Projektarbeit
Auswertung der Backup-Daten meiner Solaranlage

Nachvollziehbare Schritte

Georg Aubele, 28.04.23

1 Kurze Darstellung des Problembereichs / Aufriss des Themas

1.1 Inhaltlich

In dieser Arbeit wurden – nach Identifikation, Auswahl und Zusammenstellung – die Back-up-Daten meiner Solaranlage grafisch aufbereitet und genauer untersucht. Nach der Vorarbeit handelte es sich um einen Datensatz mit 971051 Items. Einige davon wurden nicht genauer untersucht, da sie[1] in den privaten Bereich fallen.

Ziele dieser Arbeit

1. Identifizieren der in den Backups gespeicherten Daten, Auswahl geeigneter Kategorien und Zusammenführung in einen Datensatz
2. Grafische Aufbereitung der Daten ähnlich den üblichen Überwachungsapps
3. Untersuchung weiterer interessanter Aspekte
4. Vergleich der Strings im Hinblick auf unterschiedliche Degeneration der Module.

Quellen

Backups meiner Solaranlage

1.2 Begründung des Themas

Darstellung der Relevanz des Themas

Die Solaranlage liefert beständig Daten an den Provider, von welchem man dann in vorgefertigten Apps grafische Übersichten bekommt. Ein Provider kann aber auch ausfallen oder den Dienst einstellen. Da die Solaranlage weiterhin die Überwachungsdaten produziert, sollte man auch selbst in der Lage sein, sie aufbereiten und auswerten zu können.
Darüber hinaus gibt es auch Daten bzw. Verläufe von Daten, die in diesen Apps nicht sichtbar, aber sehr wohl von Interesse sind.

Darstellung eines persönlichen Erkenntnisinteresses

Die Steuerungseinheit „Solar-Log 500“ meiner Solaranlage erstellt jede Woche in der Nacht von Sonntag auf Montag ein Backup der Daten, die in den letzten 4 Wochen gesammelt wurden. Dieses wird mir per Anhang in einer E-Mail zugesandt. Bis auf eine kurze Überprüfung der Anlage zu Beginn der Laufzeit[2] wurden die Daten bisher nur archiviert. Aufgrund der Weiterbildung im Bereich Data Science und der Aufbereitung von großen Datenmengen in einem Projekt, nahm ich mir vor, genau diese Daten grafisch aufzubereiten und ihnen vielleicht noch das eine oder andere Geheimnis zu entlocken.

2 Nachvollziehbare Schritte

2.1 Stand der Forschung

Die Daten der Solaranlage wurden bisher nur durch den Provider aufbereitet. Ob auch eine genauere Untersuchung (Fehler, auffällige Unterschiede in der Stromerzeugung, …) stattfindet, entzieht sich meiner Kenntnis.

---

1 Wie z.B. der Verbrauchsverlauf oder der kumulierte Verbrauch.

2 Ich stellte eine Diskrepanz zwischen den grafisch dargestellten Daten in der App und dem Stand meines Stromzählers fest. Dies endete damit, dass ich den Wirkungsgrad des Wechselrichters nachwies.

2.2 Fragestellung und Ansatz

2.2.1 Identifikation der Kategorien, Auswahl und Bündelung in einem Datensatz

Die wöchentlichen Backups enthalten die Daten der letzten vier Wochen, jede Zeile steht für einen Zeitpunkt, die Zeitpunkte sind immer 5 Minuten auseinander. Da die Daten für jeden Tag von 0:00 Uhr bis 23:55 Uhr gehen, ergibt das für jeden Tag 288 Zeilen, pro Woche 2016 Zeilen und vom 2. Januar 2014 bis 13. April 2023 dann insgesamt 971051 Zeilen[3].
Zu Anfang beinhalteten die Dateien in jeder Zeile 32 durch Semikolon getrennte Werte, nach einer Änderung des Formats im November 2016 waren es pro Zeile 202 Werte.
Da der Support sich auf mehrere Anfragen nicht meldete[4], war auch ein großer Teil meiner Arbeit die Identifikation der einzelnen Kategorien. Nach dieser musste eine Auswahl im Hinblick auf gemeinsame und auswertbare Daten in beiden Formaten getroffen, alle Dateien in ihrem jeweils vorliegenden Format eingelesen und alles zu einem großen Datensatz zusammengefügt werden.

Die Fragestellung ist:
Lassen sich die wichtigsten Kategorien identifizieren, eine Auswahl an auswertbaren Kategorien treffen und alles in einem Datensatz bündeln?

2.2.2 Grafischer Überblick über den Ertrag in bestimmten Zeitintervallen

Bei Kauf einer Solaranlage und der entsprechenden Überwachungshardware bekommt man auch einen Login zu einer App, mit der man die Daten der Solaranlage[5] grafisch aufbereitet serviert bekommt. Wenn man sich aber mit den Daten etwas genauer beschäftigen möchte, findet man immer wieder Dinge, die anders bzw. auch noch dargestellt werden sollten. Eine mögliche Lösung dafür ist, zu einer anderen kostenpflichtigen Software zu greifen[6] oder die Backup-Daten selbst aufzuarbeiten.

Die Fragestellung ist:
Können die Backup-Daten mit absehbarem Aufwand eine ähnliche oder vielleicht sogar überlegene Darstellung der Daten ermöglichen.

2.2.3 Untersuchung der Daten hinsichtlich weiterer interessanter Aspekte

Da die Daten sich über einen großen Zeitraum verteilen und dennoch sehr feinmaschig sind, kann man damit auch „gefühlten Wahrheiten“ das Wetter betreffend auf den Grund gehen. So gibt es – sogar durch Untersuchungen gestützt[7] - die Meinung, dass das Wetter am Wochenende im Mittel schlechter sei als unter der Woche. Mit den Daten der Solaran-

3 Dazwischen fehlen leider ein paar Daten, da die Backups unvollständig geschickt wurden oder aufgrund der Formatumstellung 2016
4 Fast zum Ende der Erstellung dieses Berichts bekam ich dann eine Antwort – siehe weiter unten.
5 momentan erzeugte Leistung, Leistungsverlauf über den Tag, die Woche, den Monat, … Verbrauchsdaten, Daten der Batterie, ...
6 Die die Anbieter der Überwachungshardware natürlich für Premiumkunden zur Verfügung stellen. Dies kostet dann aber auch mehr.

lage lässt sich zwar keine Aussage bezüglich der Temperatur geben, aber wenn an bestimmten Wochentagen der Ertrag höher wäre, wäre das ein Indiz für mehr Sonneneinstrahlung und somit besseres Wetter.
Von dieser Untersuchung motiviert, kann man auch noch untersuchen, ob sich für die einzelnen Tage im Monat ebenfalls etwas ähnliches abzeichnet.

Die Fragestellung ist:
Lassen sich mit den Daten der Solaranlage Gesetzmäßigkeiten hinsichtlich des Ertrages an bestimmten Wochen- oder Kalendertagen herausarbeiten?

2.2.4 Unterschiedliche Degeneration der Module in den Strings 1 und 2

Die Module einer Solaranlage „altern“, das bedeutet, dass aufgrund der Sonneneinstrahlung die Halbleiterschichten mit der Zeit degenerieren und der Ertrag pro eingestrahlter Leistung abnimmt. Der Hersteller meiner Module garantiert mir nach 10 Jahren noch 90% der Ausgangsleistung, nach 25 Jahren noch 80%.
Da jedes Modul einzigartig ist, gibt es auch in der Alterung sicher Unterschiede.
Meine Solaranlage besteht aus 20 Modulen, je 10 sind zu einem String gekoppelt.
Mit den Daten des Backups kann man zwar weder einzelnen Module noch die absolute Degeneration untersuchen, aber eventuelle Unterschiede in den Strings erkennen.

Die Fragestellung ist:
Lassen sich mit Hilfe der Daten unterschiedliche Alterungserscheinungen in den Strings erkennen?

2.3 Methode und Ergebnisse[8]

2.3.1 Aufbereitung des Datensatzes

2.3.1.1 Alte Backups

Nach dem Import der notwendigen Packages wurde eine CSV-Datei eingelesen und in einen Pandas Dataframe umgewandelt. Dabei musste folgendes beachtet werden:

- Die Werte waren durch Semikolon getrennt (delimiter)
- Es gab keine Spaltenüberschriften (header).
- Die ersten 6 Zeilen beinhalteten Daten, die mit dem eigentlichen Datensatz nichts zu tun hatten und auch weniger Werte (teilweise nur einen) pro Zeile hatten. Um hier Probleme beim Import zu vermeiden, wurden diese übersprungen (skiprows)
- Am Ende der Datei waren wieder Zeilen mit weniger Werten, die auch nichts mit den eigentlichen Daten zu tun hatten. Um hier Fehlermeldungen zu vermeiden wurden diese zunächst mit NaN aufgefüllt (na_values).

7 https://www.n-tv.de/wissen/frageantwort/Ist-am-Wochenende-oefter-schlechtes-Wetter-article19757366.html
8 Ich habe mich entschlossen, beides zusammen zu erläutern, da die Ergebnisse jeweils auch Einfluss auf die weitere Vorgehensweise hatten.

- Ein Datums- und Zeitwert war in der zweiten Spalte enthalten, dieser wurde in ein DateTime-Objekt umgewandelt (parse_dates).
- Im Datum selbst stand der Tag vor dem Monat. Dies hatte Pandas nicht selbst bemerkt, was zu Fehlern in den Daten führte. Ein „dayfirst=True“ behob diese Fehler.

```
# Import
# import numpy as np
import datetime as dt
import pandas as pd
import matplotlib.pyplot as plt

# %% Einlesen einer Datei

# mit Pandas

# Problem: Die ersten Zeilen haben weniger Items, danach macht er dann NaN
# Anscheinend orientiert sich Pandas nicht an der maximalen Anzahl von Einträgen
# pro Zeile, sondern an der Anzahl der Einträge in den ersten paar Zeilen
# Lösung: skiprows

df0 = pd.read_csv("backup_data_10.05.15.dat", delimiter=";", header=None,
                  skiprows=6, na_values='', parse_dates=[2], dayfirst=True)
```

Der Support meines Anbieters gab sich sehr schweigsam (erst 2 Wochen keine Antwort und dann das):

Ticket #10348967

Sehr geehrter Herr Aubele,

Vielen Dank für die Anfrage.

Leider geben wir zu den Datenstrukturen der Backups keine Informationen heraus.

