

Ein webbasiertes Multiple-Kernel-System für die automatisierte Erstellung von analytischen Berichten

Dirk Schieborn
ESB Business School der
Hochschule Reutlingen

Adresse

E-Mail

Alexander Wagner
Selbständiger Berater für
Data Science & Advanced
Analytics
Gensingerstraße/57
10315 Berlin
av3.wagner@gmail.com

Zusammenfassung

Diese Arbeit gehört zum Forschungsgebiet „Automatisierte Erstellung von Analytischen Berichten als Webbasierte Applikationen“. Das Ziel des Projektes besteht in der Automatisierung von Routineoperationen der Erstellung von Analytischen Berichten über die Standardisierung der Prozessabläufe des Datenmanagements bis hin der Platzierung des Berichtes, Software in Daten auf Web-Cloud und Verwaltung des Client-Cloud Prozesses. Zur Anwendung kommt ein breites Spektrum an Werkzeugen der Datenverarbeitung, der Softwareintegration und der Steuerung von IT-Prozessen.

Schlüsselwörter: Automatisierte Erstellung von Analytischen Berichten, Python, Streamlit, Streamlit.io-Cloud, WebApp, SAS, Autoit, MS OFFICE.

1 Das Ziel

Das übergreifende Ziel dieses Projektes ist über die Implementierung einer automatisierten Erstellung von Analytischen Berichten eine effiziente Steuerung der Kaskade aller Datenverarbeitungsschritte und der abschließenden Platzierung des Berichtes auf Web-Cloud und Verwaltung des Client-Cloud Prozesses zu ermöglichen. Die erste Aufgabe ist daher, alle Berichtsinformationen (Texte, Tabellen, Grafiken) in einem flexiblen Informationssystem/GitHub Repositorium zu organisieren. Die zweite Aufgabe besteht darin, alle notwendigen analytischen und Daten-Management Verfahren und Methoden in einem speziellen Programm-Tool zu integrieren in Form einer Projekt-GUI. In der dritten Phase werden Bericht auf Web-Cloud platziert und Verwaltung des Client-Cloud Prozesses durchgeführt.

2 Analytische Bericht

Der analytische Bericht besteht aus den folgenden Standardelementen, und zwar:

- Frontseite
- Inhaltsverzeichnis
- Überschriften und Fußnoten

- Texten
- Tabellen
- Grafiken
- Listen

Das System entwickelt alle diese Komponenten mit Hilfe von Python DOCx-Bibliothek, SAS ODS und speziellen Autoit-Funktionen. Der Bericht wird in Form eines MS-WORD - und PDF-Dokumentes erstellt und im Repository GitHub abgelegt.

2.1 Tabellen und Graphiken

Die Tabellen werden im Regelfall mit Hilfe von Python DOCx-Bibliothek erstellt, einigen Tabellen, die mit SAS Proc Report erzeugt werden, kann man in Bericht problemlos integrieren. Die Erstellung von Graphiken wiederum erfolgt mit Hilfe von Python-Bibliotheken, SAS/Graph und SAS® 9.4 ODS Graphics Prozeduren, Funktionen R-System, etc.

3 Das Konzept, die Struktur und die Funktionalitäten des Systems

Das System ist eine Integration der unterschiedlichen Software (Kerne). Die Abbildung 1 stellt das konzeptuelle Schema des webbasiertes Multiple-Kernel-System für die automatisierte Erstellung von analytischen Berichten dar.

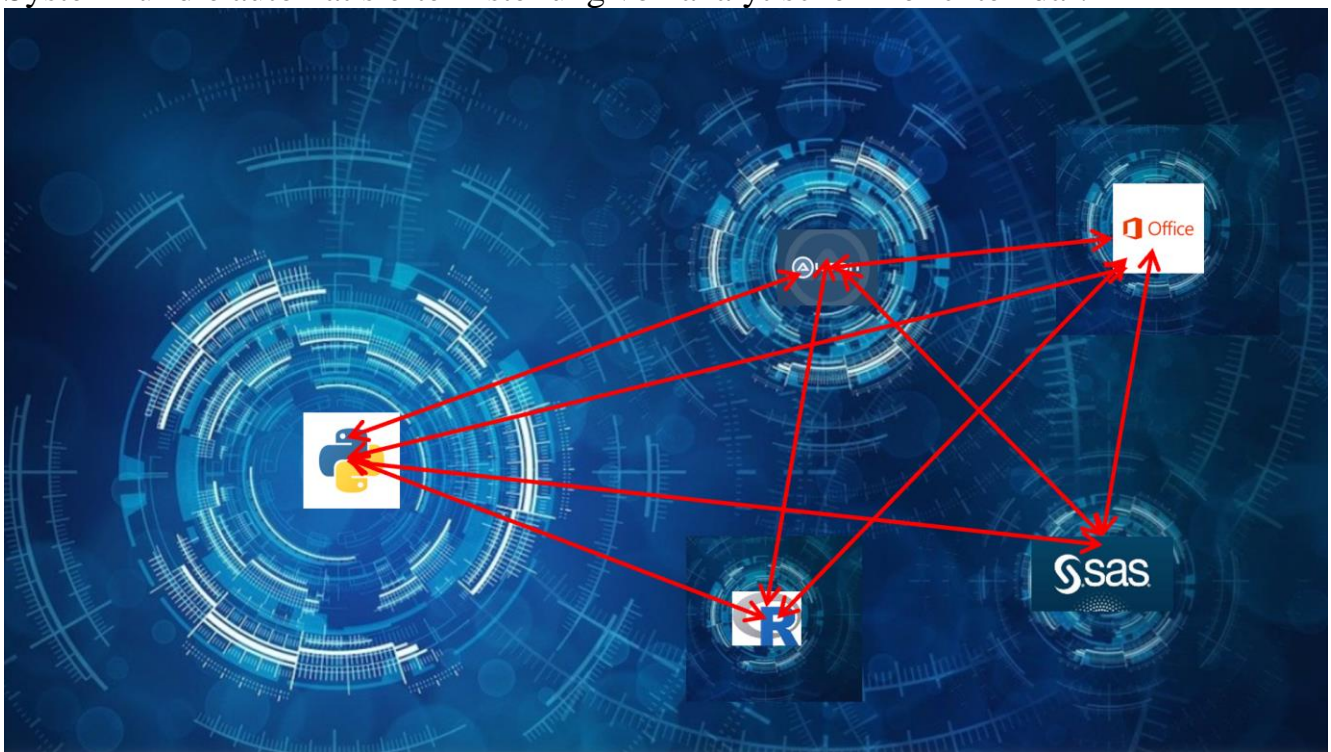


Abbildung 1: Ein webbasiertes Multiple-Kernel-System für die automatisierte Erstellung von analytischen Berichten

