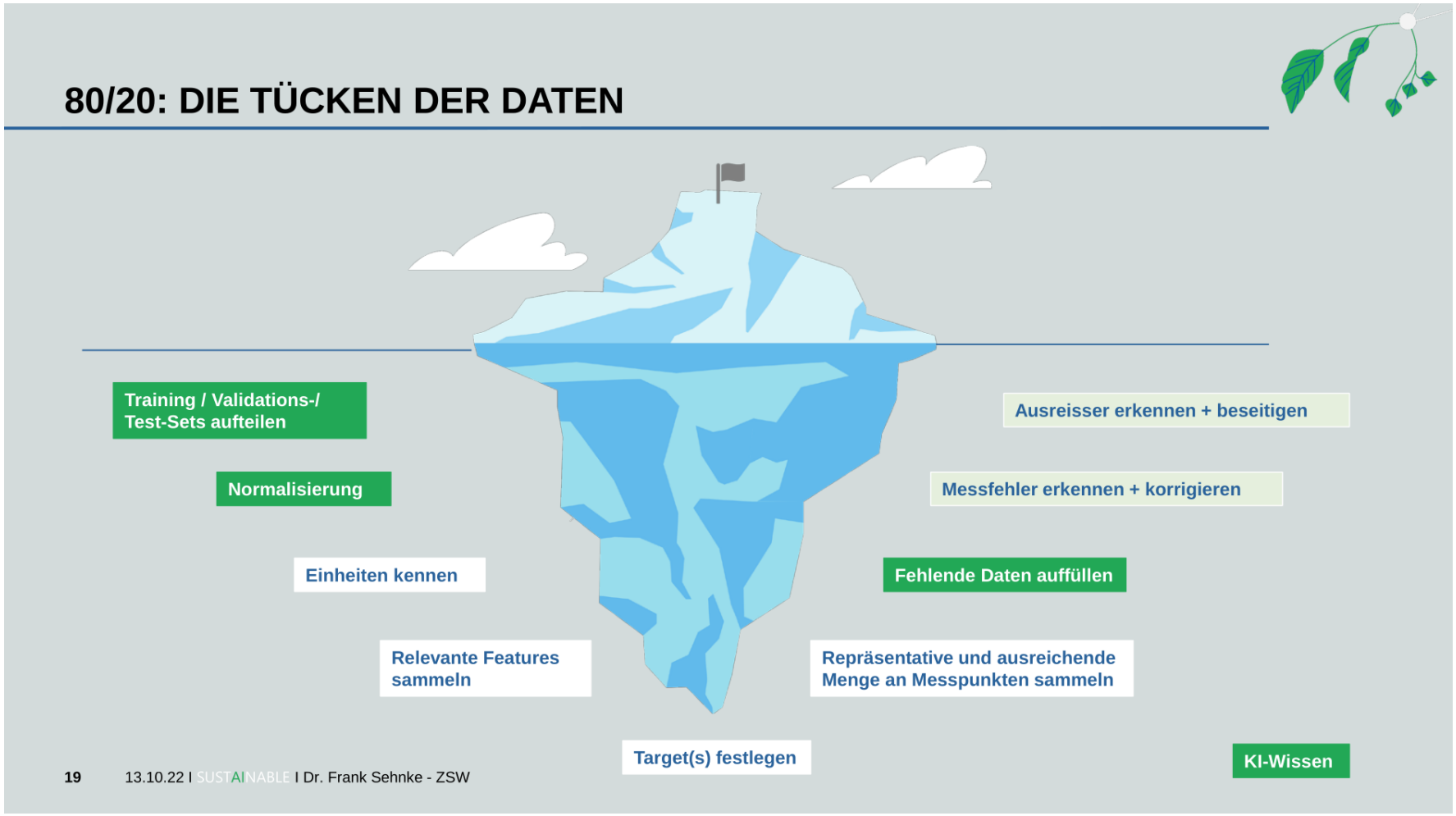
# Lehrbuch vs. Realität

Warum die Daten uns in der Praxis oft einen Strich durch die Rechnung machen.