Mit freundlichen Grüßen aus Binsdorf

Manuel Schätzle
Technischer Support

Solar-Log™
we create connections

Daraus folgte, dass ich die einzelnen Kategorien selbst untersuchen musste.
Beim Blick in den Dataframe war schnell zu erkennen, dass alle relevanten Zeilen mit einer 2 an der ersten Stelle und einer 0 an der zweiten Stelle versehen sind (anscheinend Kontrollparameter). Deswegen wurde der Dataframe auf diese Zeilen begrenzt.
Anschließend wurden alle Spalten bis auf die Datumsangabe in Integer 32 umgewandelt, um Speicherplatz zu sparen.

```
# %%
# Ausfiltern der Datensätze, die nicht zum "Stamm" gehören
# Warum auch immer, im alten Format sind die Einträge Strings, im neuen Zahlen

df1 = df0[(df0.iloc[:, 0] == "2") & (df0.iloc[:, 1] == "0")]
```

```
# %%

# Umwandeln der  Spalten außer dem Datum, um Speicherplatz zu sparen
# und Zahlen zu haben
# Anlegen einer Hilfsliste mit allen Spalten außer der 2
help_list = [i for i in range(32)]
# droppen der 2
help_list.remove(2)

# Umwandeln in int32, um Speicher zu sparen
df1.iloc[:, help_list] = df1.iloc[:, help_list].astype(int)
```

Durch weitere Beobachtungen konnte man Spalten identifizieren, die sich im Tages- und Wochenverlauf nicht ändern. Diese wurden von mir ich als nicht relevant entfernt. Die zweite Spalte wurde in „DateTime“ umbenannt[9] und als Index festgelegt.
Eine Liste mit den übrig gebliebenen Spaltenbezeichnungen wurde für spätere Zwecke angelegt.

```
# %%

# Löschen der unnötigen Spalten, da diese nur Kontroll-Parameter enthalten

drop_list_raw = [i for i in range(32)]
except_list = [2, 6, 8, 10, 15, 17, 19, 23, 25, 27, 29, 31]
drop_list = [i for i in drop_list_raw if i not in except_list]


df1 = df1.drop(df1.columns[drop_list], axis=1)


# # Umbenennen der Spalte mit dem DateTime
df1 = df1.rename(columns={2: 'DateTime'})


# # Setzen des DateTime als Index
df1 = df1.set_index(df1.columns[0])


# Kopie, damit ich mit der Liste weiter unten arbeiten kann, wenn ich die
# Erstellung der Grpahen überspringe
print_list = [6, 8, 10, 15, 17, 19, 23, 25, 27, 29, 31]
```

Um nun die restlichen Werte in den Zeilen identifizieren und weitere unnötige entfernen zu können, wurde ein Algorithmus angelegt, der jede Spalte gegen das Datum plottet. Da der Datensatz Werte von 2014 bis 2016 beinhaltet, musste er für diese Untersuchung auf einen kleineren Zeitraum beschränkt werden.
Ich suchte mir den 20.04.2015 aus, da dieser Tag einen guten Ertrag brachte und am Morgen aufgrund einer leichten Abschattung einen Unterschied in den Strings aufwies. Mit Hilfe der Darstellungen meines Providers konnten diese dann identifiziert werden[10].
Um die Vorgänge zeitlich weiter eingrenzen zu können, hielt ich mir noch die Option offen, die Zeitwerte auf der x-Achse weiter einzuschränken[11].
Jeder Plot wurde dann für genauere Betrachtung und eventuelle spätere Untersuchungen abgespeichert.

9 Ein Zugriff über einen Spaltennamen erscheint mir einfacher als über den Spaltenindex. Dies sieht man schon daran, dass ursprünglich das Datum die Spalte 2 war, nach dem Aufräumen war es Spalte 0.
10 Ansonsten hätte ich den ersten der beiden einfach als „1“ gesetzt. Dies entsprach dann auch der Wirklichkeit.
11 Zum Beispiel ist für die Untersuchung der Strings die Nacht irrelevant.

```
# %%

# Erstellen und Speichern der Graphen der übrigen Spalten

# Begrenzen für die Graphen, um etwas zu erkennen
df1_s = df1.loc["20-04-15"]


print_list = [6, 8, 10, 15, 17, 19, 23, 25, 27, 29, 31]
for i in print_list:
    fig, ax = plt.subplots()
    x = df1_s.index
    y_1 = df1_s[i]
    # Begrenzen der x-Achse, um Daten besser erkennen zu können.
    # ax.set_xlim(dt.datetime(2015, 4, 20, 6, 0),dt.datetime(2015, 4, 20, 20, 0))
    ax.set_title(str(i))
    ax.plot(x, y_1)
    plt.show()
    dateiname = str(i)+".png"
    fig.savefig(dateiname, dpi=300)
```

Ein Vergleich mit den Grafiken in meiner App bzw. dem genaueren Betrachten der Werte ergab folgendes:

```
# Alte Spalten zum Vergleich
# ==========================================================
# Übrige Spalten:
#     6 -> Verbrauch in W
#     8 -> Verbrauch kumuliert in 10^-2 Wh
#     10 -> Verbrauch kumuliert in Wh
#     15 -> P_Erzeugung Gesamt in W
#     17 -> AC Spannung
#     19 -> Kontroll Parameter, 3 wenn es läuft, 1?
#     23 -> P_Erzeugung String 1 in W
#     25 -> DC Spannung S1 in V
#     27 -> P_Erzeugung String 2 in W
#     29 -> DC Spannung S2 in V
#     31 -> Ertrag kummuliert in Wh
# ==========================================================
```

2.3.1.2 Neuere Backups

Das Vorgehen bei den neueren Backups war nahezu analog, deswegen werden hier die Schritte nur kurz erwähnt.
Einlesen:

- Beim Import wurden die ersten 31 Zeilen übersprungen, da sie irrelevant waren.
- Es waren bei den relevanten Zeilen nun 202 Einträge pro Zeile

Ein Umwandeln in Integer 32 fand ebenfalls statt. Eine erste Vorauswahl wurde getroffen, indem wieder die Veränderung der Werte im Stunden- bzw. Tagesverlauf betrachtet wurden. Danach wurde auch hier die Datumsspalte umbenannt und als Index verwendet.
Für die genauere Untersuchung habe ich mir den 16.03.2023 ausgesucht. Dabei hat sich folgende Liste von interessanten Spalten ergeben:

```
# Neue Spalten:
# ================================================================================
# Übrige Spalten:
#     2  -> DateTime
#     6 ->
#     10 -> Verbrauch kumuliert in Wh aus dem Netz
#     14 -> Verbrauch kumuliert in 10^-2 Wh aus dem Netz
#     17 -> Ertrag S1 und S1 in W (mit Verlust?)
#     19 -> Ertrag S1 in W
#     21 -> Ertrag S2 in W
#     25 -> Ertrag kumuliert in Wh
#     27 -> Spannung DC S1 in V
#     29 -> Spannung DC S2 in V
#     33 -> Spannung AC in V
#     35 -> Kontroll Parameter 0,1,3,5
#     40 -> Ladung der Batterie in W
#     42 -> Entladung der Batterie in W
#     44 -> Ladezustand der Batteris in %
#     48 -> Kontrollparameter 1, 4, 6
#     53 -> Verbruch in W
#     57 -> Verbrauch kumuliert aus der Batterie in Wh
#     61 -> Verbrauch kumuliert aus der Batterie in 10xWs
#     64 -> Ertrag S1 und S1 in W (mit Verlust?) - Seltsame Werte bei Nacht, nahezu identisch 17
#     68 -> Größe absolut 27000 Änderung in drei Tagen um 15
#     75 -> ???
#     79 -> ??? Treppenfunktion die um 3 Uhr abfällt
#     92 -> Geht nie unter 70, tagsüber oft auf 100 - Energieverbrauch Solarlog?
#     134 -> Kontrollparameter
#     136 -> ???  Evtl Speicherauslastung
#     138 -> ???  Evtl. Prozessorfreq.
```

2.3.1.3 Vergleich und Vereinheitlichung

Ein Vergleich der übriggebliebenen Spalten im neuen und alten System ergab folgende Auswahl:

```
# ========================================================
# Einigung auf bestimmte Spalten für die Auswertung:
#                              Alt     Neu
#     Verbrauch in W:          6       53
#     Verbrauch_kum in Wh:     10      10 + 57
#     Ertrag_Gesamt in W:      15      17
#     Ertrag_S1 in W           23      19
#     Ertrag_S2 in W           27      21
#     Ertrag_kum in Wh         31      25
# ========================================================
```

Im alten System wurden die entsprechend nicht weiter verwendeten Spalten gelöscht, die verbliebenen Spalten umbenannt.

```
# %%  Umformatieren und Aufräumen der Spalten

new_keep_list = [6, 10, 15, 23, 27, 31]

new_drop_list = [i for i in print_list if i not in new_keep_list]


df2 = df1.drop(new_drop_list, axis=1)
```

```
# %% Spalten Umbenennen

name_dict = {6: "Verbrauch", 10: "Verbrauch_kum", 15: "Ertrag_gesamt", 23: "Ertrag_S1",
             27: "Ertrag_S2", 31: "Ertrag_kum"}

df2 = df2.rename(columns=name_dict)
```

Zusätzliche Spalten mit Details zum Datum wurden eingefügt und am Ende wieder soweit wie möglich auf int32 konvertiert.

```
# %%
# Aufspalten des DateTime in verschiedene Spalten, um später evtl.
# einen Multiindex aufbauen zu können

# Reset Index, um den Zugriff beim Ausplitten zu vereinfachen
df2 = df2.reset_index()

# Neue Datumsspalten einfügen
df2['day'] = df2["DateTime"].dt.day
df2['month'] = df2["DateTime"].dt.month
df2['year'] = df2["DateTime"].dt.year
df2['time'] = df2["DateTime"].dt.time
df2['weekday_number'] = df2["DateTime"].dt.weekday
df2['weekday'] = df2["DateTime"].dt.strftime("%A")

# Index wieder auf DateTime umstellen
df2 = df2.set_index("DateTime")
df2['time'] = pd.to_datetime(df2["time"], format="%H:%M:%S")

# Wieder Speicher sparen
df2.iloc[:, [6, 7, 8, 10]] = df1.iloc[:, [6, 7, 8, 10]].astype(int)
```

Ein probeweises Abspeichern als Parquet ergab folgendes:
Die Dateien waren klein, der Zugriff erfolgte sehr schnell. Allerdings kann Parquet anscheinend nicht wirklich mit dem DateTime-Format von Pandas umgehen, die time-Spalte wurde mir immer auf den 01.01.1900 zurückgesetzt. Man hätte dies wohl umgehen können, indem man die zusätzlichen Spalten erst bei der Verarbeitung des Gesamtdatensatzes einfügt[12], ich habe mich aber dazu entschlossen, als Dateiformat Pickle zu verwenden. Dieses erzeugt zwar größere Dateien, der Zugriff erfolgt aber noch mit guter Geschwindigkeit und das Zeitformat wird korrekt wiedergegeben.

```
# %% Testweises Abspeichern
df2.info()
df2.to_parquet("example.parquet", engine="fastparquet", index=True)


# %% Testweises Laden
df3 = pd.read_parquet("example.parquet")
df3.info()
```

Da der kumulierte Verbrauch im neuen System auf zwei Spalten aufgeteilt war, mussten diese zusammengeführt werden. Ansonsten war die Vorgehensweise genau wie im vorigen Abschnitt beschrieben.

