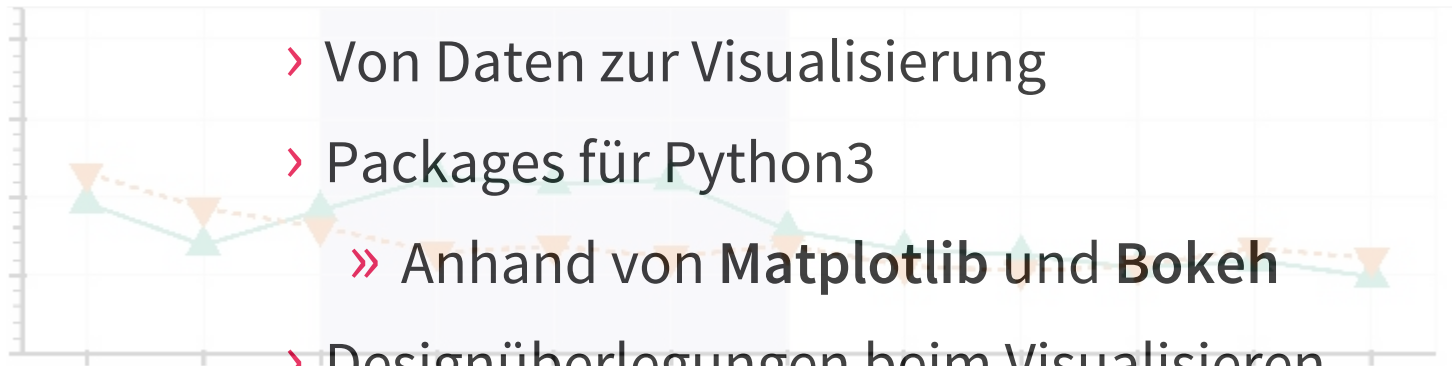


# 706.088 INFORMATIK 1

# DATEN VISUALISIEREN MIT PYTHON

## INHALT

- › Motivation
- › NumPy Wiederholung
- › Von Daten zur Visualisierung
- › Packages für Python3
  - » Anhand von **Matplotlib** und **Bokeh**
- › Designüberlegungen beim Visualisieren



# MOTIVATION

Was ist Datenvisualisierung?

- › Visuelle Representation von Daten
- › Unterstützung von geschriebenen Argumenten

Warum visualisieren wir Daten?

## Warum visualisieren wir Daten?

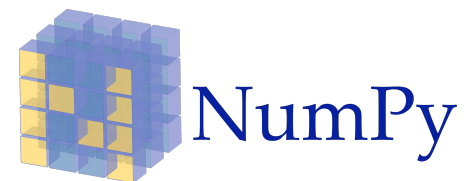
*“Never trust a diagram that you have not faked yourself.”*

(<http://www.tylervigen.com/spurious-correlations>)

# GRUNDLAGEN

- › Warum visualisieren wir Daten?
  - » Um etwas über die Daten zu lernen
  - » Um Anderen die Ergebnisse und Daten kompakt zu präsentieren
  - » Um ein Gefühl im Umgang mit visuellen Datenrepräsentationen zu bekommen


2.4



# NUMPY WIEDERHOLUNG

```
import numpy as np
```

"MATLAB in Python"

- › Mathematische Funktionalitäten für Python
- › Arrays und Matrizen
  - » Elementweise Operationen
- › Hoch performant
  - » Unterbau in C und C++
- › Ausführliche Dokumentation 

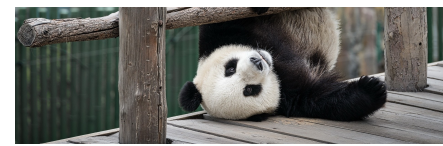
# NUMPY ARRAYS

- › Grundlegendes Konzept von Numpy
  - › Sind als Vektoren anzusehen
  - › Arrays haben einen fixen Datentyp (int, float, ...)
  - › Komplexe array (mit dtype Object) sind möglich
  - › Normale Operationen (+, -, \*, ...) Elementweise
- › Numpy arrays sind typischer Datentyp beim Plotten





