Data Analytics mit Python und SAP

Praxiserfahrungen

Magdeburger Developer Days 2023

Alexander Kramer,
Jörg Müller

Mai 2023

BA BUSINESS ADVICE

SAP Stammtisch & BA - Support für die MDDevDays

Der SAP Stammtisch Magdeburg im Internet

Besucht die SAP Stammtische im Internet!

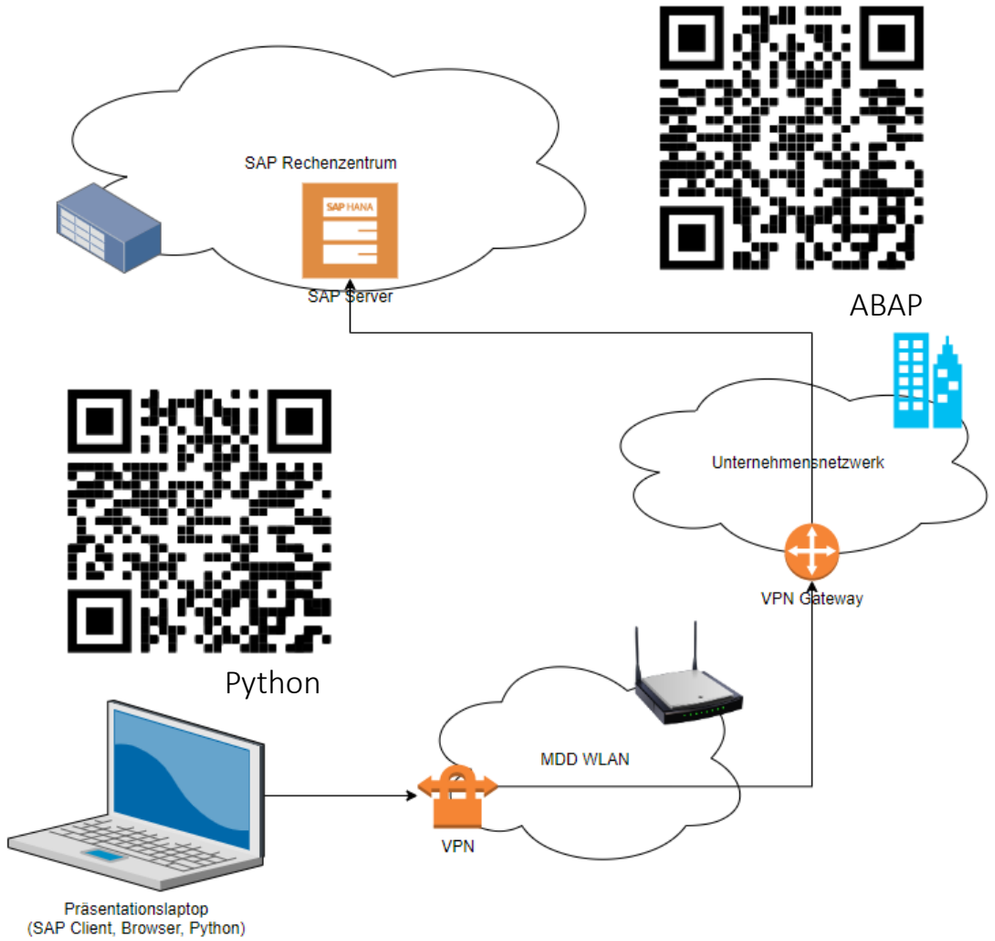
- SAP Stammtisch Magdeburg **Github Webseite**
  https://sapstammtisch.github.io/Magdeburg/
- SAP Stammtisch Magdeburg **Mastodon**
  https://machteburch.social/@SAPStammtisch
- **Email**: magdeburg@sapstammtisch.org
- Hashtags (**Twitter**, Mastodon):
  #SAPStammtischMD, #SAPStammtisch

- SAP Community
  https://community.sap.com/
- SAP Community Gruppe Magdeburg
  https://groups.community.sap.com/t5/magdeburg/gh-p/magdeburg
- SAP Community Event Kalender
  https://groups.community.sap.com/t5/events/ct-p/events
- Weitere SAP Stammtische bei Github
  https://sapstammtisch.github.io/welcome/

SAP Stammtisch Magdeburg

Nächster Termin:
12.06.2023
Hybrid

Rückblick MDDevDays 2022: Vortrag SAP und Python



- Die **Demolandschaft** entspricht einer typischen Umgebung, wie sie häufig in SAP Projekten zu finden ist:
  - Der Endanwender ist per **VPN** mit dem Unternehmen verbunden
  - Dort wird es über eine **sichere Verbindung zum SAP System** weiter geroutet
  - Das SAP System steht in einem separaten **Rechenzentrum**
  - Es ist – wie hier - häufig unmöglich, direkt **vom SAP Rechenzentrum in das Netz des Kunden** zu gelangen
    → **Herausforderung für Schnittstellen**, die vom SAP ausgehen

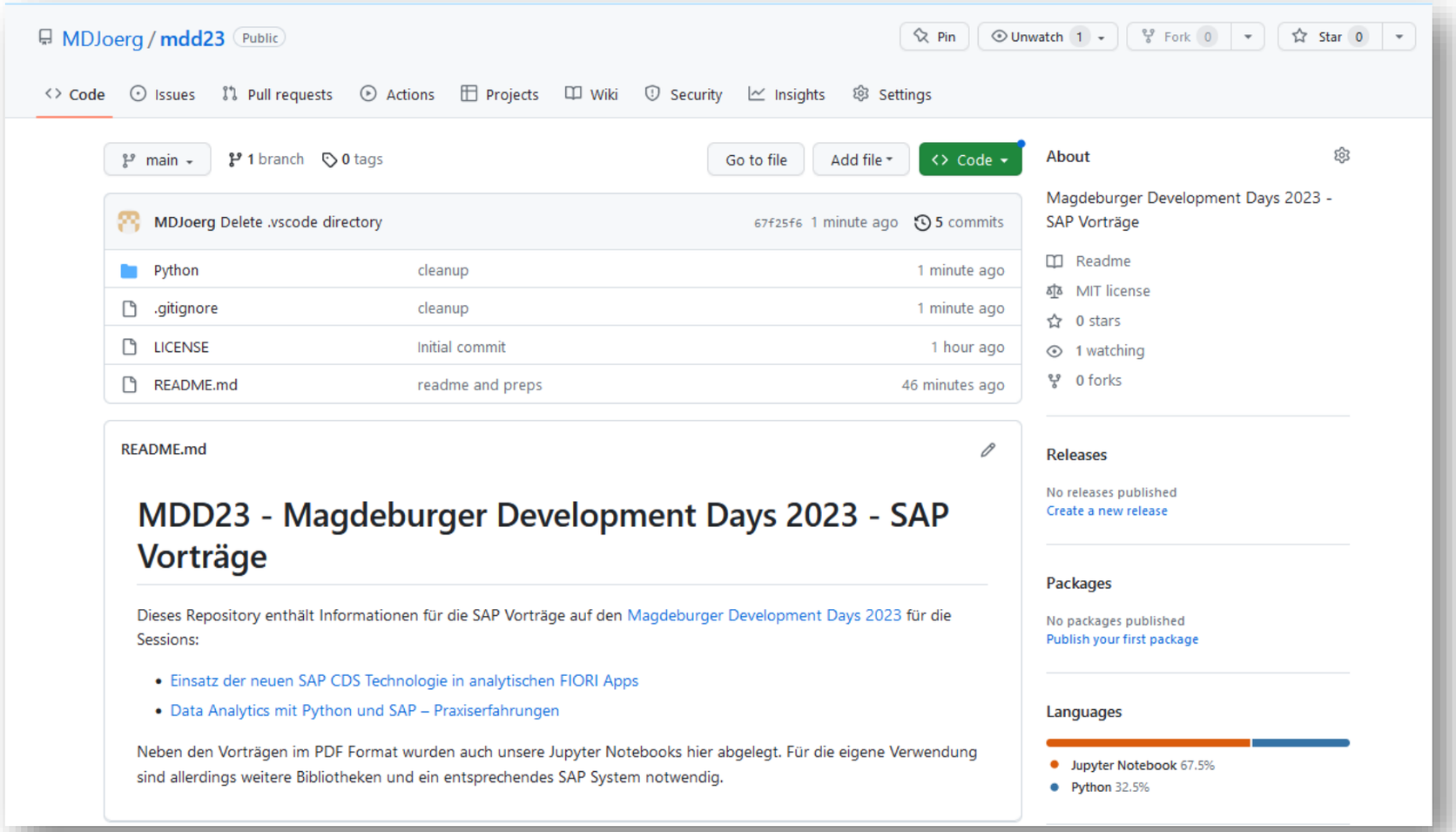
- **Github Repositories**:
  - https://github.com/MDJoerg/mdd22_python
  - https://github.com/MDJoerg/mdd22_abap

- Diese **Folien**
  → gibt es im Github Respository mdd22_python nach dieser Veranstaltung

Agenda – Was könnt Ihr heute erwarten?