3.1 Die Kerne des Systems

Das System besteht aus Folgenden Kernen/Komponenten:

- Python
- AutoIt
- SAS
- R und andere Software-Pakete
- MS OFFICE

3.1.1 Das Python Kern

Python 3.1x ist Basis-Kern für die Entwicklung des analytischen Berichts und Erstellung eines WEB-Applikation in Streamlit.io-Cloud.

3.1.1.1 Python-Docx Programm-Bibliothek

Python-Docx ist eine Python-Bibliothek, die für die Arbeit mit Microsoft Word-Dokumenten verwendet wird. Es ermöglicht Benutzern, Informationen aus Word-Dokumenten zu erstellen, zu bearbeiten und zu extrahieren, ohne die Microsoft Office Suite verwenden zu müssen. Mit Python-Docx kann man neue Word-Dokumente erstellen, vorhandene ändern und Daten wie Text, Tabellen und Bilder extrahieren.

Es unterstützt auch Formatierungsoptionen wie Fett, Kursiv, Unterstreichung, Schriftgröße und Farbe. Die Liste 1 stellt die Basis-Fragmente des Programms für die Erstellung der Tabellen und Grafiken des Berichtes.

Erstellung von analytischen Berichten

Tabelle-Erstellung

```
t = doc.add_table(n_rows+2, n_cols, style="Table Grid")
set_repeat_table_header(t.rows[0])
for j in range(n_cols):
    if include_index:
        ha=1
    else:
        t.cell(0,j).text = df3.columns[j]

for i in range(2, n_rows+2):
    for j in range(n_cols):
        if include_index:
            t.cell(i, j).text = str(df3.values[i-2,j])
            t.cell(i, j).paragraphs[0].paragraph_format.alignment =
WD_TABLE_ALIGNMENT.RIGHT
        else:
            t.cell(i, j).text = str(df3.values[i,j])
t.cell(i, 0).paragraphs[0].paragraph_format.alignment =
WD_TABLE_ALIGNMENT.LEFT
t.rows[i].height_rule = WD_ROW_HEIGHT_RULE.EXACTLY
```

Einfügen der WHO-Logo

```
cell = t.cell(0, 0)
```

```

cell.paragraphs[0].alignment = WD_PARAGRAPH_ALIGNMENT.CENTER
paragraph = cell.paragraphs[0]
run = paragraph.add_run()
run.add_picture("..\\INDATEN\\who-logo-png.png", Cm(1), Cm(1))

```

Einfügen der Tabellen-Zeile mit copyright

```

t.add_row()
g = t.cell(n_rows+2, 1)
h = t.cell(n_rows+2, 6)
g.merge(h)
cell = t.cell(n_rows+2, 6)
cell.paragraphs[0].alignment = WD_PARAGRAPH_ALIGNMENT.LEFT
cell.paragraphs[0].add_run("© Dr. Alexander Wagner. All Rights Reserved")
cell.vertical_alignment = WD_CELL_VERTICAL_ALIGNMENT.CENTER
change_table_cell(t.rows[n_rows+2].cells[2],
background_color="lightgreen", font_color="0000ff", font_size=8,
bold=True, italic=True)

```

Liste 1: Fragment der Python Def

Die Abb. 2 stellt die mit Hilfe von Python-Docx erzeugte Tabelle dar.

Tabelle 1: Altersverteilung für ausgewählte Länder nach WHO: Aruba, Australia, Austria, Azerbaijan, Bahamas, The Bahrain, Bangladesh, Barbados, Belarus, Belgium


	Gesamt Population	Altersgruppen				
	Median	0-14	15-24	25-54	55-64	65+
Aruba	39.3	17.64	12.78	41.72	14.28	13.59
Australia	38.7	17.8	12.79	41.45	11.83	16.14
Austria	44.0	14.01	11.07	42.42	13.23	19.26
Azerbaijan	31.3	22.95	14.84	45.39	10.17	6.64
Bahamas	32.0	22.55	16.4	44.14	9.16	7.75
The Bahrain	32.3	19.08	15.65	56.04	6.28	2.95
Bangladesh	26.7	27.76	19.36	39.73	6.93	6.23
Barbados	38.6	17.97	12.74	44.06	13.43	11.81
Belarus	40.0	15.78	10.29	44.76	14.21	14.95
Belgium	41.4	17.16	11.34	40.05	12.86	18.58
© Dr. Alexander Wagner. All Rights Reserved						

Abb. 2: Die mit Hilfe von Python-Docx erzeugte Tabelle

3.1.1.2 Streamlit

Streamlit (<https://streamlit.io/>) ist eine Open-Source Python-Bibliothek, die das Erstellen interaktiver Webanwendungen vereinfacht. Es wurde so konzipiert, dass es einfach und benutzerfreundlich ist, wobei der Schwerpunkt auf der Erstellung der Anwendung und nicht der zugrunde liegenden Infrastruktur liegt.

Es gibt viele Situationen, in denen Streamlit ein nützliches Werkzeug zum Erstellen von Webanwendungen in Python sein kann. Dazu gehören:

- Schnelles Prototyping und schnelle Entwicklung kleiner Anwendungen
- Erstellen interaktiver Dashboards für die Datenexploration und -visualisierung
- Erstellen benutzerdefinierter interaktiver Tools für die Datenanalyse
- Erstellen webbasierter Schnittstellen für Machine Learning-Modelle oder Datenpipelines

3.1.2 Das Autoit Kern

Autoit-Kern ist zentraler Punkt des Systems, es verwaltet den ganzen Prozess von Erstellung des Berichts bis zu pushen und Herunterladen der Dateien von Streamlit.io-Cloud auf lokalen PC und umgekehrt. Dieser Kern besteht aus mehreren UDF, die viele Daten Management- und Connection Funktionen realisieren.