Datenaufbereitung kostet Zeit + KI-Wissen:



# Lehrbücher und Tutorials: Ein guter Weg zu starten

...Wenn sie gut aufbereitet sind

Scikit Learn Decision Tree Regression:
https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html#sphx-glr-auto-examples-tree-plot-tree-regression-py

Tensorflow Basic Regression mit Keras API:
https://www.tensorflow.org/tutorials/keras/regression

# Datenvorverarbeitungsschritte:

1. Get the data
2. Clean the data
3. Split the data into training and test sets
4. Inspect the data
5. Split features from labels
6. Normalization

# 1. Get the data

In [2]:

```
dataset = pd.read_csv("wind_data_previous.csv", index_col=0)
dataset
```

Out[2]:

| | Time (UTC) | Average power output (MW) | Wind direction (°) | Wind speed (m/s) | Nacelle direction (°) | Rotational speed (s-1) |
|---|---|---|---|---|---|---|
| **0** | 01/05/18 01:00 AM | 0.05 | 329.46 | 2.73 | NaN | 6.66 |
| **1** | 01/05/18 02:00 AM | 0.05 | 333.37 | 2.03 | NaN | 3.94 |
| **2** | 01/05/18 03:00 AM | 0.12 | 340.21 | 4.04 | NaN | 10.36 |
| **3** | 01/05/18 04:00 AM | 0.14 | 339.69 | 4.59 | NaN | 12.12 |
| **4** | 01/05/18 05:00 AM | 0.11 | 338.21 | 3.33 | 351.50 | 11.74 |
| **...** | ... | ... | ... | ... | ... | ... |
| **6235** | 15/01/2019 20:00 | 0.18 | NaN | 5.55 | 348.20 | 12.47 |
| **6236** | 15/01/2019 21:00 | 0.21 | NaN | 4.73 | NaN | 12.17 |
| **6237** | 15/01/2019 22:00 | 0.06 | NaN | 3.83 | NaN | 8.96 |
| **6238** | 15/01/2019 23:00 | 0.03 | NaN | 3.04 | 344.27 | 10.40 |
| **6239** | 16/01/2019 00:00 | 0.04 | NaN | 2.95 | 344.39 | 8.18 |

6240 rows × 6 columns

# 2. Clean the data

In [3]:

```
print(f"Anzahl der Zeilen: {dataset.shape[0]}")
display(dataset.isna().sum())
print(f"Erwartete Anzahl gereinigert Daten: {6240 - 2514}")
```

```
Anzahl der Zeilen: 6240
```

```
Time (UTC)                    0
Average power output (MW)    1298
Wind direction (°)           1389
Wind speed (m/s)             2025
Nacelle direction (°)        2514
Rotational speed (s-1)       2081
dtype: int64
```

```
Erwartete Anzahl gereinigert Daten: 3726
```

In [4]:

```
dataset.dropna()
```

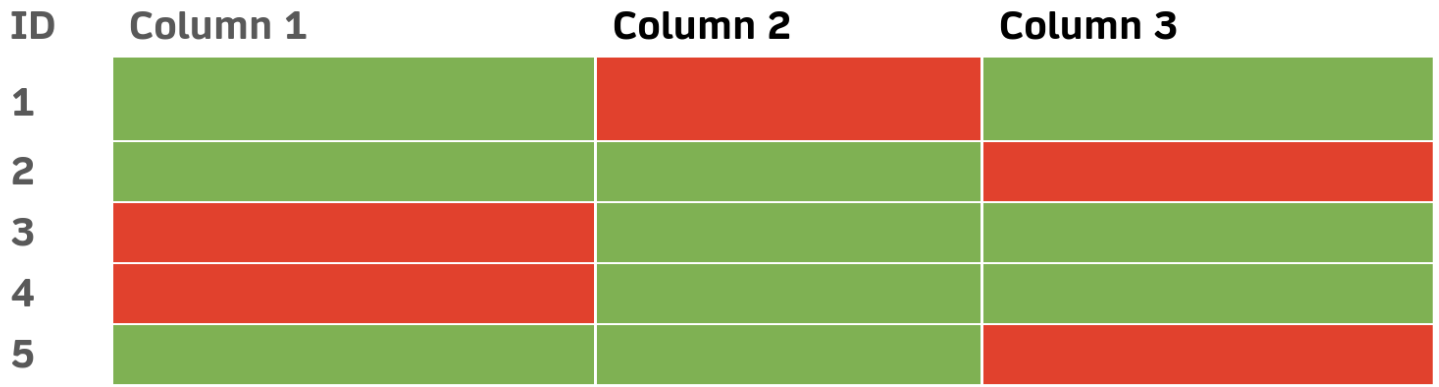
Out[4]:

| | Time (UTC) | Average power output (MW) | Wind direction (°) | Wind speed (m/s) | Nacelle direction (°) | Rotational speed (s-1) |
|---|---|---|---|---|---|---|
| **4** | 01/05/18 05:00 AM | 0.11 | 338.21 | 3.33 | 351.50 | 11.74 |
| **5** | 01/05/18 06:00 AM | 0.13 | 343.49 | 4.32 | 350.64 | 12.09 |
| **6** | 01/05/18 07:00 AM | 0.09 | 338.48 | 3.68 | 351.51 | 11.79 |
| **7** | 01/05/18 08:00 AM | 0.15 | 333.08 | 4.32 | 349.95 | 11.98 |
| **8** | 01/05/18 09:00 AM | 0.12 | 332.30 | 3.97 | 350.49 | 11.86 |
| **...** | ... | ... | ... | ... | ... | ... |
| **6207** | 14/01/2019 16:00 | 2.21 | 29.91 | 13.34 | 351.99 | 16.77 |
| **6208** | 14/01/2019 17:00 | 2.20 | 35.58 | 12.87 | 352.10 | 16.83 |
| **6209** | 14/01/2019 18:00 | 2.43 | 29.99 | 16.29 | 348.56 | 16.70 |
| **6210** | 14/01/2019 19:00 | 2.52 | 31.15 | 15.54 | 345.84 | 16.97 |
| **6211** | 14/01/2019 20:00 | 2.29 | 49.43 | 17.34 | 348.56 | 15.75 |

2858 rows × 6 columns

# ⚡Datentücke 1: NaNs

**Problem:** `dropna()` auf führt zu massivem Datenverlust, wenn in den verschiedenen Spalten zu unterschiedlichen Zeiten `NaN` vorkommt



**Mögliche Lösungen:**

- Features mit zu großen `NaN` - Anteil weglassen
- Fehlende Werte in Features und Targets Ersetzen und `dropna(how="all")`:
  - mit `-1` ( ⚠ nicht für Targets) ersetzen
  - besser: Werte interpolieren (falls möglich)
  - noch besser: Werte mit ML sinnvoll ersetzen (falls möglich)

KIlab.EE

# 3. Split the data into training, validation test sets

In [10]:

