Data Preprocessing

Analyse und Aufbereitung des **UCI Hydraulic Systems** Dataset:
https://archive.ics.uci.edu/dataset/447/condition+monitoring+of+hydraulic+systems

Übersicht der Analyseschritte

1. **Datensatz beschreiben** - Zielvariablen und Struktur verstehen
2. **Aggregation der Daten** - Time-Series Features extrahieren
3. **Missing Values prüfen** - Vollständigkeit sicherstellen
4. **Plausibilitätsprüfung** - Min/Max, Mean/Median validieren
5. **Korrelationen** - Zusammenhänge zwischen Features analysieren

1. Import der benötigten Bibliotheken

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path

# Konfiguration für bessere Plots
plt.rcParams['figure.figsize'] = (12, 6)
sns.set_style("whitegrid")
sns.set_palette("husl")

# Pfade definieren
DATA_DIR = Path('../data')
DOCS_DIR = Path('../docs')
OUT_DIR = Path('../out')

print("✓ Bibliotheken erfolgreich importiert")
```

```
✓ Bibliotheken erfolgreich importiert
```

2. Zum Datensatz

Der **UCI Hydraulic Systems Dataset** enthält 2.205 Messzyklen (je 60s) von einem hydraulischen Prüfstand. Der Prüfstand besteht u.A. aus Kühlung und Filter, am gesamten Prüfstand nehmen Sensoren

1. **Druck**
2. **Durchfluss**
3. **Temperatur**

der vier Komponenten (Kühler, Ventil, Pumpe, Akkumulator) auf. Insgesamt wurden 2205 Zyklen mit 17 Attributen aufgenommen

Zielvariablen

Das System überwacht 5 Zielgrößen, die in `docs/profile.txt` gespeichert sind:

1. **Cooler Condition** - Zustand des Kühlers (3, 20, 100)
2. **Valve Condition** - Zustand des Ventils (73, 80, 90, 100)
3. **Pump Leakage** - Pumpenleckage (0, 1, 2)
4. **Accumulator Pressure** - Akkumulatordruck (90, 100, 115, 130)
5. **Stable Flag** - Systemstabilität (0 = stabil, 1 = instabil)

Anwendung / Kontext

Mit einer Regression könnte man hier in Richtung Predictive Maintenance gehen.

In [2]:
```
# Datenstruktur der Rohdaten anzeigen
print("=" * 70)
print("STRUKTUR DER ROHDATEN (ZEITREIHEN)")
print("=" * 70)
print(f"Anzahl Messzyklen: 2.205")
print(f"Anzahl logischer Sensoren: 17")
print(f"  - 6 Drucksensoren (PS1-PS6)")
print(f"  - 1 Elektr. Leistungsaufnahme (EPS1)")
print(f"  - 2 Volumenstromsensoren (FS1-FS2)")
print(f"  - 4 Temperatursensoren (TS1-TS4)")
print(f"  - 1 Vibrationssensor (VS1)")
print(f"  - 1 Kühleffizienz (CE)")
print(f"  - 1 Kühlleistung (CP)")
print(f"  - 1 Effizienzfaktor (SE)")
print(f"\nZeitpunkte pro Sensor: bis zu 6.000 (je nach Samplingrate 1-100 Hz)")
print(f"Gesamtanzahl Spalten in Rohdaten: 43.680")
print(f"  (17 Sensoren × durchschnittlich ~2.570 Zeitpunkte)")
print("\n→ Die Rohdaten sind NICHT Teil dieses Notebooks.")
print("→ Wir arbeiten mit den bereits AGGREGIERTEN Features.")
```

```
======================================================================
STRUKTUR DER ROHDATEN (ZEITREIHEN)
======================================================================
Anzahl Messzyklen: 2.205
Anzahl logischer Sensoren: 17
  - 6 Drucksensoren (PS1-PS6)
  - 1 Elektr. Leistungsaufnahme (EPS1)
  - 2 Volumenstromsensoren (FS1-FS2)
  - 4 Temperatursensoren (TS1-TS4)
  - 1 Vibrationssensor (VS1)
  - 1 Kühleffizienz (CE)
  - 1 Kühlleistung (CP)
  - 1 Effizienzfaktor (SE)

Zeitpunkte pro Sensor: bis zu 6.000 (je nach Samplingrate 1-100 Hz)
Gesamtanzahl Spalten in Rohdaten: 43.680
  (17 Sensoren × durchschnittlich ~2.570 Zeitpunkte)

→ Die Rohdaten sind NICHT Teil dieses Notebooks.
→ Wir arbeiten mit den bereits AGGREGIERTEN Features.
```

In [3]:
```
# Zielvariablen laden - MIT MANUELLEN SPALTENNAMEN (Datei hat kein Header)
target_names = ['cooler_condition', 'valve_condition', 'pump_leakage', 'accumula
```

```
targets = pd.read_csv(DOCS_DIR / 'profile.txt', sep='\t', header=None, names=tar

print(f"Shape der Zielvariablen: {targets.shape}")
print(f"Spalten: {list(targets.columns)}\n")

# Übersicht der Zielvariablen
print("=" * 70)
print("ÜBERSICHT DER ZIELVARIABLEN")
print("=" * 70)
display(targets.head(10))