Für die Verbindung aller Dateien im jeweils alten bzw. neuen System entschloss ich mich, ein neues Script zu erstellen und die dafür notwendigen Schritte aus den vorigen Scripts entsprechend zu kopieren und als Funktionen zu deklarieren.

12 Im Nachhinein ist man immer schlauer: Beim nächsten Mal werde ich den Grunddatensatz vor dem Import so einfach wie möglich halten und dann erst auf dem importierten die Erweiterungen durchführen. Aber es gibt ja von jeder Software eine Version 2.0.

Eine Liste aller im jeweiligen Backup-Ordner befindlichen Dateien erhielt ich mit Hilfe des Packages glob:

```
# Import
# import numpy as np
import datetime as dt
import pandas as pd
import pyarrow
import glob
from os.path import exists

# %% Erstellen einer Liste aller Backup-Dateien

data_files = glob.glob("./Backups_old/*")
```

Diese Liste wurde nun abgearbeitet und dabei

- die CSV-Datei eingelesen
- der Dataframe gesäubert
- und geschrieben.

```
def read_data():
    try:
        # return pd.read_parquet("solar_data_01.parquet")
        return pd.read_pickle("solar_data_01.pkl")
    except:
        pass

# %%


def combine_and_clean_data(df1: pd.DataFrame(), df2: pd.DataFrame()):
    df_con = pd.concat([df1, df2])
    # Dups rauswerfen
    df_clean = df_con.drop_duplicates()
    return df_clean


# %%

def write_data(df):
    # df.to_parquet("solar_data_01.parquet", engine="pyarrow", index=True)
    df.to_pickle("solar_data_01.pkl")
```

Beim Schreiben musste beachtet werden, dass man an Pickle-Dateien anscheinend nicht einfach Daten anhängen kann, sondern dieser wurde eingelesen, mit den neuen Daten kombiniert und wieder abgespeichert. In diesem Schritt wurden auch eventuelle Duplikate durch Überschneidungen in den Backups entfernt und der Gesamtdatensatz wieder nach dem Datum sortiert[13].
Deswegen erfolgte zuerst eine Abfrage, ob die Datei bereits existiert. Wenn nicht, konnte direkt geschrieben werden (1. Teil), wenn doch (es bestehen schon Daten), musste das Prozedere für das Anhängen abgearbeitet werden.

Am Ende wurde zur Überprüfung der gesamte Datensatz eingelesen.

13 Bedingt durch die Dateinamen war die Einlesereihenfolge der Dateien nicht chronologisch.

```
# Jetzt geht's los!
counter = 0
for file in data_files:
    # if counter == 1:
    #     break
    counter += 1
    #     break
    # Einlesen
    df0 = read_dat_file(file)
    # SÃ¤ubern
    df1 = clean_up(df0)
    # Falls noch nichts angelegt, schreiben, ...
    if not exists("solar_data_01.pkl"):
        write_data(df1)
    # ... ansonsten lesen, anhÃ¤ngen, dups entfernen und wieder schreiben
    else:
        df2 = read_data()
        df_clean = combine_and_clean_data(df1, df2)
        df_clean.sort_index(inplace=True)
        write_data(df_clean)
        # counter += 1


df4 = read_data()

# df0 = read_dat_file(data_files[1])
# df2 = clean_up(df0)
# df3 = write_data(df2)
```

Der Counter war nur für Überprüfungszwecke, um zu sehen, ob alle Dateien eingelesen wurden.

Beim neuen System war die Vorgehensweise identisch.

Die beiden erhaltenen Dateien wurden nun nach bewährtem Vorgehen verheiratet: Einlesen, Aneinanderhängen und Abspeichern. Ein zeitliches Sortieren fand nicht statt, da gewährleistet werden konnte, dass die erste eingelesene Datei die mit dem älteren Datensatz war. Sicherheitshalber wurde auf Duplikate untersucht.

```
import pandas as pd

# %%


df1 = pd.read_pickle("solar_data_01.pkl")

df2 = pd.read_pickle("solar_data_02.pkl")

df_con = pd.concat([df1, df2])
df_clean = df_con.drop_duplicates()

df_clean.to_pickle("solar_data.pkl")
```

An der Schnittstelle gab es eine kleine Lücke in den Daten, da in dieser Zeit ein paar Backups ein seltsam gemischtes Format hatten. Es handelte sich jedoch nur um ein paar

Tage und so hielt ich den Aufwand, diese auch noch genauer zu untersuchen, für nicht erstrebenswert im Verhältnis zur Gesamtmenge der Daten.
Es ergab sich noch eine kleine Lücke in den Daten vom 29.10.2018 bis zum 7.11.2018, da in dieser Zeit die Backup-Dateien leider nicht vollständig vom Server übertragen wurden.

Der Datensatz war vollständig, die Analyse konnte beginnen.

2.3.2 Grafische Aufbereitung der Daten für Tage, Monate und Jahre

Das erste Ziel war es, mit relativ geringem Aufwand aus den Daten Grafiken zu erstellen, die einen Überblick über den Ertrag an einem bestimmten Tag, in einen bestimmten Monat oder Jahr oder über mehrere Jahre geben.

Dazu wurde – nach den entsprechenden Imports – das pickle-File in einen Dataframe eingelesen.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import datetime


# %% Das große File einlesen

df0 = pd.read_pickle("solar_data.pkl")

df0.info()

df0.describe()
```

```
In [168]: df0.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 971051 entries, 2014-01-02 00:00:00 to 2023-04-13 10:35:00
Data columns (total 12 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   Verbrauch       971051 non-null  int32
 1   Verbrauch_kum   971051 non-null  int32
 2   Ertrag_gesamt   971051 non-null  int32
 3   Ertrag_S1       971051 non-null  int32
 4   Ertrag_S2       971051 non-null  int32
 5   Ertrag_kum      971051 non-null  int32
 6   day             971051 non-null  int64
 7   month           971051 non-null  int64
 8   year            971051 non-null  int64
 9   time            971051 non-null  object
 10  weekday_number  971051 non-null  int64
 11  weekday         971051 non-null  object
dtypes: int32(6), int64(4), object(2)
memory usage: 106.3+ MB
```

2.3.2.1 Erstellung einer Tagesübersicht

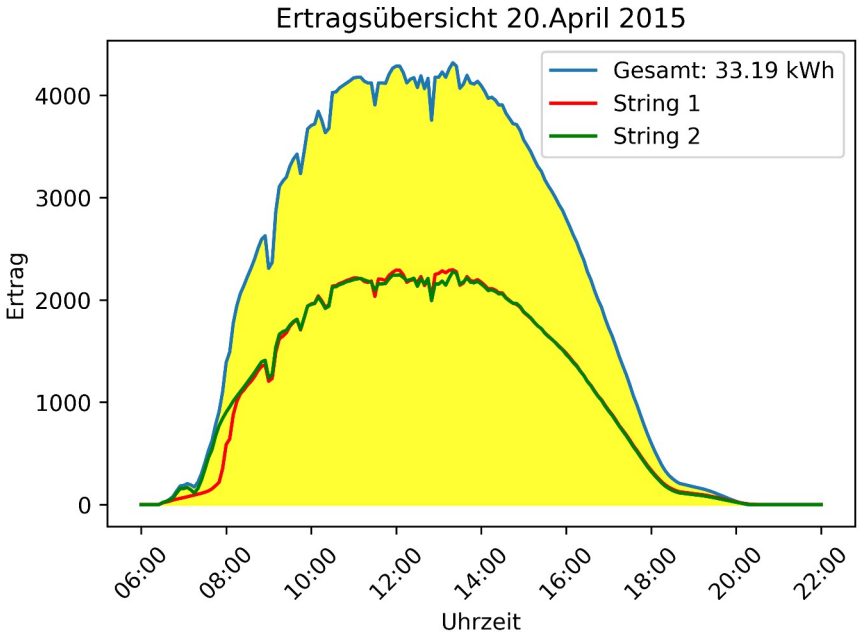
Es sollte eine Übersicht über den Verlauf des Ertrages eines Tages ausgegeben werden. Dazu wurde der Datensatz auf einen Tag und dann noch auf die Stunden von 6:00 Uhr bis 22:00 Uhr begrenzt[14].
Der Gesamtertrag wurde als Maximum der Spalte „Ertrag_kum“ ausgelesen.
Danach wurde

- der Plot des Ertrags und der Strings erstellt
- die Beschriftung der x-Achse angepasst (die Zeiten waren ohne die Anpassung nicht lesbar)
- der Platz zwischen dem Graphen und der Achse farblich gefüllt
- der Titel und die Achsenbeschriftungen hinzugefügt
- die Legende, die den Gesamtertrag (auf zwei Nachkommastellen dargestellt) beinhaltet, hinzugefügt
- der Plot ausgegeben und als png abgespeichert

```
# ================================================================================
# Erster Plot eines guten Tages, Ertrag im Tagesverlauf
# ================================================================================

df1 = df0.loc["2015-04-20"]


# Eingrenzen auf den interessanten Bereich
df1_filtered = df1.between_time("06:00:00", "22:00:00")

# Berechnen des Ertrages
ertrag = df1_filtered["Ertrag_kum"].max()/1000

# plot
fig, ax = plt.subplots()

# Linienplot
ax.plot(df1_filtered.index, df1_filtered['Ertrag_gesamt'],
        label=f"Gesamt: {ertrag:.2f} kWh")

# Strings
ax.plot(df1_filtered.index, df1_filtered['Ertrag_S1'],
        label="String 1", color="red")
ax.plot(df1_filtered.index, df1_filtered['Ertrag_S2'],
        label="String 2", color="green")

# Konvertieren der x-Achsen-Daten, damit die Uhrzeiten sauber dargestellt werden
myFmt = mdates.DateFormatter('%H:%M')
ax.xaxis.set_major_formatter(myFmt)
plt.xticks(rotation=45)

# Ausfüllen des Zwischenraums
ax.fill_between(df1_filtered.index,
                df1_filtered['Ertrag_gesamt'], color='yellow', alpha=0.8)

# Titel und Achsenbeschriftungen
plt.title("Ertragsübersicht 20.April 2015")
ax.set(xlabel='Uhrzeit', ylabel='Ertrag')
ax.legend(loc=1)

plt.show()

