HAW HAMBURG

BACHELOR THESIS
Suraj Shrestha

Evaluation von Visualisierungstypen und Bibliotheken für Energiedaten in Python

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Suraj Shrestha

Evaluation von Visualisierungstypen und Bibliotheken für Energiedaten in Python

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Regenerative Energiesysteme und Energiemanagement*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kolja Eger
Zweitgutachter: Prof. Dr. Wolfgang Renz

Eingereicht am: 26.04.2024

Suraj Shrestha

Thema der Arbeit

Evaluation von Visualisierungstypen und Bibliotheken für Energiedaten in Python

Stichworte

Energiedaten, Visualisierungstypen, Python-Bibliotheken, Evaluation

Kurzzusammenfassung

Die Grundidee der Datenvisualisierung ist die geeignete Darstellung von Daten in visueller Form, die dem Menschen einen Einblick in die Struktur der Daten, das Ziehen von Schlussfolgerungen aus den Daten und die direkte Interaktion mit den Daten ermöglicht [28]. Visuelle Datenexplorations-Techniken werden deshalb in vielen Anwendungsbereichen wie Industrie, Wirtschaft, Energiebranche, Medizin usw. eingesetzt.
Das Ziel dieser Arbeit besteht darin, unterschiedliche Visualisierungstypen für Energiedaten zu identifizieren und diese unter Verwendung gängiger Python-Bibliotheken umzusetzen. Zu den Visualisierungen gehören ein Liniendiagramm zur Darstellung der Entwicklung der jährlichen Stromverbrauchsdaten, ein Balkendiagramm zur Darstellung des monatlichen Stromverbrauchs der Gesamtnetzlast und der Residuallast sowie Histogramme, Boxplots und Violinplots zur Visualisierung der Datensatzverteilung. Mit einem Kreisdiagramm wird die monatliche Zusammensetzung dargestellt. Zur Analyse des Einflusses von Tageszeit, Wochentagen und Feiertagen auf den Stromverbrauch werden unterschiedliche Versionen der Heatmap vorgestellt. Ein Sankey-Diagramm wird benutzt, um die Energieflüsse fester Brennstoffe darzustellen. Darüber hinaus sind Karten erstellt, um Elektroladesäulen auf Bundes- und Landkreisebene darzustellen. Dashboards sind so konzipiert, dass sie eine interaktive Analyse der Daten für unterschiedliche Jahre mit unterschiedlichen Visualisierungstypen ermöglichen. Matplotlib und Seaborn werden zur statischen Visualisierung eingesetzt. Für die interaktive Visualisierung von Daten werden Vega, Bokeh und Plotly verwendet, während Folium, Geopandas und Plotly zur Erstellung von Karten genutzt werden. Dash und Streamlit werden genutzt, um Dashboards zu erstellen. Alle verwendeten Python Datenvisualisierungsbibliotheken sind leistungsfähig und erfüllen die Anforderungen, um Energiedaten zu visualisieren. Jede Bibliothek hat ihre eigenen Stärken und Schwächen. Die Visualisierungstypen und Bibliotheken, die für die Energiedaten verwendet werden sollen, können je nach den Bedürfnissen gut ausgewählt werden.

Suraj Shrestha

Title of Thesis

Evaluation of visualisation types and libraries for energy data in Python

Keywords

Energy data, Visualisation types, Python libraries, Evaluation

Abstract

The basic idea of data visualisation is to present data in an appropriate visual form, which allows people to gain insight into the structure of the data, to draw conclusions from the data and to interact directly with the data [28]. Visual data exploration techniques are therefore being used in a wide range of application areas such as industry, business, the energy sector, medicine, etc.

The aim of this work is to identify different types of visualisations for energy data and to implement them using common Python libraries. The visualisations include a line chart to show the development of the annual electricity consumption data, a bar chart to show the monthly electricity consumption of the total grid load and the residual load as well as histograms, box plots and violin plots to visualise the distribution of dataset. The monthly composition is visualised with a pie chart. Different versions of the heat map are presented to analyse the influence of time of day, weekdays and public holidays on electricity consumption. A Sankey diagram is used to visualise the energy flows of solid fuels. In addition, maps are created to show electric charging points at national and county level. Dashboards are designed to allow interactive analysis of data for different years with different visualisation types. Matplotlib and Seaborn are used for static visualisation. Vega, Bokeh and Plotly are used for the interactive visualisation of data, while Folium, Geopandas and Plotly are used to create maps. Dash and Streamlit are used to create dashboards. All Python data visualisation libraries used are powerful and meet the requirements for visualising energy data. Each library has its own strengths and weaknesses. The visualisation types and libraries for energy data can be selected according to the needs.

Inhaltsverzeichnis

Abbildungsverzeichnis **ix**

Tabellenverzeichnis **xii**

Abkürzungen **xiii**

1 Einleitung **1**
- 1.1 Hintergrund und Motivation . . . 2
- 1.2 Aufgabestellung, Forschungsziele und Forschungsfragen . . . 3
- 1.3 Gliederung der Arbeit . . . 4

2 Theoretische Grundlagen **6**
- 2.1 Visualisierungstypen . . . 6
- 2.2 Python Bibliotheken . . . 7
- 2.3 Datensatz . . . 8
- 2.4 Datenanalyse . . . 9
- 2.5 EDA . . . 10
- 2.6 Zeitreihenanalyse . . . 10

3 Anforderungen **11**
- 3.1 Analyse Visualisierungstypen für Energiedaten . . . 11
- 3.2 Anforderungen für die Visualisierung . . . 15
- 3.3 Analyse Pythonbibliotheken für Energiedaten . . . 15
- 3.4 Anforderungen für die Bibliotheken . . . 16
  - 3.4.1 Anforderungen für die statische Visualisierungs Bibliotheken (Matplotlib und Seaborn) . . . 16
  - 3.4.2 Anforderungen für die interaktive Visualisierungs Bibliotheken (Plotly, Bokeh, Vega-Altair) . . . 17
  - 3.4.3 Anforderungen für die Karten Visualisierungs Bibliotheken (Plotly, Folium, Geo-pandas) . . . 18

4 Entwurf **19**
4.1 Visualisierungstypen . . . 19
4.2 Python . . . 20
4.3 Jupyter Notebook . . . 21

5 Implementierung **23**
5.0.1 EDA . . . 23
5.1 Daten Visualisierung mit Matplotlib . . . 29
5.1.1 Liniendiagramm . . . 29
5.1.2 Balkendiagramm . . . 30
5.1.3 Boxplot . . . 31
5.1.4 Kreisdiagramm . . . 32
5.1.5 Flächendiagramm . . . 32
5.1.6 Histogramm . . . 33
5.1.7 Violinplot . . . 33
5.1.8 Wärmekarte . . . 34
5.2 Daten Visualisierung mit Seaborn . . . 38
5.2.1 Liniendiagramm . . . 38
5.2.2 Balkendiagramm . . . 38
5.2.3 Boxplot . . . 39
5.2.4 Kreisdiagramm . . . 39
5.2.5 Flächendiagramm . . . 40
5.2.6 Histogramm . . . 40
5.2.7 Violinplot . . . 41
5.2.8 Wärmekarte . . . 41
5.3 Daten Visualisierung mit Vega-Altair . . . 44
5.3.1 Liniendiagramm . . . 44
5.3.2 Balkendiagramm . . . 44
5.3.3 Boxplot . . . 45
5.3.4 Kreisdiagramm . . . 46
5.3.5 Flächendiagramm . . . 46
5.3.6 Histogramm . . . 47
5.3.7 Violinplot . . . 47
5.3.8 Wärmekarte . . . 48
5.4 Daten Visualisierung mit Bokeh . . . 51
5.4.1 Liniendiagramm . . . 51

5.4.2 Balkendiagramm . . . 51
5.4.3 Boxplot . . . 52
5.4.4 Kreisdiagramm . . . 52
5.4.5 Flächendiagramm . . . 53
5.4.6 Histogramm . . . 53
5.4.7 Violinplot . . . 54
5.4.8 Wärmekarte . . . 54
5.5 Daten Visualisierung mit Plotly . . . 56
5.5.1 Liniendiagramm . . . 56
5.5.2 Balkendiagramm . . . 57
5.5.3 Boxplot . . . 57
5.5.4 Kreisdiagramm . . . 58
5.5.5 Flächendiagramm . . . 58
5.5.6 Histogramm . . . 59
5.5.7 Violinplot . . . 60
5.5.8 Wärmekarte . . . 60
5.6 Sankey Diagramm . . . 63
5.7 Dashboard Erstellung . . . 64
5.7.1 Dashboard mit Dash . . . 65
5.7.2 Dashboard mit Streamlit . . . 67
5.8 Karten Visualisierung . . . 68
5.8.1 Karten mit Folium . . . 68
5.8.2 Karten mit Geopandas . . . 70
5.8.3 Karten mit Plotly . . . 72
5.8.4 Zeitreihenanalyse . . . 74

6 Evaluierung **82**
6.1 Statische und interaktive Visualisierung . . . 83
6.1.1 Funktionalität . . . 83
6.1.2 Dokumentation . . . 84
6.1.3 Leistung . . . 85
6.2 Dashboard Erstellung . . . 86
6.2.1 Funktionalität . . . 86
6.2.2 Benutzerfreundlichkeit . . . 86
6.2.3 Interaktivität . . . 87
6.2.4 Komplexität . . . 87

6.3 Karten Visualisierung 87
6.3.1 Funktionalität 88
6.3.2 Interaktivität 88
6.3.3 Dokumentation 88

7 Zusammenfassung **90**
7.1 Überprüfung von Forschungszielen und -fragen 90
7.2 Grenzen der Arbeit 94
7.3 Zukünftige Arbeiten 94

Literaturverzeichnis **95**

A Anhang **100**
Selbstständigkeitserklärung 101

Abbildungsverzeichnis

2.1 Zweck der Visualisierung 7
2.2 Python Visualisierungslandschaft 8

4.1 Entwurf Visualisierungstypen 19
4.2 Entwurf Pythonbibliotheken 21

5.1 Datenanalyse zur Gewinnung erster Erkenntnisse 25
5.2 Deskriptive statistische Analyse 26
5.3 Boxplot zur Erkennung von Ausreißern 27
5.4 Histogramm zur Bestimmung der Form der Verteilung 28
5.5 Punktendiagramm zur Ermittlung von Muster und Trend im Datensatz . 28
5.6 Liniendiagramm mit matplotlib 29
5.7 Balkendiagramm mit matplotlib 30
5.8 Boxplot mit matplotlib 31
5.9 Kreisdiagramm mit matplotlib 32
5.10 Flächendiagramm mit matplotlib 32
5.11 Histogramm mit matplotlib 33
5.12 Violinplot mit matplotlib 33
5.13 Heatmap Betrachtung Variante 1 mit matplotlib 34
5.14 Heatmap Betrachtung Variante 2 mit matplotlib 35
5.15 Heatmap Betrachtung Variante 3 mit matplotlib 36
5.16 Heatmap Betrachtung Variante 4 mit matplotlib 37
5.17 Liniendiagramm mit Seaborn 38
5.18 Balkendiagramm mit Seaborn 38
5.19 Boxplot mit Seaborn 39
5.20 Kreisdiagramm mit Seaborn 39
5.21 Flächendiagramm mit Seaborn 40
5.22 Histogramm mit Seaborn 40
5.23 Violinplot mit Seaborn 41

5.24 Heatmap Betrachtung Variante 1 mit Seaborn 41
5.25 Heatmap Betrachtung Variante 2 mit Seaborn 42
5.26 Heatmap Betrachtung variante 3 mit Seaborn 42
5.27 Heatmap Betrachtung Variante 4 mit seaborn 43
5.28 Liniendiagramm mit Vega-Altair 44
5.29 Balkendiagramm mit Vega-Altair 44
5.30 Boxplot mit Vega-Altair 45
5.31 Kreisdiagramm mit Vega-Altair 46
5.32 Flächendiagramm mit Vega-Altair 46
5.33 Histogramm mit Vega-Altair 47
5.34 Violinplot mit Vega-Altair 47
5.35 Heatmap Betrachtung Variante 1 mit Vega-Altair 48
5.36 Heatmap Betrachtung Variante 2 mit Vega-Altair 48
5.37 Heatmap Betrachtung Variante 3 mit Vega-Altair 49
5.38 Heatmap Betrachtung Variante 4 mit Vega-Altair 49
5.39 Liniendiagramm mit Bokeh 51
5.40 Balkendiagramm mit Bokeh 51
5.41 Kreisdiagramm mit Bokeh 52
5.42 Flächendiagramm mit Bokeh 53
5.43 Histogramm mit Bokeh 53
5.44 Heatmap Betrachtung Variante 1 mit Bokeh 54
5.45 Heatmap Betrachtung Variante 2 mit Bokeh 54
5.46 Heatmap Betrachtung Variante 3 mit Bokeh 55
5.47 Liniendiagramm mit Plotly 56
5.48 Balkendiagramm mit Plotly 57
5.49 Boxplot mit Plotly 57
5.50 Kreisdiagramm mit Plotly 58
5.51 Flächendiagramm mit Plotly 58
5.52 Histogramm mit Plotly 59
5.53 Violinplot mit Plotly 60
5.54 Heatmap Betrachtung Variante 1 mit Plotly 61
5.55 Heatmap Betrachtung Variante 2 mit Plotly 61
5.56 Heatmap Betrachtung Variante 3 mit Plotly 62
5.57 Heatmap Betrachtung Variante 4 mit Plotly 62
5.58 Sankeydiagramm für feste Brennstoff 64
5.59 Dashboard mit Dash 65

5.60 Dashboard mit Streamlit 67
5.61 Kartenvisualisierung auf Bundeslandebene 69
5.62 Kartenvisualisierung auf Landkreisebene 69
5.63 Kartenvisualisierung auf Bundeslandebene mit Geopandas 71
5.64 Kartenvisualisierung auf Landkreisebene mit Geopandas 72
5.65 Choropleth Karten auf Bundeslandebene mit Plotly 73
5.66 Choropleth Karten auf Landkreisebene mit Plotly 74
5.67 Visualisierung aller Spalten des Datensatzes 75
5.68 Resampling 76
5.69 Rolling 77
5.70 Zeitreihenzerlegung 78
5.71 Trend und Saisonalität 79
5.72 Autocorrelation 80