3.1.3 Das SAS-Kern

SAS-Kern verwendet man für die Erstellung der Tabellen und Grafiken, die später für Erstellung des Berichts verwendet werden kann. Es kann auch die Tabellen und Grafiken aus vorfertigen SAS-Berichten extrahieren und in Python-Docx Dokument integrieren. Die Liste 2 stellt ein SAS-Macro für die Erstellung der Grafiken dar.

```
%MACRO GRAPX (G=) ;
    Title;
    %COLOR (K=&G) ;
    ods graphics / reset width=8in height=5in noscale noborder
    imagename="GRAPH&G";
    proc sgplot data=DATEN.DEMOGRA (where=(GR=&G)) nowall noborder;
        styleattrs datacolors=(&STRING);
        vbar country / response=age_0_14 nostatlabel barwidth=0.6
        nooutline dataskin=preserved
            discreteoffset=-0.15
        baselineattrs=(thickness=0);
        vbar country / response=age_15_24 nostatlabel barwidth=0.5
        nooutline dataskin=preserved
            discreteoffset=-0.05
        baselineattrs=(thickness=0);
        vbar country / response=age_25_54 nostatlabel barwidth=0.4
        nooutline dataskin=preserved
            discreteoffset=0.05
        baselineattrs=(thickness=0);
        vbar country / response=age_55_64 nostatlabel barwidth=0.3
        nooutline dataskin=preserved
            discreteoffset=0.15
        baselineattrs=(thickness=0);
        vbar country / response=age_65 nostatlabel barwidth=0.2
        nooutline dataskin=preserved
            discreteoffset=0.25
        baselineattrs=(thickness=0);
        keylegend / location=outside position=topright noborder
        valueattrs=(size=8 color=gray);
```

```

        xaxis    fitpolicy=thin    display=(nolabel    noticks    noline)
valueattrs=(size=8    color=BLACK);
        yaxis    grid    display=(noline    noticks    nolabel    noline)
valueattrs=(size=8    color=gray);
run;
%MEND GRAPX;

%MACRO START;
%DO GR=1 %TO 23;
        %GRAPX (G=&GR) ;
%END;
%MEND ;

```

Liste 2: Die SAS-Macros für die Erstellung der Grafiker

Die Abb. 3 stellt die mit Hilfe von SAS GRAPH erzeugte Grafik dar.

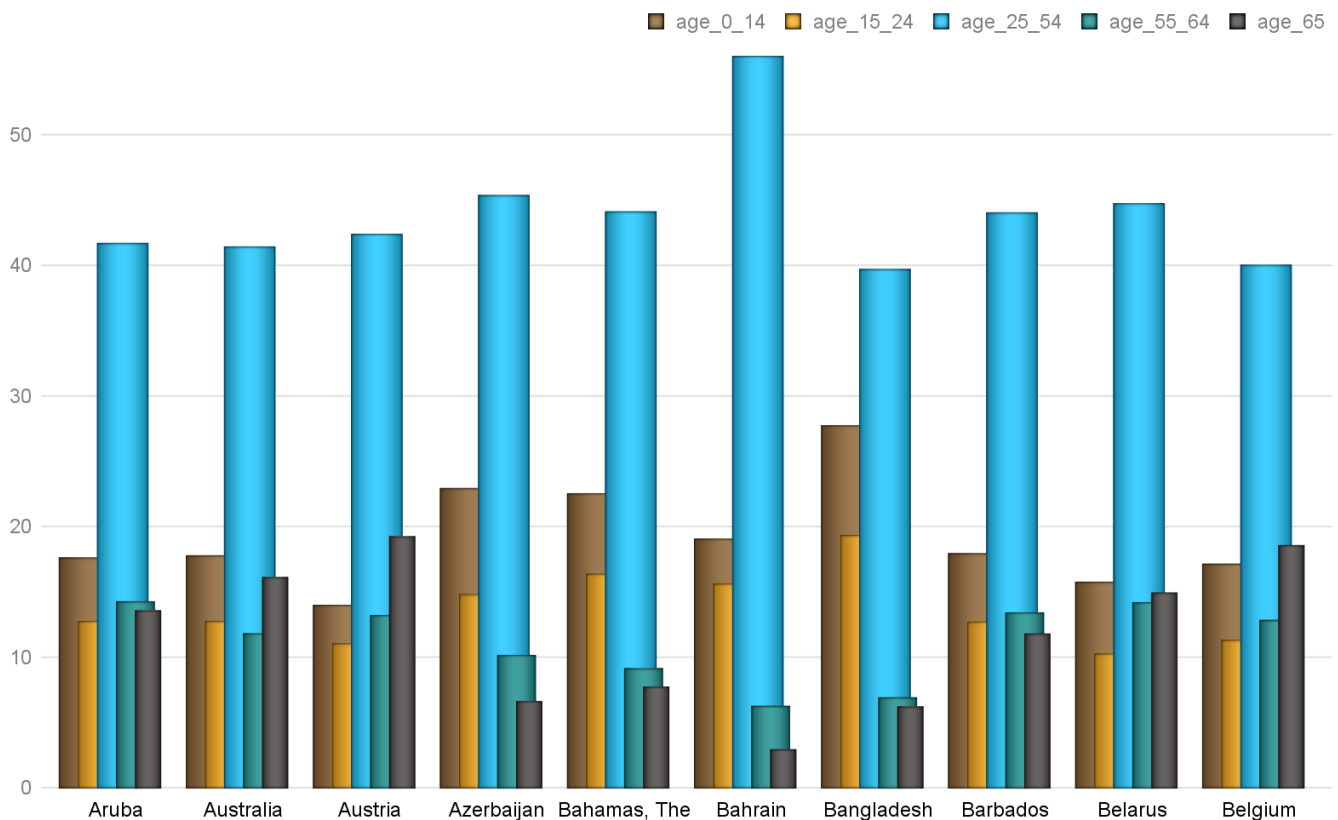


Abb. 3: SAS-Grafik: Altersverteilung für ausgewählte Länder nach WHO:
Aruba,Australia, Austria, Azerbaijan, Bahamas, The Bahrain, Bangladesh,
Barbados, Belarus, Belgium.

