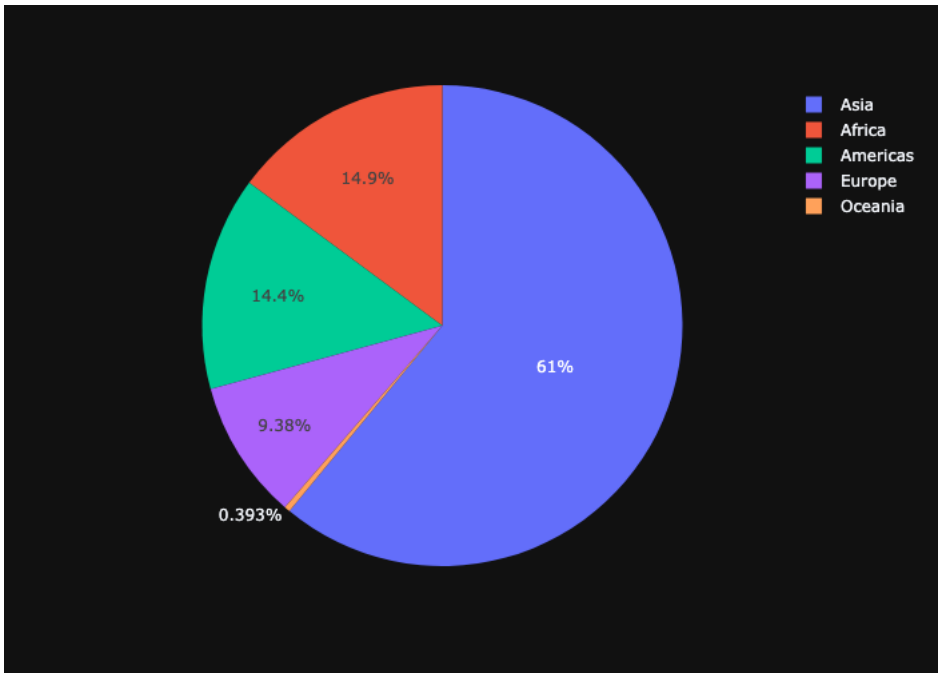


Einblendung Säulendiagramm

Kuchendiagramme

Plotly Express kann unter anderem auch Kuchendiagramme darstellen. Der Befehl dafür lautet *pie*, für das englische pie chart. Hier geben wir mit *values* an, über welche Werte die Verteilung erstellt werden soll, und mit *names*, wonach kategorisiert werden soll. Das folgende Beispiel erstellt ein Diagramm, bei dem jedes Kuchenstück angibt, wie groß der Anteil eines Kontinents an der Gesamtbevölkerung ist. Wir benutzen die Zahlen aus dem Jahr 2007.

```
px.pie(data.query("year == 2007"), values = "pop", names = "continent")
```



Einblendung Kuchendiagramm

Diagramme kombinieren

Diagramme können auch kombiniert werden. Sinnvolle Kombinationen sind z. B. zwei Liniendiagramme oder ein Liniendiagramm und ein Scatter-Plot. Erstellen wir also zuerst zwei Diagramme namens *fig1* und *fig2*.

