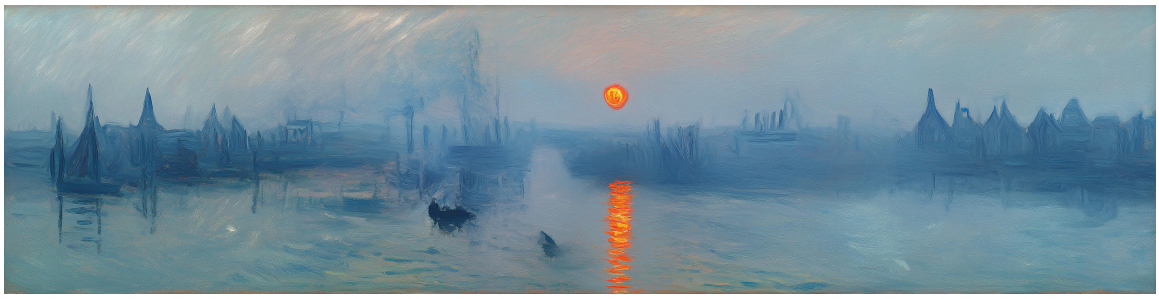
Visuelle Datenanalyse

Joern Ploennigs · AI4SC



> Die wahre Entdeckungsreise besteht nicht darin, neue Landschaften zu suchen, sondern darin, neue Augen zu haben.
>
> — Marcel Proust

Die Visualisierung von Daten ist oft einer der ersten Schritte in deren Analyse. Ziel ist es sich vertrauter mit den Daten zu machen und einfache Zusammenhänge visuell zu identifizieren. Dabei nutzen wir aus, dass der Mensch den Großteil seiner Informationen visuell aufnimmt und sehr gut in der Mustererkennung ist. Dadurch können wir schnell Muster zu erkennen, Ausreißer entdecken und geeignete Merkmale für ML-Modelle identifizieren.

Passende Diagramme

Die folgende Tabelle stellt passende Diagramme entsprechend der Zielstellungen und der Anzahl und des Datentyps der darzustellenden Variablen vor. Diese werden wir im weiteren Verlauf im Einzelnen diskutieren.

	Einzelne Variable		**Mehrere Variablen**		
Ziel	**Numerisch**	**Kategorisch**	**Numerisch**	**Kategorisch**	**Gemischt**
Trend	Line plot	Heatmap	Line plot	Heatmap	Heatmap
Saisonalität	Line plot	Heatmap	Line plot	Heatmap	Heatmap
Autokorrelation	Line plot	Heatmap	Line plot	Heatmap	Heatmap

	Einzelne Variable		**Mehrere Variablen**		
Kreuzkorrelation	Line plot	Heatmap	Scatter plot	Bar chart	Strip plot, Box plot
Verhältnisse	Area plot	Pie chart	Area plot	Stacked Bar chart	Strip plot
Verteilung	Histogramm	Pie chart	Density Heatmaps	Bar chart	Box plot, Violin plot
Streuung	Histogramm	Bar chart	Scatterplot	Bar chart	Strip plot

Wenn du z.B. den Temperaturverlauf auf einer Baustelle analysierst, eignet sich ein Lineplot. Wenn du verschiedene Lastarten auf Stützen einer Brücke vergleichen willst, kann ein Boxplot, Violinplot oder Bar Chart helfen. Zur Visualisierung eine Feuchtigkeitsverteilung in einer Wand ist ein Heatmap gut geeignet. Diese Visualisierungen sind auch vor dem Einsatz von Machine-Learning-Algorithmen entscheidend, um mögliche Probleme (z.B. ungleich verteilte Daten oder Korrelationen) zu erkennen.

Python hat viele Plotting-Bibliotheken um diese Diagramme zu erzeugen. Hier diskutieren wir eine der einfachsten namentlich *Plotly*. Plotly ist eine Open-Source-Bibliothek zur Datenvisualisierung, mit der sich interaktive, webbasierte Diagramme und Dashboards erstellen lassen. Sie ist eine beliebte Wahl für Python-Entwickler, da sie einfach zu bedienen ist und eine große Auswahl an Diagrammtypen und Anpassungsmöglichkeiten bietet.