3.2 Die Funktionen des Systems

Das System realisiert die folgenden Funktionen:

- Mit Hilfe von Python, SAS, R, etc. erstellt man die speziellen Tabellen und Grafiken für die Integration in Bericht. Alle diese Berichts-Elemente werden lokal in speziellem Projekt-Directory abgelegt.

- Die vorfertige Text-Blöcke für die Integration in Bericht wurden in speziellem Projekt-Directory als Word-Dokumente abgelegt
- Mit Hilfe von Python DOCx-Bibliothek erstellt man den analytischen Berichte
- Der Bericht wird erst lokal auf speziellem Projekt-Directory abgelegt, danach in Projekt GitHub Repositorium gepusht
- Korrektur bzw. Nachbesserung der Programm Code und Berichts wird in der Regel lokal in GUI-Fenster gemacht und danach nach GitHub gepusht
- Die spezielle Autoit UDF und Python Def realisieren mit Hilfe von speziellen Autoit Steuerelementen (Buttons, Pop- und Contents-Menu) die Download- bzw. Push/Upload Daten Transfer-Prozesse und Update des Systems.

3.3 Das System GUI mit Steuerelementen

Das Graphical User Interface (GUI) ist eine spezielle Oberfläche des Systems für die Interaktive Steuerung des Prozesses bei der Berichtserstellung und Verwaltung des Systems. Das GUI wurde mit Hilfe von Autoit [2] programmiert und steuert den automatisierten Prozess der Berichterstellung und Web-Applikation über die Integration der Python, SAS, MS-Word Kerne.

3.1.4 Die Benutzerdefinierte Funktion (UDF) des Autoit-Steuersystems

Das System ist eine Integration von benutzerdefinierten Funktionen (User Defined Functions, UDF) [18]. Die UDF wurden mit Autoit entwickelt und realisieren die verschiedenen Funktionen des Systems, wie:

- Funktionen für die Erstellung und Korrektur des Berichts
- Funktionen für die Datenmanagement (Hoch- und Herunterladen der Dateien)
- Funktionen für die Verwaltung der GutHub Repositorium und Streamlit-Cloud

3.1.5 Back-End Python-Funktionen

In Back-End Subsystem werden die folgenden Aufgaben realisiert:

- Automatisierte Erstellung des Berichts
- Korrektur bzw. Aktualisierung des Berichtes
- Herunterladen des Berichts vom GitHub Repositorium in GUI
- Hochladen/Pushen des Berichts ins GitHub Repositorium

Alle diese Aufgaben werden mit Hilfe von Autoit-UDF und Python-Def realisiert und mit Hilfe von GUI gesteuert und visualisiert

Python Code wird mit Hilfe von Standard EDA entwickelt bzw. geändert/nachgebessert, es gibt auch die Möglichkeit die Python Code direkt in GitHub Repositorium ändern/Nachgebessert.

3.1.5.1 Korrektur bzw. Aktualisierung des Berichtes

Erstens, es musste man der Bericht vom GitHub Repositorium in GUI herunterladen (siehe auch Abschnitt 3.1.5.2).

Zweitens, der Bericht musste man mit Hilfe von Microsoft Word manuell bearbeiten, in PDF-File konvertieren und lokal speichern

Drittens, die beide Datei (Docx und PDF) wird mit Hilfe von Python **def push()** in GitHub Repositorium hochladen und alte Dateien ersetzen (siehe auch Abschnitt 3.1.5.2)

3.1.5.2 Hochladen/Pushen des Berichts ins GitHub Repositorium

Für die die Korrektur bzw. Nachbesserung des Berichts gibt es speziellen Autoit und Python Funktionen.

Die Liste 3 stellt die Python Def push() dar.

```
def push() :
    #githubToken = 'ghp_vT2lf6mLRsxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
    path = os.getcwd()
    def push(file):
        with open(file, "rb") as f:
            encodedData = base64.b64encode(f.read())
            headers = {
                "Authorization": f'''Bearer {githubToken}''',
                "Content-type": "application/vnd.github+json"
            }
            data = {
                "message": "My commit message",
                "content": encodedData.decode("utf-8")
            }
            r = requests.put(githubAPIURL, headers=headers,
                json=data)
            udir = "https://api.github.com/repos/av3wagner/KSFE2023/contents/"
            ulist = ["KSFE2023.docx", "KSFE2023.pdf"]
            for l in range(len(ulist)):
                githubAPIURL = udir + ulist[l]
                file=os.path.join(os.getcwd(), "", "OUTPUT", ulist[l])
                push(file)
```

Liste 3: Fragment der Python Def push()