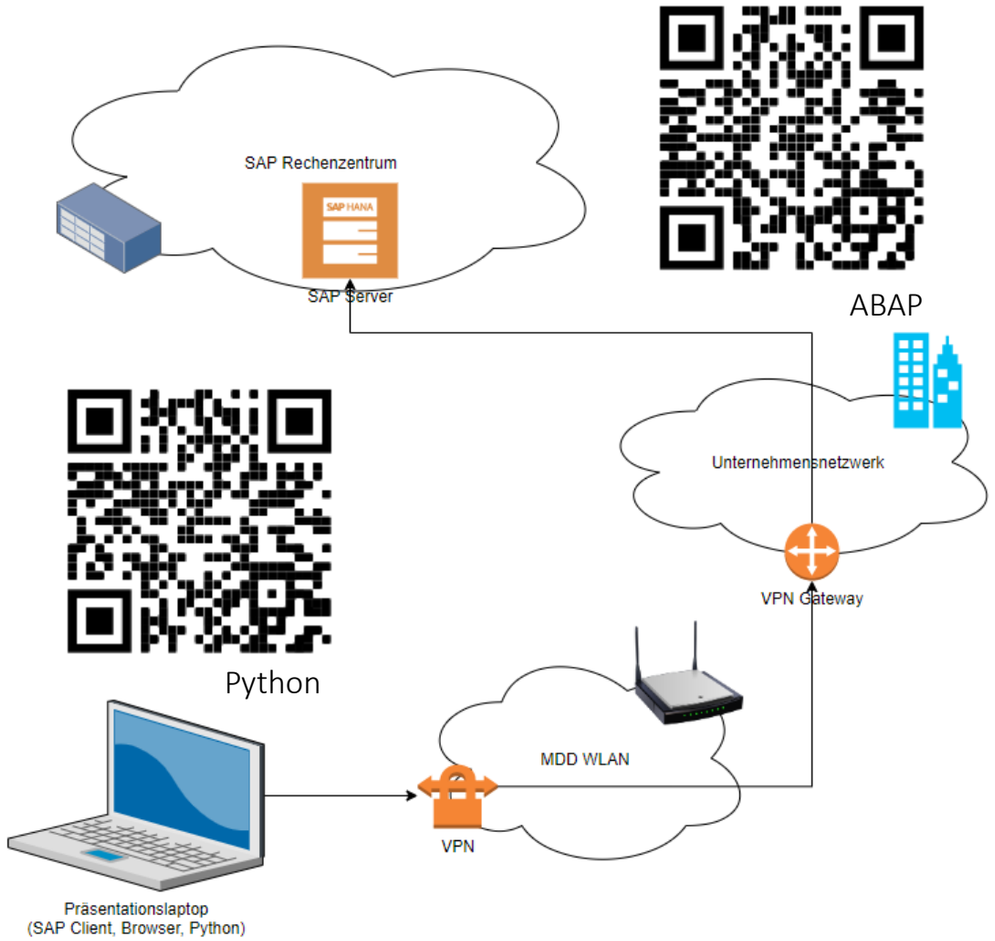
- Begrüßung
- Rückblick MDDevDays 2022
- Vorstellung PyDEEN
- Big Picture Data Science Beispielszenario
- Analytics und Data Science bei BA
- Data Science Beispiel „SAP Sensoren auswerten“
- Ausblick, Fragen, Austausch

Informationen und Dokumente zum Vortrag



- Alle SAP Vorträge von den MDDevDays 2023 sind öffentlich auf Github
- Präsentationen als PDF + Code
- https://github.com/MDJoerg/mdd23

Rückblick MDDevDays 2022: Vortrag SAP und Python



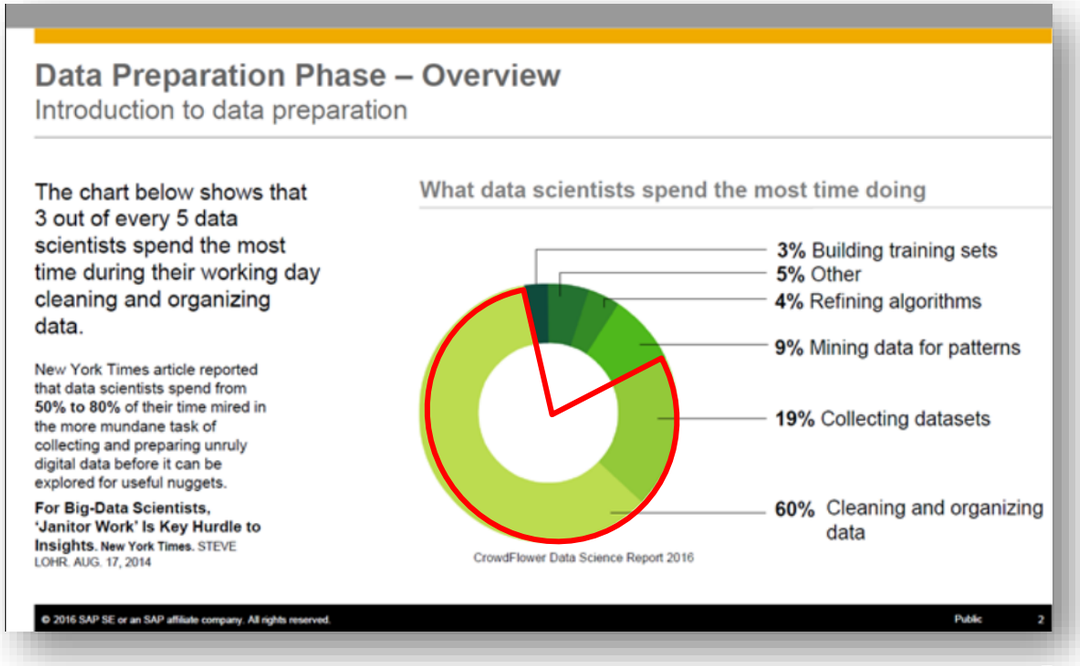
- Die **Demolandschaft** entspricht einer typischen Umgebung, wie sie häufig in SAP Projekten zu finden ist:
  - Der Endanwender ist per **VPN** mit dem Unternehmen verbunden
  - Dort wird es über eine **sichere Verbindung zum SAP System** weiter geroutet
  - Das SAP System steht in einem separaten **Rechenzentrum**
  - Es ist – wie hier - häufig unmöglich, direkt **vom SAP Rechenzentrum in das Netz des Kunden** zu gelangen
    → **Herausforderung für Schnittstellen**, die vom SAP ausgehen

- **Github Repositories**:
  - https://github.com/MDJoerg/mdd22_python
  - https://github.com/MDJoerg/mdd22_abap