Wir gehen davon aus, dass Sie die folgenden Module importiert haben:

```
import numpy as np # Import von NumPy
import pandas as pd # Import von Pandas
np.random.seed(10) # Fixierung des Zufallszahlgenerators zur beseren Erklärung
```

Plotly ist eine sehr beliebte und flexible Bibliothek, die wir als `px` importieren. Plotly Express enthält Funktionen, mit denen man sehr schnell ganze Diagramme auf einmal erstellen kann. Dabei nutzt Plotly Express intern alle Elemente aus Plotly, so dass die Diagramme mit den umfangreichen Funktionen von Plotly auch erweitert werden können. Genauso lassen sich die Plotly Express Diagramme auch durch Plotly erzeugen, jedoch mit 5- bis 100-mal mehr Code.

Plotly Express

Plotly Express eignet sich insbesondere für einfache und schnelle Datenexploration.

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

Als Beispiel für die Visualisierung laden den schon bekannten Wetterdatensatz des DWD für Warnemünde. Bevor man versucht Datensätze zu visualisieren, lohnt sich immer den Dataframe mit `head` anzusehen, so dass man sieht welche Spalten und Werte verfügbar sind.

```python
egywth = pd.read_csv("../data/UROS/Energy1D_weather_clean.csv",
parse_dates=[0])
egywth.head()
```

	EVDate_NT_HT_AE_Lab_DASIIAMESS_DATUM (overlapping headers)	QN_3	…	TMK	UPM	TXK	TNK	TGK	eor	Temperatur-Klasse	Heiz-Kuehl-Tage	QN / SQN (overlapping)	QNF_4
0	2020-12-30 00:00:00+00:00 NaN NaN NaN NaN NaN 2020-12-30 00:00:00+00:00 (overlapping: 4127, 10230.0)		…	4.0	81.0	4.6	2.9	2.2	eor	Cold	Heizgradtag	nicht alle Parameter korrigiert	5
1	2020-12-31 00:00:00+00:00 NaN NaN 125620 91.0 5.0 2020-12-31 00:00:00+00:00 (overlapping: 4127, 10231.0)		…	3.4	83.0	4.4	2.3	0.9	eor	Cold	Heizgradtag	nicht alle Parameter korrigiert	5
2	2021-01-01 00:00:00+00:00 (overlapping: 4108012212) 90.0 1.0 2021-01-01 00:00:00+00:00 (overlapping: 4127, 10101.0)		…	2.0	90.0	3.0	1.1	0.3	eor	Cold	Heizgradtag	nicht alle Parameter korrigiert	5

	EV_DATE_N4E_7AE_Lab_IDS_LAMESS_DATUM			QN_3	…	TMK	UPM	TXK	TNK	TGK	eor	Temperatur-Klasse	Heiz-Kuehl-Tage	QN_S	QN_4	F_4
3	2021-01-02 00:00:00+00:00	6301243284.0	2.0	2021-01-02 00:00:00+00:00	20210101.02.0…	3.3	95.0	4.0	2.6	2.0	eor	Cold	Heizgradtag	-	nicht alle Parameter korrigiert	5
4	2021-01-03 00:00:00+00:00	7501222383.0	2.0	2021-01-03 00:00:00+00:00	20210101.03.0…	3.5	81.0	4.6	2.4	0.8	eor	Cold	Heizgradtag	-	nicht alle Parameter korrigiert	5

Liniendiagramme (Line chart)

Ein Liniendiagramm ist eine x-y-Diagramm, das die Beziehung zwischen zwei quantitativen Variablen darstellt, wobei die Punkte, die zu derselben x-y-Gruppe gehören durch Linien verbunden werden.

Liniendiagramme eignen sich besonders zur Darstellung von Zeitreihen oder Sequenzen und erlauben die Erkennung von Trends (Korrelation), Autokorrelation, Saisonalitäten (Schwingungen) und die Autokorrelation.

Die Steigung der Linie im Mittel zeigt Trens auf, was ein Zeichen für die Korrelation zwischen x- und y-Werten ist.

- Steigt die Linie im Mittel so liegt ein Aufwärtstrend vor, das heißt die y-Werte sind positiv mit den x-Werten (z.B. der Zeit) korreliert.
- Fällt die Linie so liegt eine negative Korrelation vor, d.h. die y-Werte fallen mit steigenden x-Werten.

- Bleibt die Linie flach, so liegt kein Trend vor.