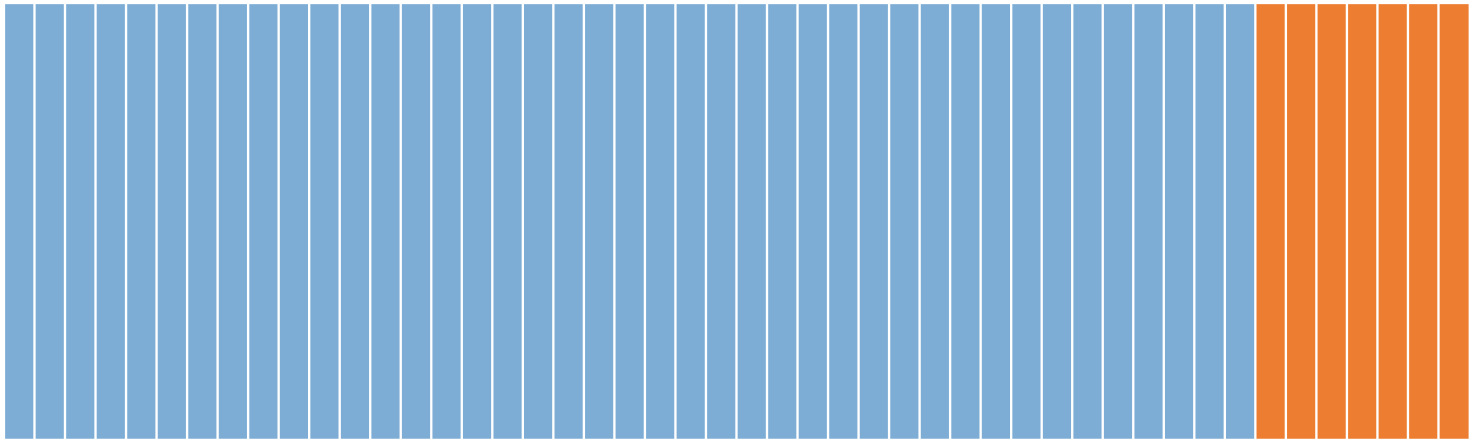
```
train_dataset = dataset.sample(frac=0.8, random_state=0)
val_dataset =  train_dataset.sample(frac=0.2, random_state=1)
test_dataset = dataset.drop(train_dataset.index)
print(f"Train Größe: {train_dataset.shape[0]-val_dataset.shape[0]},\
      Validation Größe: {val_dataset.shape[0]}, \
      Test Größe: {test_dataset.shape[0]}")
print(f"Train Anteil: {round((train_dataset.shape[0]-val_dataset.shape[0])/dataset.shape[0]*100,2)}%,\
      Validation Anteil: {round(val_dataset.shape[0]/dataset.shape[0]*100, 2)}%, \
      Test Anteil: {test_dataset.shape[0]/dataset.shape[0]*100}%,")
```

```
Train Größe: 3994,      Validation Größe: 998,
Test Größe: 1248

Train Anteil: 64.01%,   Validation Anteil: 15.99%,
Test Anteil: 20.0%,
```

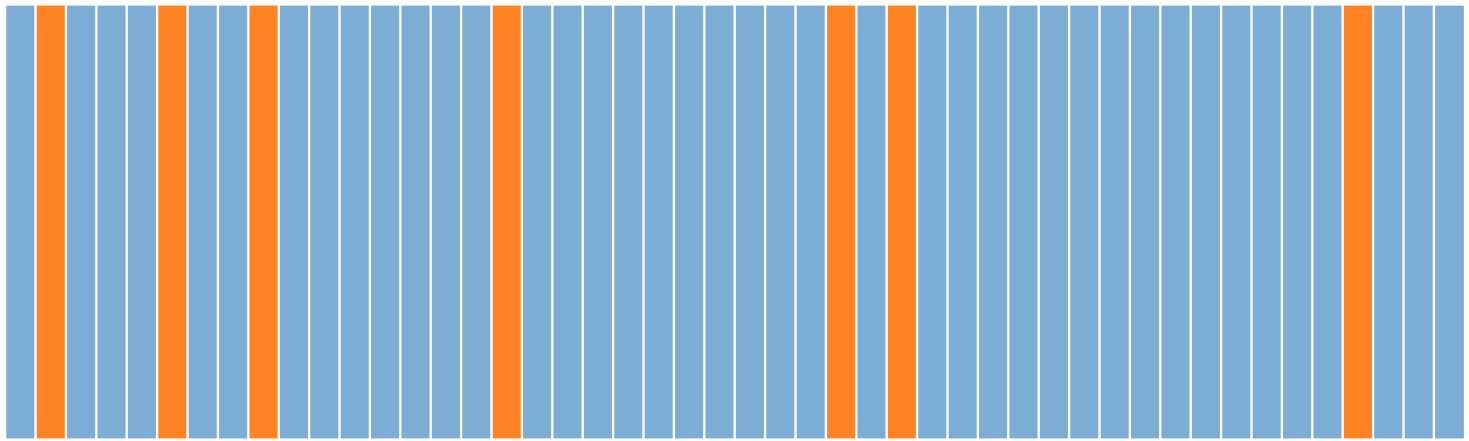
# Exkurs: Methoden zum Data-Splitting (Training / Validation Set)