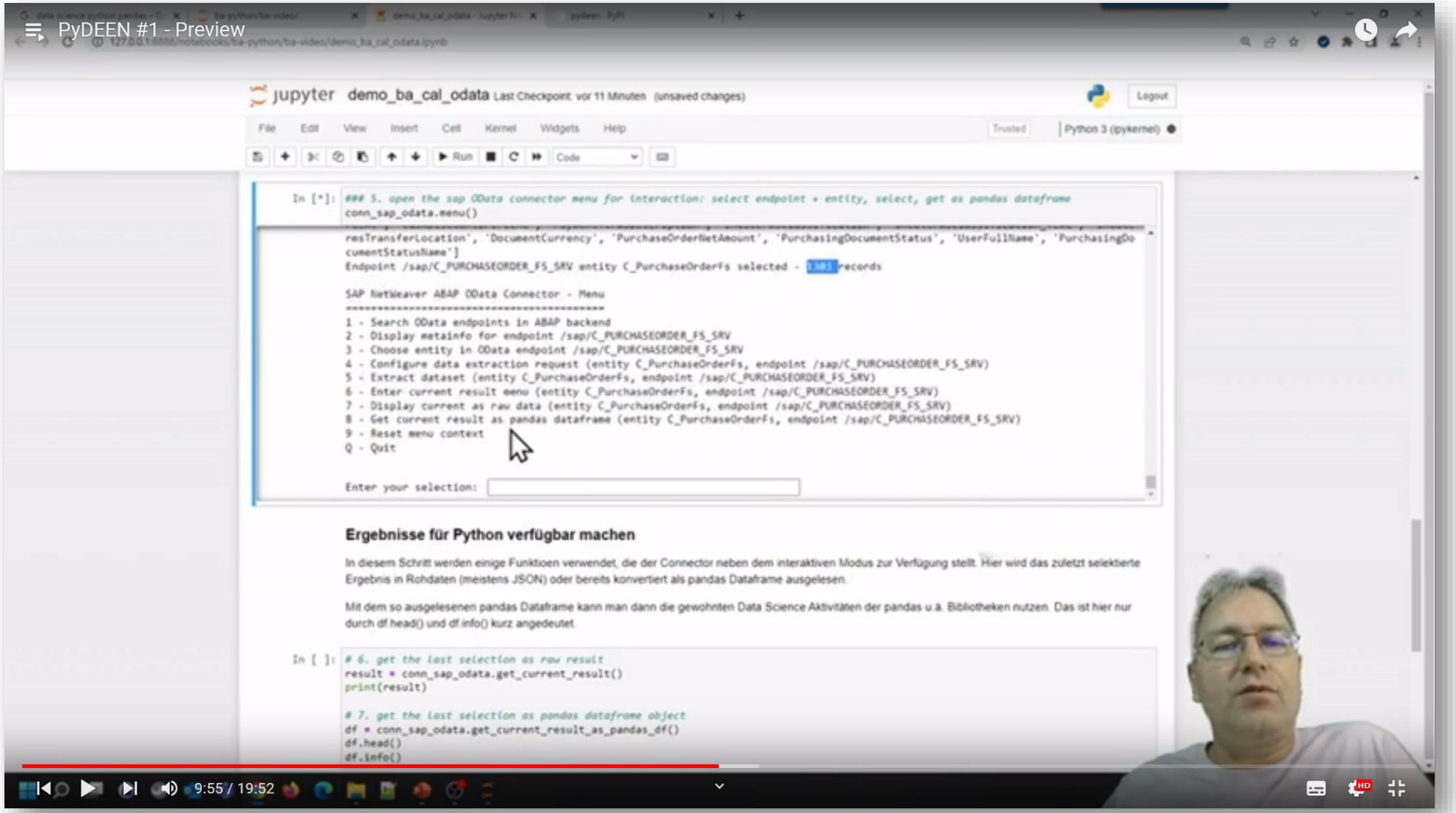
- Diese **Folien**
  → gibt es im Github Respository mdd22_python nach dieser Veranstaltung

Wie ging es weiter?

- Für den Vortrag auf den MDDevDays2022 wurden die frisch erworbenen Python Kenntnisse verwendet
- Teile des Demo-Codes waren auch für interne Projekte, Hackathons, u.ä. interessant
- der MDevDays Code aus dem Vortrag wurde als Basis für eine kleine interne Bibliothek verwendet
  → PyDEEN war geboren (= „Python Data Engineer Enterprise Notebook")
- Der Fokus lag anfangs auf SAP Connectivity für die einfache Extraktion von SAP Daten über "Connectoren" und das OData Protokoll (ohne Backend)
- Ursprünglicher Fokus wandelte sich:
  - Großer Bedarf an Jupyter Notebook Unterstützung für die einfache Handhabung der Datenextraktion und -aufbereitung
  - "Brücke" in die Data Science Welt → Pandas Dataframe
  - Weitere Connectoren (z.B. JSON REST)
- PyPi.Org Package
- Youtube Playlist mit Demoszenarien für die Features
- Fortschritt ist leider abhängig von der aktuell verfügbaren Zeit und den benötigten Features aus den Projekten



Das PyDEEN „Killer-Feature“ – der Menümodus



https://youtu.be/vI-1LB8aGEk ab Min 3:30

PyDEEN – Funktionen

Tools

- Command Line Menüs (z.B. Jupyter)
- Framework Types (z.B. Result)
- Datahub Konzept
- Configuration, Logging u.a.

Data Import

- SAP OData Connector
- SAP Deeb Connector
- REST Connector
- Datenanalyse
- File Import (Excel, CSV, Pickle)