# Statistiken für jede Zielvariable
print("\n" + "=" * 70)
print("VERTEILUNG DER ZIELVARIABLEN")
print("=" * 70)
for col in targets.columns:
    print(f"\n{col}:")
    print(targets[col].value_counts().sort_index())
```

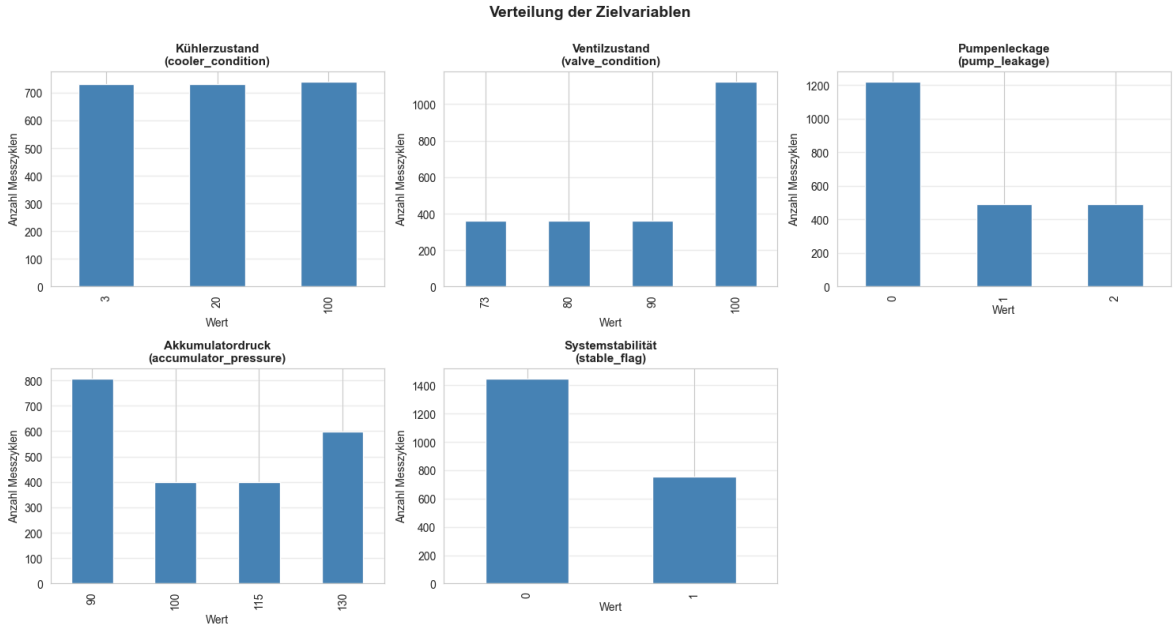
```
Shape der Zielvariablen: (2205, 5)
Spalten: ['cooler_condition', 'valve_condition', 'pump_leakage', 'accumulator_pre
ssure', 'stable_flag']

======================================================================
ÜBERSICHT DER ZIELVARIABLEN
======================================================================
```

	cooler_condition	valve_condition	pump_leakage	accumulator_pressure	stable_flag
0	3	100	0	130	1
1	3	100	0	130	1
2	3	100	0	130	1
3	3	100	0	130	1
4	3	100	0	130	1
5	3	100	0	130	1
6	3	100	0	130	1
7	3	100	0	130	1
8	3	100	0	130	1
9	3	100	0	130	1

```
======================================================================
VERTEILUNG DER ZIELVARIABLEN
======================================================================

cooler_condition:
cooler_condition
3      732
20     732
100    741
Name: count, dtype: int64

valve_condition:
valve_condition
73      360
80      360
90      360
100    1125
Name: count, dtype: int64

pump_leakage:
pump_leakage
0    1221
1     492
2     492
Name: count, dtype: int64

accumulator_pressure:
accumulator_pressure
90     808
100    399
115    399
130    599
Name: count, dtype: int64

stable_flag:
stable_flag
0    1449
1     756
Name: count, dtype: int64
```

In [4]:
```python
# Visualisierung der Zielvariablen
fig, axes = plt.subplots(2, 3, figsize=(15, 8))
axes = axes.flatten()

# Bessere Namen für die Plots
target_labels = {
    'cooler_condition': 'Kühlerzustand',
    'valve_condition': 'Ventilzustand',
    'pump_leakage': 'Pumpenleckage',
    'accumulator_pressure': 'Akkumulatordruck',
    'stable_flag': 'Systemstabilität'
}

for idx, col in enumerate(targets.columns):
    targets[col].value_counts().sort_index().plot(
        kind='bar', ax=axes[idx], color='steelblue'
    )
    # Titel mit deutscher Bezeichnung UND technischem Namen
    axes[idx].set_title(f'{target_labels.get(col, col)}\n({col})',
                        fontsize=11, fontweight='bold')
```

```
    axes[idx].set_xlabel('Wert')
    axes[idx].set_ylabel('Anzahl Messzyklen')
    axes[idx].grid(axis='y', alpha=0.3)

# Letzte Subplot leer lassen
axes[-1].axis('off')

plt.suptitle('Verteilung der Zielvariablen', fontsize=14, fontweight='bold', y=1
plt.tight_layout()
plt.show()
```



3. Aggregation / Feature Engineering

Die Rohdaten liegen als Zeitreihe vor:

- **17 Sensoren** mit unterschiedlichen Samplingraten (1 Hz bis 100 Hz)
- **2.205 Messzyklen** (Zeilen)
- **Bis zu 6.000 Zeitpunkte** pro Sensor (Spalten)
- **43.680 Spalten** in den Rohdaten

Aggregationsprozess

Für jeden Sensor werden acht Features berechnet:

- `mean` - Durchschnitt
- `std` - Standardabweichung
- `min` - Minimum
- `max` - Maximum
- `median` - Median
- `q25` - 25% Quantil
- `q75` - 75% Quantil
- `range` - Spannweite (max - min)

Resultat: 17 Sensoren × 8 Features = 136 aggregierte Features (anstatt 43.680 Spalten)

In [5]:
```python
# Aggregierte Features laden (aus prep_corrected.py generiert)
# Falls noch nicht vorhanden, führe zuerst prep_corrected.py aus

if (OUT_DIR / 'features_complete.csv').exists():
    df = pd.read_csv(OUT_DIR / 'features_complete.csv')
    print(f"✓ Aggregierte Features erfolgreich geladen")
    print(f"\nShape: {df.shape}")
    print(f"Spalten: {df.shape[1]} (136 Features + 5 Zielvariablen)")
    print(f"Zeilen: {df.shape[0]} Messzyklen\n")

    # Feature-Gruppen nach Sensor
    sensor_names = [
        'ps1', 'ps2', 'ps3', 'ps4', 'ps5', 'ps6',
        'eps1', 'fs1', 'fs2', 'ts1', 'ts2', 'ts3', 'ts4',
        'vs1', 'ce', 'cp', 'se'
    ]

    print("=" * 70)
    print("SENSOREN IM DATASET")
    print("=" * 70)
    for sensor in sensor_names:
        sensor_cols = [col for col in df.columns if col.startswith(f'{sensor}_')
        if sensor_cols:
            print(f"{sensor:5s} → {len(sensor_cols):2d} Features: {', '.join([c.