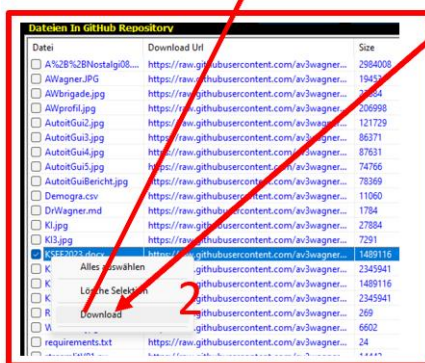
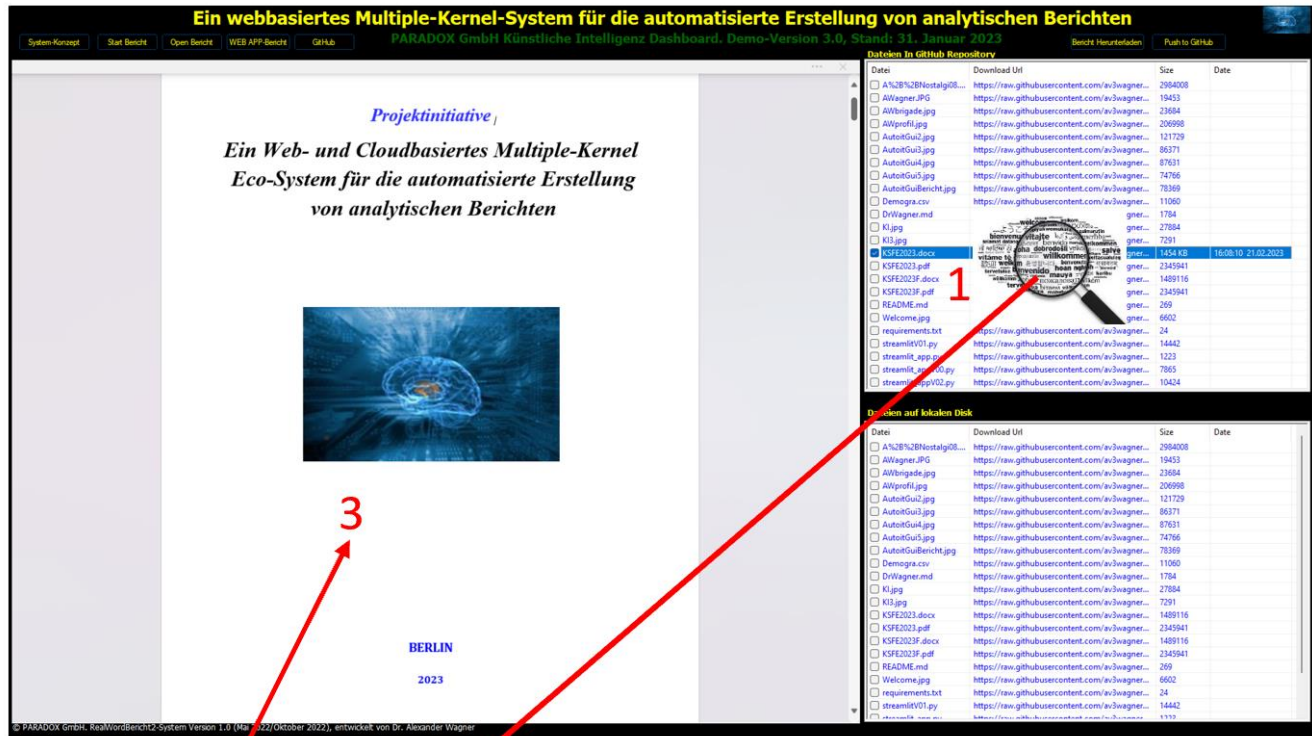
3.1.5.3 Herunterladen des Berichts vom GitHub Repositorium

Für die die Korrektur bzw. Nachbesserung des Berichts gibt es speziellen Autoit und Python Funktionen. Die Liste 1 stellt die Autoit UDF Download() dar. Die Abb. 5 stellt den herunterladenden Bericht auf GUI dar. Die Liste 4 stellt die Autoit UDF Download() dar. Die Abb. 5 stellt Download-Prozess der Datei KSFE2023.docx vom GitHub Repositorium auf System-GUI dar.

Der Prozess besteht aus den folgenden Operationen:

1. Anwender Sucht in der Liste der Dateien, die in der GitHub Repositorium liegen, ein Dokument
2. Markiert mit dem Maus dieses Dokuments
3. Mit Rechte Maus-Klick wählt in Kontext-Menü Option Download und klickt
4. Danach startet Autiot UDF, ladet die Datei herum und platziert offene Word-Dokument auf System GUI

Es konnte man mehr Dateien mit Hilfe von „Point and Klick“ Methode Herunterladen. Alternativ kann man mit Mausklick auf Button „Herunterladen“ der Prozess voll automatisch durchführen und eine Liste von Dateien Herunterladen. Die Liste von Dateien wird in spezielle Json-Datei definiert und auf GitHub- und lokalen Repositorium gespeichert.



- 1 – Selektion des Dokuments
- 2 – Mausklick mit rechte Taste auf Menu-Option „Upload“
- 3 – Ausgewählte Word-Dokument wurde runtergeladen, geöffnet und auf GUI platziert

Abb. 5: Download-Prozess der Datei KSFE2023.docx vom GitHub Repository auf System-GUI.

Das Ergebnis der Funktion kann man direkt von GitHub Repository Contents ablesen. Die Abb. 6 stellt die Information über Datei <https://api.github.com/repos/av3wagner/KSFE2023/contents/KSFE2023.docx>, die in GitHub Repository liegt dar.

Func Download()

```
Run(@ComSpec & ' /c start
https://raw.githubusercontent.com/av3wagner/KSFE2023/main/KSFE2023.docx
', ' ', @SW_SHOW)
Sleep(100)
WinWait("[CLASS:#32770]")
```

```

WinActivate("[CLASS:#32770]")
WinWaitActive("[CLASS:#32770]")
.....
.....
.....
$Bhandle=$aWinList[1][1]
WinActivate ($Bhandle)
ControlFocus($Bhandle, "", "Edit1")
Sleep(100)
ControlSend($Bhandle, "", "Edit1", @ScriptDir & "\KSFE2023")
Sleep(100)
Send("{enter}")
Sleep(100)
While not FileExists(@ScriptDir & "\KSFE2023.docx")
    Sleep(100)
WEnd
Local $sDocument = @ScriptDir & "\KSFE2023.docx"
Sleep(50)
Send("#r")
Sleep(50)
Send($sDocument)
Send("{Enter}")
.....
.....
.....
$Word.Visible=True
DllCall("user32.dll", "int", "SetParent", "hwnd", $Ahandle,
"hwnd", WinGetHandle($GUI))
DllCall("user32.dll", "long", "SetWindowLong", "hwnd",
$Ahandle, "int", -16, "long", BitOR($WS_POPUP, $WS_CHILD,
$WS_VISIBLE, $WS_CLIPSIBLINGS))
WinMove($Ahandle, "", @DesktopWidth/8, 80, 3*@DesktopWidth/4,
@DesktopHeight-105)