6.1 Codekomplexität und Ausführungszeit 85

7.1 Zusammenfassung RO1 91
7.2 Python Bibliotheken für die Datenvisualisierung 92

Tabellenverzeichnis

3.1 Überblick über Plattformen und Visualisierungstypen 12
3.2 Überblick über die verschiedenen Visualisierungszwecke, die entsprechenden Datenanforderungen und die Visualisierungstypen 14
3.3 Anforderungen für die statische Visualisierungs Bibliotheken (Matplotlib und Seaborn) 16
3.4 Anforderungen für die interaktive Visualisierungs Bibliotheken (Plotly, Bokeh, Vega-Altair) 17
3.5 Anforderungen für die Karten Visualisierungs Bibliotheken (Plotly, Folium, Geo-pandas) 18

4.1 Die beliebtesten Python-Bibliotheken für Datenvisualisierung 20

5.1 Komponenten der Programmierungsumgebung und ihre Versionen 23

6.1 Vergleich von Visualisierungsbibliotheken 83
6.2 Überblick über Bibliotheksdokumentationsstruktur und mögliche Codebeispiele 84
6.3 Überblick über Codebeispiele und Community 88

7.1 Einzelbewertungen 93

Abkürzungen

CM Cloud Manufacturing

IoT Internet der Dinge (Internet of Things)

CPS Cyber-Physische Systeme (Cyber-Physical Systems)

BDA Big Data Analytics

EnEfG Energieeffizienzgesetz

EU Europäische Union

EDA Explorative Datenanalyse

SMARD Strommarktdaten

EE Erneuerbare Energien

ENTSO-E Verband Europäischer Übertragungsnetzbetreiber (European Network of Transmission System Operators for Electricity)

PNG Portable Network Graphic

HTML Hypertext Markup Language

JPG Hypertext Markup Language

SVG Scalable Vector Graphics

JSON JavaScript Object Notation

1 Einleitung

Grafische Darstellungen von Daten sind schnell und mächtig. Schnell, da Sie die effizienteste Methode zur Vermittlung komplexer Informationen darstellen. Und mächtig, da Bilder in den Köpfen hängen bleiben. Zahlenreihen werden vergessen, Bilder nicht. In der Historie war die Datenvisualisierung enorm aufwändig, von der Datenbeschaffung bis hin zur eigentlichen Produktion des Bildes – welches in früheren Jahren üblicherweise als Holzstich, als Kupferstich oder als Lithografie ausgeführt wurde. In der Datenvisualisierungscommunity wird Charles-Joseph Minard heute oft als wichtige Figur in der Entwicklung der Datenvisualisierung angesehen. Seine Darstellung aus dem Jahre 1869 zeigt geografische Positionen, die Truppenbewegungen, den Verlust an Soldaten und die Temperaturen im Laufe von Napoleons Russlandfeldzug – technisch ausgeführt als so genanntes Sankey-Diagramm [44]. Die Arbeit von John Snow trug dazu bei, eine Cholera-Epidemie in London im Jahr 1854 zu beenden. Er identifizierte einen verseuchten Brunnen und verglich die Sterblichkeitsraten in den von zwei Unternehmen versorgten Gebieten. Seine Punktekarte und seine statistischen Vergleiche gelten als Teil der Anfänge der modernen Epidemiologie, der Reform der sanitären Grundversorgung und des Verständnisses der Infektionswege der Cholera. Während der Choleraepidemie von 1854 betreute Florence Nightingale die Patienten des Middlesex Hospitals und bemühte sich, die sanitären Verhältnisse zu verbessern. Sie visualisiert die monatlichen Todesfälle durch Infektionskrankheiten bei Soldaten, die angeblichen Auswirkungen der Verbesserung der sanitären Verhältnisse als Mittel zur Bekämpfung von Krankheiten und die Sterblichkeitsrate vor dem Eintreffen der Gesundheitskommission [29][12].

Heute sind wir an einem ähnlichen Punkt: Die Nachfrage nach Datenvisualisierungen nimmt überall zu. Industrie 4.0 verändert die Fertigung durch fortschrittliche Technologien wie CM, IoT, CPS und BDA. Sie fördert intelligente Fabriken und datengesteuerte Prozesse, die eine sorgfältige Implementierung von Visualisierungstechniken und Software erfordern. Die Echtzeit-Visualisierung ermöglicht die grafische Darstellung komplexer Prozessgrößen zu einem Bruchteil der Kosten einer vollständigen Digitalisierung [4]. In

der Wirtschaft helfen Datenvisualisierung, Verkaufszahlen, Kundenverhalten und Markttrends zu analysieren, was zu fundierteren Entscheidungen führt. In der wissenschaftlichen Forschung wird die Datenvisualisierung eingesetzt, um experimentelle Ergebnisse zu analysieren, komplexe wissenschaftliche Konzepte zu visualisieren und die Ergebnisse anderen Forschern und der Öffentlichkeit zu vermitteln. Im Gesundheitsbereich wird die Datenvisualisierung zur Analyse von Patientendaten und in der medizinischen Forschung eingesetzt und bietet Einblicke in Krankheitsmuster und Behandlungsergebnisse. Im Energiesektor bietet Datenvisualisierung einen detaillierten Einblick in den bisherigen Verbrauch und die Nachfrage und ermöglicht es Energieversorgern, fundierte Prognosen auf der Grundlage der tatsächlichen Datentrends zu erstellen [6].

1.1 Hintergrund und Motivation

Das Energieeffizienzgesetz (EnEfG) setzt klare Ziele zur Senkung des Energieverbrauchs bis 2030, im Einklang mit EU-Richtlinien. Firmen mit einem Jahresverbrauch über 7,5 GWh müssen jetzt Energie- oder Umweltmanagementsysteme einführen. Unternehmen mit einem Jahresenergieverbrauch von mehr als 2,5 Gigawattstunden müssen konkrete Pläne zu wirtschaftlichen Energieeffizienzmaßnahmen veröffentlichen. Rechenzentren sind zur Einhaltung von Energieeffizienzstandards und zur Nutzung von Abwärme verpflichtet [13]. Die Energiewende erfordert eine höhere Energieeffizienz, um den Verbrauch zu senken und die Klimaziele zu erreichen. Energieeffizienz spart Geld für Haushalte, Unternehmen und Kommunen und macht die Wirtschaft wettbewerbsfähiger. Datenvisualisierung kann dazu beitragen, die Energieeffizienz zu verbessern, indem Daten aus verschiedenen Quellen gesammelt und analysiert werden, um aktuelle Energieverbrauchsmuster zu verstehen, Einsparpotenziale zu identifizieren und realistische Ziele zu setzen. Studien zeigen, dass der Energieverbrauch durch Echtzeitinformationen deutlich gesenkt werden kann [31]. Technologische Fortschritte in der Datenvisualisierung bieten echte Möglichkeiten für die Erforschung des Energieverbrauchsbewusstseins mit Techniken wie Energiemonitor, um eine browserbasierte Applikation für die Visualisierung, Auswertung und Optimierung von Energieverbräuchen. Durch dieses Tool wird es möglich, Energieerzeugung und -verbrauch nahezu in Echtzeit zu beobachten. Grafisch aufbereitet werden die Energiedaten in einem digitalen Dashboard visualisiert und alle 15 Minuten aktualisiert. Außerdem können bestimmte Kennzahlen in verschiedenen Zeiträumen abgebildet werden, um den Verlauf des Tages, des Monats oder des Jahres zu sehen [14].

1.2 Aufgabestellung, Forschungsziele und Forschungsfragen

Ziel dieser Arbeit ist die Verkürzung der Lernkurve für zukünftige Forscherinnen und Forscher im Projekt. Dies beinhaltet die Identifikation geeigneter Visualisierungstypen für Energiedaten, die Auswahl und Evaluierung geeigneter Bibliotheken für die Visualisierung von Energiedaten in Python und die Erläuterung der erstellten Visualisierungstypen durch zahlreiche Bibliotheken anhand verschiedener Jupyter-Notebooks. Diese Notebooks enthalten Python-Code zur Erstellung von Plots sowie weitere Erläuterungen zur Nutzung und Interpretation der erstellten Visualisierungen. Um die Aufgabenstellung zu erfüllen, sind mehrere Aspekte zu berücksichtigen. Diese werden in Forschungszielen (Research Objectives, RO) zusammengefasst und in einzelnen Forschungsfragen (Research Questions, RQ) bearbeitet.

Das erste Forschungsziel behandelt die Identifikation geeigneter Visualisierungstypen für Energiedaten.

RO1: Wie können Energiedaten am besten visualisiert werden und welche Visualisierungstypen sind dafür geeignet?

Um das Forschungsziel zu erreichen, müssen die folgenden Forschungsfragen beantwortet werden:

RQ 1.1: Welche Online-Plattformen bieten Tools und Dienste zur Visualisierung von Energiedaten an?

RQ 1.2: Welche Visualisierungstypen bieten die aufgeführten Plattformen zur Darstellung von Energiedaten an?

RQ 1.3: Wann eignen sich welche Visualisierungstypen am besten für die Energiedaten?

Nach der Erarbeitung des ersten Forschungszieles werden die gängigen Pythonbibliotheken untersucht.

RO2: Welche Python-Bibliotheken sind gängig für die Datenvisualisierung?

RQ 2.1: Welche Python-Bibliotheken werden am häufigsten für die Datenvisualisierung verwendet?

RQ 2.2: Können die in RQ 1.3 genannten Visualisierungstypen mit diesen Bibliotheken realisiert werden?

RQ 2.3: Welche Anforderungen müssen Bibliotheken erfüllen, um Energiedaten zu visualisieren?

Nach der Erarbeitung des zweiten Forschungsziels werden die in RQ1 aufgelisteten Visualisierungstypen anhand der in RQ2 erforschten Python-Bibliotheken untersucht und implementiert. Für jede dieser Bibliotheken wird ein eigenes Notebook erstellt. Im nächsten RO geht es darum, verschiedene Bibliotheken anhand verschiedener Kriterien zu bewerten.

RO3: Wie können Python-Bibliotheken evaluiert werden?

RQ 3.1: Was sind die verschiedenen Bewertungskriterien, um die Bibliotheken zu vergleichen und zu bewerten?

RQ 3.2: Welche Vor- und Nachteile haben die einzelnen Bibliotheken?

RQ3.3: Für welche Anwendungsfälle sind welche Bibliotheken am besten geeignet?

Durch die Beantwortung der Forschungsziele und -Fragen entsteht am Ende ein handvolles Notebook für die Visualisierung der Energiedaten sowie Evaluierungen und Empfehlungen dazu.

1.3 Gliederung der Arbeit

- **Kapitel 1** beinhaltet neben der Einführung zum Thema auch die Aufgabestellung, Forschungsziele und Forschungsfragen.
- **Kapitel** 2 gibt eine kurze Einführung in die theoretischen Grundlagen.
- **Kapitel 3** analysiert die verschiedenen Visualisierungstypen und listet die Anforderungen an Bibliotheken auf.
- **Kapitel 4** beschreibt den Entwurf für die Umsetzung.
- **Kapitel 5** implementiert alle in unseren Anforderungen beschriebenen Visualisierungstypen mit Hilfe verschiedener gängiger Python-Bibliotheken.

- **Kapitel 6** bewertet die Bibliotheken anhand verschiedener Kriterien.
- **Kapitel 7** fasst die Arbeit zusammen und überprüft die Beantwortung der Forschungsfragen.

2 Theoretische Grundlagen

In diesem Kapitel werden allgemeine Informationen über die Visualisierungstypen, Pythonbibliotheken, Datenanalyse, EDA und Zeitreihenanalyse vermittelt.

2.1 Visualisierungstypen

Die Datenvisualisierung basiert auf Variablen, die univariate, bivariate oder multivariate Daten darstellen. Sie können quantitativ oder qualitativ sein und werden durch Zahlen oder Text dargestellt. Datenvisualisierungen werden häufig verwendet, um quantitative Datenmengen darzustellen. Bei quantitativen Variablen unterscheidet man zusätzlich noch in diskrete und stetige Variablen [33]. Entscheidungsbäume werden verwendet, um den geeigneten Visualisierungstyp auszuwählen, wobei Faktoren wie der Zweck der Visualisierung, der Datentyp und die Anzahl der Variablen berücksichtigt werden. Der Zweck der Visualisierung gibt eine Antwort auf die Frage, welche Art von Information über die Daten vermittelt werden soll. Es gibt zahlreiche Online-Lösungen, die bei der Auswahl einer geeigneten Visualisierung helfen. [19][49][1] sind in dieser Arbeit berücksichtigt. Die Anwendungszwecke bei der Anzeige von Daten sowie der empfohlene Diagrammtyp für den jeweiligen Zweck sind in der Abbildung 2.1 zusammengefasst.

Abbildung 2.1: Zweck der Visualisierung

2.2 Python Bibliotheken

Es gibt eine Reihe verschiedener Datenvisualisierungsbibliotheken und -module, die mit Python kompatibel sind. Die meisten Python-Datenvisualisierungsbibliotheken lassen sich in eine der folgenden Gruppen einteilen: Matplotlib-basierte Bibliotheken, JavaScript-Bibliotheken, JSON-Bibliotheken und WebG-Bibliotheken.

Auf Matplotlib basiert die erste große Gruppe von Bibliotheken. Matplotlib existiert nun seit mehr als zwei Jahrzehnten und ist sozusagen das Hauptwerkzeug. Es gibt viele Dinge, die um Matplotlib herum gebaut wurden. Basemap/Cartopy wird für geographische Visualisierungen verwendet. Pandas und Seaborn haben Verbindungen zu Matplotlib, während ggpy eine ggplot-Schnittstelle auf Matplotlib aufbaut. Networkx bietet Netzwerkvisualisierung. Yellowbrick und Scikit-Plot bieten Visualisierungswerkzeuge für maschinelles Lernen.