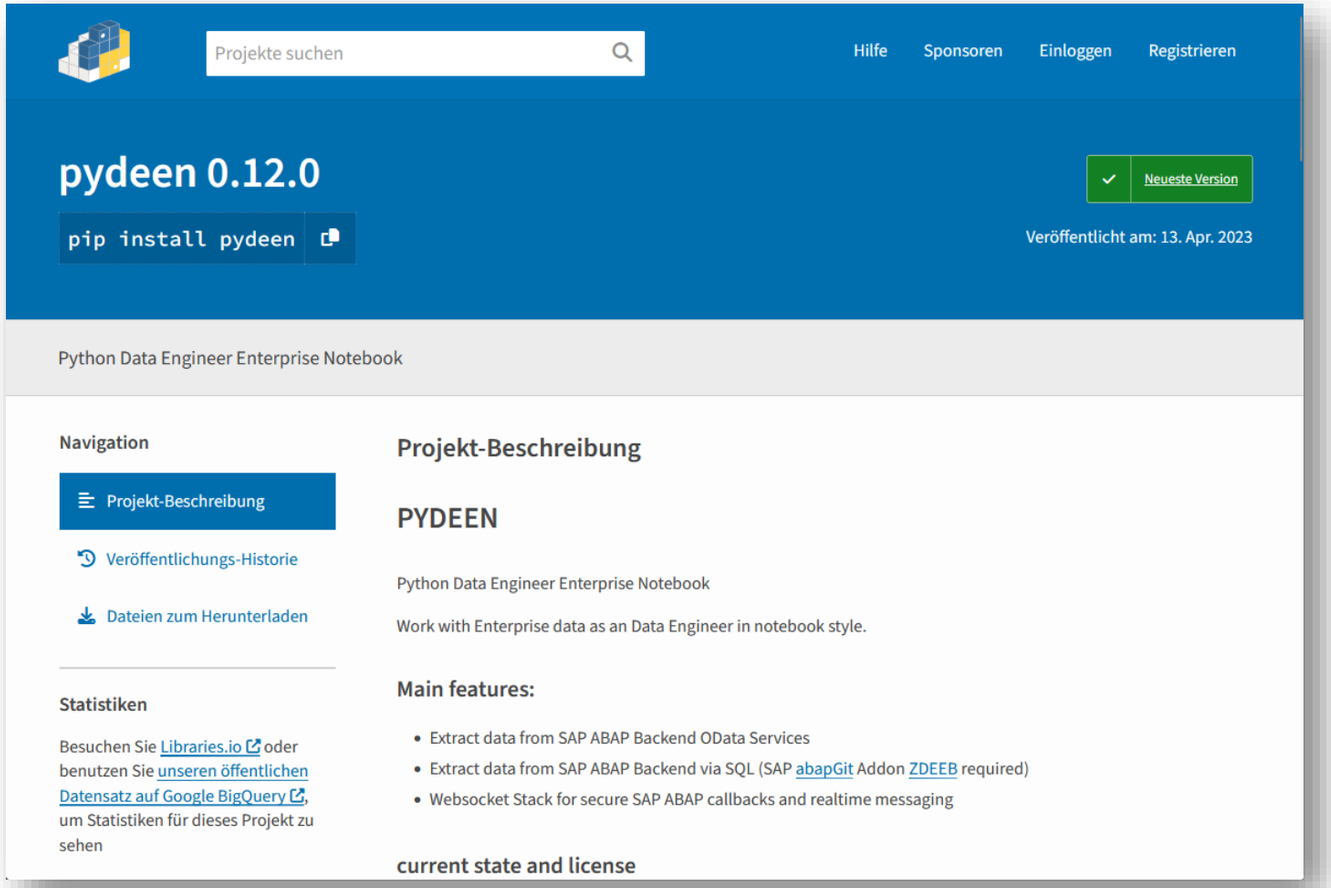
Data Export

- File Export (Pickle, CSV, Excel)
- SAP Upload
- JSON REST Post

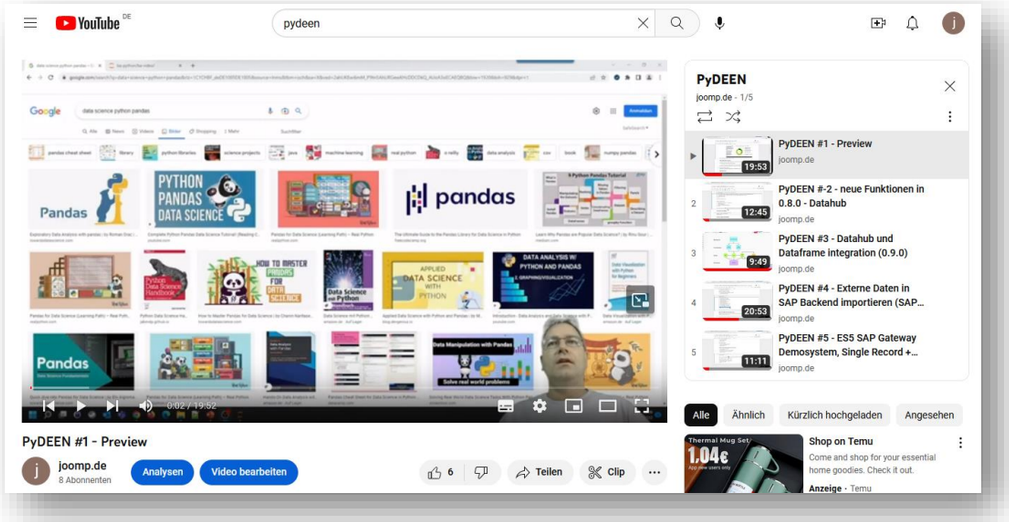
Websockets

- Socket Listener Service mit Erweiterungen
- Realtime Events
- Cloud Event Support

PyDEEN – Weitere Informationen



https://pypi.org/project/pydeen/

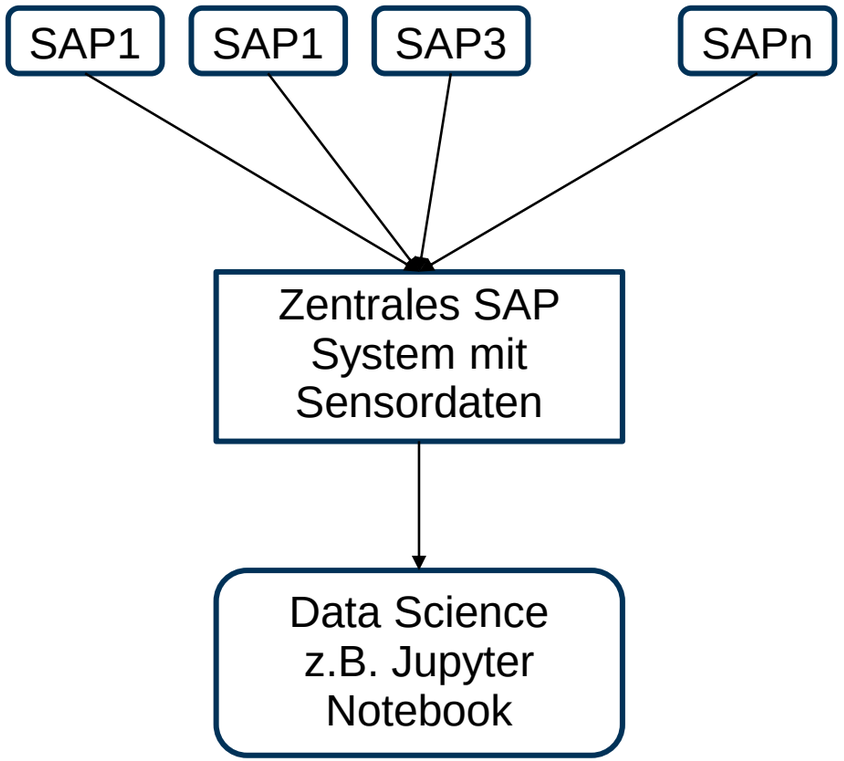


https://bit.ly/3LAEdM4

https://github.com/MDJoerg/pydeen_demos/

- PyDEEN als OpenSource geplant
- Github Repository noch nicht öffentlich
- Dokumentation noch nicht „schön“
- Alternativ: Youtube Videos und Demos im Github Repo

Beispielszenario – Big Picture

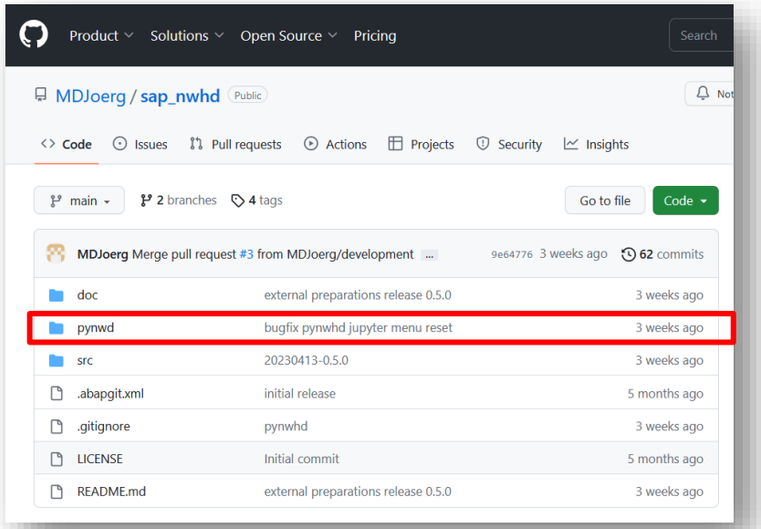


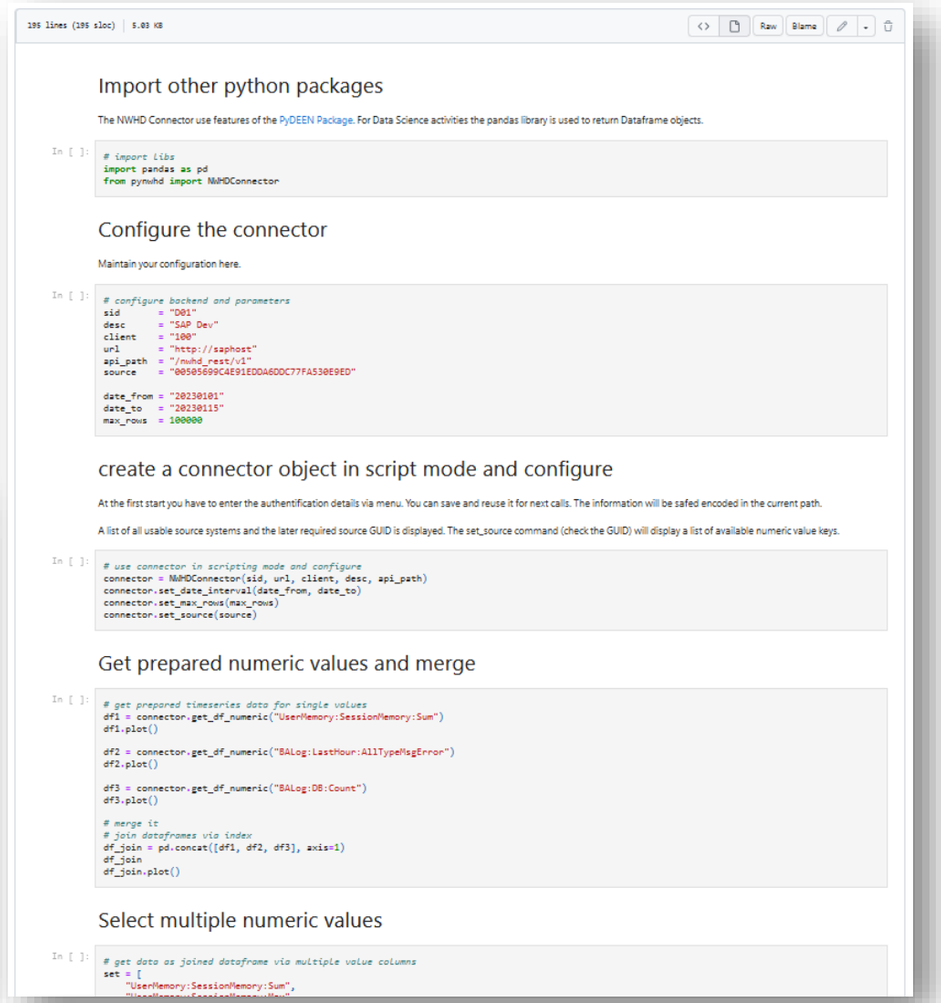
- Ausgangssituation
  - Das BA Analytics Team möchte Data Science mit SAP Daten machen (z.B. Hackathons)
  - Woher bekommen wir Massendaten?
  - Wie kommen wir an die Daten heran?
  - Sind die Daten wirklich geeignet?
- Lösung
  - Wir nutzen das SAP System selbst als Datenerzeuger → Sensordaten aus dem SAP
  - Die Sensordaten werden an ein zentrales SAP System übermittelt
  - Durch die Verwendung von IOT Konzepten können sogar übliche Security Restriktionen aufgebrochen werden (wir können fremde Rechenzentren anbinden)
  - Die Daten werden im SAP HANA optimiert gespeichert (InfluxDB Konzept) und stehen über verschiedene Wege zur Verfügung (z.B. CDS, OData, REST API)

https://docs.influxdata.com/influxdb/v1.3/concepts/key_concepts

SAP ABAP Opensource Projekt SAP_NWHD

- https://github.com/MDJoerg/sap_nwhd
- Enthält: Erzeugung der Sensordaten, Lokale Speicherung oder Versand, Zugriff über REST API, …
- Einspielen in SAP Systemen über abapgit (https://abapgit.org/)
- Dokumentation (noch im Aufbau)
- Python und Jupyter Notebook Code für den Zugriff auf die NWHD REST API
- Fokus auf schnelle Erzeugung von Pandas Dataframe Objekten (TimeSeries) auf Basis der SAP Daten
- Verwendet das PyDEEN Framework (Beispiel pynwhd.py)