# Abspeichern als png
fig.savefig("01 Tagesverlauf.png", dpi=600, bbox_inches='tight')
```

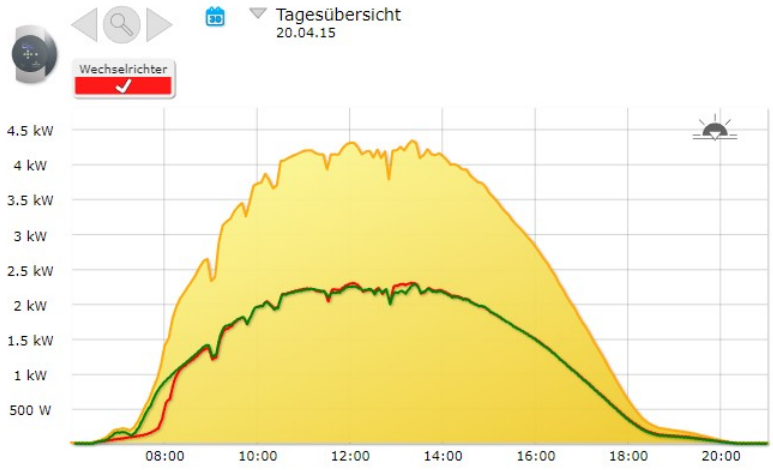
14 Da im Sommer auch schon um 6 Uhr Ertragswerte auftreten, wäre es im Allgemeinen besser, das Intervall früher starten zu lassen. Für ein Datum im April erschien mir dieser Zeitraum allerdings geeignet.

Das Ergebnis ist das folgende:



Die Verläufe von String 1 und String 2 sind nahezu identisch, der Unterschied zu Beginn liegt daran, dass damals morgens ein Teil der Anlage von einer kleinen Konifere des Nachbars beschattet wurde.

Zum Vergleich die Ausgabe vom Solar-Log:



Man erkennt, dass hier mit relativ geringem Aufwand[15] schon ein sehr ähnliches Ergebnis erzielt werden kann.

15 Abgesehen von der Erstellung des Datensatzes

2.3.2.2 Versuch einer Fit-Kurve

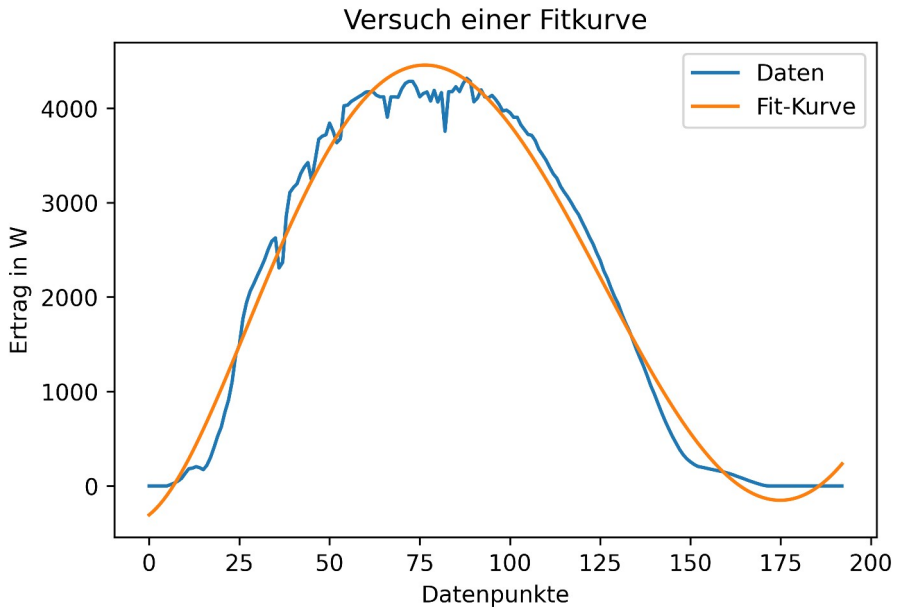
Als nächstes versuchte ich mich daran, eine Fit-Kurve durch den Tagesdatensatz zu legen. Ich benutzte eine Funktion vierten Grades. Eigentlich wäre eine Sinus-Kurve[16] besser geeignet, aber dazu hätte ich scikit o. ä. verwenden müssen, was mir an dieser Stelle ein zu hoher Aufwand erschien. Eine Funktion n-ten Grades hat – einfach ausgedrückt – höchstens n-1 Krümmungen. Da zu Beginn und am Ende des Tages genauso wie in der Mitte jeweils eine Krümmung besteht, musste mindestens eine Funktion vierten Grades benutzt werden.

```
# %% Versuch einer polynimial regression

df1 = df0.loc["2015-04-20"]
df1_filtered = df1.between_time("06:00:00", "22:00:00")

# Index Reset, weil das Fitten ein Problem mit DateTime hat.
df2 = df1_filtered.reset_index()

# Berechnung Koeffizienten des Polynoms
z = np.polyfit(df2.index, df2["Ertrag_gesamt"], 5)

# Erstelllung des Polynomobjekts
p = np.poly1d(z)

fig, ax = plt.subplots()

# Linienplot
ax.plot(df2.index, df2['Ertrag_gesamt'], label="Daten")

# Plot der Fit-Kurve
ax.plot(df2.index, p(df2.index), label="Fit-Kurve")

# Titel, Achsen und Legende
ax.set(xlabel="Datenpunkte", ylabel="Ertrag in W")
plt.title("Versuch einer Fitkurve")
ax.legend()

plt.show()

# Abspeichern als png
fig.savefig("01a Tagesverlauf fit.png", dpi=600, bbox_inches='tight')
```

Für den Fit musste ich die polyfit-Funktion von Numpy verwenden, die anscheinend mit DateTime-Indizes nicht so gut zurecht kommt. Deswegen wurde hier der Index zurückgesetzt. Danach wurden die Koeffizienten berechnet und das Objekt erstellt. Dieses wurde dann zusammen mit den Datenpunkten ausgegeben und abgespeichert.

Das Ergebnis:

16 Wahrscheinlich eher $\sin^2$



Der Verlauf der Daten wird durch die Fit-Kurve relativ gut angenähert, für weitere Untersuchungen könnte man nun die Parameter der Fit-Funktion analysieren.

2.3.2.3 Ertrag über einen Monat

Um ein Säulendiagramm mit dem Ertrag eines Monats zu bekommen, ging ich folgendermaßen vor:
Als erstes wurde der Datensatz auf Tage reduziert, da die Berechnungen für die weiteren Diagramme in diesem Abschnitt dann einfacher zu erstellen sind.
Dazu wurde ein neuer Datensatz erzeugt, der von 2015 bis 2022 geht, da ich nur von diesen Jahren (nahezu) vollständige Datensätze habe.
Eine neue Spalte „Tag“ wurde eingefügt, die nur das Datum des jeweiligen Tages beinhaltet. Danach wurden auf die Maximalwerte des kumulierten Ertrags gruppiert und die Tagesspalte zum Index gemacht.
Es wurden auch noch Spalten für den Wochentag, den Monat und das Jahr für eventuelle weitere Untersuchungen eingefügt.

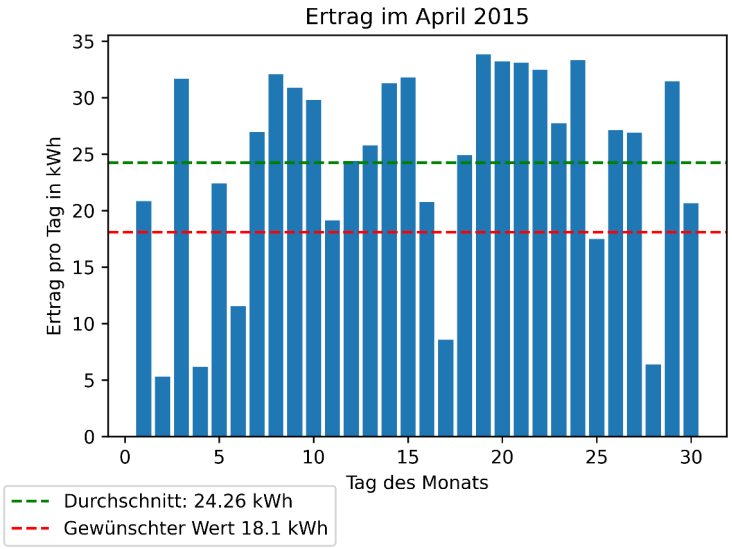
```
# %%
# Jeden Tag der maximale Ertrag 2015 bis 2020
df_cut = df0.loc["2015":"2020"]
# Tag Spalte für die Unterscheidung nach Tagen
df_cut['Tag'] = df_cut.index.date
# Gruppierung nach Tagen
df_Ertrag_daily = df_cut.groupby('Tag')['Ertrag_kum'].max().reset_index()
# Tag als Index
df_Ertrag_daily.set_index('Tag', inplace=True)
# Noch ein paar Spalten, die nützlich sein könnten
df_Ertrag_daily['Datum'] = pd.to_datetime(df_Ertrag_daily.index)
df_Ertrag_daily['Monat'] = df_Ertrag_daily['Datum'].dt.month
df_Ertrag_daily['Jahr'] = df_Ertrag_daily['Datum'].dt.year


df_Ertrag_daily.info()
```

Dieser neu angelegte Datensatz wurde auf einen Monat (April 2015) begrenzt und der Index angepasst, um auf der x-Achse nur die Tage angezeigt zu bekommen.

Der Durchschnittswert wurde für statistische Zwecke berechnet:
Es gibt für jeden Monat im Jahr einen – von der maximalen Leistung der Anlage abhängigen – Wert, der erreicht werden sollte, um lukrativ zu sein. Für meine Anlage liegt der Wert im April bei 18,1 kWh pro Tag.
Nun wurde der Plot erstellt, die beiden Linien mit dem Durchschnittswert und dem gewünschten Wert eingezeichnet, die Achsenbeschriftungen, der Titel und die Legende angelegt und das Diagramm gezeichnet und abgespeichert. Bei der Ausgabe der Legende wurde der Durchschnittswert noch auf 2 Nachkommastellen formatiert.

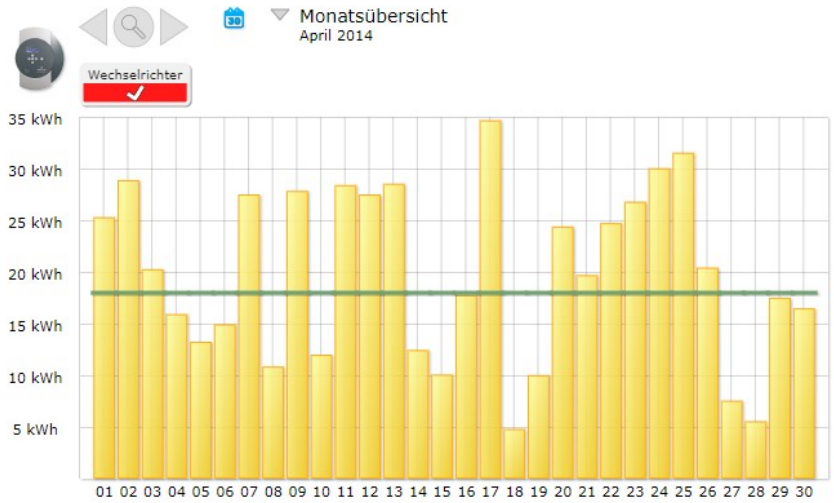
```
# ================================================================================
# Ertrag über einen Monat hinweg
# ================================================================================