Methode 1: Letzte 15 %



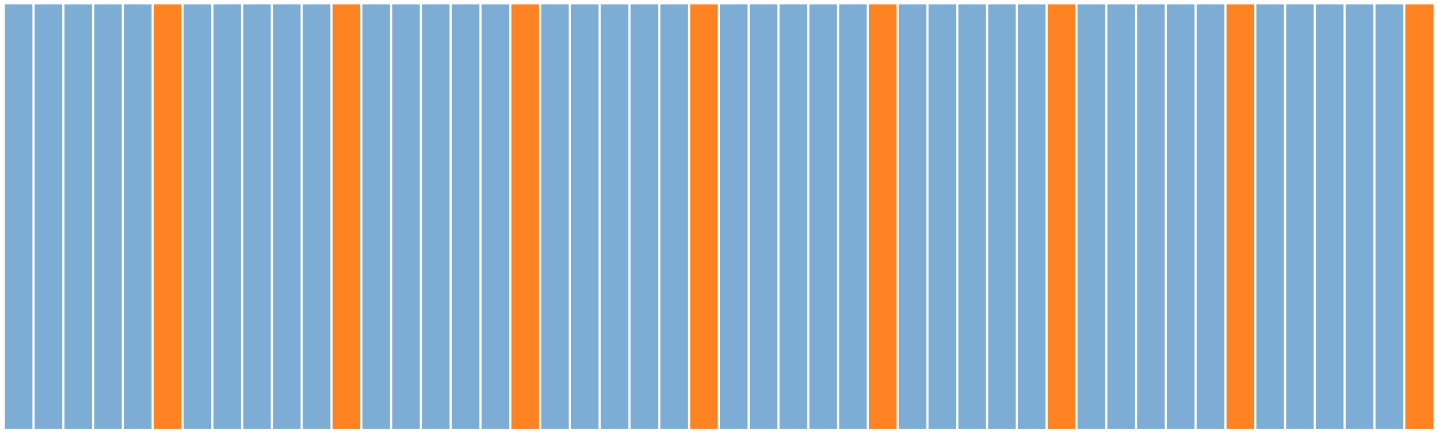
# Exkurs: Methoden zum Data-Splitting (Training / Validation Set)

Methode 2: Random 15 %



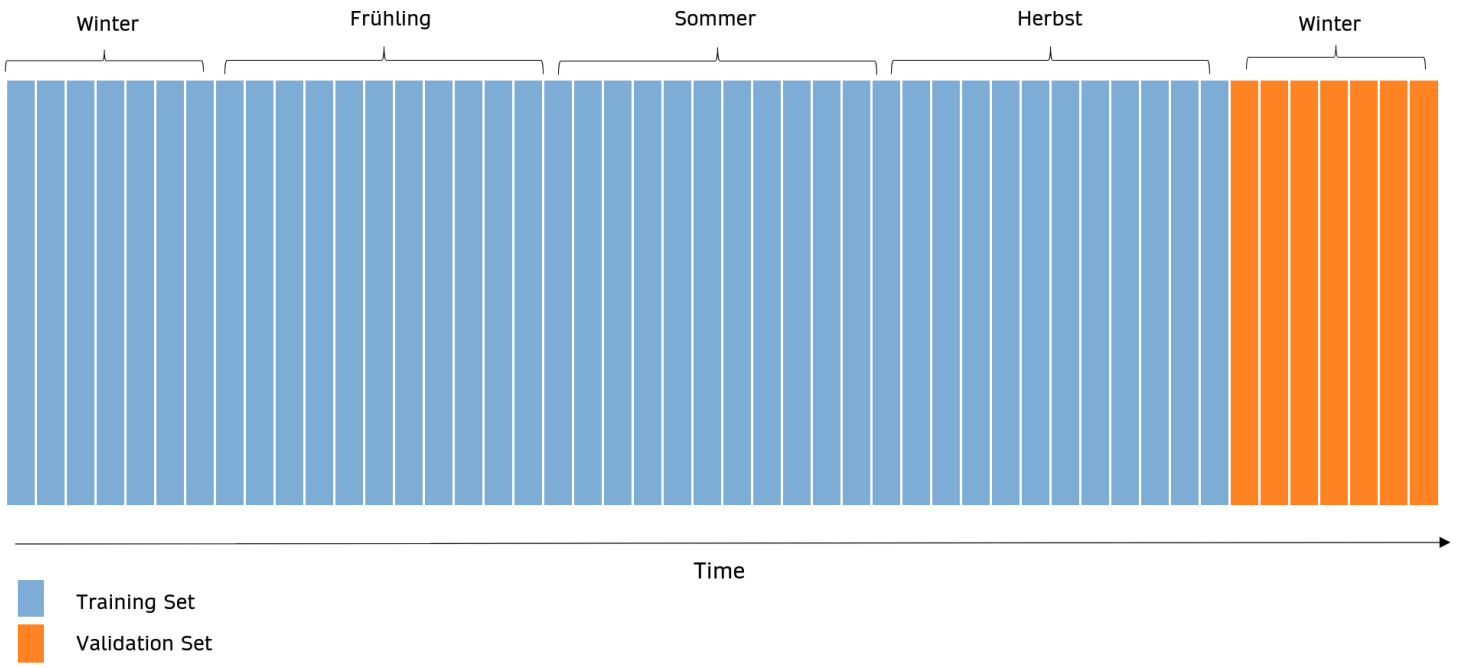
# Exkurs: Methoden zum Data-Splitting (Training / Validation Set)