BA – Analytics Überblick

- Als Expertenteam begleiten wir S/4HANA Einführungsprojekte mit dem Fokus auf das Berichtswesen unserer Kunden
- Die Aufgabenbereiche umfassen vor allem:
  - Technische und inhaltliche Beratung
    - Erfahrung in der Umsetzung von konzernübergreifendem Berichtswesen
    - Anforderungsaufnahme
    - Auf die Kundenanforderung zugeschnittene Best-Practice Beratung
  - Projektbegleitende Umsetzung
    - CDS-View Development
    - FIORI-App Entwicklung
    - SAP Analytics Cloud
    - Embedded Analytics
  - Innovationsthemen und Data Science z.B. mit Python

Data Science im IT Service mit Jupyterhub

Live-Demo

Einstieg ins System – laden der Bibliotheken

```
# import libs
import pandas as pd
from pynwhd import NWHDConnector


from pydeen.types import Factory
import matplotlib.pyplot as plt
import seaborn as sns
import calendar
import panel as pn
import numpy as np


pn.extension('tabulator')


import hvplot.pandas
```

```
# configure backend and parameters
sid        = "S4D"
desc       = "BA Entwicklung"
client     = "100"
url        = "[redacted]"
api_path   = "/nwhd_rest/v1"
source     = "[redacted]"

date_from = "20230301"
date_to   = "20230430"
max_rows  = 100000
```

```
PyDEEN logging initialized (version 0.12.0)
URL: http://[redacted]/nwhd_rest/v1/sources

Available sources:
00505699C[redacted]E6A9ED - S4P100                BA S/4 Produktion
00505699C[redacted]C29ED - BED020                 BA ECC
00505699C[redacted]F0A9ED - S4D100                BA S/4 Entwicklung
00505699C[redacted]98449ED - CK_DS4100            C[redacted] - DS4 - S/4 HANA Entwicklung
00505699C[redacted]A530E9ED - CK_PS4100           C[redacted]e - PS4 - S/4 HANA Produktion
/nwhd_rest/v1/numeric_available/00505699[redacted]A530E9ED
URL: http[redacted]/nwhd_rest/v1/numeric_available/00505699[redacted]A530E9ED

Source 00505699[redacted]A530E9ED selected.

Available numeric values:
BALog:DB:Count                                =      35928 values
BALog:Last24h:AllTypeCount                    =      31213 values
BALog:Last24h:AllTypeMsgAbort                 =          1 values
BALog:Last24h:AllTypeMsgCount                 =      31841 values
BALog:Last24h:AllTypeMsgError                 =      21274 values
BALog:Last24h:AllTypeMsgWarning               =      31839 values
BALog:Last24h:Type1Count                      =       5417 values
BALog:Last24h:Type1MsgAbort                   =         11 values
BALog:Last24h:Type1MsgCount                   =       5777 values
BALog:Last24h:Type1MsgError                   =       4203 values
BALog:Last24h:Type1MsgWarning                 =       1972 values
BALog:Last24h:Type2Count                      =      25621 values
BALog:Last24h:Type2MsgAbort                   =          8 values
```

```
# Konnektor im Skripting-Modus verwenden und konfigurieren
connector = NWHDConnector(sid, url, client, desc, api_path)
connector.set_date_interval(date_from, date_to)
connector.set_max_rows(max_rows)


#Quellsystem auswählen
connector.set_source(source)
```

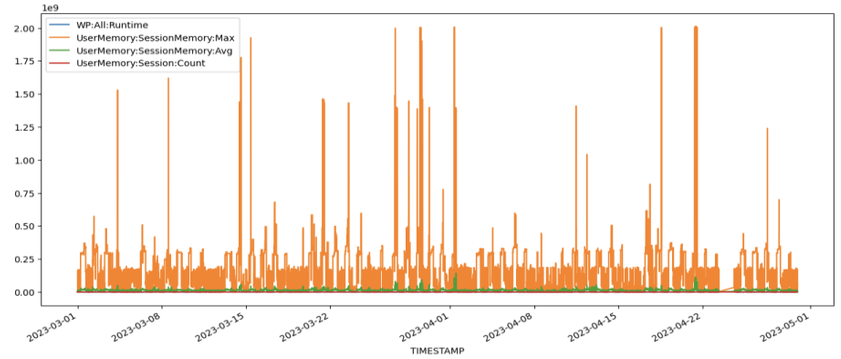
```
PyDEEN logging initialized (version 0.12.0)
URL: http://s[redacted]/nwhd_rest/v1/sources
```

DataFrame als Auswertungsbasis erzeugen 1/2

Rohdaten laden

```
# get data as joined dataframe via multiple value columns


set = [
    "WP:All:Runtime"
    ,"UserMemory:SessionMemory:Max"
    ,"UserMemory:SessionMemory:Avg"
    ,"UserMemory:Session:Count"
]

df_UserMem = connector.get_df_numeric_multiple(set)
#print(df_list)    , df_list
#if df_UserMem:
df_UserMem.plot(figsize=(15, 7))
df_UserMem.dtypes
df_UserMem.index



#else:
#    print("No joined Data")
```

```
[4]: DatetimeIndex(['2023-02-28 23:00:47+00:00', '2023-02-28 23:05:47+00:00',
                    '2023-02-28 23:25:47+00:00', '2023-02-28 23:30:47+00:00',
                    '2023-02-28 23:35:47+00:00', '2023-02-28 23:40:47+00:00',
                    '2023-02-28 23:45:47+00:00', '2023-02-28 23:50:47+00:00',
                    '2023-02-28 23:55:47+00:00', '2023-03-01 00:00:47+00:00',
                    ...
                    '2023-04-29 21:10:50+00:00', '2023-04-29 21:15:50+00:00',
                    '2023-04-29 21:20:50+00:00', '2023-04-29 21:25:50+00:00',
                    '2023-04-29 21:30:50+00:00', '2023-04-29 21:35:50+00:00',
                    '2023-04-29 21:40:50+00:00', '2023-04-29 21:45:50+00:00',
                    '2023-04-29 21:50:50+00:00', '2023-04-29 21:55:50+00:00'],
                   dtype='datetime64[ns, UTC]', name='TIMESTAMP', length=16220, freq=None)
```



Spaltenbezeichnung anpassen

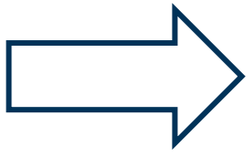
```
# List of variable with original name and wanted name

rename_cols = [
    ('WP:All:Runtime', 'Runtime'),
    ('UserMemory:SessionMemory:Max', 'SessionMemoryMax'),
    ('UserMemory:SessionMemory:Avg', 'SessionMemoryAvg'),
    ('UserMemory:Session:Count' , 'SessionCount')
    ]

# Iteration über die Liste und Umbenennung der Spalten
for old_name, new_name in rename_cols:
    df_UserMem.rename(columns={old_name: new_name}, inplace=True)


df_UserMem.fillna(0, inplace=True)
df_UserMem
```