    # Erste Zeilen anzeigen
    print("\n" + "=" * 70)
    print("ERSTE ZEILEN DES AGGREGIERTEN DATASETS")
    print("=" * 70)
    display(df.head())
else:
    print("❌ Datei nicht gefunden!")
    print("Führe zuerst 'python prep_corrected.py' aus, um die Features zu gener
```

```
✓ Aggregierte Features erfolgreich geladen

Shape: (2205, 141)
Spalten: 141 (136 Features + 5 Zielvariablen)
Zeilen: 2205 Messzyklen

======================================================================
SENSOREN IM DATASET
======================================================================
ps1   →  8 Features: mean, std, min, max, median, q25, q75, range
ps2   →  8 Features: mean, std, min, max, median, q25, q75, range
ps3   →  8 Features: mean, std, min, max, median, q25, q75, range
ps4   →  8 Features: mean, std, min, max, median, q25, q75, range
ps5   →  8 Features: mean, std, min, max, median, q25, q75, range
ps6   →  8 Features: mean, std, min, max, median, q25, q75, range
eps1  →  8 Features: mean, std, min, max, median, q25, q75, range
fs1   →  8 Features: mean, std, min, max, median, q25, q75, range
fs2   →  8 Features: mean, std, min, max, median, q25, q75, range
ts1   →  8 Features: mean, std, min, max, median, q25, q75, range
ts2   →  8 Features: mean, std, min, max, median, q25, q75, range
ts3   →  8 Features: mean, std, min, max, median, q25, q75, range
ts4   →  8 Features: mean, std, min, max, median, q25, q75, range
vs1   →  8 Features: mean, std, min, max, median, q25, q75, range
ce    →  8 Features: mean, std, min, max, median, q25, q75, range
cp    →  8 Features: mean, std, min, max, median, q25, q75, range
se    →  8 Features: mean, std, min, max, median, q25, q75, range

======================================================================
ERSTE ZEILEN DES AGGREGIERTEN DATASETS
======================================================================
```

	ce_mean	ce_std	ce_min	ce_max	ce_median	ce_q25	ce_q75	ce_range	cp_me
0	39.601350	6.370535	28.866	47.438	40.6755	33.61075	45.93825	18.572	1.8627
1	25.786433	1.686129	23.320	29.208	25.3855	24.40025	27.24150	5.888	1.2555
2	22.218233	0.638345	21.220	23.554	22.1040	21.64050	22.74900	2.334	1.1132
3	20.459817	0.455755	19.673	21.565	20.4805	20.13825	20.71025	1.892	1.0621
4	19.787017	0.290156	19.133	20.460	19.7985	19.63250	19.96675	1.327	1.0704

5 rows × 141 columns

4. Missing Values

Laut Doku sollte der Datensatz keine fehlenden Werte enthalten.

Dennoch kurze Absicherung:

In [6]:
```python
if 'df' in locals():
    # Anzahl fehlender Werte pro Spalte
    missing_counts = df.isnull().sum()
    missing_total = missing_counts.sum()

    print("=" * 70)
```

```
    print("MISSING VALUES ANALYSE")
    print("=" * 70)
    print(f"Gesamtanzahl fehlender Werte: {missing_total}")
    print(f"Prozentsatz: {(missing_total / (df.shape[0] * df.shape[1]) * 100):.2

    if missing_total > 0:
        print("\nSpalten mit fehlenden Werten:")
        print(missing_counts[missing_counts > 0].sort_values(ascending=False))
    else:
        print("\n✓ KEINE FEHLENDEN WERTE GEFUNDEN!")
        print("Der Datensatz ist vollständig.")

    # Visualisierung (falls fehlende Werte existieren)
    if missing_total > 0:
        plt.figure(figsize=(12, 6))
        missing_counts[missing_counts > 0].sort_values(ascending=False).plot(kin
        plt.title('Missing Values pro Spalte', fontsize=14, fontweight='bold')
        plt.xlabel('Spalte')
        plt.ylabel('Anzahl fehlender Werte')
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()
else:
    print("❌ Dataset nicht geladen. Führe zuerst die vorherigen Zellen aus.")
```

```
======================================================================
MISSING VALUES ANALYSE
======================================================================
Gesamtanzahl fehlender Werte: 0
Prozentsatz: 0.00%

✓ KEINE FEHLENDEN WERTE GEFUNDEN!
Der Datensatz ist vollständig.
```

5. Plausibilitätsprüfung der Attribute

Wir prüfen die statistischen Kennzahlen jedes Features, um Ausreißer oder Probleme zu identifizieren:

- **Min/Max:** Wertebereich plausibel?
- **Mean vs. Median:** Hinweise auf Schiefe (Skewness)?
- **Stbw:** Variabilität der Daten

In [7]:
```
if 'df' in locals():
    # Nur numerische Features (ohne Zielvariablen)
    feature_cols = [col for col in df.columns if '_' in col and col not in targe
    features_only = df[feature_cols]

    # Deskriptive Statistiken
    stats = features_only.describe().T

    print("=" * 70)
    print("DESKRIPTIVE STATISTIKEN DER FEATURES")
    print("=" * 70)
    print(f"Anzahl Features: {len(feature_cols)}\n")

    # Erste 20 Features anzeigen
    display(stats.head(20))
```

```
    print("\n" + "=" * 70)
    print("PLAUSIBILITÄTSPRÜFUNG")
    print("=" * 70)

    # Features mit negativen Werten (falls nicht erwartet)
    negative_mins = stats[stats['min'] < 0]
    if len(negative_mins) > 0:
        print(f"\n⚠ Features mit negativen Werten: {len(negative_mins)}")
        print(negative_mins[['min', 'max', 'mean']].head(10))
    else:
        print("\n✓ Keine negativen Werte gefunden")