# PANDAS



```
import pandas as pd
```

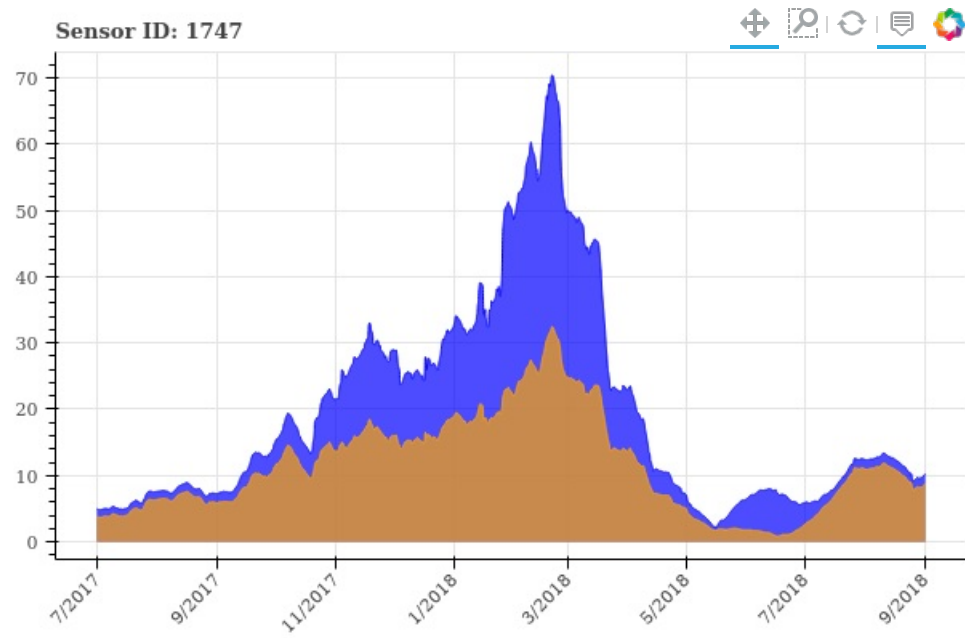
- › Mächtiges Package auf NumPy aufbauend
- › **Dataframe** als grundlegender Datentyp
  - » Simpel gesagt "Dictionary aus numpy arrays"
- › Inkludiert sehr viel Funktionalität für Analyse großer Datenmengen
- › Inkludiert Visualisierungstools (Auf Basis von Matplotlib)
  - » Nur Basisfunktionalitäten simpel anwendbar
- › Moderne Visualisierungspackages verwenden Dataframes als Input
- › Kann direkt mit CSV (und anderen Dateitypen) umgehen



# VON DATEN ZUR VISUALISIERUNG

	sensor_id	sensor_type	location	lat	lon	timestamp	P1	
0	1503	SDS011	743	47.08	15.49	2017-07-01 00:00:06	2.60	2
1	1693	SDS011	844	47.06	15.47	2017-07-01 00:00:11	2.40	2
2	2910	SDS011	1465	47.09	15.44	2017-07-01 00:00:25	2.55	2
3	1438	SDS011	713	47.08	15.43	2017-07-01 00:01:02	5.10	3
4	1747	SDS011	871	47.03	15.39	2017-07-01 00:01:04	2.60	2
5	3211	SDS011	1618	47.06	15.47	2017-07-01 00:01:14	2.13	1
6	2782	SDS011	1399	47.07	15.45	2017-07-01 00:01:18	3.92	3
7	3667	SDS011	1850	47.10	15.44	2017-07-01 00:01:37	4.20	2
8	834	SDS011	399	47.05	15.44	2017-07-01 00:02:16	4.93	1



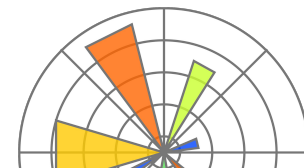


# GRUNDSÄTZLICHE VORGANGSWEISE

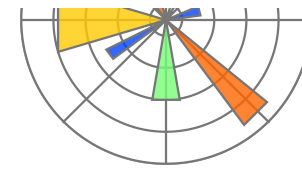
1. Rohdaten
  - › Filtern, Bereinigen, Anonymisieren?
2. Visualisierungsart festlegen
  - › z.B., Linien, Heatmaps, Boxplots
3. Dokumentation öffnen!
4. Visualisierung erstellen
  - › Stil Anpassen
5. Prüfen ob die Visualisierung die gewollte Aussage unterstützt

# VISUALISIERUNG UND PYTHON

- › Python hat keinen nativen Support
  - ›› Externe Packages füllen diese Lücke
  - ›› Große Unterschiede zwischen Web und Print



# MATPLOTLIB



```
from matplotlib import pyplot as plt
```

- › Eines der ältesten Package
  - » Erstellt Rastergrafiken (.jpg, .png) oder Vektorgrafiken (.pdf)
- › Weit verbreitet und gut supportet
- › Simple Anwendung
  - » Viele Plottypen nativ implementiert
- › Großes Update in 2018
  - » Plots sehen nun besser aus!
- › Viele gute Erweiterungen (z.B., **seaborn**)
  - » Neue Plottypen und Anbindung an Pandas





# GRUNDLAGEN

- › Zwei Anwendungswege
  - » Objektorientiert
  - » State Machine
  - » (der schlechtere Weg!)