TIMESTAMP	Runtime	SessionMemoryMax	SessionMemoryAvg	SessionCount
2023-02-28 23:00:47+00:00	0.0	2198032.0	960121.0	12.0
2023-02-28 23:05:47+00:00	0.0	0.0	974771.0	11.0
2023-02-28 23:25:47+00:00	0.0	0.0	974771.0	0.0
2023-02-28 23:30:47+00:00	0.0	0.0	968813.0	0.0
2023-02-28 23:35:47+00:00	0.0	0.0	974771.0	11.0
...	...	...	...	...
2023-04-29 21:35:50+00:00	0.0	0.0	974771.0	11.0
2023-04-29 21:40:50+00:00	0.0	2198032.0	976505.0	12.0
2023-04-29 21:45:50+00:00	306.0	168810151.0	13904620.0	13.0
2023-04-29 21:50:50+00:00	602.0	23029567.0	2690728.0	0.0
2023-04-29 21:55:50+00:00	908.0	64795719.0	5903509.0	13.0

DataFrame als Auswertungsbasis erzeugen 2/2

Spalten in dynamische Variablen setzen und Daten aggregieren

```
cols = list(df_UserMem.columns)
col1 = cols[0]
col2 = cols[1]
col3 = cols[2]
col4 = cols[3]


df_UM_resampled = df_UserMem.resample('H')

df_base = df_UM_resampled.agg({col1: 'mean',
                               col2: 'mean',
                               col3: 'mean',
                               col4: 'max'})
pd.options.display.float_format = '{:.0f}'.format
df_base[[col2 , col3]] /= 1024
df_base
```

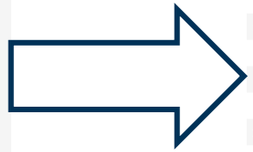


TIMESTAMP	Runtime	SessionMemoryMax	SessionMemoryAvg	SessionCount
2023-02-28 23:00:00+00:00	201	27400	3003	13
2023-03-01 00:00:00+00:00	165	28437	3074	13
2023-03-01 01:00:00+00:00	182	32805	3602	37
2023-03-01 02:00:00+00:00	152	29005	3463	14
2023-03-01 03:00:00+00:00	182	30718	3538	14
...	...	...	...	...
2023-04-29 17:00:00+00:00	181	22618	2641	13
2023-04-29 18:00:00+00:00	227	41464	4073	13
2023-04-29 19:00:00+00:00	202	23767	2724	13
2023-04-29 20:00:00+00:00	151	19550	2413	13
2023-04-29 21:00:00+00:00	151	21243	2544	13

1439 rows × 4 columns

Daten um Datumsinformationen anreichern

```
# Neue Spalten erstellen mit den Datumsinformationen

df_monthly['date'] = pd.to_datetime(df_monthly['TIMESTAMP'])
df_monthly['year'] = df_monthly['date'].dt.year
df_monthly['month'] = df_monthly['date'].dt.month
df_monthly['day'] = df_monthly['date'].dt.day
df_monthly['week'] = df_monthly['date'].dt.isocalendar().week # Iso Woche
df_monthly['weekday'] = df_monthly['date'].dt.weekday
df_monthly['Month_Year'] = pd.to_datetime(df_monthly['TIMESTAMP'].dt.to_period('M').astype(str)).dt.strftime('%b %Y')


# Setting Index for the next Function
df_monthly = df_monthly.set_index('TIMESTAMP')

# Create a new column for the weekday
df_monthly['Wochentag'] = df_monthly.index.day_name()
weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
df_monthly['Wochentag'] = pd.Categorical(df_monthly['Wochentag'], categories=weekdays, ordered=True)


# did it work?
df_monthly
```

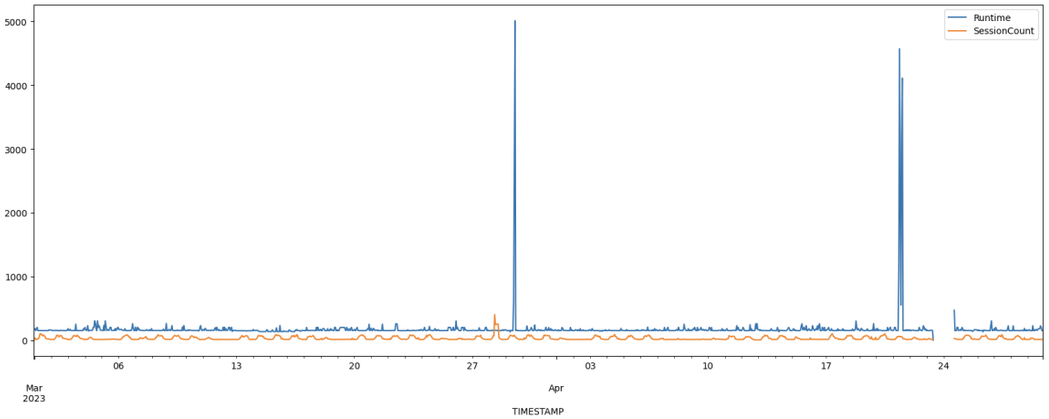


TIMESTAMP	Runtime	SessionMemoryMax	SessionMemoryAvg	SessionCount	date	year	month	day	week	weekday	Month_Year	Wochentag
2023-02-28 23:00:00+00:00	201	27400	3003	13	2023-02-28 23:00:00+00:00	2023	2	28	9	1	Feb 2023	Tuesday
2023-03-01 00:00:00+00:00	165	28437	3074	13	2023-03-01 00:00:00+00:00	2023	3	1	9	2	Mar 2023	Wednesday
2023-03-01 01:00:00+00:00	182	32805	3602	37	2023-03-01 01:00:00+00:00	2023	3	1	9	2	Mar 2023	Wednesday
2023-03-01 02:00:00+00:00	152	29005	3463	14	2023-03-01 02:00:00+00:00	2023	3	1	9	2	Mar 2023	Wednesday
2023-03-01 03:00:00+00:00	182	30718	3538	14	2023-03-01 03:00:00+00:00	2023	3	1	9	2	Mar 2023	Wednesday
...	...	...	...	...	...	...	...	...	...	...	...	...
2023-04-29 17:00:00+00:00	181	22618	2641	13	2023-04-29 17:00:00+00:00	2023	4	29	17	5	Apr 2023	Saturday
2023-04-29 18:00:00+00:00	227	41464	4073	13	2023-04-29 18:00:00+00:00	2023	4	29	17	5	Apr 2023	Saturday
2023-04-29 19:00:00+00:00	202	23767	2724	13	2023-04-29 19:00:00+00:00	2023	4	29	17	5	Apr 2023	Saturday
2023-04-29 20:00:00+00:00	151	19550	2413	13	2023-04-29 20:00:00+00:00	2023	4	29	17	5	Apr 2023	Saturday
2023-04-29 21:00:00+00:00	151	21243	2544	13	2023-04-29 21:00:00+00:00	2023	4	29	17	5	Apr 2023	Saturday

Erste Auswertungen des DataFrame

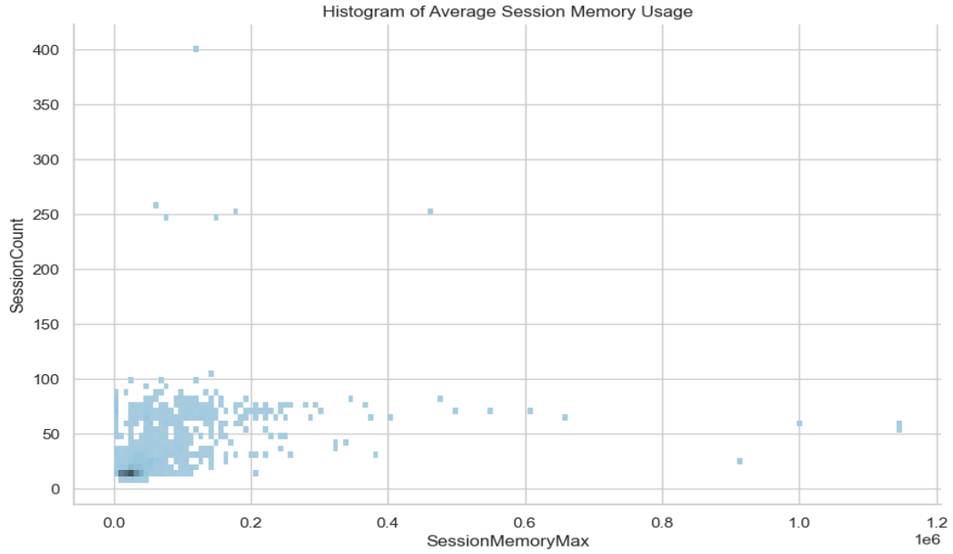
Erste grafische Analysen – Auffälligkeiten im Zeitverlauf

```
cols = list(df_base.columns)
#cols_subset = cols[:2] #[:3]
cols_subset = [cols[0], cols[3]]
cols_subset


# Nur gezielte Spalten plotten
pd.options.display.float_format = '{:.0f}'.format
df_base[cols_subset].plot(figsize=(20, 7))
# Grafik anzeigen
plt.show()
```