In den letzten Jahren haben viele dieser Python-Bibliotheken begonnen, von JavaScript abzuhängen und JavaScript zu nutzen, um eine großartige interaktive Visualisierung zu erhalten. Die beiden Größten davon sind Plotly und Bokeh. Es gibt noch Toyplot und

BQplot, die interessant sind. Ipyleaflet, ipyvolume, pythreejs mit denen man verschiedene Aspekte von JavaScript für die interaktive Visualisierung im Notebook nutzen kann, was ziemlich cool ist. Es gibt noch andere Dinge, wie Cufflinks, das auf Plotly aufbaut ist.

Um Javascript in Matplotlib einzubinden, gibt es die Möglichkeit, d3.js zu verwenden und MPLd3 verbindet Matplotlib mit d3. Allerdings ist diese Methode nicht optimal unterstützt. Es gibt Vega und Vega-Lite für interaktive Grafiken. Vega-Spezifikationen definieren, wie interaktive Visualisierungen in JavaScript-Object-Notation (JSON) erstellt werden. Altair basiert auf den Standards Vega und Vega-Lite.

Die Open Graphics Library OpenGL ist eine API-Spezifikation für 2D- und 3D- Grafikanwendungen [18][30].

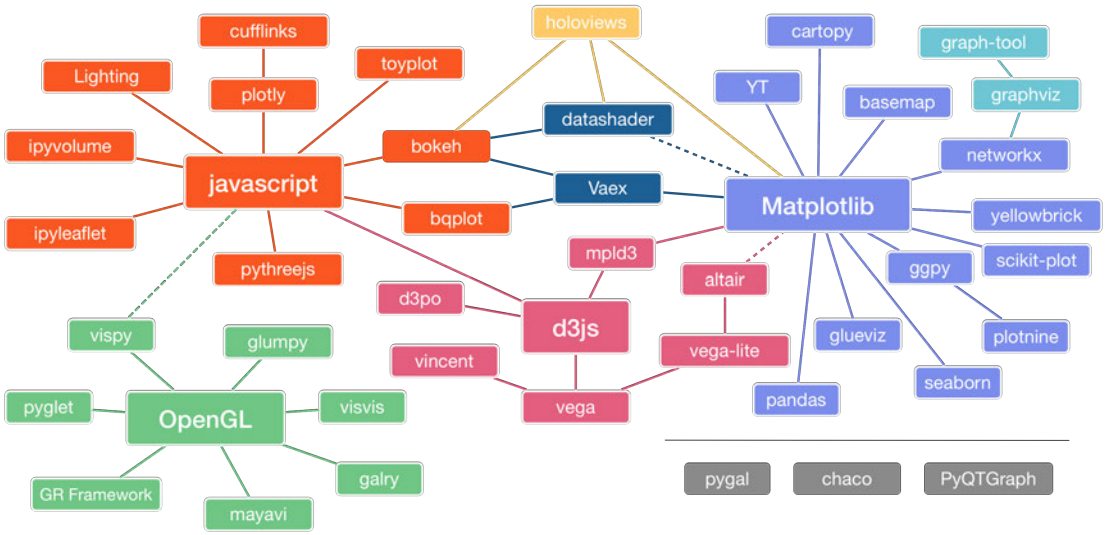

Abbildung 2.2: Python Visualisierungslandschaft

2.3 Datensatz

Im Rahmen des EcoCharge-Projekts werden die Daten von der Open-Source-Website (SMARD) heruntergeladen und für alle Projektteilnehmer auf GitHub zur Verfügung gestellt. SMARD ist eine Daten- und Informationsplattform mit dem Ziel, die Transparenz im deutschen Strommarkt durch die Bereitstellung von Echtzeitdaten zu Erzeugung,

Verbrauch, Import und Export von Strom zu erhöhen. In dieser Arbeit werden Stromverbrauchsdaten betrachtet. Der Datensatz für den realisierten Stromverbrauch ab dem Jahr 2015 enthält Datentypen wie die Gesamtnetzlast und die Residuallast für den realisierten Stromverbrauch [9]. Für das Sankey-Diagramm wird der Datensatz von Eurostat verwendet [15][16]. In dieser Arbeit wird nur der Fluss der festen Brennstoffe betrachtet. Die Kartenvisualisierung basiert auf dem von der Bundesnetzagentur veröffentlichten Datensatz, der Informationen zu Elektroladesäulen in Deutschland auf Bundesland- sowie Landkreisebene bereitstellt [8].

2.4 Datenanalyse

Von historischen Trends bis hin zu zukünftigen Erkenntnissen kann das Interesse an den Daten reichen. Für ein stabiles statistisches Umfeld sind verschiedene Analyseebenen erforderlich. Die verschiedenen Analyseebenen umfassen die deskriptive, prädiktive und diagnostische Analyse. Bei der deskriptiven Analyse werden Daten aggregiert, um vergangene Ereignisse zu analysieren, während bei der prädiktiven Analyse statistische Modelle und Prognosetechniken verwendet werden, um zukünftige Ergebnisse vorherzusagen. Die diagnostische Analyse liefert in Verbindung mit der deskriptiven Analyse eine detaillierte Erklärung eines Szenarios durch das Verständnis von Verhaltensmustern [41][22].

Der Fokus der Arbeit liegt auf der deskriptiven Analyse. Dabei werden Daten aggregiert, um vergangene Ereignisse zu analysieren und Fragen zum Geschehen zu beantworten. Zunächst werden Daten aus verschiedenen Quellen wie SMARD, Eurostat und der Bundesnetzagentur gesammelt. Nach der Datenerfassung beginnen die Bereinigung und Vorverarbeitung, d. h. die Umwandlung der Daten in eine einheitliche Struktur, die Standardisierung der Formate und die Behandlung fehlender oder falscher Werte. Ziel dieser Datenanalyse ist das Verständnis der Struktur und der Merkmale des Datensatzes durch den Einsatz explorativer Datenanalysemethoden wie z. B. Histogramme, Boxplot, Punktediagramm, Balkendiagramm und deskriptive (beschreibende) Statistik. Die Deskriptive Statistik beinhaltet somit alle Verfahren, mit denen sich durch die Beschreibung von Daten Informationen gewinnen lassen. Zu diesen Methoden bzw. Verfahren gehören unter anderem die Erstellung von Grafiken und Tabellen und die Berechnung von deskriptiven Kennzahlen bzw. Parametern. Mit Ihrer Hilfe können die zentrale Tendenz, die Streuung und die Verteilung eines Datensatzes ermittelt werden [11][42].

2.5 EDA

Die explorative Datenanalyse bezeichnet statistische Verfahren zur Aufdeckung von Datenstrukturen, Abhängigkeiten und Abweichungen einer vorhandenen Grundstruktur [vgl. Gabler Wirtschaftslexikon, 2015]. Sie umfasst häufig grafische Verfahrensweisen und dient dazu, Daten zunächst zu explorieren, d. h. sie zu erkunden, um darin durch Visualisierungsmethoden besser die enthaltenen Muster und Strukturen zu erkennen, Schlussfolgerungen zu ziehen sowie mit den Daten interagieren zu können. Der Begriff explorative Datenvisualisierung bezeichnet die interaktive Visualisierung der Ergebnisse aus der explorativen Datenanalyse (EDA) oder auch explorativen Statistik [34].

> Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone as the first step. John W. Turkey, 1977.

2.6 Zeitreihenanalyse

Die Zeitreihenanalyse ist eine statistische Methode zur Untersuchung von Datenpunkten über einen bestimmten Zeitraum. Die Zeitreihenanalyse verfolgt vier Ziele: Beschreibung, Erklärung, Prognose und Kontrolle. Der erste Schritt der Analyse besteht darin, die Daten grafisch darzustellen und die grundlegenden Merkmale der Reihe zu ermitteln. Dies kann so einfach sein wie die Suche nach Trends oder so komplex wie die Analyse saisonaler Veränderungen. Diagramme können verwendet werden, um nach Ausreißern zu suchen, die nicht mit dem Rest der Daten übereinstimmen. Durch die grafische Darstellung der Zeitreihe ist es möglich, einen Wendepunkt zu erzeugen, an dem ein Aufwärtstrend in einen starken Abwärtstrend übergeht. Darüber hinaus ermöglicht die Zeitreihenanalyse die Erklärung von Ereignissen in der Vergangenheit. Basierend auf der beobachteten Zeitreihe können zukünftige Werte vorhersagen. Bei den Kontrolltechniken werden die Beobachtungen in eine Kontrollkarte eingetragen und die Reihe wird mit Hilfe eines stochastischen Modells angepasst. Eine Zeitreihe besteht aus vier Komponenten: einem Trend, der auf regelmäßige Muster in den Daten hinweist, saisonalen Schwankungen, d. h. kurzfristigen Veränderungen in einer Zeitreihe, die typischerweise innerhalb eines Jahres auftreten. Zu diesem Zweck werden häufig stündliche, tägliche, wöchentliche, vierteljährliche und monatliche Daten für die Analyse aggregiert. Außerdem gibt es zyklische Schwankungen, die über einen Zeitraum von mehr als einem Jahr auftreten, und zufällige Schwankungen, die unregelmäßig und zufällig sind [27].

3 Anforderungen

Die zentrale Aufgabe der Arbeit besteht darin, verschiedene Visualisierungstypen für Energiedaten zu bestimmen und diese mit gängigen Python-Bibliotheken zu implementieren. Ziel dieses Kapitels ist die Bearbeitung der Forschungsziele 1 und 2.

3.1 Analyse Visualisierungstypen für Energiedaten

Es werden verschiedene Onlineplattformen untersucht, die Werkzeuge und Dienste zur Visualisierung von Energiedaten anbieten. Es wird analysiert, welche Visualisierungstypen die aufgeführten Plattformen zur Darstellung von Energiedaten verwenden. Tabelle 3.1 gibt einen Überblick über Plattformen und Visualisierungstypen.

Plattform zur Visualisierung der Energiedaten	**Verwendete Visualisierungs-typen**	**Was für Energiedaten?**
SMARD.de [9]	Flächendiagramm, Liniendiagramm	Realisierter Erzeugung, Prognostizierter Erzeugung Day-Ahead, Prognostizierter Erzeugung Intraday, Realisierter Stromverbrauch, Prognostizierter Stromverbrauch, Stromhandel (Importe und Exporte)
	Balkendiagramm (gestapelt)	Installierter Erzeugungsleistung durch unterschiedliche Energieträger
	Karte (Maps)	Kraftwerke
Energy-Charts [20]	Flächendiagramm, Liniendiagramm	Öffentliche Nettostromerzeugung, Speicherfüllstände
	Wärmekarten	Öffentliche Nettostromerzeugung aus konventionellen und EE-Quellen.
	Punktendiagramme	Prognose vs. Real werte darzustellen
	Kreisdiagramme	Öffentliche Nettostromerzeugung aus konventionellen und EE-Quellen.
	Säulendiagramme	Monatlicher Anteil Erneuerbarer Energien an der öffentlichen Nettostromerzeugung
	Karte (Maps)	Anteil EE an der elektrischen Last, Stromhandel (Import & Export), Durchschnittliche Day-Ahead-Börsenstrompreise von EU
ENTSO-E [45]	gestapeltes Balkendiagramm	Tatsächliche Erzeugung nach Erzeugungsart
	Balkendiagramm, Kreisdiagramm	Installierte Leistung nach Erzeugungsart
	Flächendiagramm, Liniendiagramm	Prognostizierter Erzeugung Day-Ahead, Wasserspeicher und Wasserspeicherkraftwerke

Tabelle 3.1: Überblick über Plattformen und Visualisierungstypen

Weitere Online-Plattformen zur Visualisierung von Energiedaten sind die IEA (Internationale Energieagentur) [24], das Energy Visualisation Portal (Eurostat) [17] und der Optenda Energiemonitor [32]. Alle Plattformen verwenden ähnliche Visualisierungstypen für vergleichbare Energiedaten.

Abbildung 2.1 zeigt verschiedene Visualisierungstypen für unterschiedliche Zwecke. Als Nächstes werden die Anforderungen an die zu vermittelnden Informationen sowie die passenden Visualisierungstypen für SMARD-Datensatz analysiert und in einer Tabelle zusammengefasst.

Zweck der Visualisierung	**Datenanforderungen**	**Visualisierungstypen**
Anzeigen von Verteilungen	Darstellung der numerischen Daten des Datensatzes durch ihre Quartile mit Anzeige von Median, höherem/geringerem Quartil und Maximum/Minimum und Ausreißern.	Boxplot
	Darstellung der Verteilung des Datensatzes, wie oft Werte in Bereiche(bins) fallen.	Histogramm
	Darstellung Boxplot mit einer gedrehten Kernel-Dichte auf jeder Seite	Violinenplot
Anzeigen von Vergleichen	Anzeige der jährlichen Gesamtnetz- und Residuallast	Liniendiagramme
	Die Saisonalität, d.h. der Einfluss unterschiedlicher Zeiten wie Tageszeit, Wochenzeit, Feiertage auf den Energieverbrauch durch verschiedene Varianten der Wärmekarte verdeutlichen.	Wärmekarte
	Verlauf von jährlichen Gesamtnetz- und Residuallast anzeigen	Punktediagramme
	Darstellung des Gesamtnetz- und Residuallast nebeneinander und Gruppierung nach Monaten auf derselben Achse.	Balkendiagramm
Anzeigen von Beziehungen	Korrelation zwischen Gesamtnetzlast und residuallast	Punktediagramme
Anzeigen von Trend	Anzeige der jährlichen Gesamtnetzlast und wöchentlicher gleitender Mittelwert	Liniendiagramm
	Anzeige der jährlichen Gesamtnetz- und Residuallast	Flächendiagramm
Anzeigen von Zusammensetzungen	Anzeige der numerischen Proportionen des monatlichen Energieverbrauchs	Kreisdiagramm
Anzeigen von Geografie	Darstellung der Elektroladesäule in Deutschland	Kartendiagramm, Punktdichtekarte
Anzeigen von Energieflüsse	Darstellung Energieflüsse	Sankeydiagramm

Tabelle 3.2: Überblick über die verschiedenen Visualisierungszwecke, die entsprechenden Datenanforderungen und die Visualisierungstypen

3.2 Anforderungen für die Visualisierung

Im Rahmen dieser Arbeit sollen die Visualisierungstypen der Tabelle 3.2 mit Hilfe verschiedener Python-Bibliotheken implementiert werden. Dabei wird zwischen einfachen und komplexen Visualisierungen unterschieden. Zu den einfachen Visualisierungen gehören Liniendiagramme, Balkendiagramme, Kreisdiagramme, Flächendiagramme, Punktdiagramme, Histogramme, Boxplots und Violinplots. Zu den komplexen Visualisierungen gehören Wärmekarten, Sankeydiagramme und Karten. Darüber hinaus sollten interaktive Visualisierungstechnologien eingesetzt werden, um Trends schnell zu erkennen, Zusammenhänge in den Daten besser zu verstehen und komplexe Daten zu vereinfachen. Interaktive Datenvisualisierung verwendet Interaktionstools, um Parameter zu ändern, Details zu zeigen und neue Erkenntnisse zu gewinnen. Eine wichtige Aufgabe dieser Arbeit ist die Implementierung des Dashboards. Der Benutzer kann mit dem Dashboard interagieren und sich die Kennzahlen anzeigen lassen, die seinen Wünschen entsprechen.