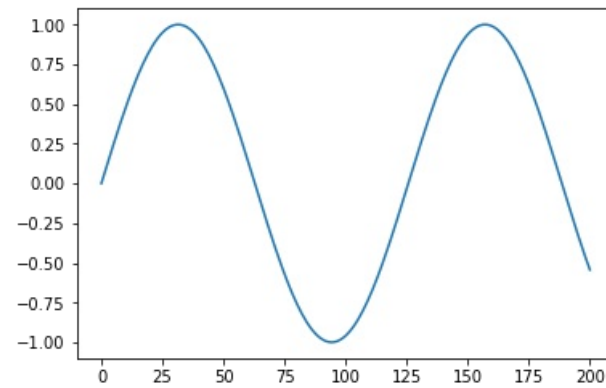
# BASISOBJEKTE

```
figure, axis = plt.subplots()
```

- › figure
  - » Containerobjekt für gesamte Visualisierung
- › axis
  - » Beinhaltet einen Plot
  - » Z.B.: Linenplot oder Bar Chart
- › Figure besteht oft aus mehreren Achsen
  - » Teilen der X oder Y Dimension
- › Speichern über Figure und nicht über Axis
  - » `figure.savefig()`

# FIRST STEPS

```
data = np.sin(np.linspace(0,10,201))  
plt.plot(data) # plot() is generally a line plot
```




# ACHSEN UND LABELS

Visualisierungen ohne Beschriftungen sind wertlos!

- › Jedes `axis` Objekt kann eine Beschriftung auf X und Y Achse haben
  - » `axis.set_xlabel("<text>")`
  - » `axis.set_ylabel("<text>")`
- › Unterstützt Mathematische Symbole
  - » Via LaTeX encoding (`$<symbol>$`)
  - » Z.B.:  $\lambda = \$\backslash\lambda\$$

# MARKER


- › Im `axis.plot()` Aufruf definiert
- › Markieren die einzelnen Werte bei Linienplots
  - » Hervorheben einzelner Werte
  - » Z.B., Monatsmarker auf Zeitserie
- › Helfen mehrere Linien zu unterscheiden
- › Viele mögliche Zeichen bei Matplotlib ()

```
marker="<symbol>",  
markersize=<size>,  
markeredgecolor=<color of outline>,  
markerfacecolor=<fill color>
```

# LEGENDEN

- › Um jedem Plot eine Bezeichnung zu geben
- › Via `label="<name>` im `plot()` Aufruf
- › Zeigt Linientyp und Name an
  - » Linientyp von `color` und `marker` übernommen
- › Platzierung beachten
  - » Sollte nicht über Linien stehen
  - » Default von Matplotlib ist meist ausreichend
  - » Platzierung außerhalb des Plots umständlich
  - » Angezeigt erst nach `axis.legend()` Befehl

# ANNOTIERUNGEN

- › `axis.annotate()` 
- › Text und Marker (Pfeile) am Plot
  - » Auf Besonderheiten Hinweisen
- › Z-Indices beachten
  - » Annotierungen immer am Ende plotten.

## MEHRERE PLOTS AUF EINER ACHSE

- › Mehrfacher Aufruf von `axis.plot()`
- › Übersichtlichkeit beachten
  - » Mithilfe von Farben, Markern und Legende

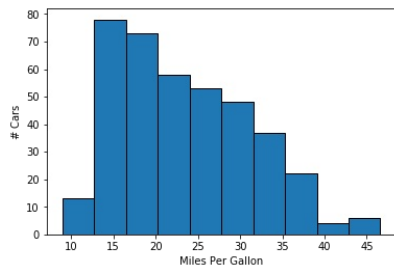


# SPEICHERN

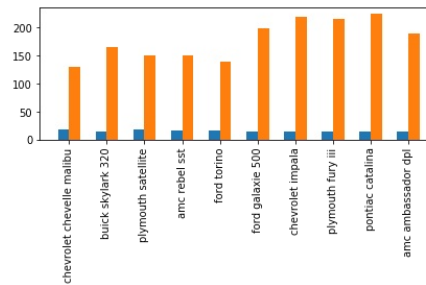
```
figure.savefig("<path/name/ext>", transparent=True, bbox_inches="tight")
```