Als Saisonalitäten bezeichnet man sich wiederholende Muster oder Schwankungen. Sie deuten auf Frequenzabhängigkeit der y-Werten von den x-Werten hin. Es können dabei mehrere Frequenzen sichtbar sein. So findet man bei natürlichen Zeitreihen zum Beispiel Jahres oder Tagesschwankungen.

Eine hohe Autokorrelation erkennt daran das nachfolgende Werte sich nicht stark von den direkten Vorgängerwerten unterscheiden.

Wir können Liniendiagramme in Plotly mit `px.line` erstellen. Man gibt immer zuerst den Dataframe an und spezifiziert dann die Spaltenamen der Werte die als `x` und `y` verwendet werden sollen.

```
px.line(egywth, x="Date", y="TMK")
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

Für unser Beispiel können wir erstmal keinen klaren Aufwärtstrend erkennen. Stattdessen sehen wir die Saisonalitäten im Jahr sehr gut, also das Temperaturen im Winter niedriger und im Sommer höher sind. Wir erkennen auch, dass es Schwingungen im Wochentakt gibt, wo sich das Wetter langsam durch wechselnde Hoch- und Tiefeinflüsse ändert. Hätten wir Stundenwerte würden wir auch die Tagessaisonalität sehen, da es nachts kälter ist.

Für die Autokorrelation zoomen wir uns die letzten 30 Tage rein. Das können wir, indem wir mit der Maus in dem Diagramm oben über die letzten 30 Tage ziehen. Dann zoomt er in diesen Bereich (mit Doppelklick können wir wieder raus zoomen). Diese Interaktivität ist ein sehr großer Vorteil von Plotly.

Wir können in dem Liniendiagramm auch mehrere Zeitreihen gleichzeitig anzeigen, indem wir als y-Werte mehrere Spalten angeben. Wollen wir zum Beispiel neben dem Mittelwert, die maximale und minimale Temperatur anzeigen, so können wir das mit:

```
px.line(egywth, x="Date", y=["TNK", "TMK", "TXK"])
```

```
Unable to display output for mime type(s): text/html
```

Diese Darstellung ist etwas unübersichtlich. In solchen Fällen empfiehlt es sich die Zeitreihen nebeneinander in einzelnen Facetten darzustellen. Hierfür gibt es die Option `facet_row` (oder `facter_col`). Hierfür müssen wir den Datensatz in eine Reihendarstellung bringen, die wir im Abschnitt Dataframes Transformieren kennen gelernt haben.

```
import plotly.express as px
egywth_melt=egywth.melt(id_vars=['Date', "TemperaturKlasse"],
value_vars=["TXK","TMK","TNK"])
px.line(egywth_melt, x="Date", y="value", facet_row ="variable")
```

```
Unable to display output for mime type(s): text/html
```

Hier erkennen wir auch leichte Unterschiede zwischen dem minimal und maximal Werten (TNK, TXK) die stärker variieren als die mittlere Temperatur (TMK). Das ist auf die niedrigere Robustheit von Min und Max versus dem Mittelwert zurückzuführen, die wir im Statistikteil diskutiert haben.

In Facetten lassen sich auch Werte mit unterschiedlichen Skalen nebeneinander gut darstellen. Zum Beispiel die Temperatur mit dem Energieverbrauch „ES_Lab“, wo wir eine erkennbare Parallelität sehen, allerdings mit einer zeitlichen Verschiebung, da die Energie der Temperatur ca. 30 Tage voran schreitet.

```
egywth_melt=egywth.melt(id_vars=['Date', "TemperaturKlasse"],
value_vars=["TMK", "ES_Lab"])
px.line(egywth_melt, x="Date", y="value", facet_row
="variable").update_yaxes(matches=None)
```

```
Unable to display output for mime type(s): text/html
```

Flächendiagramme (Area chart)

Flächendiagramme sind Liniendiagrammen, bei denen die Fläche unter der Linie gefüllt ist. Sie sind dann sinnvoll, wenn es um die Visualisierung dieser Fläche geht, z.B. bei Integralen, Wahrscheinlichkeitsverteilungen. Zu beachten ist, dass die Flächen standartmäßig aufaddiert werden (stacking). Damit kann man zum Beispiel unterschiedliche Energieverbräuche gut visualisieren.

Flächendiagramme sind genauso wie Liniendiagramme zu benutzen.