3.3 Analyse Pythonbibliotheken für Energiedaten

Es gibt viele verschiedene Tools, von denen jedes auf seine eigene Anwendung spezialisiert ist oder seine eigenen Stärken hat. Ziel dieser Arbeit ist es, einen Überblick über die Landschaft der Data-Visualisierungstools in Python zu geben und zu ermitteln, welche Bibliotheken für die Energiedatenvisualisierung am besten geeignet sind. Abbildung 2.2 gibt einen groben Überblick über die Python-Visualisierungslandschaft. Zur Erfüllung der Visualisierungsanforderungen sind die auf Matplotlib basierenden Bibliotheken Matplotlib und Seaborn ausgewählt. Matplotlib ist eine Kernbibliothek für die Datenvisualisierung in Python und Seaborn ist eine auf Matplotlib aufbauende High-Level-Schnittstelle zum Zeichnen ansprechender statistischer Grafiken. Unter den auf Java-Script basierenden Bibliotheken sind Plotly und Bokeh ausgewählt. Plotly und Bokeh sind die Kernbibliotheken von Python, auf denen mehrere übergeordnete Bibliotheken aufbauen. Als JSON-basierte Bibliothek ist Vega-Altair ausgewählt, das eine kurze deklarative JSON-Syntax für die Erstellung der Visualisierungen bietet. Diese Arbeit behandelt die populärsten Python-Datenvisualisierungsbibliotheken, die in die oben definierten Kategorien fallen. Diese Bibliotheken können je nach Bedarf statische und interaktive Visualisierungen implementieren. Fast jede Python-Bibliothek kann verwendet werden, um statische PNG-, SVG-, HTML- oder andere Ausgaben zu erstellen, die in eine Präsentation eingefügt, per E-Mail versendet oder als Abbildung in einem Dokument veröffentlicht werden können.

Viele möchten Python-basierte Dashboards erstellen, mit denen Benutzer Daten untersuchen oder analysieren können. Python bietet mehrere Bibliotheken für diesen Zweck. Dash und Streamlit werden verwendet, um webbasierte Dashboards zu erstellen. Für die Kartenvisualisierung werden Plotly, Folium und Geopandas ausgewählt, um mit geographischen Koordinaten zu arbeiten. Die in dieser Arbeit behandelten Bibliotheken sind: Matplotlib, Seaborn für die statische Visualisierung, Plotly, Bokeh, Altair für die interaktive Visualisierung, Dash und Streamlit für die Erstellung von Dashboards und Folium, Geopandas und Plotly für die Kartenvisualisierung.

3.4 Anforderungen für die Bibliotheken

3.4.1 Anforderungen für die statische Visualisierungs Bibliotheken (Matplotlib und Seaborn)

"Muss"-Anforderungen	**"Soll"-Anforderungen**	**"Kann"-Anforderungen**
Visualisiert alle in Tabelle 3.2 aufgeführten Visualisierungstypen.	Integrierte Themen, Figurenästhetik und Farbpaletten	Interaktivität
Anpassung des Diagramms (X-Y-Achse und Ticks formatieren, Beschriftung, Legende, Titel hinzufügen)	Erweiterte Visualisierungstypen	Integration mit Dashboard Bibliothek wie Streamlit
Möglichkeit zum Export von Plots in einer Vielzahl von Formaten	Leistungseffizienz bei großen Datensätzen	Erweiterte Anpassungen von Text und Anmerkungen wie z.B. Logo und Fußnote für Quelle
Integration mit Pandas zur Datenmanipulation	Umfassende Dokumentation und aktive Support-Community	

Tabelle 3.3: Anforderungen für die statische Visualisierungs Bibliotheken (Matplotlib und Seaborn)

3.4.2 Anforderungen für die interaktive Visualisierungs Bibliotheken (Plotly, Bokeh, Vega-Altair)

"Muss"-Anforderungen	**"Soll"-Anforderungen**	**"Kann"-Anforderungen**
Alle Muss-Anforderungen von statischer Visualisierung erfüllen.	Alle Soll-Anforderungen von statischer Visualisierung erfüllen.	Erweiterte Diagrammtypen wie Karten und Sankey-Diagramme.
Interaktive Visualisierung (Hover infos, zoom, pan, reset, download)	Hinzufügen von benutzerdefinierten Steuerelementen wie Schaltflächen, Dropdown-Menüs, Schiebereglern und Selektoren.	globale Bereitstellung von interaktiven Plots.
Exportmöglichkeiten (statisches Bildformat (JPG,PNG,SVG) für Bericht und Veröffentlichung, interaktives Web-Format für Dashboard)	Benutzerfreundlichkeit (Intuitive API und übersichtliche Dokumentation)	

Tabelle 3.4: Anforderungen für die interaktive Visualisierungs Bibliotheken (Plotly, Bokeh, Vega-Altair)

3.4.3 Anforderungen für die Karten Visualisierungs Bibliotheken (Plotly, Folium, Geo-pandas)

“Muss”-Anforderungen	**“Soll”-Anforderungen**	**“Kann”-Anforderungen**
Choroplethkarte der Elektroladesäule auf Bundesland-und Landkreiseebene	Erweiterung mit interaktiven Funktionen wie HTML Popups, Search usw.	Karte auf PLZ Ebene
Hovern Funktion um Information über Bundesländer ggf. Landkreise und Anzahl der Ladesäulen zu erfahren	ClusterMarker und weiter Plugins zur Verbesserung der Sichtbarkeit und Benutzerfreundlichkeit	Layerliste auf Bundesland-, Landkreis- und PLZ-Ebene
Alle Standorte von Ladestationen markieren	Leistungseffizienz bei großen Datenpunkten	Filter für normale und schnelle Ladesäulen

Tabelle 3.5: Anforderungen für die Karten Visualisierungs Bibliotheken (Plotly, Folium, Geo-pandas)

4 Entwurf

In diesem Kapitel werden die analysierten Visualisierungstypen und Bibliotheken zusammengefasst.

4.1 Visualisierungstypen

In der Tabelle 3.2 wird ein Überblick über die verschiedenen Visualisierungszwecke gegeben. Es wird gezeigt, welche Informationen aus den Daten gewonnen werden sollen und welche Visualisierungstypen dafür geeignet sind. Abbildung 4.1 gliedert die ausgewählten Visualisierungstypen in drei Gruppen: einfache Visualisierungen, statistische Diagramme und komplexe Visualisierungen.

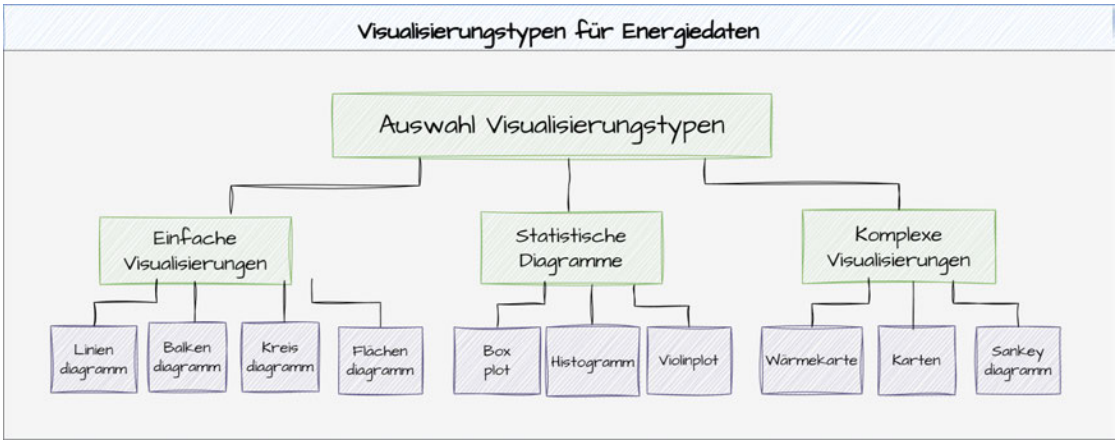

Abbildung 4.1: Entwurf Visualisierungstypen

4.2 Python

Für die Auswahl der Python-Bibliotheken werden die Anzahl der Github-Sterne, die Anzahl der Downloads und die Anzahl der Mitwirkenden als Kriterien herangezogen. Darüber hinaus werden auch einzelne Gruppen aus der Python-Visualisierungslandschaft berücksichtigt. Dazu gehören Matplotlib und Seaborn für Matplotlib-basierte Bibliotheken, Bokeh und Plotly für JavaScript-basierte Bibliotheken und Vega-Altair für JSON-basierte Bibliotheken. Zusätzlich werden bei der Auswahl der Bibliotheken Benutzeranforderungen berücksichtigt. Z. B. statische Visualisierung, interaktive Visualisierung, Erstellung von Dashboards und Visualisierung von Karten.

Bibliotheken	**Gemeinschaft**		
	Sterne	**Mitwirkenden**	**Downloads**
Matplotlib	19k	417	59M/monat
Seaborn	12k	185	18M/monat
Vega-Altair	8.9k	144	22M/monat
Bokeh	19k	391	4.9M/monat
Plotly	15k	231	14M/monat
Dash	20k	124	2.8M/monat
Streamlit	31k	208	4M/monat
Folium	6.6k	153	978k/monat
Geopandas	4.1k	209	6.4/monat

Tabelle 4.1: Die beliebtesten Python-Bibliotheken für Datenvisualisierung

Die Tabelle 4.1 zeigt die gängigen Python-Bibliotheken, die den Auswahlkriterien für Bibliotheken entsprechen.

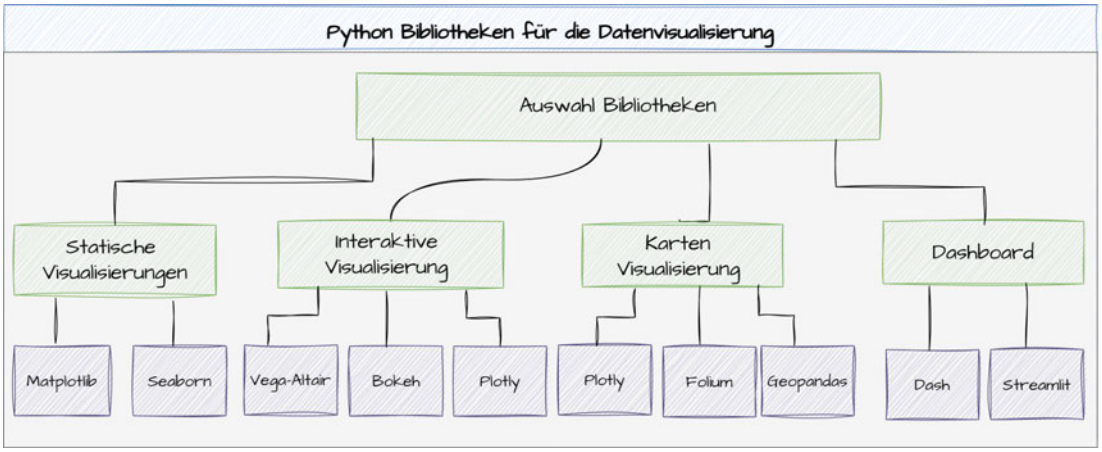

Abbildung 4.2: Entwurf Pythonbibliotheken

Die Abbildungen 4.1 und 4.2 geben einen Überblick über die Struktur der vorliegenden Arbeit.

4.3 Jupyter Notebook

Die Visualisierungstypen werden von verschiedenen Pythonbibliotheken implementiert und von jeder Bibliothek wird ein Jupyternotebook erstellt. Im Rahmen dieser Arbeit werden den künftigen Forscherinnen und Forschern die nachfolgend aufgeführten Notizbücher in GitHub des EcoCharge Projekts zur Verfügung gestellt.

- Datenvisualisierung mit Matplotlib-Bibliothek
- Datenvisualisierung mit Seaborn-Bibliothek
- Datenvisualisierung mit Vega-Altair-Bibliothek
- Datenvisualisierung mit Bokeh-Bibliothek
- Datenvisualisierung mit Plotly-Bibliothek
- Sankey-Diagramm mit Plotly
- Kartenvisualisierung mit Plotly
- Kartenvisualisierung mit Folium

- Kartenvisualisierung mit Geopandas
- EDA und Zeitreihenanalyse
- Dashboard mit Streamlit

5 Implementierung

In Abschnitt 4 wird ein Entwurf zur Implementierung im Rahmen dieser Arbeit vorgestellt. Die folgende Tabelle 5.1 bietet einen Überblick über die verwendete Komponente der Programmierumgebung. Darüber hinaus werden Informationen über die erste, aktuelle und für die vorliegende Arbeit verwendete Version bereitgestellt. Alle Visualisierungstypen werden anhand ausgewählter Bibliotheken implementiert. Die implementierten interaktiven Funktionen lassen sich anhand der gespeicherten HTML-Dateien in GitHub nachvollziehen.