```

EndFunc

Liste 4: Fragment der Autoit UDF DownLoad()

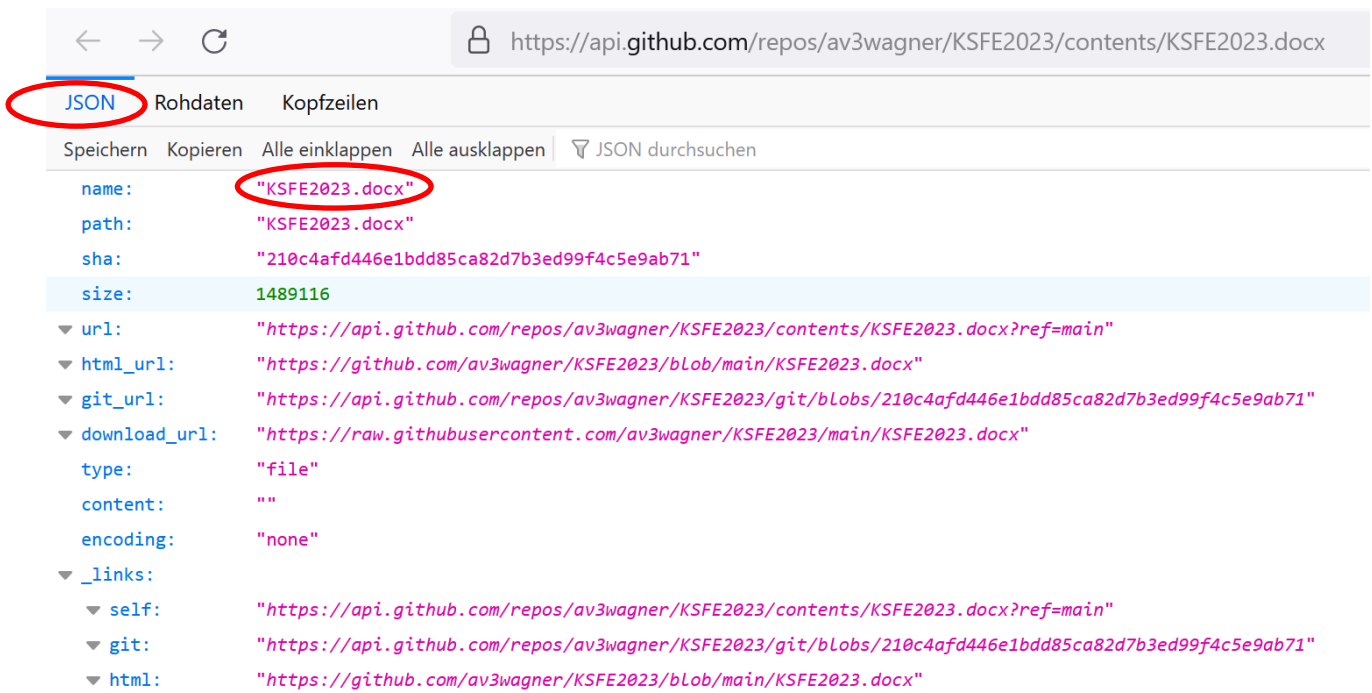


Abb. 6: Information über die KSFE.docx Datei n GitHub Repositorium

4. Web-App (Front-End) des Systems

Wir verwenden Streamlit Community Cloud, eine Open-Source-Plattform für die Community, um Streamlit-Apps bereitzustellen und freizugeben.

Der Prozess zum Bereitstellen der Streamlit-App in Streamlit Cloud funktioniert in 3 einfachen Schritten:

1. Generierung der requirements.txt die alle Abhängigkeiten für den Projekt auflistet. Dies wird benötigt, um Streamlit mitzuteilen, welche Bibliotheken installiert werden müssen, um den Code auszuführen. Es kann man Pipregs verwenden, um die Datei basierend auf den Importen in Programm Code automatisch zu generieren. Es kann man in Terminal einfach zu dem Speicherort / Pfad, in dem Sie die app.py Datei für den Projekt speichern, und "pipreqs" eingeben. Das war's, eine requirements.txt Datei wird generiert.
2. Das Hosting den Code in einem öffentlichen GitHub-Repository und Sicherstellung, dass das Repository auch die requirements.txt Datei enthält.
3. Registrieren an der für share.streamlit.io. Es muss man auf Neue App klicken, wählen das "Aus vorhandenem Repository", fügen dann Projekt GitHub-URL und den Namen der App ein (in diesem Fall app.py - die Datei, die den auszuführenden Code enthält) und klicken dann auf Bereitstellen!

Die Abb. 7. Stellt das Bereitstellen (Deploy) der Web-App in Streamlit Community Cloud dar.

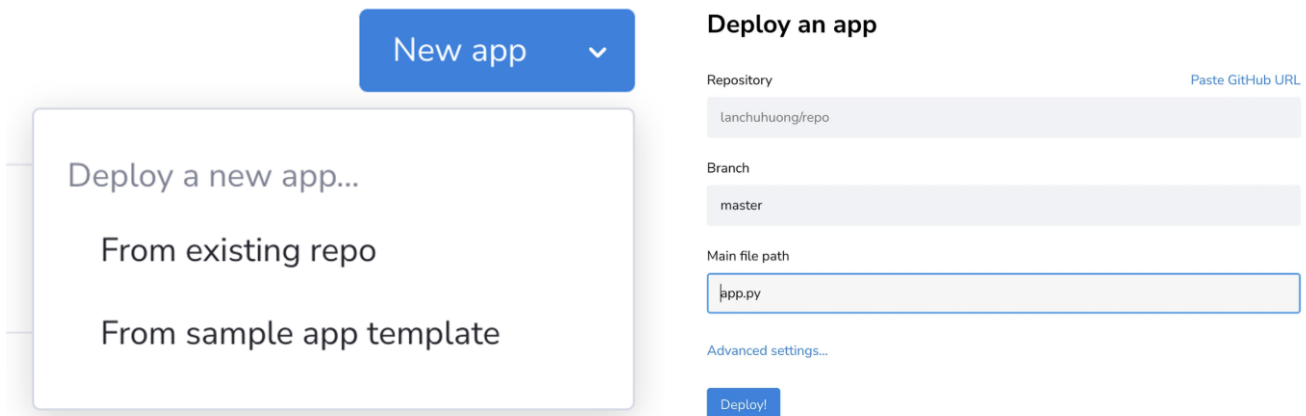


Abb. 7: Bereitstellen (Deploy) der Web-App in Streamlit Community Cloud

Die Abb. 8 stellt das Streamlit Web-App auf der GUI des Systems dar.