# Begrenzen auf einen Monat
start_date = datetime.date(2015, 4, 1)
end_date = datetime.date(2015, 4, 30)
df_month = df_Ertrag_daily.loc[start_date:end_date]

# Datum umwandeln, damit man auf der x-Achse nur die Tage hinbekommt
df_month.index = pd.to_datetime(df_month.index)

# Durchschnitt berechnen
mean_max_val = df_month.Ertrag_kum.mean()/1000

# Plot
fig, ax = plt.subplots()

# Balkendiagramm
ax.bar(df_month.index.day, df_month['Ertrag_kum']/1000)

# Durschnittslinie
ax.axhline(mean_max_val, color='green', linestyle='--',
           label=f"Durchschnitt: {mean_max_val:.2f} kWh")
ax.axhline(18.1, color='red', linestyle='--',
           label="Gewünschter Wert 18.1 kWh")

# Achsen, Titel und Legende
ax.set_xlabel('Tag des Monats')
ax.set_ylabel('Ertrag pro Tag in kWh')
ax.set_title('Ertrag im April 2015')
legend = ax.legend(loc='upper center', bbox_to_anchor=(0.1, -0.1))

plt.show()

fig.savefig("03 Monatsverlauf.png", dpi=600, bbox_inches='tight')
```

Das Ergebnis:



Man erkennt sofort, dass der April 2015 ein außergewöhnlich guter Monat war, was den Ertrag angeht.

Zum Vergleich die Ausgabe von Solar-Log:



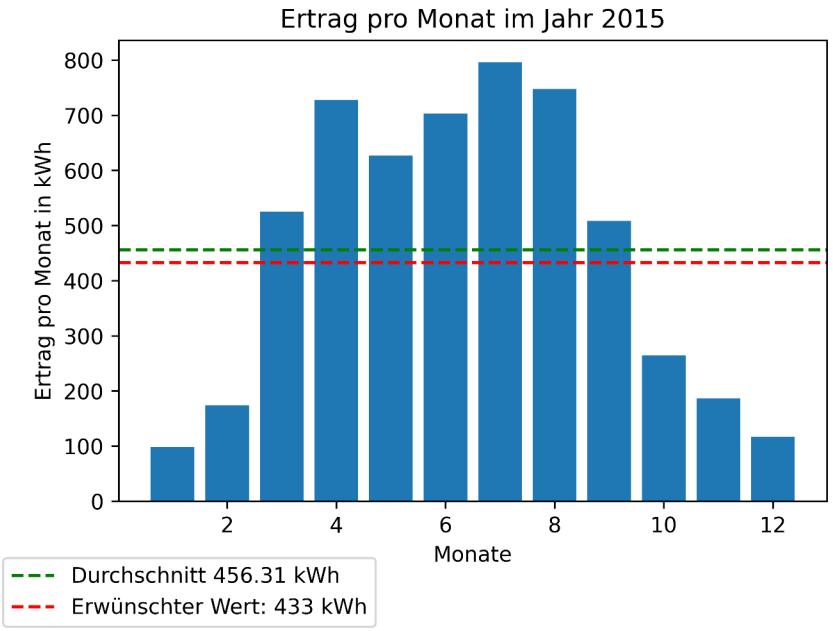
2.3.2.4 Ertrag über ein Jahr

Für die Übersicht über 2015 konnte der oben erstelle Datensatz weiterverwendet werden[17]. Dafür fand eine Beschränkung auf das Jahr 2015 statt und monatlich mit der Summe über die Tageserträge gruppiert. Der Durchschnitt über die Monate wurde für die Darstellung im Diagramm berechnet.
Danach wurde wie immer der Plot mit seinen Einzelheiten erzeugt und abgespeichert.

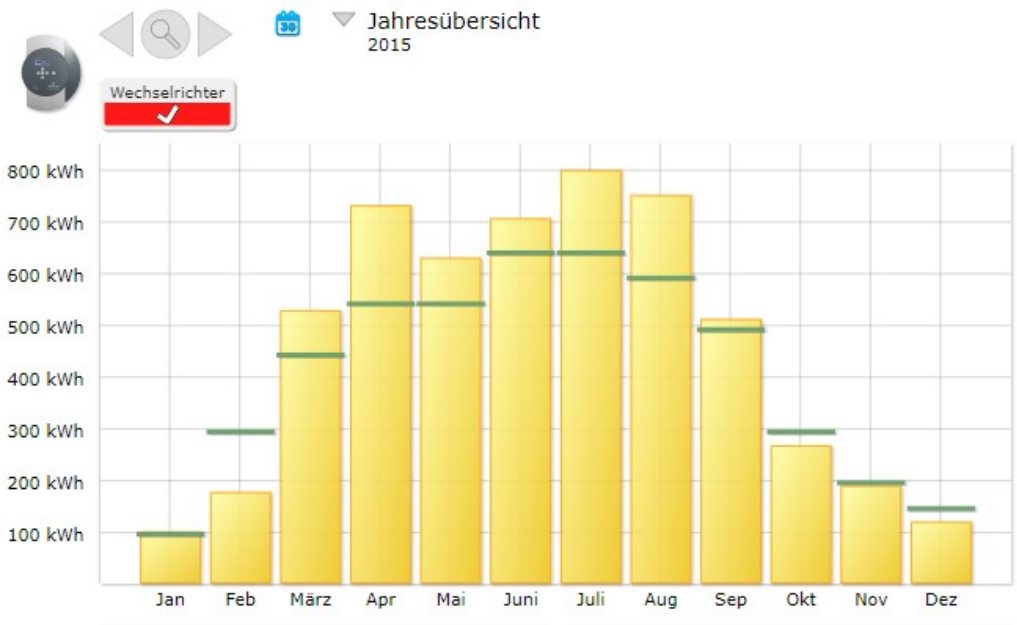
```
# ================================================================
# Ertrag über das Jahr 2015 hinweg
# ================================================================

# df eingrenzen
df_year = df_Ertrag_daily[df_Ertrag_daily['Jahr'] == 2015]

# df gruppieren und summieren
df_monthly = df_year.groupby('Monat')['Ertrag_kum'].sum()

# Durchschnit berechnen
avg_val = df_monthly.mean()/1000


# Plot
fig, ax = plt.subplots()

# Säulendiagramm
ax.bar(df_monthly.index, df_monthly/1000)
# Durchschnittslinie
ax.axhline(avg_val, color='green', linestyle='--',
           label=f"Durchschnitt {avg_val:.2f} kWh")
ax.axhline(433, color='red', linestyle='--',
           label="Erwünschter Wert: 433 kWh")

# Achsen, Titel und Legende
ax.set_xlabel('Monate')
ax.set_ylabel('Ertrag pro Monat in kWh')
ax.set_title('Ertrag pro Monat im Jahr 2015')
legend = ax.legend(loc='upper center', bbox_to_anchor=(0.1, -0.1))


plt.show()
fig.savefig("04 Jahresverlauf.png", dpi=600, bbox_inches='tight')
```

17 Zu Beginn habe ich versucht, alle Diagramme mit dem ersten kompletten Datensatz zu erstellen, der das 5-Minuten-Raster hatte. Das wurde zunehmend komplizierter. Deswegen der neu erstellte Datensatz.

Das Ergebnis:



Im Vergleich dazu das Diagramm von Solar-Log:



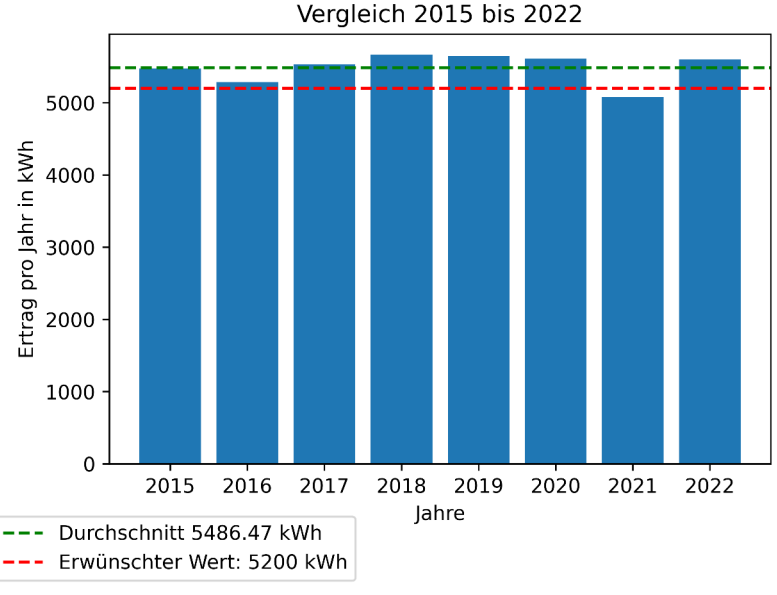
Hier sind die gewünschten Mengen pro Monat angegeben, was ebenfalls nicht schlecht ist, ein Gesamtdurchschnitt fehlt aber. Die Monats-Wunschwerte könnten noch mit nicht zu hohem Aufwand in das Diagramm eingebaut werden (axhline).

2.3.2.5 Vergleich der Jahre

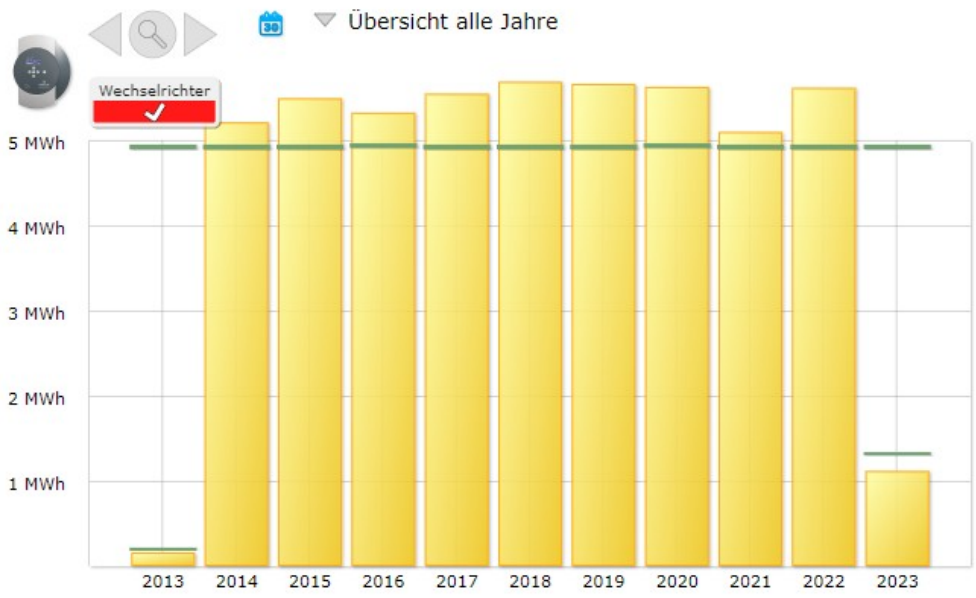
Als letztes Diagramm in diesem Abschnitt sollte nun ein Vergleich der Jahre 2014 bis 2022 erstellt werden. Eine Begrenzung des Ausgangsdatensatzes war nicht notwendig, es fand nur eine Gruppierung auf Jahre mit der Summe der kumulierten täglichen Erträge statt.
Anschließend wurde wieder der Durchschnitt berechnet, um ihn mit dem angestrebten Wert von 5200 kWh vergleichen zu können.
Der Plot wurde mit seinen Einzelheiten (Säulendiagramm, waagrechte Linien für Durchschnitt und Wunschwert, Achsenbeschriftungen, Legenden und Titel) erstellt und abgespeichert.