    # Features mit sehr hoher Standardabweichung (möglicherweise Ausreißer)
    high_std = stats[stats['std'] > stats['mean']].sort_values('std', ascending=
    if len(high_std) > 0:
        print(f"\n⚠ Features mit STD > MEAN (hohe Variabilität): {len(high_std)
        print(high_std[['mean', 'std', 'min', 'max']].head(10))

    # Features mit konstanten/sehr geringen Werten
    low_std = stats[stats['std'] < 0.01].sort_values('std')
    if len(low_std) > 0:
        print(f"\n⚠ Features mit sehr geringer Variabilität (STD < 0.01): {len(
        print(low_std[['mean', 'std', 'min', 'max']].head(10))
    else:
        print("\n✓ Alle Features haben ausreichende Variabilität")

else:
    print("❌ Dataset nicht geladen.")
```

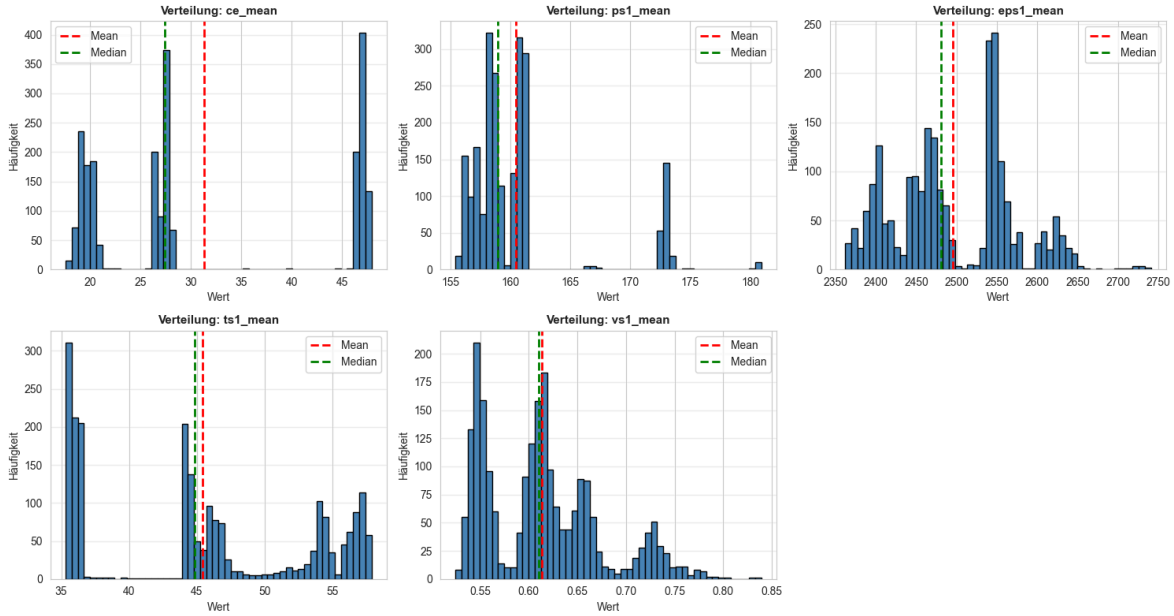
```
======================================================================
DESKRIPTIVE STATISTIKEN DER FEATURES
======================================================================
Anzahl Features: 136
```

	count	mean	std	min	25%	50%	
ce_mean	2205.0	31.299077	11.575330	17.555983	20.084650	27.392533	46.6
ce_std	2205.0	0.287325	0.199760	0.063164	0.217363	0.264125	0.3
ce_min	2205.0	30.785665	11.583481	17.042000	19.513000	26.862000	46.1
ce_max	2205.0	31.880312	11.602740	18.142000	20.694000	27.983000	47.2
ce_median	2205.0	31.287446	11.570829	17.501000	20.083000	27.363500	46.6
ce_q25	2205.0	31.078011	11.574876	17.362000	19.875750	27.151250	46.4
ce_q75	2205.0	31.506510	11.574789	17.746000	20.345000	27.592000	46.8
ce_range	2205.0	1.094648	0.585637	0.375000	0.871000	1.048000	1.2
cp_mean	2205.0	1.808399	0.278263	1.062150	1.550100	1.739683	2.1
cp_std	2205.0	0.022692	0.011138	0.006914	0.018377	0.021472	0.0
cp_min	2205.0	1.769092	0.282245	1.016000	1.502000	1.701000	2.1
cp_max	2205.0	1.855768	0.278504	1.105000	1.604000	1.786000	2.1
cp_median	2205.0	1.806442	0.277476	1.066000	1.550000	1.737000	2.1
cp_q25	2205.0	1.791137	0.279099	1.045500	1.532750	1.719750	2.1
cp_q75	2205.0	1.824328	0.277262	1.077000	1.570000	1.756750	2.1
cp_range	2205.0	0.086676	0.033147	0.034000	0.073000	0.083000	0.0
eps1_mean	2205.0	2495.509203	73.836682	2361.747267	2442.933467	2480.926633	2548.2
eps1_std	2205.0	203.489501	27.766181	185.105121	188.986584	196.153664	198.9
eps1_min	2205.0	2267.717914	65.422202	2097.800000	2202.800000	2260.000000	2341.6
eps1_max	2205.0	2900.794921	49.647523	2813.400000	2853.000000	2897.400000	2951.4

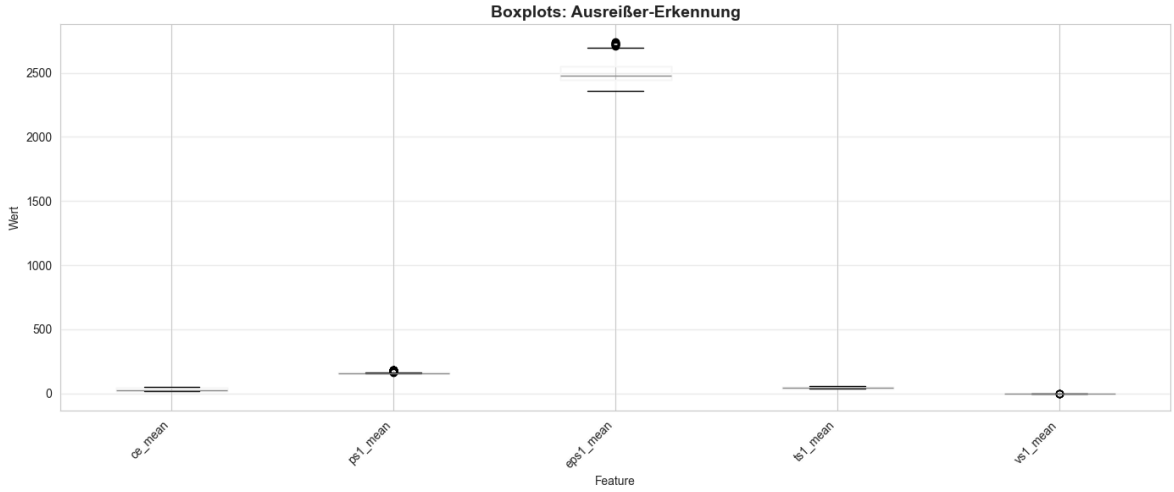
```
======================================================================
PLAUSIBILITÄTSPRÜFUNG
======================================================================