```
px.area(egywth, x='Date', y=["EV_HT_740", "EV_NT_740", "E_AV_Lab", "E_SV_Lab",
"ES_Lab"])
```

```
Unable to display output for mime type(s): text/html
```

In dem Flächendiagramm erkennt man gut, dass „EV_HT_740“ am 18. Oktober 2022 mit auf „EV_NT_740“ aufgeschaltet wurde und danach 0 ist. Dadurch singt der Gesamtverbrauch (die Gesamtfläche) nicht. Weiter erkennen wir die Abfälle im Energieverbrauch zum Jahreswechsel. Auch fehlen Daten zwischen 11. Mai und 19. Mai 2022.

Heatmaps

Heatmaps eignen sich zur Visualisierung von kategorischen Zeitreihen. Hierbei geht es genauso um die Erkennung von Saisonalität, Korrelation und Trends. Sie können auch für numerische Zeitreihen verwendet werden, wenn insbesondere die Mustererkennung im Vordergrund steht.

Hierfür verwendet man in Plotly die Funktion `imshow`. Sie ist eigentlich dazu da 2D Bilder darzustellen und akzeptiert deshalb nur eine Matrix an Werten. Diese können wir aus dem Datensatz erzeugen durch Auswahl spezifischer Spalten und Transposition dieser

```
px.imshow(egywth[["EV_HT_740", "EV_NT_740", "E_AV_Lab", "E_SV_Lab",
"ES_Lab"]].T, x=egywth["Date"], color_continuous_scale='Viridis')
```

```
Unable to display output for mime type(s): text/html
```

Dadurch kann man nun gut die wöchentliche Saisonalität im Energieverbrauch sehen. Ferner erkennen wir, dass am Wochenende vor Oktober 2022 bereits, immer „EV_HT_740“ auf „EV_NT_740“ umgelegt wird.

Dies können wir auch für kategorische Variablen nutzen. Hierfür müssen wir allerdings diese in eine numerische Darstellung überführen. In dem Abschnitt Konvertierung kategorischer Variablen haben wir kennen gelernt, dass das mit der Funktion `factorize` geht.

```
TK_Fact, TK_Map = pd.factorize(egywth.TemperaturKlasse) # Wandel die
kategorische Variable in eine Numerische um
TK_Fact = np.array([TK_Fact]) # Wandel die Serie in eine Matrix um, die von
imshow erwartet wird
px.imshow(TK_Fact, x=egywth["Date"], width=1000, height=400,
color_continuous_scale='Viridis')
```

```
Unable to display output for mime type(s): text/html
```

Streudiagramme (Scatterplot)

Ein Streudiagramm oder Punktewolke ist ein Diagramm, bei dem eine numerische Variable `y` über einer numerischen Variable `x` als Punkt dargestellt wird.

Das Streudiagramm eignet sich besonders zur Visualisierung der Korrelation und deren Streuung zwischen beiden Variablen. Im Gegensatz zu Liniendiagrammen werden zwischen aufeinander folgenden Werten keine Linien gezogen. Dadurch blendet man saisonale Aspekte und Autokorrelation bewusst aus.

Scatterplots können genauso verwendet werden, wie Liniendiagramme in Plotly (Liniendiagramme sind ein Untertyp von Scatterplots). Prinzipiell können wir damit auch Zeitreihen der Temperatur visualisieren.

```
px.scatter(egywth, x="Date", y="TMK")
```

```
Unable to display output for mime type(s): text/html
```

Dabei zeigt sich, dass Saisonalität und Autokorrelation schlechter zu erkennen sind und die Streuung stärker betont wird.

Die Stärke von Scatterplots offenbart sich, wenn man damit keine Zeitreihen visualisiert, sondern numerische Variablen vergleicht. Vergleichen wir zum Beispiel die Entwicklung der minimalen und maximalen Temperatur im Vergleich zur mittleren Tagestemperatur.

```
px.scatter(egywth, x="TMK", y=["TNK", "TXK"], opacity=.2) # es empfieht sich
bei vielen Werten die opacity zu reduzieren
```

```
Unable to display output for mime type(s): text/html
```

Jetzt ist der wenig überraschende lineare Zusammenhang zwischen den Variablen gut zu erkennen.

Interessanter ist es, wenn wir uns den Zusammenhang zwischen der Temperatur und dem Dampfdruck ansehen, welcher mist wie stark Wassermoleküle versuchen, von einer Oberfläche in die Luft überzugehen.