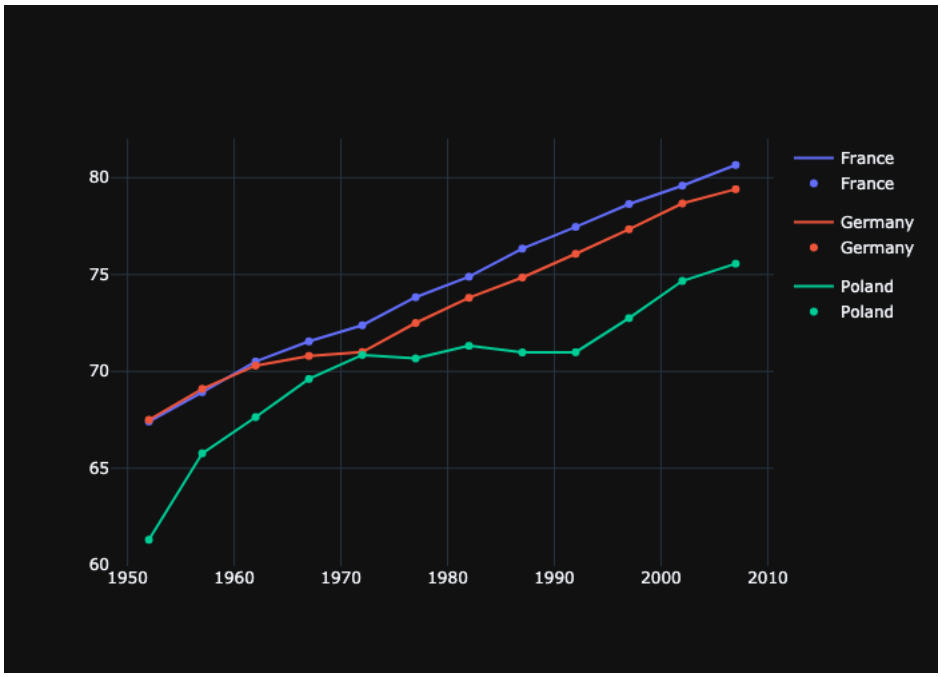
```
fig1 = px.line(dataDeFrPo, x = 'year', y= 'lifeExp', color = 'country')
fig2 = px.scatter(dataDeFrPo, x = 'year', y= 'lifeExp', color = 'country')
```

Der Einfachheit halber nehmen wir als fig1 das Liniendiagramm, bei dem die Lebenserwartung von Deutschland, Polen, und Frankreich verglichen wird, und erzeugen als fig2 noch den dazugehörigen Scatter-Plot dazu.
Kombinieren funktioniert jetzt folgendermaßen: Du benötigst das Modul *plotly.graph_objects*, was häufig mit *go* abgekürzt wird. Um zwei Diagramme zu kombinieren, kannst du den folgenden Befehl verwenden.

```
import plotly.graph_objects as go
fig = go.Figure(fig1.data + fig2.data)
fig.show()
```

Dabei werden alle Daten von *fig1* und *fig2* inklusive der Layoutoptionen genommen und mit dem + verknüpft. Der *show*-Befehl zeigt einfach nur ein Diagramm an, das wir zuvor in einer Variable gespeichert haben. Das Resultat sieht so aus.