✓ Keine negativen Werte gefunden

⚠ Features mit STD > MEAN (hohe Variabilität): 10
                    mean       std       min        max
ps4_max     3.172446e+00  4.349040  0.000000  10.266000
ps4_q75     2.636674e+00  4.302525  0.000000  10.226000
ps4_median  2.572349e+00  4.295579  0.000000  10.207000
ps4_q25     2.536717e+00  4.290085  0.000000  10.190000
ps4_mean    2.600266e+00  4.279355  0.000000  10.207068
ps4_min     2.475427e+00  4.257247  0.000000  10.133000
ps4_range   6.970190e-01  1.733458  0.000000  10.238000
ps4_std     1.107360e-01  0.390756  0.000000   4.529653
ts4_std     4.352992e-02  0.052920  0.008342   1.442105
fs1_min     4.535147e-07  0.000021  0.000000   0.001000

⚠ Features mit sehr geringer Variabilität (STD < 0.01): 7
                  mean       std       min       max
ps3_min  0.000000e+00  0.000000  0.000000  0.000000
ps2_min  0.000000e+00  0.000000  0.000000  0.000000
se_min   0.000000e+00  0.000000  0.000000  0.000000
fs1_min  4.535147e-07  0.000021  0.000000  0.001000
ps5_std  2.225296e-02  0.002095  0.020142  0.088154
ps6_std  2.245055e-02  0.002104  0.020164  0.085366
fs2_std  1.716132e-02  0.004020  0.012419  0.091772
```

In [8]:

```python
if 'df' in locals():
    # Verteilung einiger wichtiger Features visualisieren
    important_features = ['ce_mean', 'ps1_mean', 'eps1_mean', 'ts1_mean', 'vs1_m
    available_features = [f for f in important_features if f in df.columns]

    if len(available_features) > 0:
        fig, axes = plt.subplots(2, 3, figsize=(15, 8))
        axes = axes.flatten()

        for idx, feature in enumerate(available_features):
            df[feature].hist(bins=50, ax=axes[idx], color='steelblue', edgecolor
            axes[idx].set_title(f'Verteilung: {feature}', fontsize=11, fontweigh
            axes[idx].set_xlabel('Wert')
            axes[idx].set_ylabel('Häufigkeit')
            axes[idx].axvline(df[feature].mean(), color='red', linestyle='--', l
            axes[idx].axvline(df[feature].median(), color='green', linestyle='--
            axes[idx].legend()
            axes[idx].grid(axis='y', alpha=0.3)

        # Nicht verwendete Subplots ausblenden
        for idx in range(len(available_features), len(axes)):
            axes[idx].axis('off')

        plt.tight_layout()
        plt.show()
else:
    print("❌ Dataset nicht geladen.")
```



In [9]:
```
if 'df' in locals():
    # Boxplots für Ausreißer-Erkennung
    sample_features = ['ce_mean', 'ps1_mean', 'eps1_mean', 'ts1_mean', 'vs1_mean
    available_features = [f for f in sample_features if f in df.columns]

    if len(available_features) > 0:
        plt.figure(figsize=(14, 6))
        df[available_features].boxplot(figsize=(14, 6))
        plt.title('Boxplots: Ausreißer-Erkennung', fontsize=14, fontweight='bold
        plt.ylabel('Wert')
        plt.xlabel('Feature')
        plt.xticks(rotation=45, ha='right')
        plt.grid(axis='y', alpha=0.3)
        plt.tight_layout()
        plt.show()
else:
    print("❌ Dataset nicht geladen.")
```



6. Korrelationsanalyse

Grundlegende Korrelationsanalyse um:

- **Redundante Features** zu identifizieren (hohe Korrelation)

- **Wichtige Zusammenhänge** zu verstehen
- **Features für theoretische Modellierung** auszuwählen

In [10]:
```
if 'df' in locals():
    # Korrelationsmatrix berechnen (nur Features, keine Targets)
    feature_cols = [col for col in df.columns if '_' in col and col not in targe
    corr_matrix = df[feature_cols].corr()

    print("=" * 70)
    print("KORRELATIONSMATRIX")
    print("=" * 70)
    print(f"Shape: {corr_matrix.shape}")
    print(f"Anzahl Korrelations-Paare: {(corr_matrix.shape[0] * (corr_matrix.sha

    # Hohe Korrelationen finden (> 0.9)
    high_corr = []
    for i in range(len(corr_matrix.columns)):
        for j in range(i+1, len(corr_matrix.columns)):
            if abs(corr_matrix.iloc[i, j]) > 0.9:
                high_corr.append({
                    'Feature 1': corr_matrix.columns[i],
                    'Feature 2': corr_matrix.columns[j],
                    'Correlation': corr_matrix.iloc[i, j]
                })

    if len(high_corr) > 0:
        print(f"\n⚠ Hohe Korrelationen gefunden (|r| > 0.9): {len(high_corr)}")
        high_corr_df = pd.DataFrame(high_corr).sort_values('Correlation', key=ab
        display(high_corr_df.head(20))
    else:
        print("\n✓ Keine extrem hohen Korrelationen gefunden (|r| > 0.9)")
else:
    print("❌ Dataset nicht geladen.")
```