```
px.scatter(egywth, x="TMK", y="VPM", opacity=.3)
```

```
Unable to display output for mime type(s): text/html
```

Hier sehen wir einen exponentiellen Zusammenhang. Der ist erklärbar, da an warmen Tagen, die Wassermoleküle mehr Energie haben und somit eher verdampfen. Jetzt könnte man annehmen, dass deshalb an warmen Tagen auch die Luftfeuchtigkeit höher ist. Wenn wir nachschauen, sehen wir, dass sich diese Annahme in unserem Datensatz nicht bestätigt wird. Zur Verdeutlichung nutzen wir die kategorische Variable `TemperaturKlasse`

```
px.scatter(egywth, x="TMK", y="UPM", opacity=.2)
```

```
Unable to display output for mime type(s): text/html
```

Betrachten wir den zuvor entdeckten Zusammenhang zwischen der Temperatur und der Energie „ES_Lab“ ansehen. Hier sehen wir, dass der im Linienplot sichtbare Zusammenhang nicht mehr so klar zu erkennen ist, das liegt an der zuvor diskutierten zeitlichen Verschiebung. So eine Verschiebung ist beim Trainieren von ML-Modellen später problematisch, wenn sie nicht adressiert wird, weil das Modell diesen Zusammenhang nicht erkennen kann.

```
px.scatter(egywth, x="TMK", y="ES_Lab", opacity=.2)
```

```
Unable to display output for mime type(s): text/html
```

All diese Untersuchungen macht man meist gemeinsam, indem man ein Streudiagrammatrix über alle numerischen Daten in einem Datensatz plottet. Hierfür bietet Plotly die Funktion

```
px.scatter_matrix(egywth, dimensions=["TMK", "VPM", "UPM", "ES_Lab"],
opacity=.1)
```

```
Unable to display output for mime type(s): text/html
```

Hier sehen wir in der Diagonalen immer einen linearen Zusammenhang, da die Werte der gleichen Variable auf sich selbst abgebildet werden. Interessant sind die obere oder die untere Dreiecksmatrix, wo man nach lernbaren Zusammenhängen sucht.

Balkendiagramme

Balkendiagramm verwendet man zur Darstellung von kategorischen Werten. Hierbei werden auf der x-Achse die Kategorien gelistet und darüber auf der y-Achse ein rechteckiger Balken, der die Häufigkeit, Proportionen oder andere Werte für jede Kategorien darstellt.

Balkendiagramme eignen sich besonders gut, um Häufigkeiten von Kategorien und Kreuzkombination dieser zu vergleichen.

Plotly bietet hierfür die Funktion `px.bar`. Man kann sie einfach mit einer Kategorie aufrufen, allerdings ist das Ergebnis nicht sehr lesbar, weil Plotly die Häufigkeit nicht aufaddiert.

```
px.bar(egywth, x='TemperaturKlasse')
```

```
Unable to display output for mime type(s): text/html
```

Deshalb müssen wir die Anzahl vorher zählen, was wir mit einer Gruppierung via `groupby` und der Zählung via `count` der Spalte bestimmen können.

```
egywth_tmpcls=egywth[["Date","TemperaturKlasse"]].groupby("TemperaturKlasse").count().reset_index
egywth_tmpcls
```

	TemperaturKlasse	Date
0	Cold	659
1	Cool	260
2	Mild	170

	TemperaturKlasse	Date
3	Warm	9

```
px.bar(egywth_tmpcls, x='TemperaturKlasse', y="Date")
```

```
Unable to display output for mime type(s): text/html
```

Balkendiagramme eignen sich auch um Kreuzkombination von Kategorien zu untersuchen. Zum Beispiel wollen wir wissen in welchen Monaten welche Temperaturklassen auftreten. Dafür erzeugen wir uns eine Spalte Monat und beziehen diese in unsere Aggregation mit ein.