- › Immer über `figure`- Objekt`
- › Dateityp über Dateiname definiert
  - » Normalerweise `.png` oder `.pdf`
- › Schnelle outputs über Rastergrafiken (`.png`)
- › Publizieren immer via Vektorgrafik (`.pdf`)

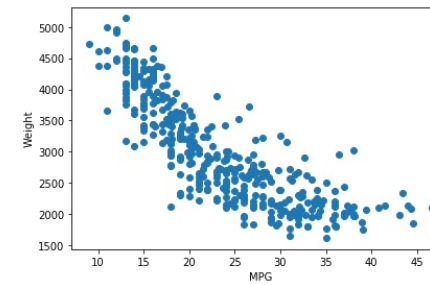
# WEITERE ÜBLICHE PLOTTYPEN



Histogramme



Bar Charts



Scatterplots





- › Sehr moderne Library
  - » v1.0 Oktober 2018
- › Fokus auf Web
  - » Via Overlays und Interaktivität
  - » Erstellt Canvas oder SVG Objekte (.html Output)
- › Nativ sehr gute Plot Designs
- › Anwendung komplizierter
  - » Dokumentation umständlich
  - » Viele übliche Plots per Hand
  - » Z.B., Box Plots über positionierte Rechtecke
- › PDF-Konvertierung nur über externe Programme



# JUPYTER NOTEBOOKS

- › Müssen manuell geladen werden
  - » `output_notebook()` lädt die JavaScript Library
  - » `show(plot)` rendert das Bild im Browser

```
from bokeh.io import output_notebook, show
output_notebook()
```

# DATA SOURCES

```
from bokeh.models import ColumnDataSource  
source = ColumnDataSource(pandas_dataframe)
```

- › Datenquelle für Bokeh Plots
  - » Nicht verpflichtend aber sehr hilfreich!
- › Ähnlich zu Pandas Dataframes
  - » Können direkt konvertiert werden
- › plot via Labels und source Angabe

```
# Example where we have a list of timestamps and a list of values "P"  
p1_line = plot.line(x="timestamp", y="P1", source=source)
```

# TOOLBOX

- › Viele hilfreiche Tools implementiert
  - » zoom, drag, save, ...
  - » Ohne Extraaufwand für Programmierer

# HOVERTOOL

- › Zeigt Daten bei mouseover an
- › Manuelle Erstellung
  - » Einbinden via `tools=[]
  - » Restliche Tools manuell hinzufügen!

```
from bokeh.models import HoverTool
# {} defines our formatting. Somewhat similar to formatstrings
hover = HoverTool(tooltips=[
   ("<display_name_A>", "@<field_A>{%F}"),          # use custom format
   ("<display_name_B>", "@<field_B>{0,0.000}"),      # format as float
],
    formatters={'field_B': 'datetime'}, # custom formatter
    mode="vline")
```



# SPEICHERN

```
from bokeh.io import output_file  
output_file("<filename.html>")
```

- › Bokeh Output ist HTML
  - » Browser is Renderer
- › Benutzer kann Bild manuell speichern
  - » Via save tool
  - » Default als .png

# KONVERTIERUNG ZU PDF

1. Bokeh Backend umstellen

› `plot.output_backend = "svg"`

2. Öffnen via externen Editor

› Z.B., **InkScape**

3. Als PDF speichern

# DESIGNENTSCHEIDUNGEN

- › Erstellen einer Checkliste
  - » Annotierungen und Labels
  - » Größen von Plot und Text
  - » Dimension der Daten
  - » Skalierung
  - » Limits for X und Y
  - » Farbwahl und Marker

# FARBEN UND MARKER

- › Korrekte Wahl ist wichtig
- › Quantitative oder qualitative Daten
- › Beschränkte Auswahl
  - » Photokopierer
  - » Schlechte / Schwarz-Weiß Drucker
  - » Farbenblinde Menschen
- › Externe Tools helfen
  - » Z.B., **Colorbrewer**

*“matplotlib tries to make easy things easy  
and hard things possible”*

- `https://matplotlib.org/`