Komponente	**Erstes Veröffentlichungsjahr**	**Erste Version**	**Aktuelle Version**	**Verwendete Version**
Python	1991	0.9.0	3.12.3	3.11.5
Visual Studio Code	2015	1	1.88	1.85.2
Jupyter Notebook	2015	4	7.1.3	7.0.8
Matplotlib	2003	0.1	3.8.3	3.8.2
Seaborn	2012	0.1	0.13.2	0.12.2
Vega-Altair	2016	1.0.0	5.3.0	5.2.0
Bokeh	2013	0.1.0	3.4.1	2.4.3
Plotly	2013	0.6	5.21.0	5.18.0
Folium	2014	0.1.3	0.16.0	0.15.1
Geopandas	2014	0.1.0	1.0.0	0.14.3
Streamlit	2019	0.1.0	1.33.0	1.31.0

Tabelle 5.1: Komponenten der Programmierungsumgebung und ihre Versionen

5.0.1 EDA

Die explorative Datenanalyse ist ein wichtiger Schritt in jedem Datenanalyseprojekt. Sie hilft, die Daten besser zu verstehen und Muster, Beziehungen, Trends und Anomalien zu erkennen. Diese Arbeit gibt einen kurzen Überblick über die Schritte der EDA mit Python für den Datensatz 2022.

Schritte 1. Bibliotheken und Datensätze importieren

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Read CSV file into a DataFrame
def read_SMARD_data(path):
   df = pd.read_csv(path, delimiter=';', thousands='.', decimal=',',
       parse_dates=[[0,1]], dayfirst="True")

   # Spalten umbenennen
   df = df.rename(
      columns={
      'Datum_Anfang': 'Datum',
      'Gesamt (Netzlast) [MWh] Originalauflsungen': 'Gesamt_Netzlast',
      'Residuallast [MWh] Originalauflsungen': 'Residuallast',
      'Pumpspeicher [MWh] Originalauflsungen' : 'StrombezugPumpspeicher'
      }
   )
   df.pop('Ende')

   return df
#Load the data into a pandas dataframe
SMARD_df = read_SMARD_data("../../data/SMARD Data/Real data/
    Realisierter_Stromverbrauch_202201010000_202212312359_Viertelstunde.csv")
```

Listing 5.1: Schritte 1: Bibliotheken und Datensätze importieren

Schritte 2. Datenanalyse zur Gewinnung erster Erkenntnisse

```
#check the shape of dataset
print("Data shape :", SMARD_df.shape)

#view the first few rows of the dataset
print("Data's first few rows :\n", SMARD_df.head())

#check the data types of each feature
print("Data type of the attributes :\n", SMARD_df.dtypes)

print("Checking for missing values :\n",
SMARD_df.isnull().sum())
```

Listing 5.2: Schritte 2: Datenanalyse zur Gewinnung erster Erkenntnisse

```
Data shape : (35040, 4)

 Data's first few rows :
                 Datum  Gesamt_Netzlast  Residuallast  StrombezugPumpspeicher
0 2022-01-01 00:00:00         11161.50       3194.75                  599.75
1 2022-01-01 00:15:00         11058.75       3136.00                  595.00
2 2022-01-01 00:30:00         10931.75       3129.00                  649.50
3 2022-01-01 00:45:00         10763.50       2992.50                  695.00
4 2022-01-01 01:00:00         10549.00       2914.50                  586.75

 Data type of the attributes :
 Datum                     datetime64[ns]
Gesamt_Netzlast                  float64
Residuallast                     float64
StrombezugPumpspeicher           float64
dtype: object

Checking for missing values :
 Datum                     0
Gesamt_Netzlast            0
Residuallast               0
StrombezugPumpspeicher     0
dtype: int64
```

Abbildung 5.1: Datenanalyse zur Gewinnung erster Erkenntnisse

Der Datensatz enthält 35.040 individuelle Datensätze und vier Variablen. Die Einträge sind in 15-Minuten-Intervallen aufgeführt. Die Variablen sind Datum vom Typ datetime64[ns], das für die Verarbeitung von Datum-Zeit-Daten verwendet wird, und die Gesamtnetzlast, die Residuallast und der StrombezugPumpspeicher sind alle quantitativ kontinuierliche Variablen vom Typ float64. Die Anzahl der Nullen in jeder Spalte des Datensatzes zeigt, dass keine Werte im Datensatz fehlen.

Schritte 3. Deskriptive statistische Analyse

Nach einem ersten Überblick über die Daten wird eine deskriptive statistische Analyse durchgeführt, bei der deskriptive Statistiken wie Mittelwert, Median, Minimum und Standardabweichung mit der Funktion ‚describe()' erstellt werden.

```
1 SMARD_df.describe()
```

Listing 5.3: Schritte 3: Deskriptive statistische Analyse

	Datum	Gesamt_Netzlast	Residuallast	StrombezugPumpspeicher
count	35040	35040.000000	35040.000000	35040.000000
mean	2022-07-02 12:28:10.273972736	13773.905051	8619.509703	409.541924
min	2022-01-01 00:00:00	7963.750000	628.500000	0.000000
25%	2022-04-02 06:56:15	11801.750000	6516.750000	27.750000
50%	2022-07-02 12:52:30	13777.750000	8679.875000	185.000000
75%	2022-10-01 18:48:45	15636.500000	10709.250000	658.000000
max	2022-12-31 23:45:00	19707.000000	17855.750000	2188.250000
std	NaN	2400.054835	3098.664417	490.204631

Abbildung 5.2: Deskriptive statistische Analyse

Die Gesamtnetzlast variiert zwischen 7963,75 MWh/15min (Grundlast) und 19707 MWh/15min (Spitzenlast). Der Mittelwert beträgt 13773,90 MWh/15min, während der Median mit 13777,75 MWh/15min etwas höher liegt, was auf eine symmetrische Verteilung der Daten um den Mittelwert hinweist. Eine Standardabweichung von 2400,05 MWh/15min zeigt eine mäßige Schwankung der Gesamtnetzlast.

Schritte 4. Datenbereinigung

Im nächsten Schritt erfolgt eine Datenbereinigung, bei der fehlende Werte im Datensatz behandelt und visualisiert, irrelevante Merkmale entfernt und Ausreißer mit Techniken wie Boxplots, Z-Score und IQR erkannt werden. In Schritt 1 ist festgestellt worden, dass es keine fehlenden Werte im Datensatz gibt. Die Variable ‚StrombezugPumpspeicher' wird bei der Analyse nicht berücksichtigt und die Spalte wird aus dem Dataframe entfernt. Ausreißer werden mit Boxplots veranschaulicht.

```
1  import seaborn as sns
2  import matplotlib.pyplot as plt
3
4  fig, axes = plt.subplots(1, 2)
5
6  sns.boxplot(ax = axes[0], data = SMARD_df['Gesamt_Netzlast'], notch=True,
       flierprops={"marker": "x"},
7              medianprops={"color": "r", "linewidth": 1}, whiskerprops={'
                   color': 'red', 'linewidth': 1, 'linestyle': ':'},
8              capprops={'color': 'black', 'linewidth': 1}, palette='bright',
                   orient='h', width=.2)
9
10 axes[0].set_xlabel('Realisierter Stromverbrauch in [MWh/15min]')
```