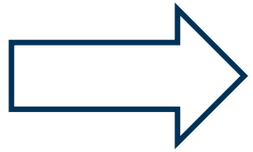
Erste grafische Analysen – Verteilungsdiagramm

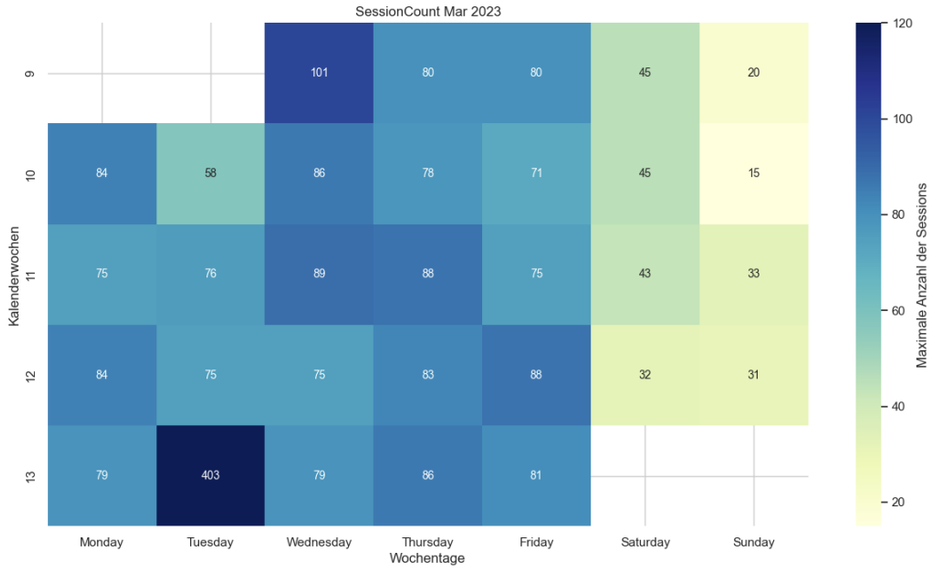
```
pd.options.display.float_format = '{:.0f}'.format
g = sns.displot(
    data=df_base,
    x=col2,
    y=col4,
    kind='hist',
    height=6,
    aspect=1.5,
    color='skyblue')

g.set(title='Histogram of Average Session Memory Usage', xlabel=col2, ylabel=col4)
plt.show()
```

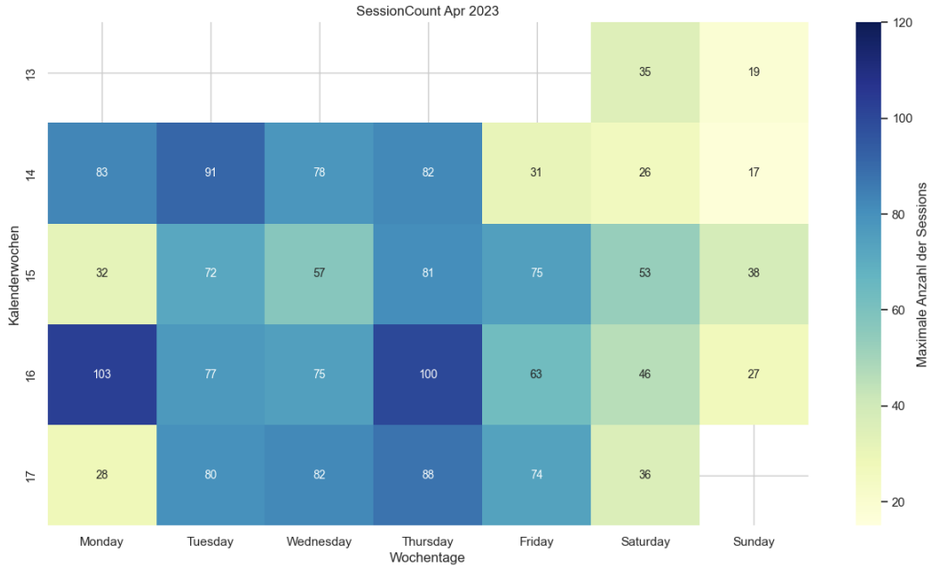


Auswertung zur Mustererkennung 1/3

```python
cols = list(df_monthly.columns)
col1 = cols[0]
col2 = cols[1]
col3 = cols[2]
col4 = cols[3]
value_col = cols[3]

cols_subset = cols[:-1] #[:3]-Nur die letzte #[:-1] Nur die letzte
cols_subset

# Heatmaps erstellen

selected_month = ['Mar 2023', 'Apr 2023']

#df_monthly['Month_Year'].unique():

for month_year in selected_month:
    month_df = df_monthly[df_monthly['Month_Year'] == month_year]
    cal_df = month_df.pivot_table(values=col4
                                  ,index='week'
                                  ,columns='Wochentag'
                                  ,aggfunc='max'
                                  #,fill_value=0
                                 )

    cal_df = cal_df.reindex(index=cal_df.index[::1]) # Kalenderwochen umdrehen
    weekday_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
    #cal_df = cal_df[weekday_order]


# Heatmap plotten
    fig, ax = plt.subplots(figsize=(15, 8))
    ax.set_title(col4 + ' ' + month_year)
    sns.heatmap(cal_df,
                cmap='YlGnBu',
                annot=True,
                fmt='.0f',
                annot_kws={"size": 10},
                cbar=True, ax=ax,
                vmin=15,
                vmax=120,
                cbar_kws={'label': 'Maximale Anzahl der Sessions'}
               )
    ax.set_xlabel('Wochentage')
    ax.set_ylabel('Kalenderwochen')
    plt.show()
```