Methode 3: Jeder *n* te Wert



# ⚡Datentücke 2 : Zeitreihen-Daten und Korrelation

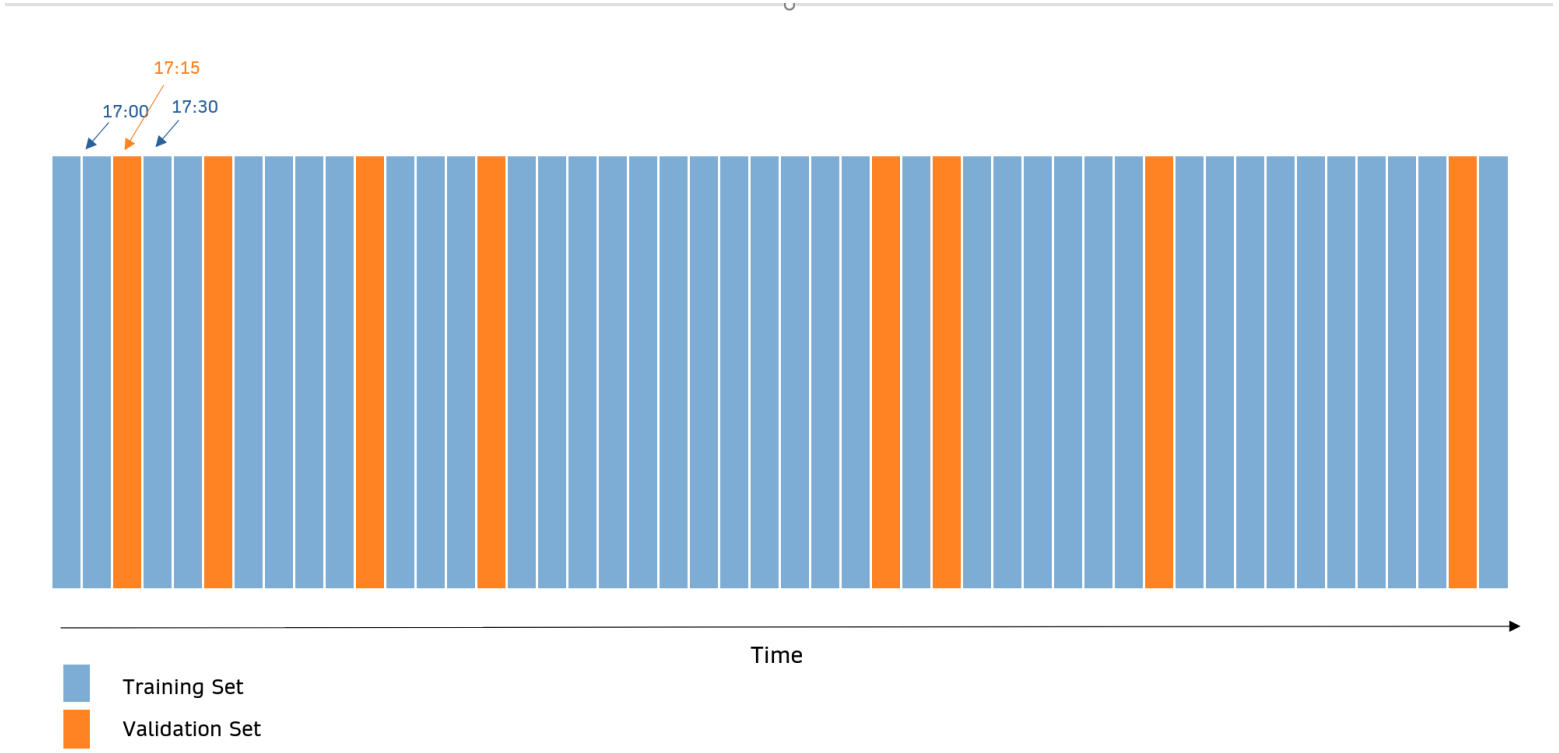
## Methode 1 & Methode 3:




**Problem:** Es besteht hier das Risiko das Modell auf einen nicht repräsentativen Teil der Daten auszuwerten

# ⚡Datentücke 2 : Zeitreihen-Daten und Korrelation

## Methode 2:




**Problem:** Es besteht hier das Risiko, dass Training- und Validationsdaten zu stark korrelieren ➡ Modell Overfitting

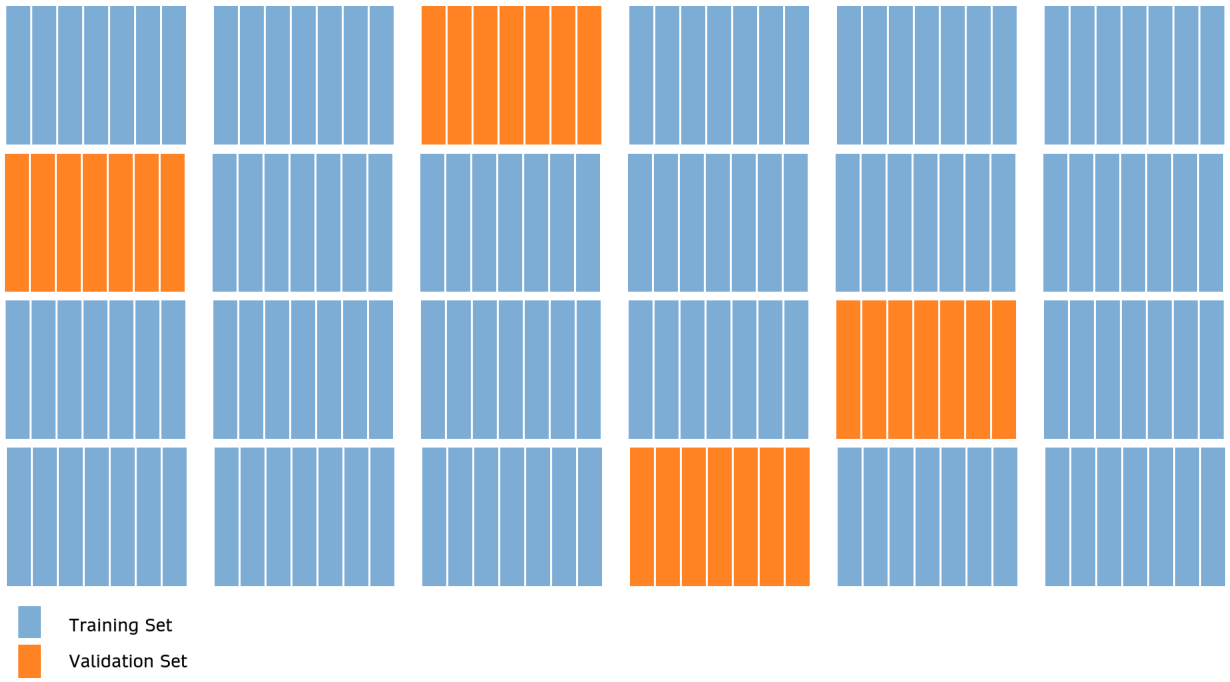
⚠ Sehr problematisch, da es sich nicht im Trainings-Fehler bemerkbar macht!