```
======================================================================
KORRELATIONSMATRIX
======================================================================
Shape: (136, 136)
Anzahl Korrelations-Paare: 9180

⚠ Hohe Korrelationen gefunden (|r| > 0.9): 1992
```

⚠ Hohe Korrelationen gefunden (|r| > 0.9): 1992

	Feature 1	Feature 2	Correlation
1673	se_max	se_range	1.000000
1240	ps2_max	ps2_range	1.000000
1287	ps3_max	ps3_range	1.000000
793	fs1_max	fs1_range	1.000000
1512	ps6_mean	ps6_median	0.999999
1316	ps5_mean	ps5_median	0.999999
1959	ts4_mean	ts4_median	0.999997
824	fs2_mean	fs2_median	0.999997
1901	ts3_mean	ts3_median	0.999996
1595	ps6_median	ps6_q75	0.999996
1677	ts1_mean	ts1_median	0.999996
1514	ps6_mean	ps6_q75	0.999995
1418	ps5_median	ps5_q75	0.999995
1318	ps5_mean	ps5_q75	0.999995
1513	ps6_mean	ps6_q25	0.999995
1317	ps5_mean	ps5_q25	0.999995
1807	ts2_mean	ts2_median	0.999994
1902	ts3_mean	ts3_q25	0.999994
1417	ps5_median	ps5_q25	0.999993
1594	ps6_median	ps6_q25	0.999993

In [11]:
```
if 'df' not in locals() or 'targets' not in locals():
    print("❌ FEHLER: df oder targets nicht geladen!")
    print("Bitte führe die vorherigen Zellen aus (Abschnitte 1-3).")
else:
    # Feature-Spalten extrahieren (alle mit '_' außer Zielvariablen)
    feature_cols = [col for col in df.columns if '_' in col and col not in targe
    print(f"✓ Setup für Korrelationsanalyse erfolgreich")
    print(f"  - DataFrame shape: {df.shape}")
    print(f"  - Feature-Spalten: {len(feature_cols)}")
    print(f"  - Ziel-Spalten: {len(targets.columns)}")
    print(f"  - Ready for Section 6!")
```

```
✓ Setup für Korrelationsanalyse erfolgreich
  - DataFrame shape: (2205, 141)
  - Feature-Spalten: 136
  - Ziel-Spalten: 5
  - Ready for Section 6!
```

💡 Interpretation der hohen Korrelationen

Die obige Tabelle zeigt **perfekte Korrelationen (r ≈ 1.0)** zwischen bestimmten Feature-Paaren. Das ist **mathematisch erwartbar** und kein Fehler:

Perfekte Korrelationen (r = 1.0):

- `max` ↔ `range` : Da `range = max - min` ist, und `min` konstant ist, sind diese Features perfekt korreliert
- Beispiel: `ps2_max` und `ps2_range` haben r = 1.0

Sehr hohe Korrelationen (r > 0.999):

- `mean` ↔ `median` ↔ `q25` ↔ `q75` : Bei symmetrischen Verteilungen (ohne Ausreißer) sind diese Lagemaße nahezu identisch
- Beispiel: `ps6_mean` , `ps6_median` , `ps6_q25` , `ps6_q75` korrelieren alle mit r > 0.999

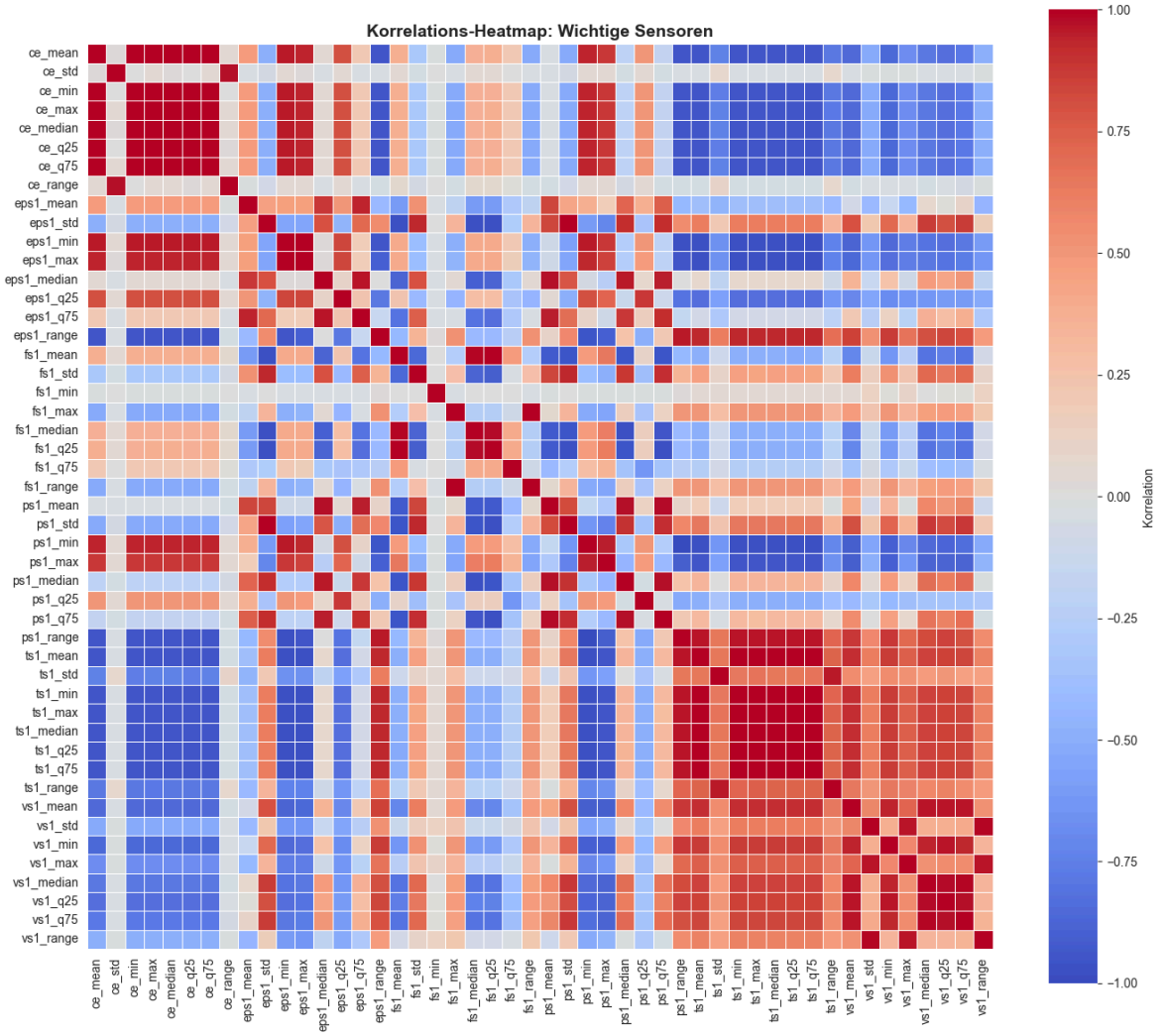
Konsequenzen für die Modellierung:

- ✅ **Feature Selection notwendig:** Redundante Features können entfernt werden (z.B. `range` wenn `max` vorhanden)
- ✅ **Dimensionality Reduction:** PCA oder Lasso Regularization könnten helfen
- ⚠️ **Multikollinearität:** Bei linearen Modellen (z.B. Lineare Regression) problematisch
- ✓ **Tree-based Modelle** (Random Forest, XGBoost) sind robust gegenüber hohen Korrelationen

Fazit: Die hohen Korrelationen sind **plausibel** und zeigen, dass die Sensoren in stabilen Bereichen messen. Für ML-Modelle sollten wir redundante Features entfernen.

In [12]:
```
if 'df' in locals():
    # Heatmap der Korrelationsmatrix (Ausschnitt für bessere Lesbarkeit)
    # Wähle eine Teilmenge von Features für die Visualisierung
    important_sensors = ['ce', 'ps1', 'eps1', 'ts1', 'vs1', 'fs1']
    subset_cols = [col for col in feature_cols if any(col.startswith(f'{s}_') fo