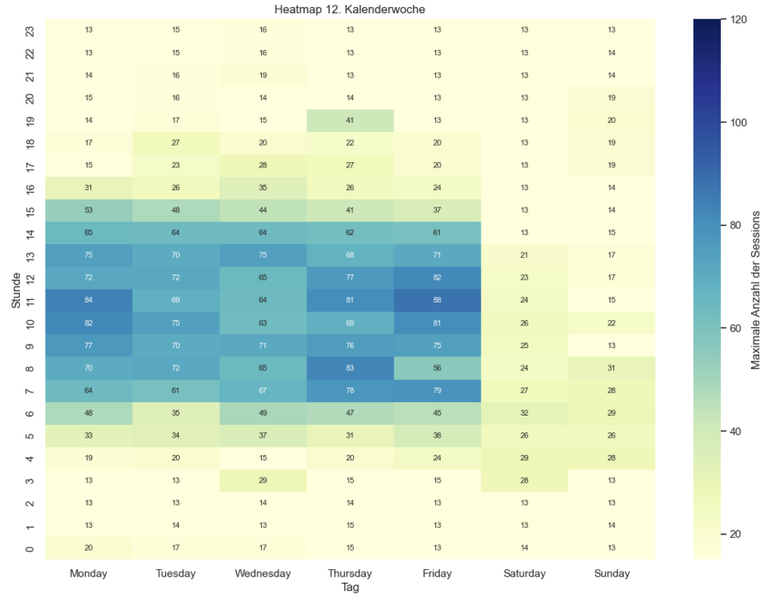


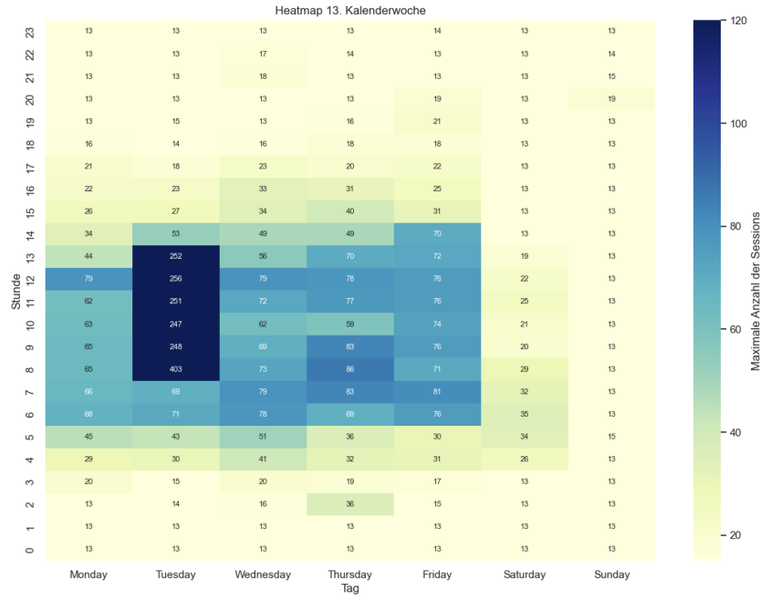




Auswertung zur Mustererkennung 2/3

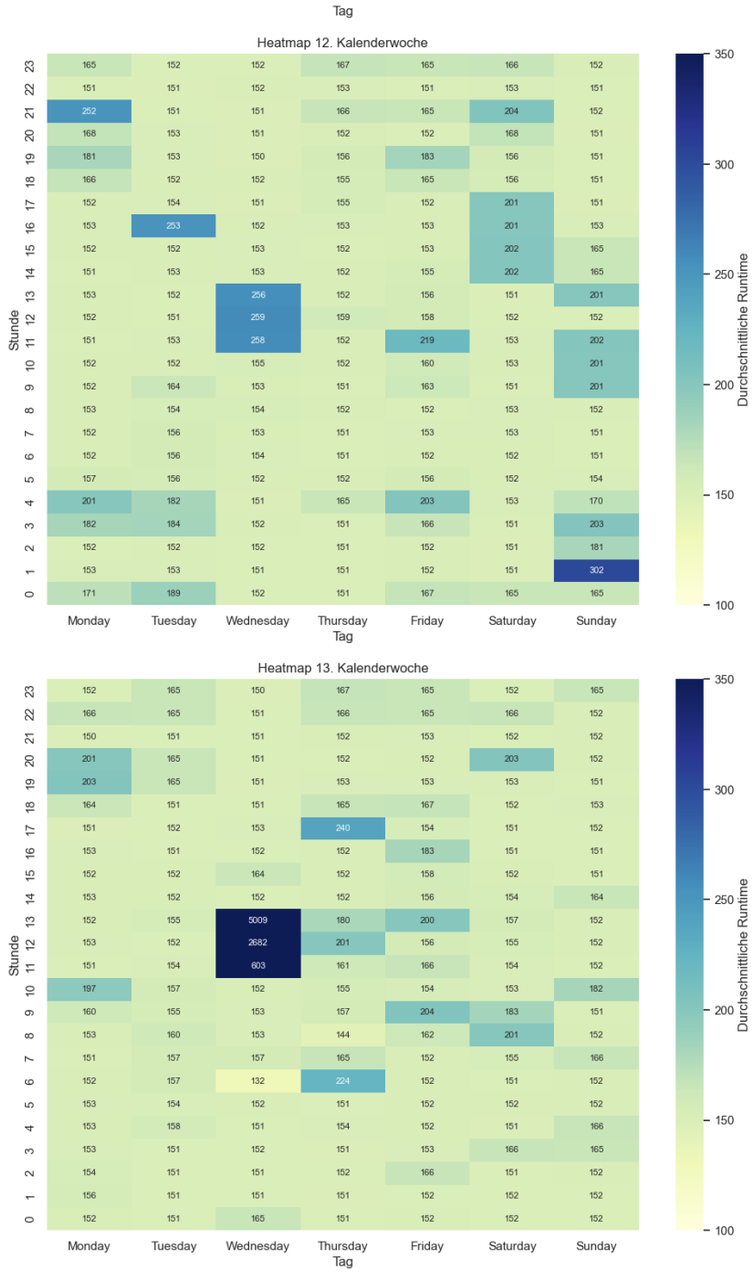
```python
df_weekly2 = df_base.reset_index()
#df_UM2.fillna(0, inplace=True)


# Erstelle zusätzliche Spalten für Jahr, Monat, Tag, Stunde und Kalenderwoche
df_weekly2['date'] = pd.to_datetime(df_weekly2['TIMESTAMP'])
df_weekly2['Year'] = df_weekly2['TIMESTAMP'].dt.year
df_weekly2['Month'] = df_weekly2['TIMESTAMP'].dt.month
df_weekly2['Day'] = df_weekly2['TIMESTAMP'].dt.day
df_weekly2['Hour'] = df_weekly2['TIMESTAMP'].dt.hour
df_weekly2['Week'] = df_weekly2['TIMESTAMP'].dt.isocalendar().week

weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']


# Setting Index for the next Function
df_weekly2 = df_weekly2.set_index('TIMESTAMP')

# Create a new column for the weekday
df_weekly2['Wochentag'] = df_weekly2.index.day_name()
df_weekly2['Wochentag'] = pd.Categorical(df_weekly2['Wochentag'], categories=weekdays, ordered=True)


#Variablen erstellen
cols = list(df_weekly2.columns)
col1 = cols[0]
col2 = cols[1]
col3 = cols[2]
col4 = cols[3]
value_col = cols[2]

cols_subset = cols[3] #[:3]-Nur die letzte #[:-1] Nur die letzte
cols_subset


# Liste mit den Kalenderwochen erstellen, für die eine Heatmap erstellt werden soll
selected_weeks = [ 10, 11, 12, 13, 14, 15, 16]

for week in selected_weeks:

    # Subset des DataFrames für die aktuelle Kalenderwoche erstellen
    df_week = df_weekly2[df_weekly2["Week"] == week]

    # Pivot-Tabelle erstellen, um Stunden pro Tag als Zeilen und Tage als Spalten anzuzeigen
    df_heatmap = df_week.pivot_table( index='Hour', columns='Wochentag', values=cols_subset)
    df_heatmap = df_heatmap.reindex(index=df_heatmap.index[::-1]) # Stundenanzeige umdrehen
    #df_heatmap.fillna(0, inplace=True)

    # Heatmap erstellen
    plt.figure(figsize=(14, 10))
    sns.heatmap(df_heatmap,
                cmap="YlGnBu",
                annot=True,
                fmt=".0f",
                annot_kws={"size": 8},
                vmin=15,vmax=120,
                cbar_kws={'label': 'Maximale Anzahl der Sessions'}
               )
    plt.title(f"Heatmap {week}. Kalenderwoche")
    plt.xlabel("Tag")
    plt.ylabel("Stunde")

    ax.set_xticklabels(df_week['date'].dt.date.unique(), rotation=90)
    plt.show()
```



Auswertung zur Mustererkennung 3/3

```
## Der DataFrame df_weekly2 existiert ja bereits – er ist auch die Basis für die Auswertung der


#Variablen erstellen
cols = list(df_weekly2.columns)
col1 = cols[0]
col2 = cols[1]
col3 = cols[2]
col4 = cols[3]
value_col = cols[0]

cols_subset = cols[0] #Hier ändere ich das Subset
cols_subset


# Liste mit den Kalenderwochen erstellen, für die eine Heatmap erstellt werden soll
selected_weeks = [ 10, 11, 12, 13, 14, 15, 16]

for week in selected_weeks:

    # Subset des DataFrames für die aktuelle Kalenderwoche erstellen
    df_week = df_weekly2[df_weekly2["Week"] == week]

    # Pivot-Tabelle erstellen, um Stunden pro Tag als Zeilen und Tage als Spalten anzuzeigen
    df_heatmap = df_week.pivot_table( index='Hour', columns='Wochentag', values=cols_subset)
    df_heatmap = df_heatmap.reindex(index=df_heatmap.index[::-1]) # Stundenanzeige umdrehen
    #df_heatmap.fillna(0, inplace=True)


    # Heatmap erstellen
    plt.figure(figsize=(12, 9))
    sns.heatmap(df_heatmap,
                cmap="YlGnBu",
                annot=True,
                fmt=".0f",
                annot_kws={"size": 8},
                vmin=100,
                vmax=350,
                cbar_kws={'label': 'Durchschnittliche Runtime'}
               )
    plt.title(f"Heatmap {week}. Kalenderwoche")
    plt.xlabel("Tag")
    plt.ylabel("Stunde")

    ax.set_xticklabels(df_week['date'].dt.date.unique(), rotation=90)
    plt.show()
```



BA RISE UP WITH NETWORK

Bester Arbeitgeber: join the team

BA BUSINESS ADVICE

Infos an unserem Messestand –

direkt hier!

Check unsere Website:
www.ba-gmbh.com

- New Work Balance: arbeiten im Digital Workspace
- New Pay: lohnende Gehaltsmodelle & Benefits, Sportangebot, Firmenevents – inclusive legendärer Partys
- Enabling: individuelle Karrierewege und eine steile Lernkurve
- Onboarding: ein Coach steht Dir zu Seite
- Mobility Optionen: Bike & Car (inkl. Flatrate)
- Equipment: born digital needs Technik & Tools

Top 2023 Company kununu

Top 2022 Company kununu

SAP Gold Partner