```
11
12 sns.boxplot(ax = axes[1], data = SMARD_df['Residuallast'], notch=True,
       flierprops={"marker": "x"},
13                 medianprops={"color": "r", "linewidth": 1}, whiskerprops={'
                       color': 'red', 'linewidth': 1, 'linestyle': ':'},
14                 capprops={'color': 'black', 'linewidth': 1}, palette='bright'
                       , orient='h', width=.2)
15 axes[1].set_xlabel('Residuallast in [MWh/15min]')
16
17 plt.show()
```

Listing 5.4: Schritte 3: Deskriptive statistische Analyse

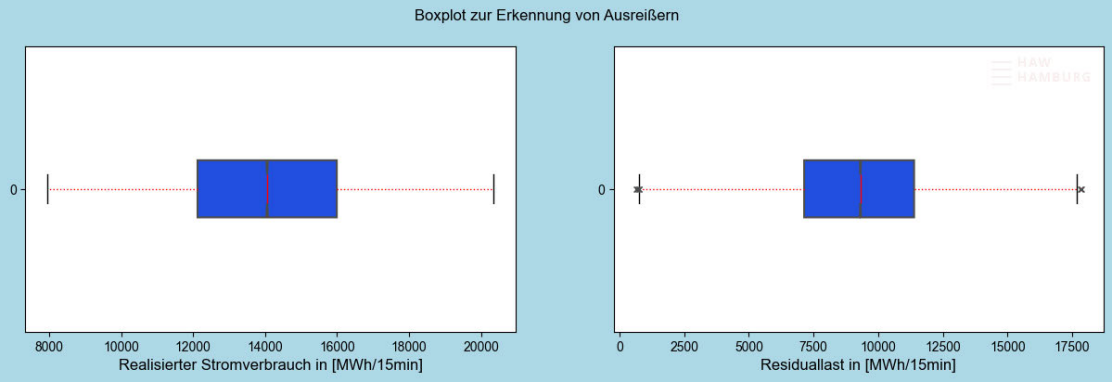

Abbildung 5.3: Boxplot zur Erkennung von Ausreißern

Aus der Abbildung 5.3 ist ersichtlich, dass es Ausreißer in Residuallast gibt.

Schritte 5. Datenvisualisierung

Bei der Datenvisualisierung wird eine univariate Analyse durchgeführt, um Ausreißer und ungewöhnliche Beobachtungen mittels Boxplot zu identifizieren. Zur Beschreibung der Streuung und Dispersion der Variablen werden die Spannweite, Varianz oder Standardabweichung berechnet. Außerdem wird die Form der Verteilung der Variablen durch Erstellen eines Histogramms identifiziert. Anschließend wird eine bivariate Analyse durchgeführt, um die Beziehung zwischen zwei Variablen in einem Datensatz zu untersuchen. Dabei werden Muster und Trends in einem Datensatz identifiziert, indem ein Punktediagramm erstellt wird.

Univariate Analyse

Ein Beispiel für einen Boxplot ist in Abbildung 5.3 zu sehen. Die Funktion pandas describe() liefert Mittelwert und Median zur Bestimmung der Streuung, während die Standardabweichung zur Bestimmung der Streuung der Variablen dient (siehe 5.0.1). Der Medianwert liegt nur wenig über dem Mittelwert, was auf eine symmetrische Verteilung der Daten um den Mittelwert hindeutet, was auch in Abbildung 5.4 erkennbar ist.

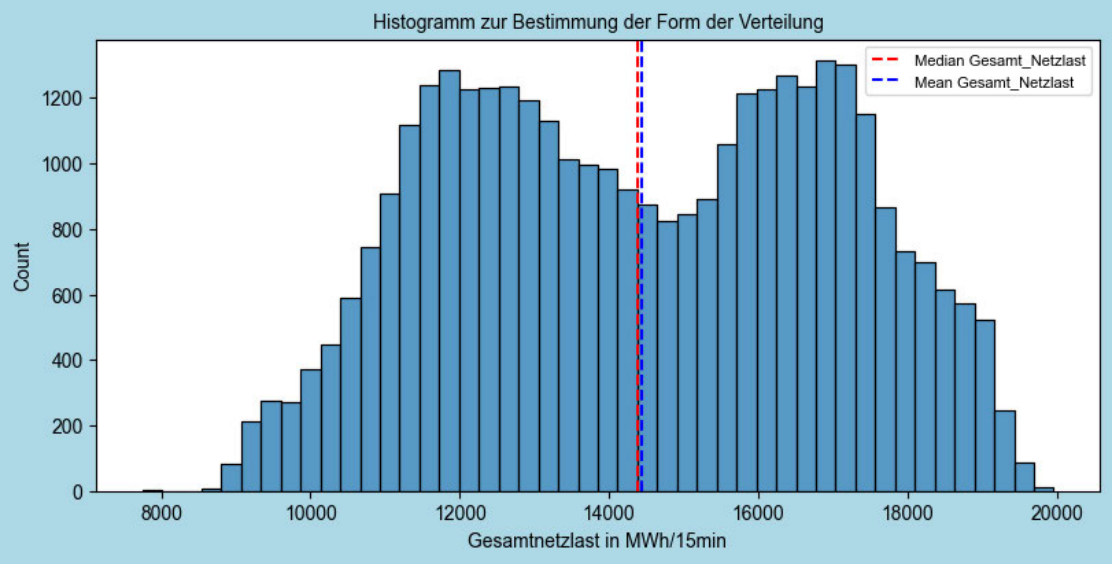

Abbildung 5.4: Histogramm zur Bestimmung der Form der Verteilung

Bivariate Analyse Die Abbildung 5.5 zeigt ein Punktendiagramm zur Ermittlung möglicher Trends und Muster im Datensatz.

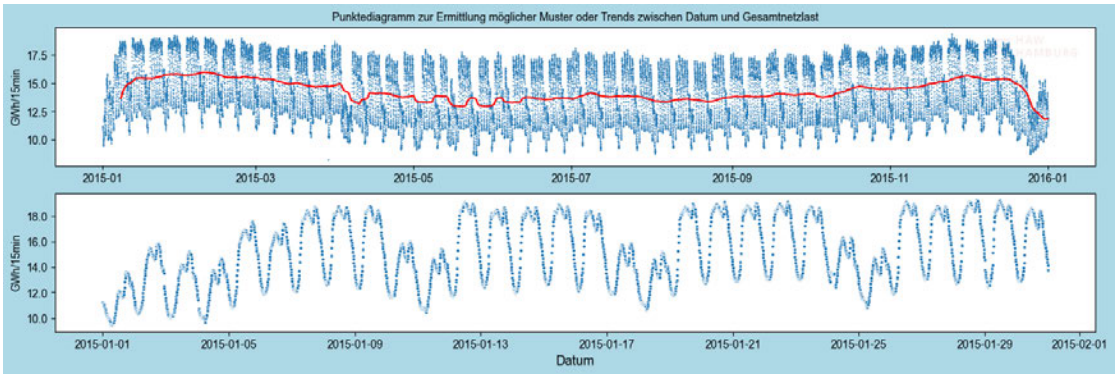

Abbildung 5.5: Punktendiagramm zur Ermittlung von Muster und Trend im Datensatz

Die Abbildung 5.5 zeigt die täglichen Höchst- und Tiefstwerte. Der gleitende Mittelwert zeigt eine steigende Tendenz des Stromverbrauchs im Winter und eine fallende Tendenz im Sommer. Die monatliche Darstellung der Stromverbrauchskurve zeigt den Einfluss von Wochenenden, Feiertagen und Wochentagen auf den Stromverbrauch.

Im nächsten Abschnitt werden verschiedene Visualisierungstypen mit Hilfe verschiedener Python-Bibliotheken implementiert. Dadurch können zusätzliche Informationen über den Datensatz gewonnen werden.

5.1 Daten Visualisierung mit Matplotlib

5.1.1 Liniendiagramm

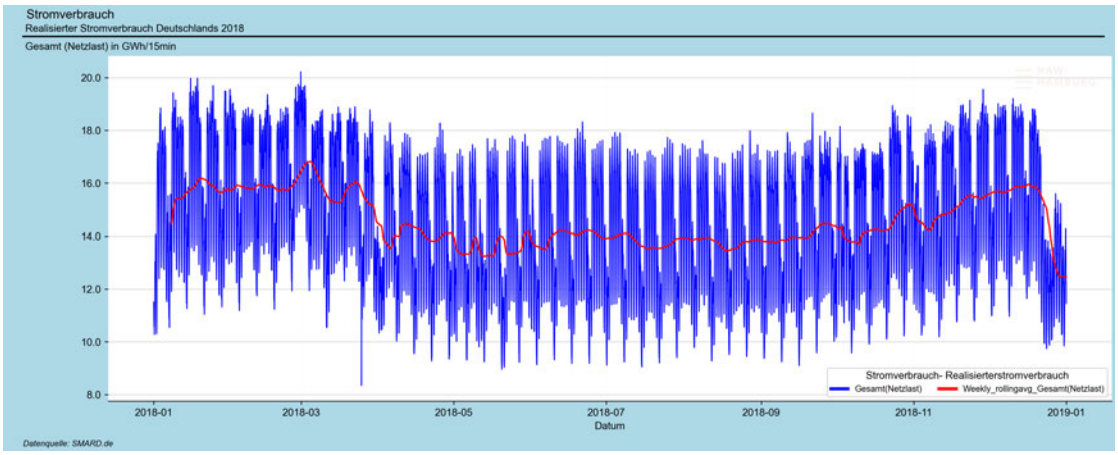

Abbildung 5.6: Liniendiagramm mit matplotlib

Das Liniendiagramm zeigt die sich täglich wiederholenden Höchst- und Tiefstwerte des Energieverbrauchs. Damit wird der typische Tagesverlauf dargestellt. Ein gleitender Durchschnitt zeigt einen höheren Stromverbrauch im Winter und einen niedrigeren Stromverbrauch im Sommer. Dies weist auf saisonale Einflüsse hin. Die geglättete Durchschnittslinie zeigt auch den allgemeinen Trend des Energiebedarfs im Jahresverlauf.

5.1.2 Balkendiagramm

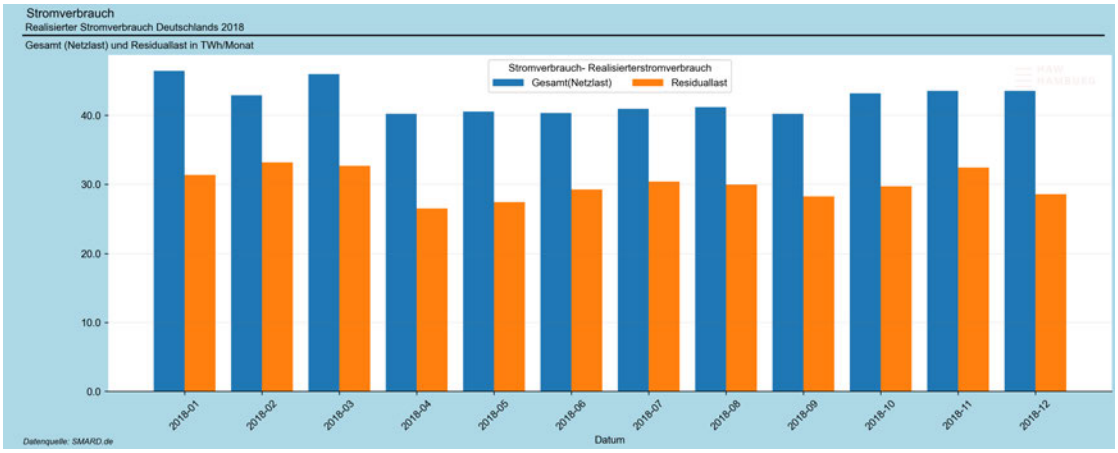

Abbildung 5.7: Balkendiagramm mit matplotlib

Das Balkendiagramm zeigt die monatlichen Stromverbrauchsdaten für Deutschland im Jahr 2018. Es unterscheidet zwischen zwei verschiedenen Arten von Energielasten: der Gesamt(Netzlast) und der Residuallast. Die Residuallast stellt die Differenz zwischen der gesamten Netzlast und der Stromerzeugung aus erneuerbaren Energiequellen dar und ist durch kleinere Balken dargestellt. Der Stromverbrauch zeigt ein saisonales Muster mit höherem Verbrauch in den kälteren Monaten (Januar, Februar) und gegen Ende des Jahres (Oktober, November, Dezember). Dies könnte auf einen erhöhten Heizbedarf in diesen Zeiträumen zurückzuführen sein.

