Python Vorgehensweise

ich möchte einen pythonskript schreiben, bei dem ich eine datei mit etwa 1300000 datenwerten hochlade und dann mithilfe des skriptes verschiedene parameter wie beispielsweise cosinus phi berechnen lasse. außerdem sollen die werte mithilfe einer häufigkeitsfunktion (Gauss) in einen 15 minuten Takt wiedergegeben werden und nicht wie zurzeit im minuten oder anderen rhythmus. Das ganze soll dann in einer neuen Datei ausgegeben werden.

es kommen noch weitere sachen die das programm machen soll aber lass uns schritt für schritt anfangen.

Hier die schritte wie das ablaufen soll:

1. csv datei soll geladen bzw eingelesen werden

2. cos(phi) soll ausgerechnet werden anhand der werte die vorhanden sind

3. hilfsgrößen sollen erstellt werden , bspw. neben dem datum soll stehen welcher tag das ist, Montag - Sonntag

4. die daten sollen sortiert werden, alle montage als erstes, dann alle dienstage und so weiter bis sonntage

5. aus diesen werten sollen dann anhand der häufigkeitsfunktion (Gauss) die werte im 15 minuten takt erstellt werden, es soll beginnen mit 00:00-23:45

Nächster schritt in python:

hier kommt jetzt als nächstes hinzu, dass die tage nacheinander sortiert werden, erst kommen alle montage nacheinander, dann die dienstage, dann mittwoche usw. bis sonntage.

danach möchte ich anhand der gauss verfahren was ich gerade eben gesagt habe die werte pro tag im 15 minuten takt in einer neuen spalte haben.

also wenn ich davor pro tag 3600 werte hatte möchte ich danach nur 96 werte pro tag. das soll mit allen spalten und werten in der tabelle gemacht werden insbesondere mit der wirkleistung. das ganze soll dann in neuen spalten dargestellt werden. am besten soll es dann in einer neuen tabelle dargestellt werden wo dann erst das datum steht, dann der tag, und anschließend alle werte.

Code Alperen:

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

# Funktion zur Bestimmung der Jahreszeit für ein gegebenes Datum

def get\_season(date):

if pd.isnull(date):

return 'Unbekannt'

month, day = date.month, date.day

if (11 <= month <= 12) or (1 <= month <= 3 and day <= 20):

return 'Winter'

elif (5 <= month <= 9) or (month == 10 and day <= 14):

return 'Sommer'

elif (3 <= month <= 5 and day >= 21) or (9 <= month <= 11 and day >= 15):

return 'Übergang'

return 'Unbekannt'

# CSV-Daten in einen pandas DataFrame einlesen

df = pd.read\_csv('Transformator\_Alle.csv', delimiter=';', decimal=',')

# Mögliche führende/nachfolgende Leerzeichen entfernen und Spalten konvertieren

df['Datum'] = df['Datum'].str.strip()

df['Uhrzeit'] = df['Uhrzeit'].str.strip()

df[['P L Sum', 'S L Sum', 'U L1', 'I L1', 'U L2', 'I L2', 'U L3', 'I L3']] = df[['P L Sum', 'S L Sum', 'U L1', 'I L1', 'U L2', 'I L2', 'U L3', 'I L3']].apply(pd.to\_numeric, errors='coerce')

# 'Wirkleistung' und Leistungsfaktor berechnen

df['cos\_phi'] = df['P L Sum'] / df['S L Sum']

df['Wirkleistung'] = (df['U L1'] \* df['I L1'] + df['U L2'] \* df['I L2'] + df['U L3'] \* df['I L3']) \* df['cos\_phi']

# 'Datum' und 'Uhrzeit' in datetime umwandeln und den Wochentag extrahieren

df['Datetime'] = pd.to\_datetime(df['Datum'] + ' ' + df['Uhrzeit'], format='%d.%m.%y %H:%M', errors='coerce')

df['Wochentag'] = df['Datetime'].dt.day\_name()

# Jahreszeit für jede Zeile bestimmen

df['Jahreszeit'] = df['Datetime'].apply(get\_season)

# Nach Wochentag und Jahreszeit gruppieren für durchschnittliche 'Wirkleistung'

seasonal\_avg = df.groupby(['Wochentag', 'Jahreszeit'])['Wirkleistung'].mean().reset\_index()

# DataFrame mit Wirkleistung, Wochentag und Jahreszeit speichern

df.to\_csv('aktualisierte\_daten\_mit\_wirkleistung\_wochentag\_jahreszeit.csv', index=False)

# Durchschnittswerte der Jahreszeiten speichern

seasonal\_avg.to\_csv('saisonale\_durchschnitte\_wirkleistung.csv', index=False)

# Zeilen inspizieren, bei denen 'Datetime' nicht geparst werden konnte

print("1")

print(df[pd.isnull(df['Datetime'])])

print(60\*"-")

print("2")

print(df[df['Datetime'].isna() & df['Datum'].notna()])