    if len(subset_cols) > 0:
        corr_subset = df[subset_cols].corr()

        plt.figure(figsize=(14, 12))
        sns.heatmap(
            corr_subset,
            annot=False,  # Keine Zahlen in Zellen (zu viele Features)
            cmap='coolwarm',
            center=0,
            vmin=-1,
            vmax=1,
            square=True,
            linewidths=0.5,
            cbar_kws={'label': 'Korrelation'}
        )
        plt.title('Korrelations-Heatmap: Wichtige Sensoren', fontsize=14, fontwe
        plt.tight_layout()
        plt.show()
```

```
else:
    print("❌ Dataset nicht geladen.")
```



In [13]:
```
if 'df' in locals() and 'targets' in locals():
    # Feature-Spalten sicherstellen
    feature_cols = [col for col in df.columns if '_' in col and col not in targe

    print("=" * 70)
    print("KORRELATION: FEATURES vs. ZIELVARIABLEN")
    print("=" * 70)

    for target_col in targets.columns:
        if target_col in df.columns:
            # Top 10 Features mit höchster Korrelation zur Zielvariable
            target_corr = df[feature_cols].corrwith(df[target_col]).abs().sort_v

            print(f"\n{target_col.upper()}:")
            print("-" * 70)
            print(target_corr.head(10))
        else:
            print(f"\n⚠ {target_col} nicht in DataFrame gefunden")
else:
    print("❌ Dataset oder Zielvariablen nicht geladen.")
```

```
======================================================================
KORRELATION: FEATURES vs. ZIELVARIABLEN
======================================================================

COOLER_CONDITION:
----------------------------------------------------------------------
ce_q25       0.992062
ce_min       0.992038
ce_mean      0.991943
ce_median    0.991902
ce_q75       0.991507
ce_max       0.991238
cp_min       0.957230
cp_q25       0.957230
cp_mean      0.956220
cp_median    0.955641
dtype: float64

VALVE_CONDITION:
----------------------------------------------------------------------
se_mean      0.231354
ps2_std      0.207383
fs1_mean     0.190141
fs1_max      0.181891
fs1_range    0.181890
se_q75       0.177244
eps1_mean    0.174618
se_median    0.172288
se_q25       0.171737
fs1_q25      0.167301
dtype: float64

PUMP_LEAKAGE:
----------------------------------------------------------------------
se_mean        0.469243
se_q75         0.449402
eps1_mean      0.426320
eps1_q75       0.423309
fs1_mean       0.421702
se_q25         0.406394
eps1_median    0.403787
se_median      0.401875
fs1_q25        0.387679
fs1_median     0.378952
dtype: float64

ACCUMULATOR_PRESSURE:
----------------------------------------------------------------------
fs1_max      0.397148
fs1_range    0.397148
ps3_max      0.342438
ps3_range    0.342438
ts1_range    0.281249
ts1_std      0.277756
cp_max       0.241880
cp_min       0.241573
cp_q75       0.241478
cp_mean      0.241432
dtype: float64
```