5.1.3 Boxplot

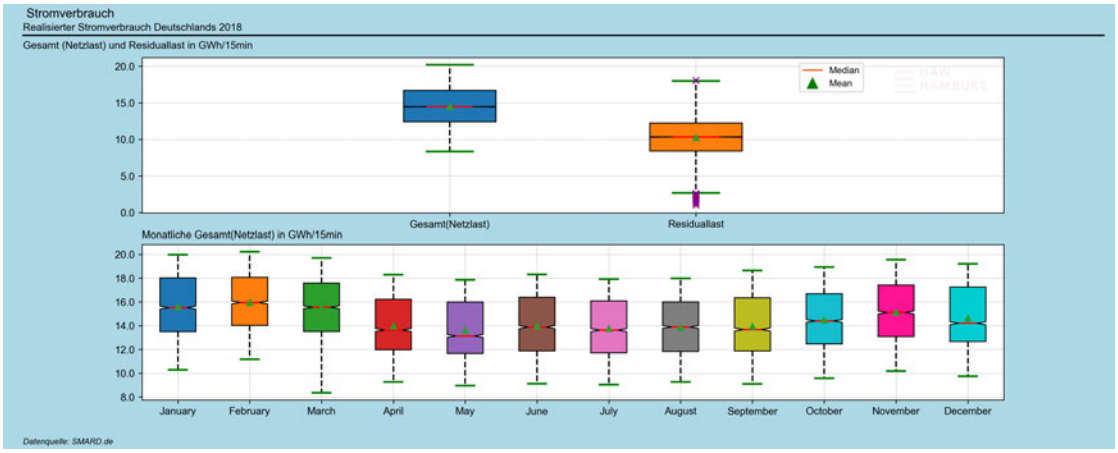

Abbildung 5.8: Boxplot mit matplotlib

Das erste Subplot zeigt die Verteilung der Gesamtnetz- und Residuallast für das gesamte Jahr. Es ist zu erkennen, dass der Median und der Mittelwert nahe beieinander liegen, was auf eine symmetrische Verteilung der Daten um die zentrale Tendenz hinweist. Die Whisker erstrecken sich, um den Bereich der Daten zu zeigen, und die Box zeigt den Interquartilsbereich (IQR), der die mittleren 50% der Datenpunkte darstellt. Die Residuallast enthält Ausreißer, die von den typischen Tageswerten abweichen und auf Extremwerte hinweisen.

Im zweiten Subplot ist die monatliche Verteilung der Gesamtnetzlast dargestellt. Ähnlich wie bei der vorherigen Analyse scheint es einen saisonalen Trend zu geben. In den kälteren Monaten sind die Medianwerte höher als in den wärmeren Monaten. Die Länge der Kästchen und die Whisker zeigen die Variabilität innerhalb jedes Monats. Die kälteren Monate weisen eine höhere Variabilität mit längeren Whiskern auf.

5.1.4 Kreisdiagramm

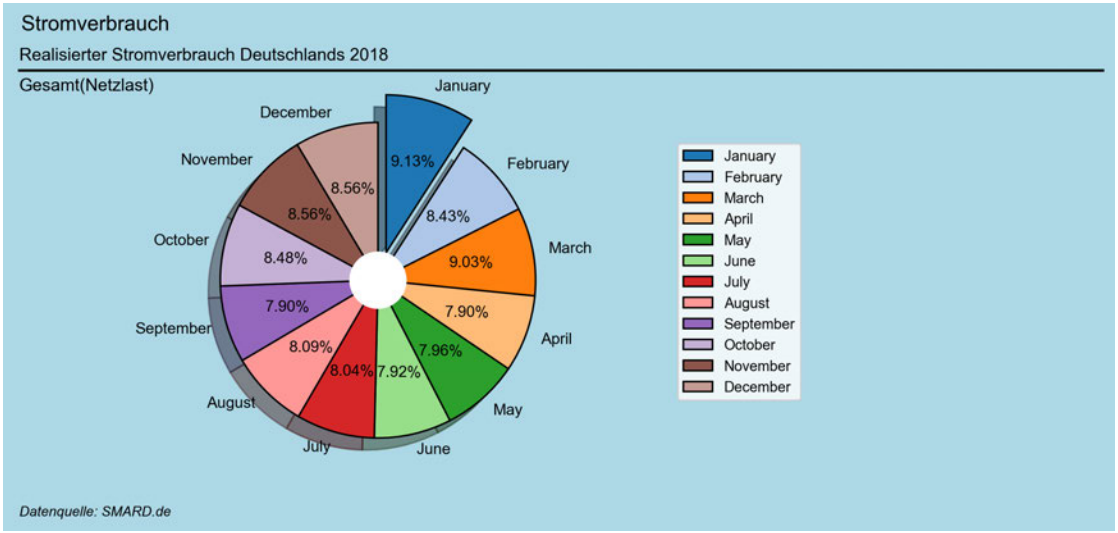

Abbildung 5.9: Kreisdiagramm mit matplotlib

Große Anteile sind in den Monaten Januar, Februar und Dezember zu sehen, was auf einen höheren Stromverbrauch in den kälteren Monaten aufgrund des Heizbedarfs hinweist, was auch die Auswirkungen der Jahreszeiten auf den Stromverbrauch verdeutlicht.

5.1.5 Flächendiagramm

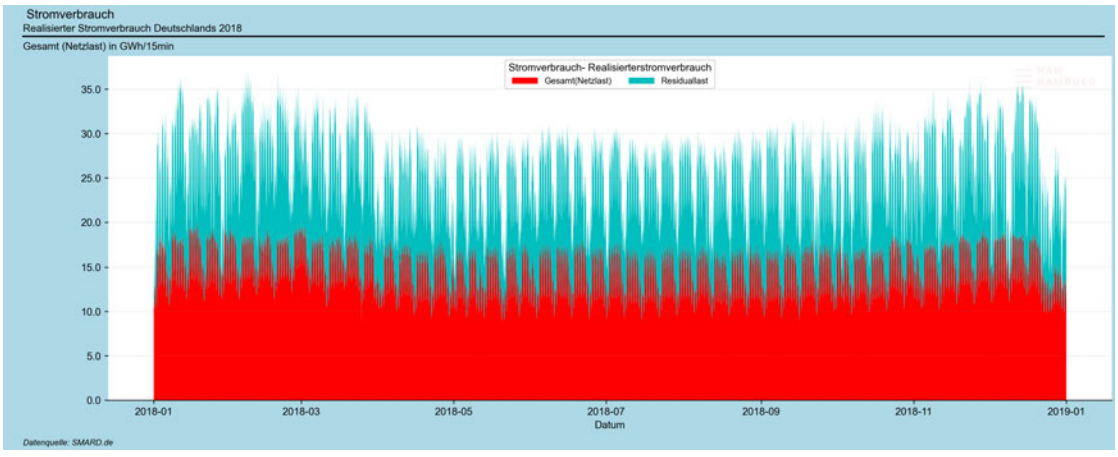

Abbildung 5.10: Flächendiagramm mit matplotlib

5.1.6 Histogramm

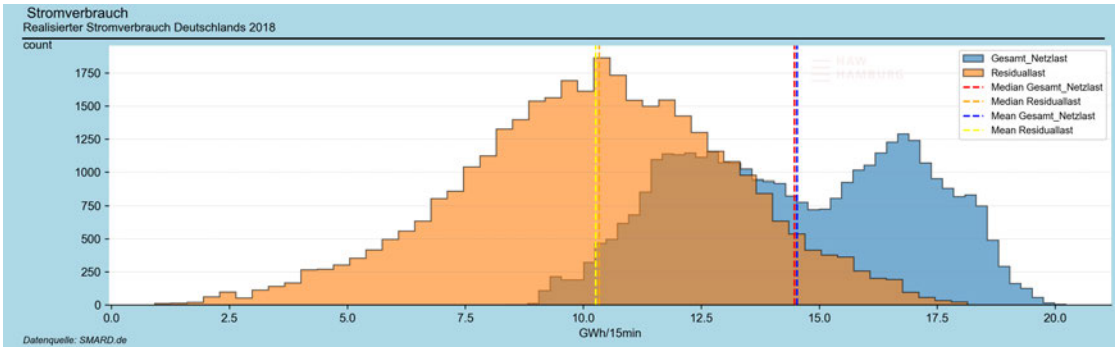

Abbildung 5.11: Histogramm mit matplotlib

Das Histogramm zeigt die Häufigkeitsverteilung von Gesamtnetzlast und Residuallast, wobei die gestrichelte Linie den Median, den Mittelwert der beiden Werte, darstellt. Die x-Achse zeigt den Wertebereich des Datensatzes, die y-Achse die Anzahl der Datenpunkte in jedem Bereich.

Die Daten sind symmetrisch, wobei der Mittelwert und der Median annähernd gleich sind, und das Histogramm weist auf beiden Seiten seiner Mitte eine ähnliche Form auf. Im Residuallastdiagramm gibt es zwei markante Spitzen, die auf einen bimodalen Datensatz hinweisen.

5.1.7 Violinplot

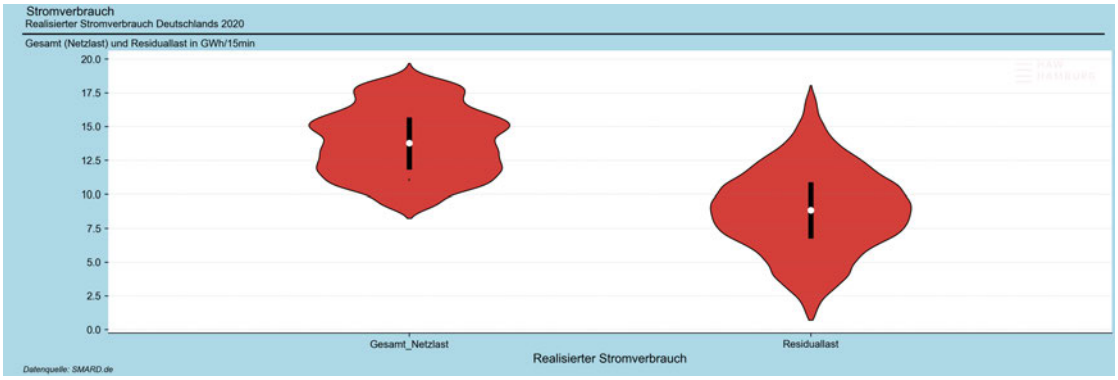

Abbildung 5.12: Violinplot mit matplotlib

Die gedrehte Kerneldichte der Gesamtnetzlast und Restiduallast zeigt die Wahrscheinlichkeitsdichte der Daten bei verschiedenen Werten. Der Median und der Interquartilbereich sind durch Markierungen und Kästen gekennzeichnet.

5.1.8 Wärmekarte

Betrachtung variante 1

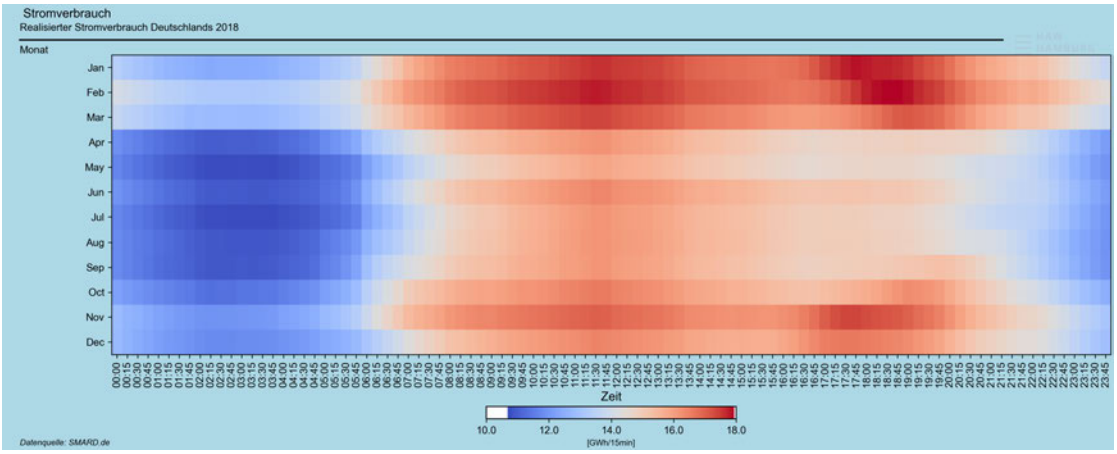

Abbildung 5.13: Heatmap Betrachtung Variante 1 mit matplotlib

Diese Heatmap zeigt den durchschnittlichen Stromverbrauch für jede Viertelstunde des Tages für jeden Monat.

Spitzenzeiten: Es scheint bestimmte Tageszeiten zu geben, zu denen der Stromverbrauch am höchsten ist, was an den helleren (oder wärmeren) Farben zu erkennen ist. Dies könnte mit der Rush Hour zusammenhängen, wenn die Menschen wach sind, wenn die Geschäfte geöffnet sind und wenn die industrielle Aktivität am intensivsten ist.

Übergangszeiten: Der Stromverbrauch steigt ab den frühen Morgenstunden allmählich an, erreicht seinen Höhepunkt gegen Mittag oder am frühen Nachmittag und sinkt dann gegen Abend wieder ab. Dieses Muster entspricht dem typischen Tagesablauf von Stromverbrauch.

Saisonale Schwankungen: Im Winter ist der Gesamtstromverbrauch höher. Es gibt leichte Schwankungen in den Spitzenzeiten des Stromverbrauchs während der verschiedenen Jah-

reszeiten, die auf Veränderungen der Tageslichtstunden und des menschlichen Verhaltens zurückzuführen sind.

Konstanz: Die Ähnlichkeit der Muster über die Monate hinweg (auch wenn die Intensität variiert) deutet auf konsistente Alltagsroutinen und Stromverbrauchsgewohnheiten in Haushalten, Gewerbe und Industrie hin.

Betrachtung variante 2

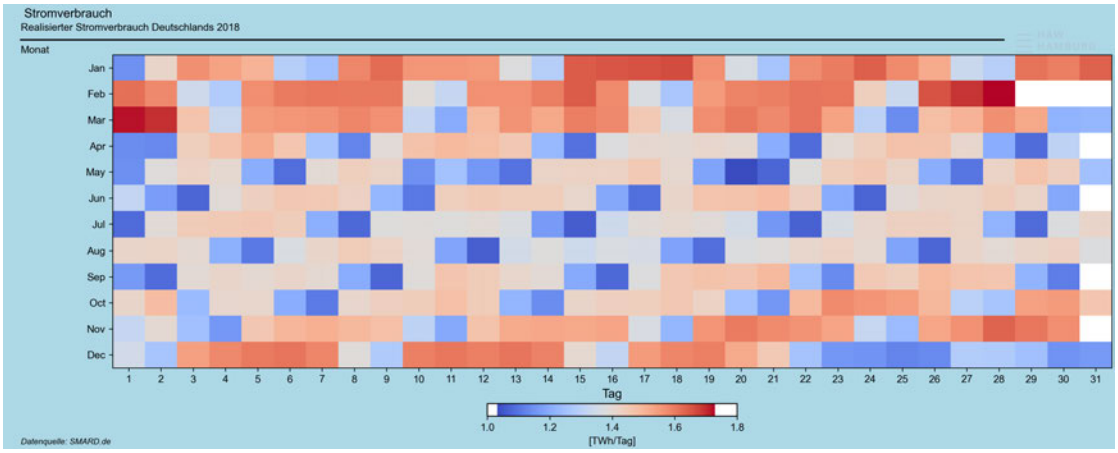

Abbildung 5.14: Heatmap Betrachtung Variante 2 mit matplotlib