# Die 'Wirkleistung' über die Zeit plotten

#plt.figure(figsize=(10, 5))

#plt.plot(df['Datetime'], df['Wirkleistung'], label='Wirkleistung', linestyle='-', linewidth=0.1)

#plt.xlabel('Datum und Uhrzeit')

#plt.ylabel('Wirkleistung (Aktive Leistung)')

#plt.title('Wirkleistung über die Zeit')

#plt.legend()

#plt.grid(True)

#plt.show()

import os

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import holidays

from scipy.ndimage import gaussian\_filter1d

# Stelle sicher, dass das Verzeichnis 'data' existiert

input\_file\_path = 'data/Transformator\_Alle.csv'

os.makedirs('data', exist\_ok=True)

os.makedirs('plots', exist\_ok=True)

# Lade und verarbeite den Datensatz

df = pd.read\_csv(input\_file\_path, delimiter=';', decimal=',')

df['Datum'] = df['Datum'].str.strip()

df['Uhrzeit'] = df['Uhrzeit'].str.strip()

df['Datum\_Uhrzeit'] = pd.to\_datetime(df['Datum'] + ' ' + df['Uhrzeit'], format='%d.%m.%y %H:%M', errors='coerce')

# Überprüfe auf Zeilen mit NaT-Werten in 'Datum\_Uhrzeit'

nat\_rows = df[df['Datum\_Uhrzeit'].isna()]

if not nat\_rows.empty:

print("Zeilen mit NaT-Werten in 'Datum\_Uhrzeit':")

print(nat\_rows[['Datum', 'Uhrzeit']])

# Entferne Zeilen mit NaT-Werten

df = df.dropna(subset=['Datum\_Uhrzeit'])

df.set\_index('Datum\_Uhrzeit', inplace=True)

# Konvertiere zu numerischen Werten und berechne 'Wirkleistung'

df[['P L Sum', 'S L Sum', 'U L1', 'I L1', 'U L2', 'I L2', 'U L3', 'I L3']] = df[['P L Sum', 'S L Sum', 'U L1', 'I L1', 'U L2', 'I L2', 'U L3', 'I L3']].apply(pd.to\_numeric, errors='coerce')

df['cos\_phi'] = df['P L Sum'] / df['S L Sum']

df['Wirkleistung'] = (df['U L1'] \* df['I L1'] + df['U L2'] \* df['I L2'] + df['U L3'] \* df['I L3']) \* df['cos\_phi']

df['Wirkleistung'] = df['Wirkleistung'] / 1000 # Umrechnung in kW

# Kategorisiere nach Zeitraum

df['time\_period'] = df.index.to\_series().apply(lambda dt: 'winter' if dt.month == 11 or dt.month <= 3 else 'sommer' if 5 <= dt.month <= 9 else 'übergang')

# Gruppiere nach Wochentag und Uhrzeit

df['day\_of\_week'] = df.index.dayofweek

df['time'] = df.index.time

de\_holidays = holidays.Germany(prov='NW')

# Funktion zur Anpassung für Feiertage

def adjust\_for\_holidays(row):

try:

if row.name.date() in de\_holidays or (row.name.date().month == 12 and row.name.date().day in [24, 31] and row.name.weekday() != 6):

return 6 if row.name.date().weekday() != 6 else row['day\_of\_week'] # Behandle als Sonntag, es sei denn, es ist bereits Sonntag

elif row.name.date().month == 12 and row.name.date().day in [24, 31]:

return 5 # Behandle als Samstag

return row['day\_of\_week']

except Exception as e:

print(f"Fehler bei der Verarbeitung der Zeile: {row.name}, Ausnahme: {e}")

raise e

df['day\_of\_week'] = df.apply(adjust\_for\_holidays, axis=1)

# Aufteilen der Daten in Arbeitstage, Samstag und Sonntag

workdays = df[df['day\_of\_week'] < 5]

saturday = df[df['day\_of\_week'] == 5]

sunday = df[df['day\_of\_week'] == 6]

# Funktion zum Erstellen und Speichern von Plots

def create\_and\_save\_plots(workdays, saturday, sunday, sigma=2, save\_path='plots/combined\_plot.png'):

plt.figure(figsize=(18, 6))

# Erstelle den Plot für Arbeitstage

plt.subplot(1, 3, 1)

plot\_wirkleistung(workdays, 'Wirkleistung Arbeitstage (Mo-Fr)', sigma)

# Erstelle den Plot für Samstag

plt.subplot(1, 3, 2)

plot\_wirkleistung(saturday, 'Wirkleistung Samstag', sigma)

# Erstelle den Plot für Sonntag

plt.subplot(1, 3, 3)

plot\_wirkleistung(sunday, 'Wirkleistung Sonntag', sigma)

plt.tight\_layout()

plt.savefig(save\_path)

plt.close()

# Funktion zur Darstellung der Wirkleistung mit Glättung

def plot\_wirkleistung(data, title, sigma=2):