```
# %%

# ============================================================================
# Vergleich von mehrern Jahren
# ============================================================================

# df gruppieren und summieren
df_yearly = df_Ertrag_daily.groupby('Jahr')['Ertrag_kum'].sum()

# Durchschnitt
avg_yearly = df_yearly.mean()/1000

# Plot
fig, ax = plt.subplots()

# Säulendiagramm
ax.bar(df_yearly.index, df_yearly/1000)

# Durschnittslinie
ax.axhline(avg_yearly, color='green', linestyle='--',
           label=f"Durchschnitt {avg_yearly:.2f} kWh")

ax.axhline(5200, color='red', linestyle='--',
           label="Erwünschter Wert: 5200 kWh")

# Achsen, Titel und Legende
ax.set_xlabel('Jahre')
ax.set_ylabel('Ertrag pro Jahr in kWh')
ax.set_title('Vergleich 2015 bis 2022')
legend = ax.legend(loc='upper center', bbox_to_anchor=(0.1, -0.1))


plt.show()

fig.savefig("05 Jahresvergleich.png", dpi=600, bbox_inches='tight')
```

Das Ergebnis:



Im Vergleich dazu das Diagramm von Solar-Log:



Hier sieht man noch, dass bei den Jahren 2013 und 2023 eine zusätzliche Markierung zu sehen ist, die den gewünschten Wert auf den momentanen Zeitpunkt herunterrechnet. Da ich keine „unvollendeten“ Jahre darstellte, war dies bei mir aber hinfällig[18].
Solar-Log geht von einem leicht kleineren angestrebten Wert aus, der knapp unterhalb von 5 MWh pro Jahr liegt.

2.3.3 Untersuchung weiterer interessanter Aspekte

Es gäbe sicher vieles, was man aus den Daten herauslesen könnte, mich aber interessierten zwei Dinge:

- Gibt es einen Wochentag, der besonders ergiebig ist?
- Gibt es einen Kalendertag im Monat, der besonders ergiebig ist?

2.3.3.1 Untersuchung bezüglich der Wochentage

Für diese Untersuchung wurde eine neue Spalte mit dem Namen des Wochentags in den vorbereiteten Dataframe, der bereits seit der Untersuchung der Tage in einem Monat Verwendung findet, eingepflegt.

18 Die Formel dafür ist ziemlich einfach: 5200 kWh / 365 · t, wenn t die Anzahl der vergangenen Tage ist.

```
# %%
# Für den Vergleich von Wochentagen
df_wochentag = df_Ertrag_daily.copy()
# Wochentag-Spalte einfügen
df_wochentag["Wochentag"] = df_Ertrag_daily["Datum"].dt.strftime("%A")
```

Daraufhin wurden die Werte nach dieser Spalte gruppiert und der Mittelwert für jeden Wochentag ermittelt. Der Mittelwert sollte die saisonalen Unterschiede herausfiltern und – falls es eine geringe Abweichung in der Anzahl der jeweiligen Wochentage gäbe – noch verbleibende Unterschiede ausmitteln. Die Stichprobe sollte groß genug sein, um eine Aussage machen zu können: Grob sollte jeder Wochentag $8 \cdot 365 : 7 \approx 417$ mal vorkommen. Auch Effekte wie zeitliche oder örtliche teilweise Abschattung spielten keine Rolle, da man sich ja die gesamten Tageserträge anschaut und nicht irgendwelche Teile davon oder die Erträge der einzelnen Strings. Ob eine teilweise oder komplette Schneebedeckung bestimmte Tage bevorzugte, hätte eine Rolle spielen können, da dann anhand des Ertrages nicht entscheiden werden kann, ob gutes oder schlechtes Wetter herrscht.

Ein Problem ergab sich dadurch, dass die Wochentage, die den Index darstellten, alphabetisch sortiert waren. Also musste der Index zurückgesetzt und die Wochentage in die korrekte Reihenfolge gebracht werden.[19]

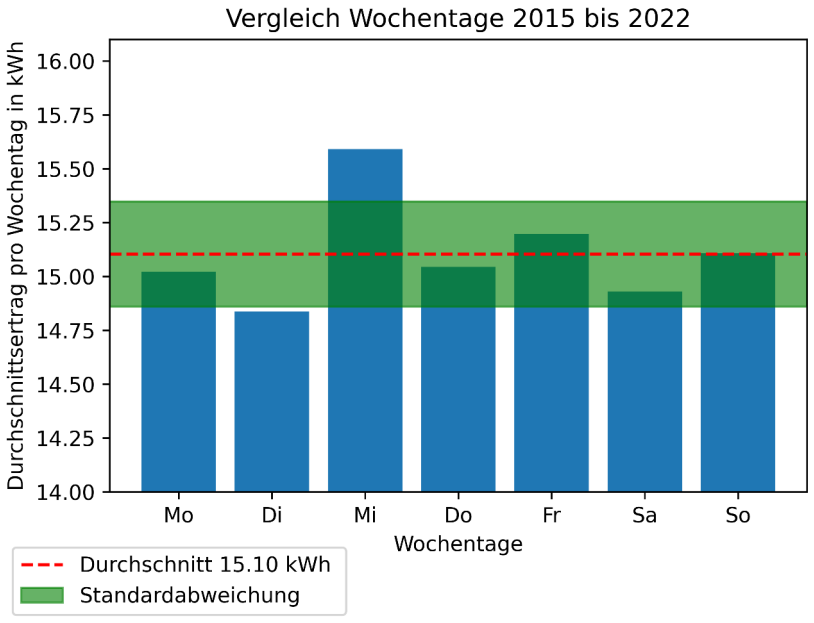
Anschließend wurden der Mittelwert und die Standardabweichung für die Darstellung im Diagramm berechnet. Das Diagramm wurde in bereits bekannter Weise erstellt, die Darstellung der Wochentage auf der x-Achse wurde durch die gebräuchlichen Abkürzungen ersetzt und der Bereich der y-Achse eingegrenzt, um Unterschiede besser sehen zu können.

Am Ende wurde wie immer das Diagramm dargestellt und abgespeichert.

19 Im Nachhinein würde ich die Wochentage als Zahl belassen und dann die Achsenbeschriftung anpassen.

```python
# ======================================================================
# Vergleich der Wochentage über die Jahre 2015 bis 2022
# ======================================================================

# Dataframe day of week

df_dow = df_wochentag.groupby("Wochentag")["Ertrag_kum"].mean()

# Sortieren nach der korrekten Reihenfolge
df_dow = df_dow.reset_index()
cats = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday', 'Saturday', 'Sunday']
df_dow['Wochentag'] = pd.Categorical(
    df_dow['Wochentag'], categories=cats, ordered=True)
df_dow = df_dow.sort_values('Wochentag')


# Durchschnitt und Standardanweichung
avg_dow = df_dow["Ertrag_kum"].mean()/1000
std_dow = df_dow["Ertrag_kum"].std()/1000

# Plot
fig, ax = plt.subplots()

# Säulendiagramm, Durchschnitt und Bereich darum
ax.bar(df_dow["Wochentag"], df_dow["Ertrag_kum"]/1000)
ax.axhline(avg_dow, color='red', linestyle='--',
           label=f"Durchschnitt {avg_dow:.2f} kWh ")

ax.axhspan(avg_dow - std_dow, avg_dow + std_dow, color='green',
           alpha=0.6, label="Standardabweichung")

# Achsenbegrenzung zur besseren Sichtbarkeit
ax.set_ylim([14, 16.1])

# Achsen, Titel und Legende
ax.set_xticklabels(["Mo", "Di", "Mi", "Do", "Fr", "Sa", "So"])
ax.set_xlabel('Wochentage')
ax.set_ylabel('Durchschnittsertrag pro Wochentag in kWh')
ax.set_title('Vergleich Wochentage 2015 bis 2022')
legend = ax.legend(loc='upper center', bbox_to_anchor=(0.1, -0.1))

plt.show()

fig.savefig("06 Wochentage.png", dpi=600, bbox_inches='tight')
```

Das zugehörige Diagramm bringt eine Überraschung zu Tage:



Der Mittwoch ist signifikant besser beim Ertrag als die anderen Wochentage. Man sieht, dass er um fast zwei Standardabweichungen vom Mittelwert abweicht. Der Dienstag hingegen ist der schlechteste Wochentag, er ist mehr als eine Standardabweichung unter dem Mittelwert.
Ich wüsste keine Begründung für dieses Ergebnis, habe auch keine dazu passenden Artikel oder Informationen gefunden.

2.3.3.2 Untersuchung bezüglich der Tage im Monat

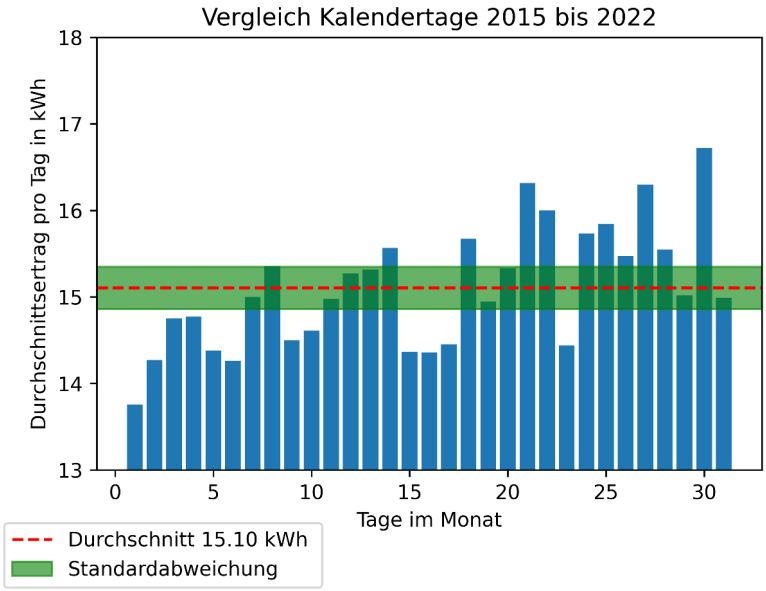
Von diesem Ergebnis angestachelt wollte ich nun noch sehen, ob es einen bestimmten Tag im Monat gibt, der ähnlich heraussticht.
Dafür wurde wieder der Datensatz df_Ertrag_daily angepasst, indem er eine neue Spalte „MT“ für die Monatstage erhielt.