```
egywth["Monat"]=egywth.Date.dt.month

egywth_tmpclsM=egywth[["Date","TemperaturKlasse","Monat"]].groupby(["TemperaturKlasse","Monat"]).
egywth_tmpclsM.head()
```

	TemperaturKlasse	Monat	Date
0	Cold	1	93
1	Cold	2	83
2	Cold	3	93
3	Cold	4	89
4	Cold	5	54

Anstatt dass wir jetzt jede Kombination zusammenführen müssen, können wir die neue Spalte einfach als `color` Parameter in dem Balkendiagramm mit angeben. Plotly coloriert jetzt automatisch die unterschiedlichen Kategorien

```
px.bar(egywth_tmpclsM, x='TemperaturKlasse', y="Date", color="Monat") # durch
setzen von color erhalten wir auch farbliche kategorien
```

```
Unable to display output for mime type(s): text/html
```

```
egywth_tmpclsM=egywth[["Date","HeizKuehlTage","TemperaturKlasse"]].groupby(["TemperaturKlasse","H
egywth_tmpclsM.head()
```

	TemperaturKlasse	HeizKuehlTage	Date
0	Cold	Heizgradtag	659
1	Cool	Heizgradtag	81

	TemperaturKlasse	HeizKuehlTage	Date
2	Cool	Normaltag	179
3	Mild	Kühlgradtag	22
4	Mild	Normaltag	148

```
px.bar(egywth_tmpclsM, x='TemperaturKlasse', y="Date", color="HeizKuehlTage")
# durch setzen von color erhalten wir auch farbliche kategorien
```

```
Unable to display output for mime type(s): text/html
```

Tortendiagramme

Tortendiagramme sind eine Alternative zu Balkendiagrammen die insbesondere Verwendet werden, wenn die Proportionen zwischen den Werten wichtig sind. Sie erlauben allerdings nicht Kreuzkombinationen von Kategorien darzustellen.

```
px.pie(egywth_tmpcls, names='TemperaturKlasse', values="Date")
```

```
Unable to display output for mime type(s): text/html
```

Histogramme

Histogramme sind eine besondere Art von Balkendiagramm, wobei die Länge jedes Balkens auf der y-Achse die Häufigkeit oder den Anteil der Datenpunkte in einem bestimmten Bereich oder Intervall auf der x-Achse darstellen. Sie werden benutzt, um die Verteilungsfunktion einer Variable zu analysieren.

Als erstes schauen wir uns die Verteilung des Luftdrucks.

```
px.histogram(egywth, x="PM")
```

```
Unable to display output for mime type(s): text/html
```

Wir sehen hier schön ausgeprägte Normalverteilung, die leicht Linkslastig ist.

Bei der Temperatur ergibt sich eine bimodale Verteilung, was eine Überlagerung zweier annähernd normaler Teilverteilungen ist.

```
px.histogram(egywth, x="TMK")
```

```
Unable to display output for mime type(s): text/html
```

Der Energieverbrauch „ES_Lab“ ist eher Gammaverteilt, da er oft 0 ist, wenn keine Last anliegt.

```
px.histogram(egywth, x="ES_Lab")
```

```
Unable to display output for mime type(s): text/html
```

Auch hier können wir mehrere Spalten angeben, um die Histogramme zu vergleichen. So sehen wir zum Beispiel, dass „E_AV_Lab“ und „E_SV_Lab“ beide Normalverteilt sind. Hierfür erhöhen wir die Anzahl der Bins mit `nbins`.

```
px.histogram(egywth, x=["E_AV_Lab", "E_SV_Lab"], nbins=200)
```

```
Unable to display output for mime type(s): text/html
```

Boxplots

Boxplots werden verwendet, um die Verteilungsform von Variablen in der Abhängigkeit von anderen meist kategorischen Variablen zu analysieren. In diesen Fällen werden Histogramme zu unübersichtlich und man reduziert das Histogramm auf die wichtigen statistischen Kenngrößen und zeigt diese in Form einer Box.

Um diese zu verstehen, visualisieren wir den Boxplot mal direkt über dem Histogramm des Luftdrucks.