# Filtern der Daten nach Zeitraum und andere Vorbereitungen

data\_winter = data[data['time\_period'] == 'winter']

data\_summer = data[data['time\_period'] == 'sommer']

data\_transition = data[data['time\_period'] == 'übergang']

# Erstelle eine Liste von Zeitmarken im 15-Minuten-Intervall

time\_range = pd.date\_range(start='00:00', end='23:45', freq='15T').time

time\_indices = np.arange(len(time\_range))

# Mapping der Zeitstempel auf die X-Achsen-Indizes

time\_to\_index = {time: index for index, time in enumerate(time\_range)}

if not data\_winter.empty:

# Gruppiere und berechne den Durchschnitt für die Wirkleistung, glätte die Daten

winter\_resampled = data\_winter.groupby(data\_winter.index.time)['Wirkleistung'].mean().reindex(time\_range, fill\_value=0)

smoothed\_winter = gaussian\_filter1d(winter\_resampled, sigma=sigma)

plt.plot(time\_indices, smoothed\_winter, label='Winter', color='black')

if not data\_summer.empty:

summer\_resampled = data\_summer.groupby(data\_summer.index.time)['Wirkleistung'].mean().reindex(time\_range, fill\_value=0)

smoothed\_summer = gaussian\_filter1d(summer\_resampled, sigma=sigma)

plt.plot(time\_indices, smoothed\_summer, label='Sommer', color='gray')

if not data\_transition.empty:

transition\_resampled = data\_transition.groupby(data\_transition.index.time)['Wirkleistung'].mean().reindex(time\_range, fill\_value=0)

smoothed\_transition = gaussian\_filter1d(transition\_resampled, sigma=sigma)

plt.plot(time\_indices, smoothed\_transition, label='Übergang', color='white', linestyle='--')

plt.title(title)

plt.xlabel('Tageszeit')

plt.ylabel('Wirkleistung [kW]')

plt.legend()

# Setze X-Achsen-Ticks und Labels

plt.xticks(time\_indices[::4], [time.strftime('%H:%M') for time in time\_range[::4]]) # Alle Stunden anzeigen

# Generiere und speichere die kombinierten Plots

create\_and\_save\_plots(workdays, saturday, sunday, sigma=2, save\_path='plots/combined\_plot.png')

# Speichere den aktualisierten DataFrame in einer CSV-Datei

output\_file\_path = 'data/Transformator\_Alle\_Updated.csv'

df.reset\_index().to\_csv(output\_file\_path, index=False)

GridCal (Programm 1) was noch fehlt:

* Nachdem alle Tage sortiert werden, hat man pro Tag 00:00-23:59 Uhr jeweils einen Wert (1-Minute Takt)
* Daraus müssen mit Gauß´sche Normalverteilung Werte im 15-Minuten Takt erzeugt werden -> 96 Werte pro Takt
* Wenn möglich Boxplot für 15-Minuten Takt Werte -> es wird ein Wert aus 15 Werten erstellt, min und max angeben, um bei eventuellen falschen Ergebnissen zu sehen wozwischen sich dieser Wert befindet
* Diese Werte die im 15-Minuten Takt sind müssen in der Endtabelle sein
* Endtabelle sieht folgendermaßen aus:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Jahreszeit | Tag | Datum/Uhrzeit | U | I | P | Q | S | cos phi | Wirkleistung |
| w/s/ü1/ü2 | Montag | 01.01.24/00:00 |  | - |  |  |  |  |  |
| w/s/ü1/ü2 | Montag | 01.01.24/00:15 |  | - |  |  |  |  |  |
| w/s/ü1/ü2 | Montag | 01.01.24/00:30 |  | - |  |  |  |  |  |

* Plotten von den Graphen
* -> 1. Werktag Mo-Fr; 2. Samstag; 3. Sonntag; 4. Montag; 5. Dienstag;

6. Mittwoch; 7. Donnerstag; 8. Freitage

* Die werte die fürs Plotten benutzt werden müssen in Endtabelle ausgegeben werden so dass man da alles herausbekommen kann
* Endtabelle in txt Format wenn möglich

Smight (Programm 2):

* Dateien werden eingelesen -> mehrere Dateien, da pro Tag eine Tabelle für i und eine für u vorhanden ist
* Funktion, dass mehrere Dateien gleichzeitig ausgewählt werden können und nicht alle einzeln (da ggf. mehrere Tabellen ausgewählt werden müssen)
* Die i-Datei ist in einem 10 Sekunden Takt (manchmal weniger Werte da muss dann aus denen ein Wert für die Minute bestimmt werden)
* Erstmal die spalten L1active+L2active+L3active rechnen, damit man nur einen Wert für Strom hat
* Daraus die Mittelwerte nehmen, so dass man pro Minute nur einen Wert hat
* Dann wird die Datei von i mit u zusammengefügt -> so hat man pro u wert auch einen i Wert für jede Minute
* Damit werden die restlichen Parameter berechnet -> P, Q, S cos phi, Wirkleistung
* Ab da kommt der Rest wie im 1. Programm