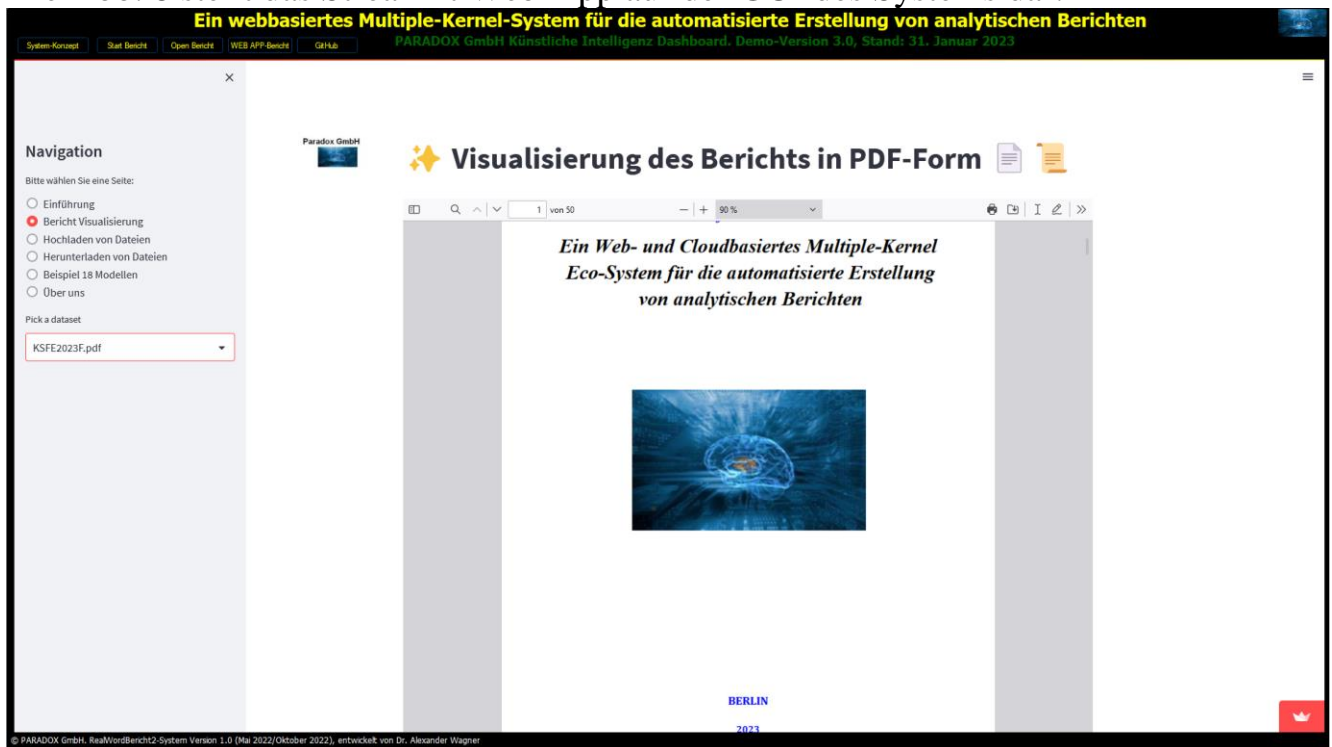


Abb. 8: Streamlit.io-Cloud Web-App auf der GUI des Systems

5 Prozessablauf der automatisierten Erstellung des analytischen Berichts

Der Prozessablauf für die Erstellung des analytischen Berichts läuft vollautomatisch. Durch Standard Point-and-Click Manipulationen wird der gesamte Prozess mit Hilfe von GUI-Elementen gesteuert. Die Abb. 9 stellt das Prozessablauf der automatisierten Erstellung des analytischen Berichts dar.

Der Prozessablauf besteht aus Ertappen bzw. Elementen:

- System GUI als Kommando-Zentrale

- Data-Pushing
- Deploy des Web-App
- GitHub-Repositorium
- Streamlit.io-Cloud
- Produktion des Systems von der Seite der Endanwender

6 Weiterentwicklung des Systems

Das System funktioniert als Streamlit.io Web-Applikation und konnte mit seiner Effizienz vielfach überzeugen. Dennoch bestehen Ansatzpunkte für Nachbesserung und Weiterentwicklung. Die folgenden Erweiterungs- und Optimierungspotentiale werden priorisiert:

- Verwendung etwas funktional und technologisch überlegenen Django-System (<https://www.djangoproject.com/>) statt Streamlit.
- Verwendung etwas funktional und technologisch überlegenen Cloud-Systeme (Amazon EC2, Microsoft Azure Cloud, etc.) statt Streamlit.io
- Korrektur bzw. Nachbesserung des Berichts in Cloud von mehreren Anwendern gleichzeitig
- Verwendung der Json Steuer/Config-Datei für voll automatisierte Berichtserstellung
- Automatisierung des Uploads/Download Prozesses z.B. mit Hilfe von Windows-Task