```
# %%
# Für Vergleich der Monatstage
df_monatstag = df_Ertrag_daily.copy()
# Spalte "MT" gibt den Tag im Monat an
df_monatstag["MT"] = df_Ertrag_daily["Datum"].dt.day
```

Das Vorgehen ist analog zum vorigen Abschnitt.

```
# ================================================================
# Vergleich der Kalendertage in den Monaten
# ================================================================

df_dom = df_monatstag.groupby("MT")["Ertrag_kum"].mean()

avg_dom = df_dom.mean()/1000

std_dom = df_dom.std()/1000

# Plot
fig, ax = plt.subplots()

# Säulendiagramm, Durchschnitt und Bereich darum
ax.bar(df_dom.index, df_dom/1000)
ax.axhline(avg_dow, color='red', linestyle='--',
           label=f"Durchschnitt {avg_dow:.2f} kWh ")

ax.axhspan(avg_dow - std_dow, avg_dow + std_dow, color='green',
           alpha=0.6, label="Standardabweichung")

# Achsenbegrenzung zur besseren Sichtbarkeit
ax.set_ylim([13, 18])

# Achsen, Titel und Legende
ax.set_xlabel('Tage im Monat')
ax.set_ylabel('Durchschnittsertrag pro Tag in kWh')
ax.set_title('Vergleich Kalendertage 2015 bis 2022')
legend = ax.legend(loc='upper center', bbox_to_anchor=(0.1, -0.1))

plt.show()

fig.savefig("07 Monatstage.png", dpi=600, bbox_inches='tight')
```

Das Ergebnis:



Hier zeigt sich ein sehr uneinheitliches Bild. Vielleicht liegt es daran, dass die Stichprobengröße für diese Art der Untersuchung nicht ausreicht: $365 \cdot 8 : 30 \approx 97$ Items pro Ausprägung. An der unterschiedlichen Länge der Monate kann es nicht liegen, da es sich ja um Mittelwerte handelt und nicht um Summen. Auch saisonale Unterschiede kommen nicht in Frage, denn warum sollte am 1. Dezember, Januar, Februar, … ein stark anderes Wetter als am 30. Tag des Vormonats sein.
Nichtsdestotrotz, die Daten legen zumindest für meine Solaranlage für die letzten 8 Jahre nahe, dass der 30. Tag eines Monats statistisch den höchsten Ertrag liefert und der erste statistisch am schlechtesten ist.

2.3.4 Untersuchung der Strings auf Unterschiede

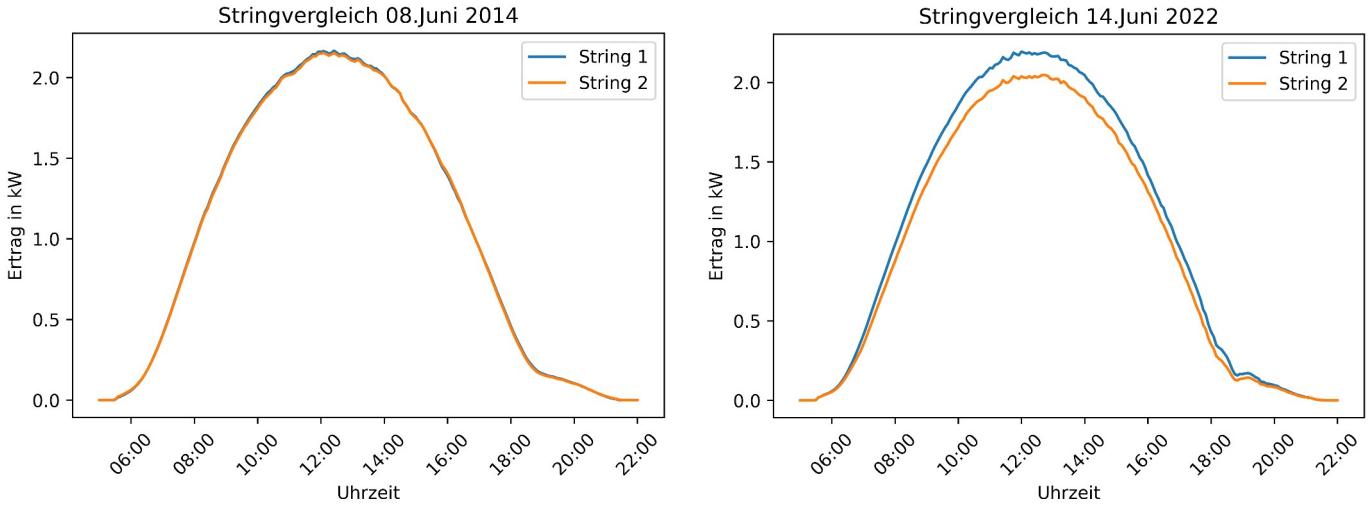
Im allerersten Diagramm wurden die Erträge der Strings mit dargestellt. Eine leichte Abweichung war aufgrund einer Abdeckung erkennbar. Da die Module mit der Zeit an Leistung verlieren, interessierte ich mich noch dafür, ob eventuell in den beiden Strings Unterschiede in der Degeneration erkennbar wären.
Da Unterschiede am besten bei höherer Leistung erkennbar sind, suchte ich zwei Tage im Juni aus, da zu dieser Zeit die Einstrahlung im besten Winkel erfolgt: 08.06.2014 und 14.06.2022

Der dazugehörige Code sieht folgendermaßen aus:

```
# %%

# ================================================================================
# String-Vergleichsplot
# ================================================================================

# Auswahle eines "guten" Tags mit viel Sonne

# df3 = df0.loc["2014-06-08"]
df3 = df0.loc["2022-06-14"]

# Eingrenzen auf den interessanten Bereich
df3_filtered = df3.between_time("05:00:00", "22:00:00")

# Plot erstellen

fig, ax = plt.subplots()

ax.plot(df3_filtered.index, df3_filtered['Ertrag_S1']/1000, label="String 1")
ax.plot(df3_filtered.index, df3_filtered['Ertrag_S2']/1000, label="String 2")

# Konvertieren der x-Achsen-Beschriftung in das richtige Format
myFmt = mdates.DateFormatter('%H:%M')
ax.xaxis.set_major_formatter(myFmt)
plt.xticks(rotation=45)
# plt.title("Stringvergleich 08.Juni 2014")
plt.title("Stringvergleich 14.Juni 2022")
plt.legend()
ax.set(xlabel='Uhrzeit', ylabel='Ertrag in kW')

plt.show()

# Abspeichern als png
# fig.savefig("02a Stringvergleich 2014.png", dpi=600, bbox_inches='tight')
fig.savefig("02b Stringvergleich 2022.png", dpi=600, bbox_inches='tight')
```

Der Datensatz wurde auf das entsprechende Datum und auf die Zeit zwischen 5 Uhr und 22 Uhr begrenzt (zwischen den Daten wurde mit Auskommentieren gewählt). Der Plot wurde mit Titel, Legende und Achsenbeschriftungen erstellt, ausgegeben und abgespeichert, nachdem die x-Achsenbeschriftung angepasst wurde.

Die beiden Diagramme:



Zu erkennen ist ein deutlicher Unterschied zwischen den Erträgen der Strings im Jahr 2022 im Vergleich zu 2014!

Um genauer zu untersuchen, ob und wie unterschiedlich die Strings gealtert sind[20] wurde ein neuer Dataframe erstellt.
Dabei wurde der Ausgangs-Dataframe auf die Zeit zwischen 10 Uhr und 16 Uhr und auf die Monate Mai bis August beschränkt, um eine eventuelle Abschattung eines Strings durch Bäume (Zeitbeschränkung) zu verhindern und maximale Einstrahlungsleistung (Monatsbeschränkung) zu gewährleisten. Der Index musste nachträglich wieder sortiert werden, da ansonsten erste alle Mai-Monate kommen, dann alle Juni-Monate usw.. Unnötige Spalten wurden entfernt und eine Spalte mit den Ertragsdifferenzen erstellt.

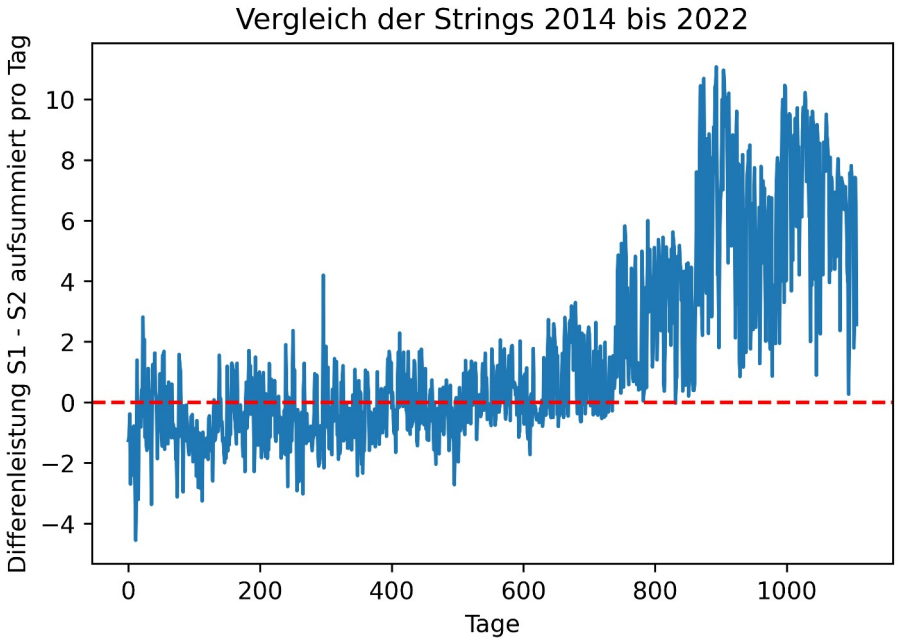
```
# %%
# Vergleich der Strings
# Beschränkung auf die Zeiten zwischen 10 Uhr und 16 Uhr
# 10 Uhr um Beschattung auszuschließen
start_time = datetime.time(10, 0)
end_time = datetime.time(16, 0)
df_time = df0.between_time(
    start_time, end_time, inclusive="both")

# Beschränkung auf die Monate mit der besten Einstrahlung,
# hier werden Unterschiede am deutlichsten.
df_strings = pd.concat([df_time.loc[df_time.index.month == 5],
                        df_time.loc[df_time.index.month == 6],
                        df_time.loc[df_time.index.month == 7],
                        df_time.loc[df_time.index.month == 8]])

# Sortieren wieder nach Jahren
df_strings = df_strings.sort_index()
# Löschen der nicht gebrauchten Spalten
df_strings = df_strings.drop(
    columns=["Verbrauch", "Verbrauch_kum", "weekday", "weekday_number"])

# Differenzspalte: 1-2
df_strings["Diff"] = df_strings["Ertrag_S1"] - df_strings["Ertrag_S2"]

df_strings.info()
```

Zur Erzeugung des Diagramms wurden die Daten nun nach Tagen gruppiert und dabei die Summe der Erträge innerhalb der Strings gebildet.
Da beim Resample die fehlenden Tage eingefügt werden, musste man diese wieder entfernen. Der Index wurde zurückgesetzt, damit zwischen den Monaten August und Mai keine entsprechend großen Lücken auf der x-Achse entstehen. Dadurch wurde die x-Achse zwar einfach nur in Tagen gezählt, das spielte für die Auswertung jedoch keine Rolle. Der Plot wurde in gewohnter Weise erstellt und zusätzlich eine Nulllinie zur Verdeutlichung eingefügt.

20 Mit den zur Verfügung stehenden Daten sind nur relative Unterschiede erkennbar. Wie stark die Module in den Strings absolut gealtert sind, kann man nicht feststellen, da man dazu genormte Lichteinstrahlung benötigen würde.