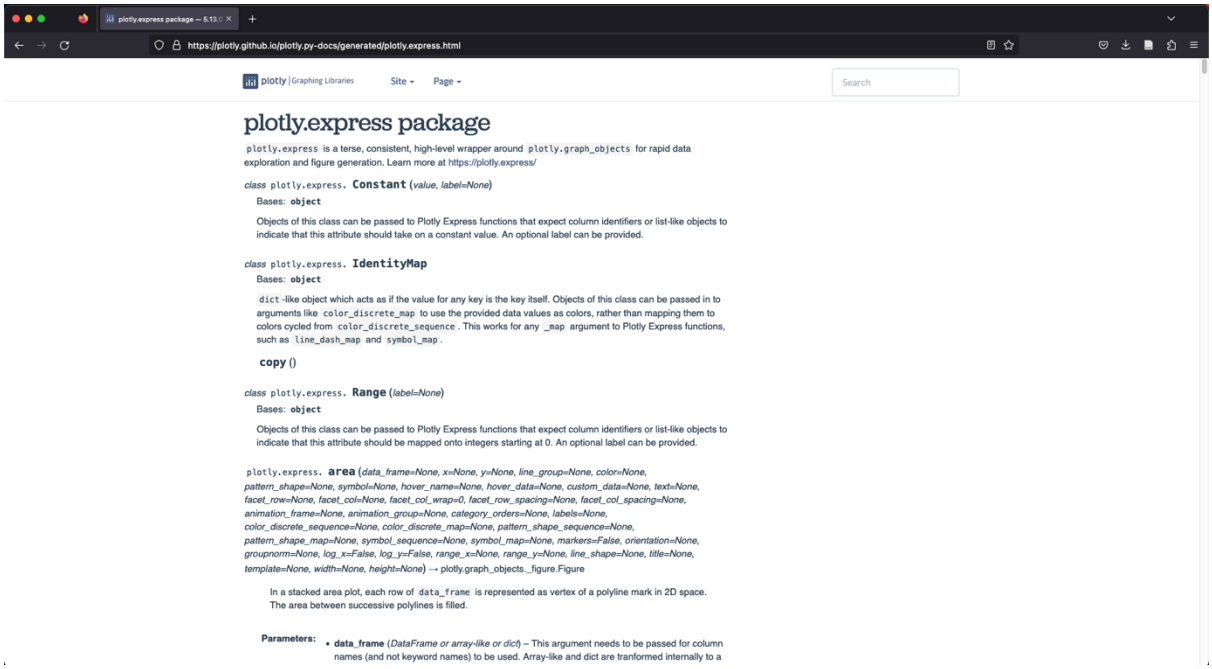
CC BY



Einblendung kombiniertes Diagramm

Handbuch

Natürlich musst du nicht alle gegebenen Befehle und Parameter auswendig können. Dafür gibt es zu jedem Modul ein Handbuch, das die enthaltenen Funktionen und dazugehörigen Parameter erläutert. Das Handbuch für Plotly Express sieht so aus:



CC BY

Einblendung Handbuch (Quelle [2])

Wir haben in diesem Video nur rudimentäre Diagramme erstellt. Es gibt sehr viele Gestaltungsmöglichkeiten, was Farben, Formen, Beschriftungen usw. angeht. Auch hier wirst du sicher im Handbuch fündig!
Sobald du deine Grafik fertiggestellt hast, willst du sie sicher auch außerhalb eines Jupyter-Notebooks verwenden. Zum externen Speichern musst du das Modul Kaleido installiert haben. Du kannst dann auf folgende Weise deine Diagramme speichern. *fig* bezeichnet dabei deine zu speichernde Grafik. Der Befehl *write_image(Pfad)* speichert eine Grafik statisch ab, wohingegen bzw. *write_html(Pfad)* eine html-Datei erzeugt und damit die Interaktivität erhält. Der Pfad ist dabei ein String, wo das Bild gespeichert werden soll, inklusive Dateiname und -endung. Willst du also z. B. das eben erstellte kombinierte Diagramm namens *fig* unter dem Namen *plot* in demselben Ordner speichern, in dem auch das hier gezeigte Notebook liegt, verwendest du diese Befehle.

```
fig.write_image("plot.png")
fig.write_html("plot.html")
```

Wie du siehst, wurden beide Dateien erzeugt.

Einblendung Notebook-Dateien

Abschluss

In diesem Video hast du gelernt, mit dem Modul Plotly Express kleine Visualisierungen zu erstellen und selbstständig mithilfe des Handbuchs deine Visualisierungen weiterzuentwickeln.

Quellen

Quelle [1] https://gapminder.org

Quelle [2] https://plotly.github.io/plotly.py-docs/generated/plotly.express.html

Weiterführendes Material

https://plotly.github.io/plotly.py-docs/generated/plotly.express.html

https://plotly.com/python/plotly-express/

Disclaimer

Transkript zu dem Video „Woche 05 Programmierung: Visualisierung“, Ann-Kathrin Selker. Dieses Transkript wurde im Rahmen des Projekts ai4all des Heine Center for Artificial Intelligence and Data Science (HeiCAD) an der Heinrich-Heine-Universität Düsseldorf unter der Creative Commons Lizenz CC-BY 4.0 veröffentlicht. Ausgenommen von der Lizenz sind die verwendeten Logos, alle in den Quellen ausgewiesenen Fremdmaterialien sowie alle als Quellen gekennzeichneten Elemente.