# ⚡Datentücke 2 : Zeitreihen-Daten und Korrelation

## Lösungen

- Varianten von Methode 1: Cross-Validation ( ⚠ viel Trainingszeit)
- Training- / Validation- und Test-Set manuell designen

- oder einen allgemein gültigen Algorithmus finden KIlab.EE



# 4. Normalization

In [6]:
```
train_dataset.describe().transpose()
```

Out[6]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Average power output (MW)** | 3945.0 | 0.553820 | 0.718131 | 0.00 | 0.0100 | 0.210 | 0.8500 | 2.58 |
| **Wind direction (°)** | 3896.0 | 227.497841 | 109.307114 | 2.63 | 142.3300 | 255.830 | 330.4475 | 358.39 |
| **Wind speed (m/s)** | 3352.0 | 6.840456 | 4.217716 | 0.28 | 3.1500 | 6.235 | 9.7250 | 23.03 |
| **Nacelle direction (°)** | 2969.0 | 212.610179 | 109.595266 | 0.00 | 128.6000 | 231.260 | 320.7800 | 359.70 |
| **Rotational speed (s-1)** | 3300.0 | 9.768745 | 6.505898 | 0.01 | 1.6475 | 12.210 | 15.8600 | 26.77 |

# ⚡(Kleine) Datentücke 3 : Normalisierung und zyklische Daten

---

Wind Direction: 360 ➡ 1.0

Wind Direction: 0 ➡ 0.0

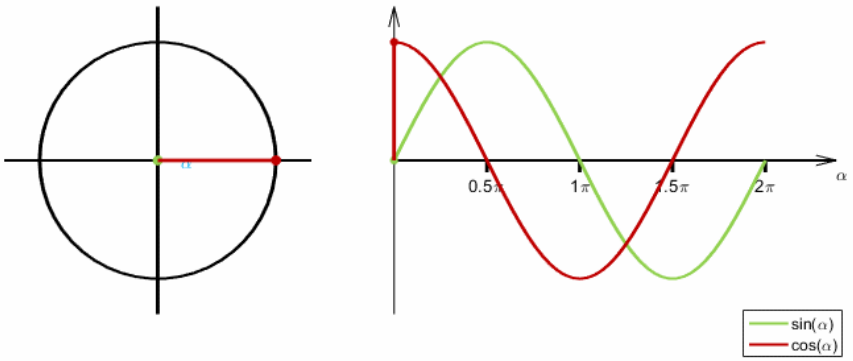
---

Hour of Day: 24 ➡ 1.0

Hour of Day: 0 ➡ 0.0

**Lösung**

- Sinus / Cosinus Normalisierung: Aus einer Variable werden Zwei



# ⚡Datentücke nach dem Training: Experimente richtig vergleichen

**Paxis-Beispiel:**

- Training verschiedener KI-Modelle um Wind-Leistung verschiedener Anlagen vorherzusagen
- Vorhersage wird mit dem NRMSE oder NMAE bewertet.
    - Anlage A: Fehler von 0.08 also 8%
    - Anlage B: Fehler von 0.14 also 14%
- Schlussfolgerung: Anlage A lässt sich einfacher vorhersagen

**Problem:** Anlage A und Anlage B wurden evtl. nicht mit denselben Werten nomiert, daher ist ein Vergleich des NRMSEs und NMAEs nicht angebracht

**Lösung:** Für Modell-Vergleiche immer absolute Fehler verwenden oder eine Experiment-übergreifende Normierung festlegen