```
# Summe der Differenzen pro Tag zwischen den Strings: 1 - 2
df_s = df_strings.resample('D').sum(numeric_only=True)
# Entfernen der übrigen Tage
df_s = df_s[df_s["Ertrag_gesamt"] != 0]
# Index resetten, damit der Graph keine Lücken hat
df_s = df_s.reset_index()


# plot
fig, ax = plt.subplots()

ax.plot(df_s.index, df_s["Diff"]/1000)

# Nulllinie
ax.axhline(0, color='red', linestyle='--')

# Achsen und Titel
ax.set_xlabel('Tage')
ax.set_ylabel("Differenleistung S1 - S2 aufsummiert pro Tag")
ax.set_title('Vergleich der Strings 2014 bis 2022')

plt.show()

fig.savefig("08 Stringvergleich_01.png", dpi=600, bbox_inches='tight')
```

Das Diagramm:



Man erkennt zwei Dinge:

1. Der Unterschied ist bei hoher Leistung am deutlichsten: Die Unterschiede „zittern“ stark hin und her, was auf eine Änderung in der Ertragsdifferenz im Tagesverlauf schließen lässt[21]. Pro Jahr (es sind neun „Hügel“ für neun Jahre erkennbar) sind in der Mitte (also in den Monaten Juni und Juli) die Unterschiede (mit der optimalen Einstrahlung) am größten.
2. Die Differenzen steigen gut sichtbar im Laufe der Jahre an. Zu Beginn ist String 1 schlechter (negative Differenz), dann jedoch nimmt die Leistung von String zwei im Verhältnis ab.

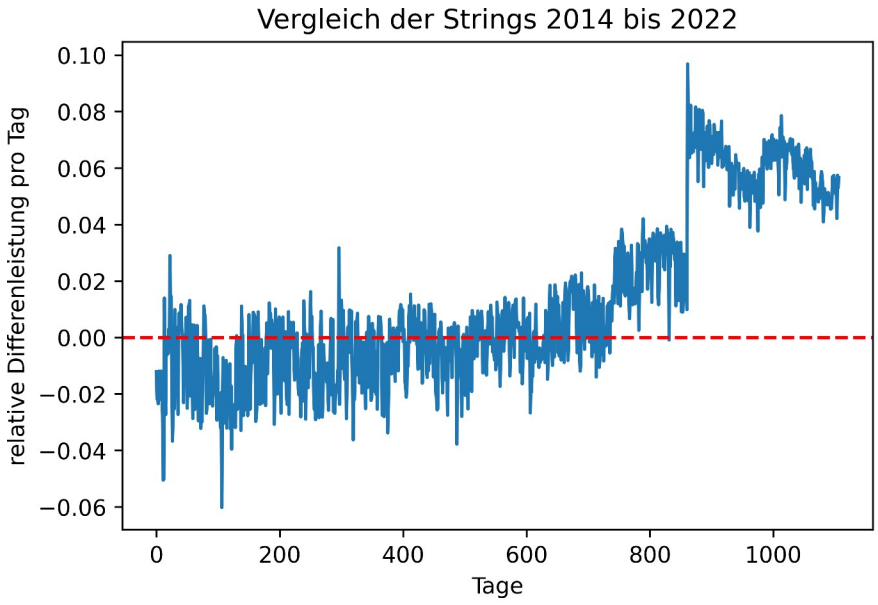
21 Eine genauere Untersuchung mit einem Zeitraum von nur ein paar Tagen wäre möglich. Im Anbetracht der Länge dieser Arbeit wurde darauf verzichtet.

Die Einheit auf der y-Achse hat wenig Aussagekraft, die Tendenz ist aber deutlich zu sehen.

Eine mögliche Quelle für Fehler wäre nun noch gewesen, dass es sich hier um absolute Differenzen handelte, deswegen wurde noch ein Diagramm mit relativen Werten erstellt. Der Code dafür ist nahezu identisch, es wurde nur die Differenz noch durch den Ertrag von String 1 geteilt und dieses Verhältnis dann gegen die Tage angetragen.

```
# ================================================================================
# Das ganze mal realtiv zum Ertrag von String 1
# ================================================================================
df_s["rel"] = df_s["Diff"]/df_s["Ertrag_S1"]

# plot
fig, ax = plt.subplots()

ax.plot(df_s.index, df_s["rel"])

# Nulllinie
ax.axhline(0, color='red', linestyle='--')

# Achsen und Titel
ax.set_xlabel('Tage')
ax.set_ylabel("relative Differenleistung pro Tag")
ax.set_title('Vergleich der Strings 2014 bis 2022')

plt.show()

fig.savefig("08a Stringvergleich_01_rel.png", dpi=600, bbox_inches='tight')
```

Das Diagramm:

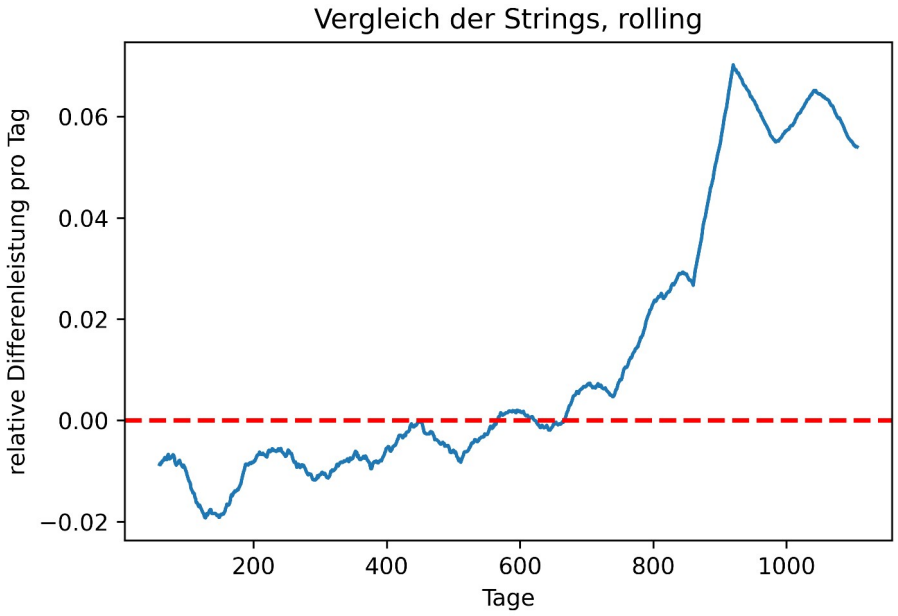


Die Kurve hat sich leicht geglättet, die Unterschiede zwischen den Monaten erscheinen nicht mehr so stark. Die Darstellung hat auch den Vorteil, dass nun die y-Achse die Differenz in Prozent der Leistung von String 1 anzeigt. In den letzten beiden Jahren war der Unterschied also bei ca. 5 bis 8 %.
Am Ende scheint es einen Trend in die andere Richtung zu geben.

Ich war mit der Darstellung noch nicht ganz zufrieden. Um das „Zittern“ zu entfernen versuchte ich noch einen Plot mit einem Rolling Window.
Das Rolling Window erschien nach einigen Versuchen mit einer Größe von 60 Tagen am sinnvollsten, der Plot weißt den üblichen Aufbau auf. Das Abspeichern erfolgte leicht verändert, da es sich um einen Pandas-Plot und nicht um einen Matplotlib-Plot handelt.

```
# ============================================================
# Rolling Window über die Tage
# ============================================================
res = df_s.rel.rolling(60).mean()
# Achsen und Titel

fig = res.plot()

# # Nulllinie
fig.axhline(y=0, color='red', linewidth=2, linestyle="--")

fig.set_xlabel('Tage')
fig.set_ylabel("relative Differenleistung pro Tag")
fig.set_title('Vergleich der Strings, rolling')

fig2 = fig.get_figure()


fig2.savefig("08b Stringvergleich rolling", dpi=600)
```

Das Diagramm:



Die Kurve hat sich sichtlich geglättet, die Wölbungen pro Jahr sind immer noch erkennbar, ebenso wie der starke Anstieg des Unterschieds zum Jahr 2021 hin und der anscheinend fallende Trend im Jahr 2022.
Es bliebt interessant und abzuwarten, wie sich die Daten weiter entwickeln, ob es nur ein „zeitlicher Vorsprung“ der Module in String 2 ist, oder ob sich der Unterschied hält oder gar wieder vergrößert.

2.4 Ausblick

Durch die gerade im letzten Teil gefundenen Ergebnisse werde ich die Daten meiner Solaranlage weiterhin im Auge behalten und die Ertragsunterschiede beobachten. Auch die Unterschiede in den Wochen- und Kalendertagen werden mit einer größeren Grundgesamtheit zunehmend interessanter.
Ich überlege auch, ein Frontend (Web oder GUI) für die Auswertung zu erstellen, das auf Knopfdruck die entsprechenden Grafiken darstellt.
Die Skripte können auch auf andere Backup-Dateien mit relativ wenig Aufwand angewandt werden. So könnte man auch Solaranlagen mit Steuerungshardware anderer Provider damit untersuchen. Sind die Daten erst in einem homogenen Format, ist die Auswertung identisch. Zusätzliche Daten (mehrere Strings oder die Verwendung einer Batterie) können leicht eingepflegt werden.
In den Fußnoten sind schon einige Ideen wiedergegeben, die ich bei einer Überarbeitung der Skripte ändern würde.

Das Projekt hat mir sehr viel Spaß gemacht!