```
STABLE_FLAG:
----------------------------------------------------------------------
se_mean      0.324695
fs1_max      0.290922
fs1_range    0.290921
fs1_mean     0.288568
se_q25       0.274363
se_q75       0.273904
se_median    0.273156
eps1_q75     0.270526
ps2_std      0.267960
eps1_mean    0.267649
dtype: float64
```

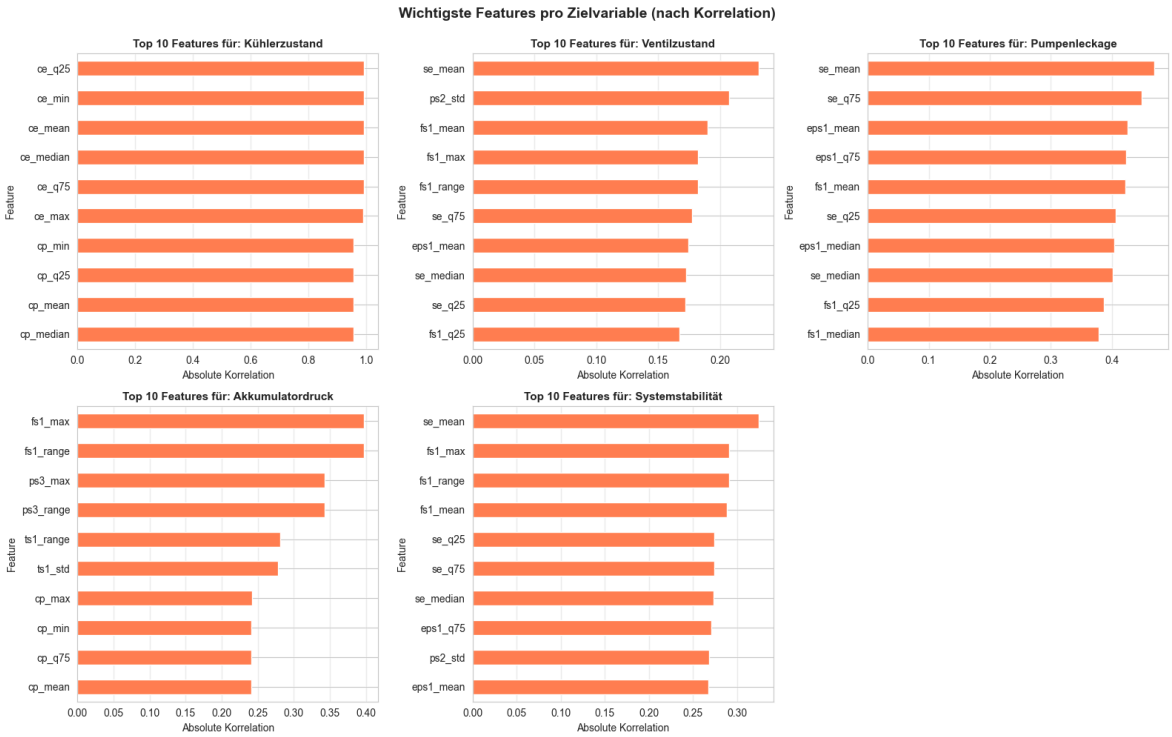
```
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3065: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[:, None]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3066: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[None, :]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3065: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[:, None]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3066: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[None, :]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3065: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[:, None]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3066: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[None, :]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3065: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[:, None]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3066: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[None, :]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3065: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[:, None]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3066: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[None, :]
```

In [14]:

```
if 'df' in locals() and 'targets' in locals():
    # Feature-Spalten sicherstellen
    feature_cols = [col for col in df.columns if '_' in col and col not in targe

    # Visualisierung: Top Features pro Zielvariable
    fig, axes = plt.subplots(2, 3, figsize=(16, 10))
```

```
    axes = axes.flatten()

    target_labels = {
        'cooler_condition': 'Kühlerzustand',
        'valve_condition': 'Ventilzustand',
        'pump_leakage': 'Pumpenleckage',
        'accumulator_pressure': 'Akkumulatordruck',
        'stable_flag': 'Systemstabilität'
    }

    plot_count = 0
    for idx, target_col in enumerate(targets.columns):
        if target_col in df.columns:
            # Top 10 Features mit höchster Korrelation
            target_corr = df[feature_cols].corrwith(df[target_col]).abs().sort_v

            if len(target_corr) > 0:
                target_corr.plot(kind='barh', ax=axes[idx], color='coral')
                axes[idx].set_title(f'Top 10 Features für: {target_labels.get(ta
                                    fontsize=11, fontweight='bold')
                axes[idx].set_xlabel('Absolute Korrelation')
                axes[idx].set_ylabel('Feature')
                axes[idx].grid(axis='x', alpha=0.3)
                axes[idx].invert_yaxis()  # Höchste Korrelation oben
                plot_count += 1
            else:
                axes[idx].text(0.5, 0.5, f'Keine Daten für\n{target_col}',
                               ha='center', va='center', fontsize=12)
                axes[idx].axis('off')

    # Letzte Subplot leer lassen
    axes[-1].axis('off')

    plt.suptitle('Wichtigste Features pro Zielvariable (nach Korrelation)',
                 fontsize=14, fontweight='bold', y=0.995)
    plt.tight_layout()
    plt.show()
    print(f"✓ {plot_count} Plots erfolgreich generiert")
else:
    print("❌ Dataset oder Zielvariablen nicht geladen.")
```

```
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3065: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[:, None]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3066: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[None, :]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3065: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[:, None]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3066: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[None, :]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3065: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[:, None]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3066: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[None, :]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3065: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[:, None]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3066: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[None, :]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3065: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[:, None]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3066: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[None, :]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3065: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[:, None]
c:\Users\casam\Documents\06_Coding\Python\DataProcessing\hydraulicSystems\.venv\L
ib\site-packages\numpy\lib\_function_base_impl.py:3066: RuntimeWarning: invalid v
alue encountered in divide
  c /= stddev[None, :]
```




```
✓ 5 Plots erfolgreich generiert
```

Zusammenfassung

Analyseschritte

1. **Beschreibung** - 2.205 Messzyklen mit 5 Zielvariablen
2. **Aggregation** - Von 43.680 Spalten zu 136 Features
3. **Missings** - Keine fehlenden Werte im Dataset
4. **Plausibilität** - Min/Max, Mean/Median, Standardabweichung
5. **Korrelationen** - Zusammenhänge zwischen Features und Zielvariablen

Output

Alle Ergebnisse wurden in `out/` gespeichert:

- `features_complete.csv` - Vollständiger Datensatz mit Features
- `feature_stats.csv` - Deskriptive Statistiken
- `correlation.csv` - Korrelationsmatrix