```
px.histogram(egywth, x="PM", marginal="box")
```

```
Unable to display output for mime type(s): text/html
```

Schauen wir uns den Aufbau genauer an:

- Die Box: erstreckt sich von Quartil 1 bis Quartil 3, die wir aus dem Teil der Statistische Datenanalyse kennen gelernt. Das Quartil 1 (Q1) ist der Wert, unterhalb dessen sich 25% der Datenpunkte befinden, während Quartil 3 (Q3) der Wert ist, unterhalb dessen sich 75% der Datenpunkte befinden. Eine breite Box deutet an, dass die Werte sehr breit streuen.
- Der Median wird durch einen Strich innerhalb der Box dargestellt. Es ist der Wert, der die Daten in zwei Hälften teilt, wobei 50% der Datenpunkte unterhalb und 50% oberhalb dieses Wertes liegen. Ist der Strich näher an der oberen Grenze der Box, ist die Verteilung rechts-schief, liegt er unten, ist sie links-schief verteilt.
- Die Antennen (Whisker): Die Linien, die von der Box abgehen, heißen Whisker. Sie zeigen die Spannweite der Daten, die noch nicht als Ausreißer gelten. Ihr Ende markiert in der Regel den größten Wert, der innerhalb von 1,5-fachen des Interquartilsabstands (Differenz zwischen Q3 und Q1) vom Q1 bzw. Q3 entfernt liegt.
- Ausreißer: Datenpunkte, die außerhalb der 1,5-fachen Interquartilsabstands-Grenze liegen, werden als kleine Kreise oder Sterne oberhalb oder unterhalb der Whisker dargestellt.

Mit einem Boxplot kannst du also auf einen Blick erkennen:

- Wie die Daten verteilt sind (symmetrisch, links- oder rechts-schiefe Verteilung)
- Wo der Median liegt (zentraler Wert)
- Wie groß die Streuung der Daten im mittleren Bereich ist (Länge der Box)
- Ob es Ausreißer gibt (Kreise oder Sterne außerhalb der Whisker)

Boxplots sind besonders praktisch, wenn du mehrere Datensätze miteinander vergleichen werden. Durch das Nebeneinanderstellen mehrerer Boxplots kann man schnell erkennen, in welchem Datensatz die Verteilung am meisten gestreut ist, wo die Medianwerte liegen oder ob es in einem Datensatz mehr Ausreißer gibt.

Analysieren wir noch einmal den Zusammenhang zwischen dem Energieverbrauch „ES_Lab“ und den Temperaturkategorien, so sehen wir wieder das bereits im Scatterplot beobachtete Verhalten.

```
px.box(egywth, x="TemperaturKlasse", y="ES_Lab")
```

```
Unable to display output for mime type(s): text/html
```

Boxplots erlauben es dabei vor allem mehrere Kreuzkombinationen von kategorischen Variablen zu visualisieren. Zum Beispiel wollen wir wissen wie der Energieverbrauch in welchen Monaten und Temperaturklassen verteilt.

```
px.box(egywth, x="TemperaturKlasse", y="ES_Lab", color="Monat")
```

```
Unable to display output for mime type(s): text/html
```

Violindiagramme

Violindiagramme sind eine Alternative zu Boxplots bei denen die Verteilungsdichte aus dem Histogramm anstatt der Box anzeigt wird, was meist einer Violine gleicht.

```
px.violin(egywth, x="TemperaturKlasse", y="ES_Lab")
```

```
Unable to display output for mime type(s): text/html
```

Streudiagramme (Strip)

Eine weitere Alternative zu Boxplots sind Streudiagramme. Sie kombinieren die Idee von Scatterplots mit dem des Violinplots und ergeben ein Scatterplot mit kategorischen Variablen. Hierbei werden die Variablenwerte über den Kategorien so aufgetragen, dass sie sich minimal in `x` überlagern.

```
px.strip(egywth, x="TemperaturKlasse", y="ES_Lab")
```

```
Unable to display output for mime type(s): text/html
```

Plots formatieren und abspeichern

Plotly bietet viele Funktionen um die erzeugten Diagramme zu formatieren und abzuspeichern.

Dafür ist es sinnvoll als erstes, das Diagramm einer Variable zuzuweisen, z.B. `fig`.

```
fig = px.line(egywth, x="Date", y="TMK")
```

Für die Abbildung `fig` können wir nun einen Titel und Achsenbeschriftungen festlegen. Dabei können wir HTML-Tags zur Formatierung benutzen.

```
fig.update_layout(title="<b>Wetterstation:</b> Rostock-Warnemuende",
                  xaxis_title="Datum",
                  yaxis_title="Tagesmittel der <br> Lufttemperatur in °C")
```

Jetzt können wir die Schriftarten ändern und die fertige Abbildung darstellen.

```
fig.update_layout(
    title_font=dict(size=16, family="Times New Roman", color="darkblue"),
    font_family="Times New Roman",
    font_color="green",
)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

Diese Abbildung können wir nun mit `write_image` abspeichern:

```
fig.write_image("temperaturverlauf.png")
```

Bibliographie