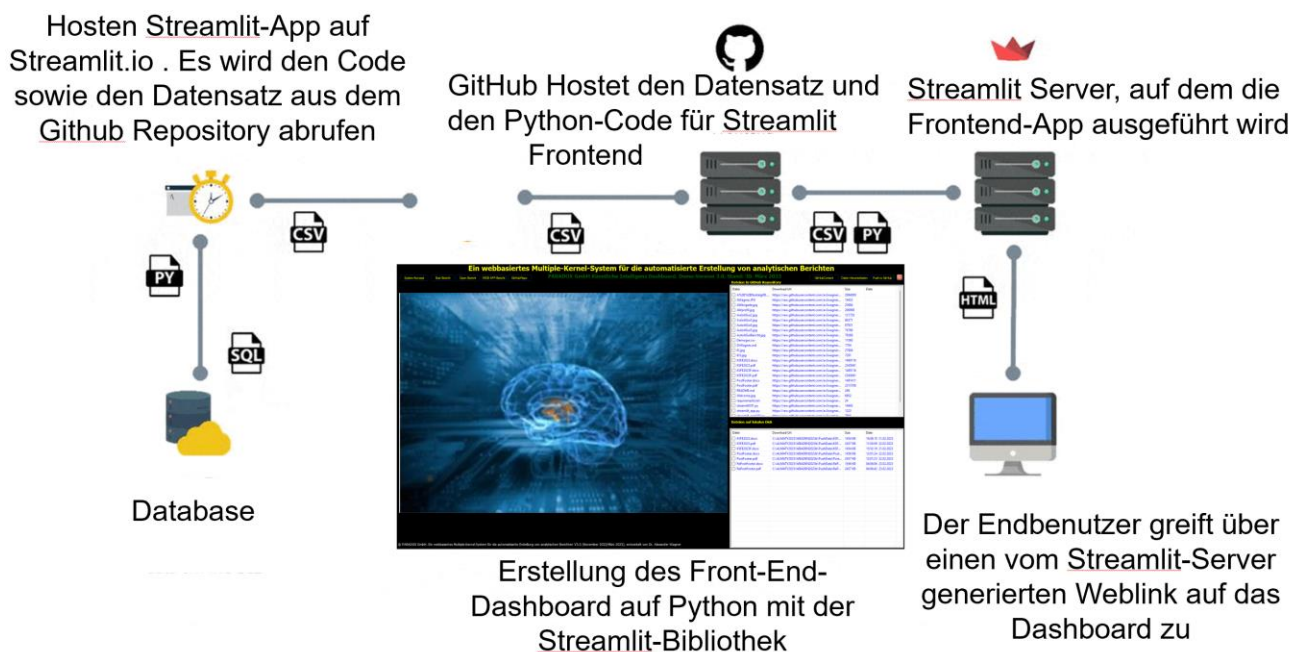


Abb. 9: Prozessablauf der automatisierten Erstellung des analytischen Berichts

Literatur

- [1] AutoIt website. <https://www.autoitscript.com/site/>
- [2] User Defined Function. https://de.wikipedia.org/wiki/User_Defined_Function
- [3] Base SAS. SAS Notes and Concepts for ODS, The RTF Destination. https://support.sas.com/rnd/base/ods/templateFAQ/Template_rtf.html
- [4] Lauren Haworth, Applying Microsoft Word Styles to ODS RTF Output. <http://www2.sas.com/proceedings/sugi30/043-30.pdf>
- [5] Kirk Paul Lafler Output Delivery System (ODS) – Simply the Basics. Software Intelligence Corporation, Spring Valley, California. <http://www.scsug.org/wp-content/uploads/2012/11/Output-Delivery-System-ODS-%E2%80%93-Simply-the-Basics-SCSUG-2012.pdf>
- [6] Carol Matthews, Elena Kalchenko, Pretty Please?! Making RTF Output “Pretty” with SAS. United Biosource Corporation, Blue Bell, Pennsylvania. <https://www.pharmasug.org/proceedings/2013/IB/PharmaSUG-2013-IB08.pdf>
- [7] SAS® 9.4 ODS Graphics: Procedures Guide, Sixth Edition. <http://support.sas.com/documentation/cdl/en/grstatproc/69716/HTML/default/viewer.htm#titlepage.htm>
- [8] python-docx 0.8.11 documentation. <https://python-docx.readthedocs.io/en/latest/>
- [9] python-docx-template. <https://docxtpl.readthedocs.io/en/latest/>
- [10] Automate Word Document (.docx) With Python-docx And pywin32
- [11] Jay, Automate Word Document (.docx) With Python-docx And pywin32. <https://pythoninoffice.com/automate-docx-with-python/>
- [12] Dr. Rik Voorhaar, How to edit Microsoft Word documents in Python. <https://www.rikvoorhaar.com/python-docx/>
- [13] PyGithub- documentation <https://pygithub.readthedocs.io/en/latest/introduction.html>
- [14] Abdou Rockikz, Python script to create GitHub repository. <https://iq.opengenus.org/create-github-repository-python/>
- [15] Code for How to Use Github API in Python Tutorial. <https://www.thepythoncode.com/article/using-github-api-in-python>
- [16] Streamlit-Dokumentation: <https://docs.streamlit.io/>
- [17] Marc, Introduction to Streamlit: Building Interactive Web Apps using Python. <https://marccodess.medium.com/introduction-to-streamlit-building-interactive-web-apps-using-python-62c250ba858d>
- [18] Alan Jones, Getting Started With Streamlit Web Based Applications. <https://towardsdatascience.com/getting-started-with-streamlit-web-based-applications-626095135cb8>

- [19] Alan Jones, Streamlit from Scratch: Embedding Images, Video and Audio.
<https://towardsdatascience.com/streamlit-from-scratch-embedding-images-video-and-audio-8b2e8b98fad4>
- [20] Navid Mashinchi, The Current State of COVID-19 From 3 Different Perspectives.
<https://towardsdatascience.com/the-current-state-of-covid-19-from-3-different-perspectives-3fbaabcd0348>
- [21] Ponsriharini, Deploy a Machine Learning model using streamlit.
<https://medium.com/featurepreneur/deploy-a-machine-learning-model-using-streamlit-a7ef163c19b0>
- [22] Sharad Mittal, Creating a Web app of a ML model using Streamlit.
<https://sharadmittal.hashnode.dev/creating-a-web-app-of-a-ml-model-using-streamlit>.
- [23] Deliver Your Customer Behavior Analysis Report with Streamlit.
<https://medium.com/octopus-id-data/deliver-your-customer-behavior-analysis-report-with-streamlit-36f052d0a43c>