Diese Heatmap zeigt die tägliche Summe des Stromverbrauchs für jeden Tag des Monats im Laufe des Jahres.

Tägliche Schwankungen: Die Stromnachfrage ist an Wochenenden und Feiertagen im Allgemeinen niedriger als an Wochentagen, was auf eine geringere gewerbliche und industrielle Nachfrage zurückzuführen ist.

Saisonale Schwankungen: Höherer Stromverbrauch in den Wintermonaten.

Sie können auch erkennen, dass nach jeweils 5 Kästchen mit warmer Farbe (hohem Stromverbrauch) ein helles Kästchen erscheint, das für Wochenenden steht. Wenn mehr als 2 helle Kästchen erscheinen, handelt es sich um einen Feiertag, wie z. B. im Dezember wegen der Weihnachtsferien oder in der ersten Aprilwoche wegen der Osterferien.

Betrachtung variante 3

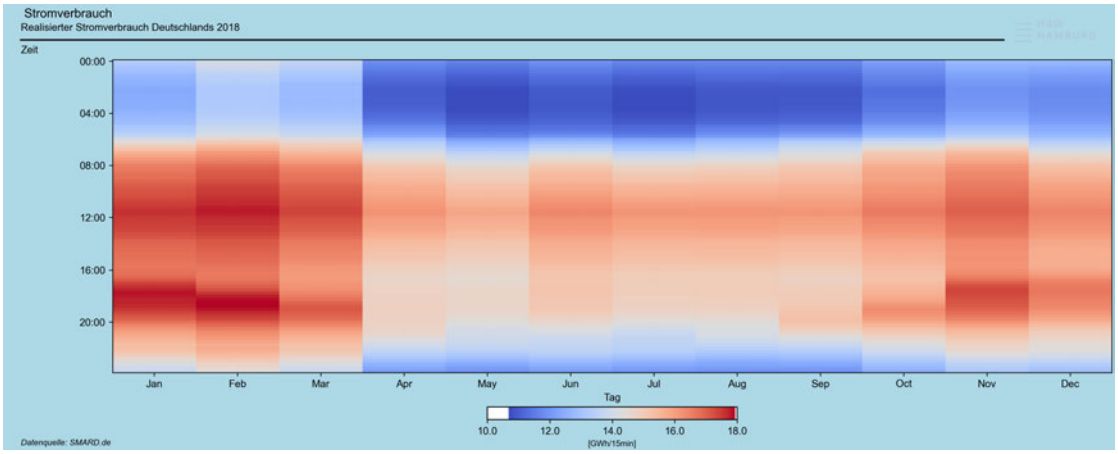

Abbildung 5.15: Heatmap Betrachtung Variante 3 mit matplotlib

Konsistenz im Tagesverlauf: Der Farbverlauf von Blau zu Rot und wieder zurück zu Blau zeigt ein konsistentes Muster des Stromverbrauchs, der im Laufe des Tages ansteigt, seinen Höhepunkt erreicht und wieder abfällt. Dies deutet auf eine regelmäßige tägliche Aktivität und einen regelmäßigen Stromverbrauch hin.

Tageszeitliche Schwankungen: Der Stromverbrauch ist tagsüber höher als in der Nacht.

Saisonale Schwankungen: Die Verbrauchsintensität folgt einem eindeutigen saisonalen Muster, das im Winter höher und im Sommer niedriger ist.

Spitzenstunden: In Winter gibt es zwei Spitzen im Stromverbrauch – eine am Nachmittag und eine am Abend und im Sommer weist der Stromverbrauch eine einzige Spitze auf, typischerweise während des Tages.

Betrachtung variante 4

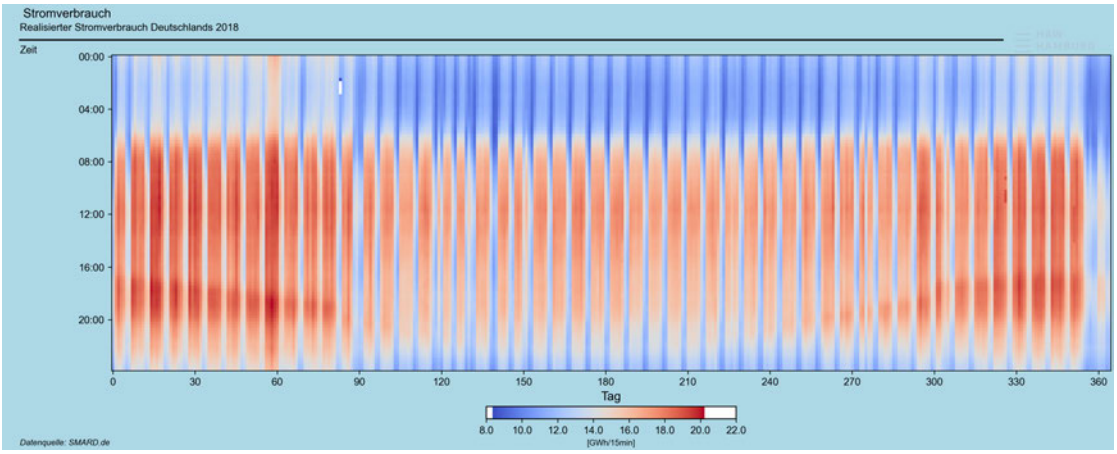

Abbildung 5.16: Heatmap Betrachtung Variante 4 mit matplotlib

Die Heatmap stellt den viertelstündlichen Stromverbrauch für jeden Tag des Jahres dar. Diese Variante der Heatmap zeigt die Konsistenz des Stromverbrauchs über den gesamten Tag, die täglichen und saisonalen Schwankungen, die Spitzenzeiten und den Einfluss von Wochentagen und Wochenenden auf den Energieverbrauch.

Matplotlib ist eine vielseitige und leistungsstarke Bibliothek zur Erstellung von hochwertigen Grafiken für wissenschaftliche Publikationen. Sie bietet sowohl eine einfache und intuitive Schnittstelle (Pyplot) als auch eine objektorientierte Architektur, die es erlaubt, alles innerhalb eines Plots zu verändern [23]. Alle grundlegenden Plots und komplexen Visualisierungen, wie Wärmekarten und deren verschiedene Varianten, sind mit Matplotlib implementiert. Für fast jede Eigenschaft lassen sich in Matplotlib Voreinstellungen definieren: Größe und DPI der Grafik, Linienbreite, Farbe und Stil, Achsen, Achsen- und Rastereigenschaften, Text- und Schrifteigenschaften und so weiter [25]. „Nice-to-have“-Funktionen für statische Plots, wie z. B. die erweiterte Anpassung von Text und Anmerkungen wie Logo und Fußnote für die Quelle, sind ebenfalls implementiert. Die Community von Matplotlib ist riesig, da sie externe Ressourcen wie Bücher, Kapitel, Artikel, Videos und Tutorials zur Verfügung stellt und eine ausführliche Dokumentation für Anfänger bietet, um mit der Visualisierung zu beginnen [26].

5.2 Daten Visualisierung mit Seaborn

5.2.1 Liniendiagramm

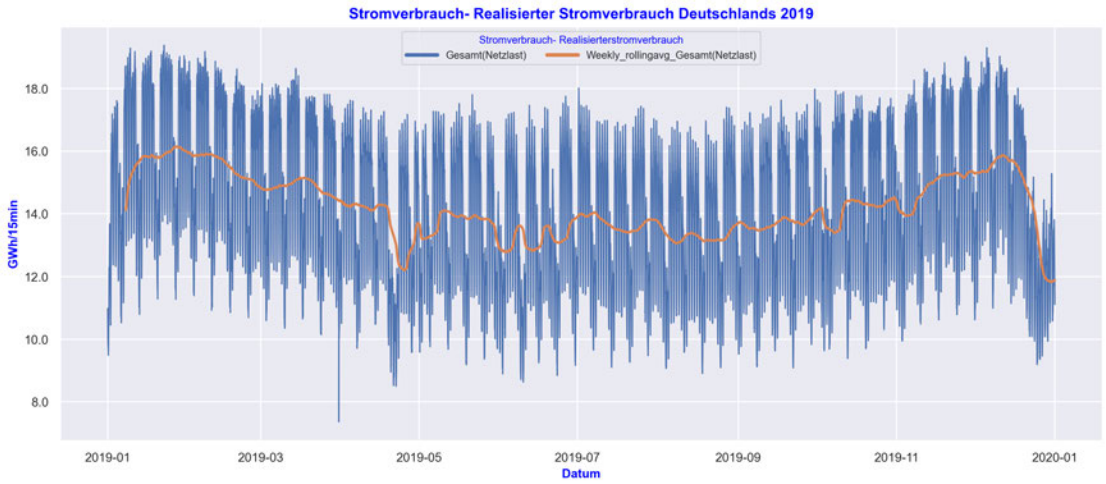

Abbildung 5.17: Liniendiagramm mit Seaborn

5.2.2 Balkendiagramm

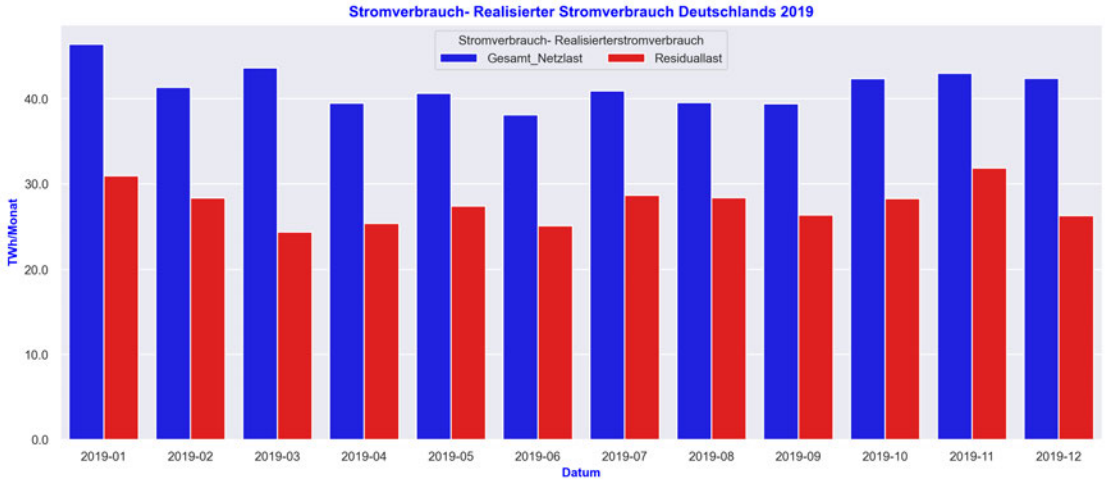

Abbildung 5.18: Balkendiagramm mit Seaborn

5.2.3 Boxplot

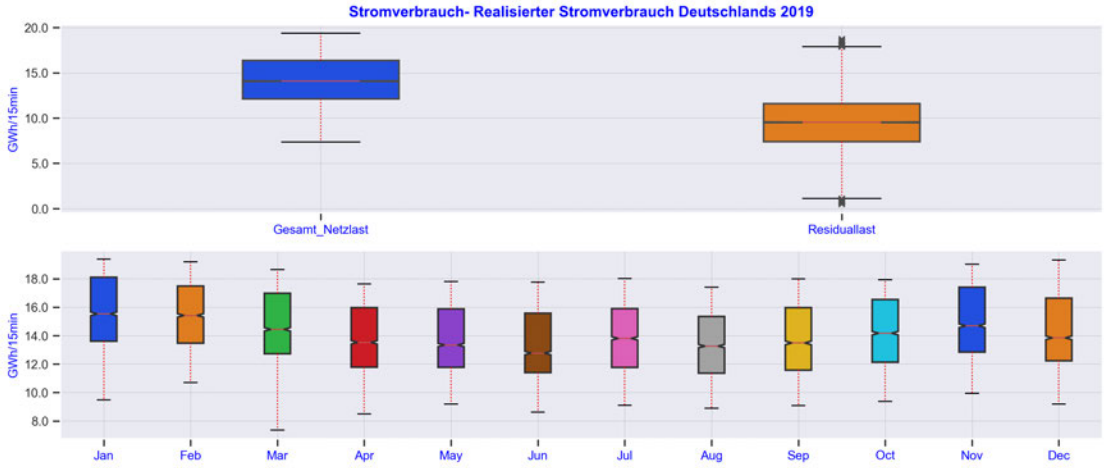

Abbildung 5.19: Boxplot mit Seaborn

5.2.4 Kreisdiagramm

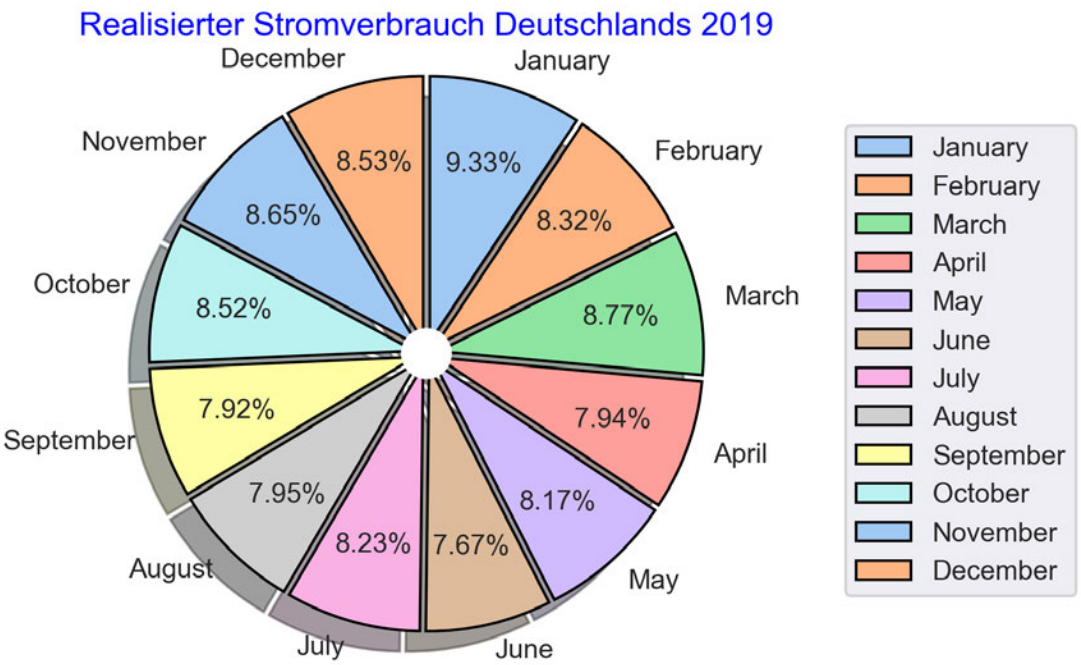

Abbildung 5.20: Kreisdiagramm mit Seaborn

5.2.5 Flächendiagramm

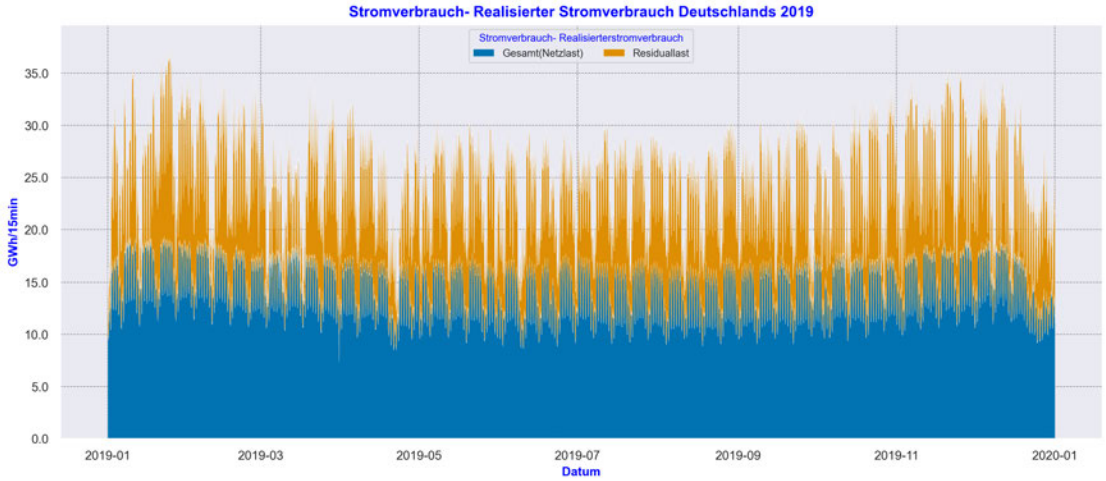

Abbildung 5.21: Flächendiagramm mit Seaborn

5.2.6 Histogramm

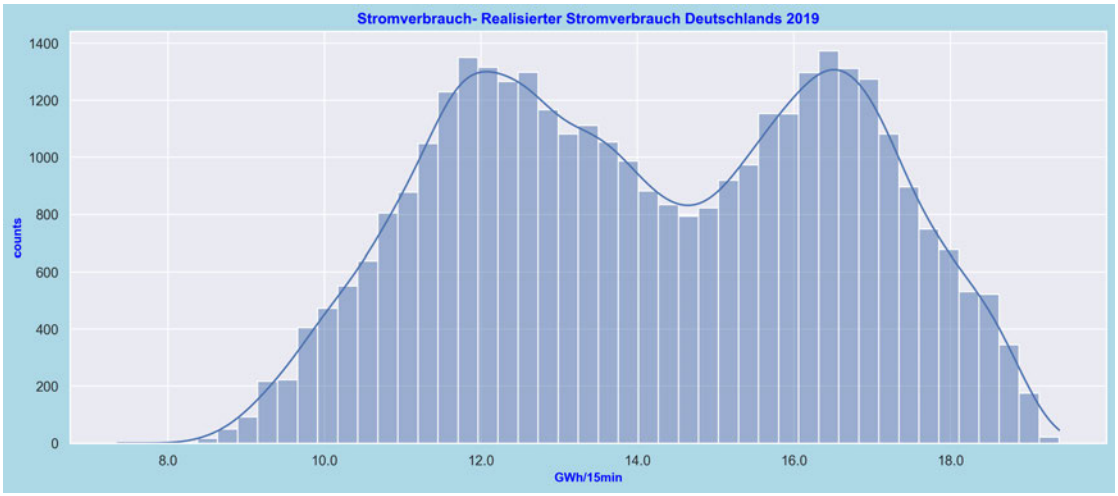

Abbildung 5.22: Histogramm mit Seaborn

5.2.7 Violinplot

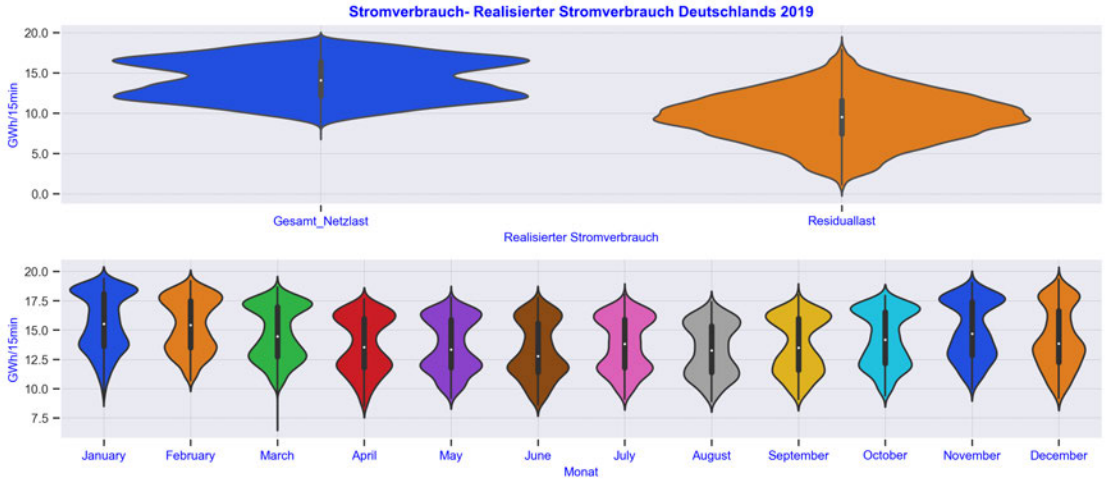

Abbildung 5.23: Violinplot mit Seaborn

5.2.8 Wärmekarte

Betrachtung variante 1

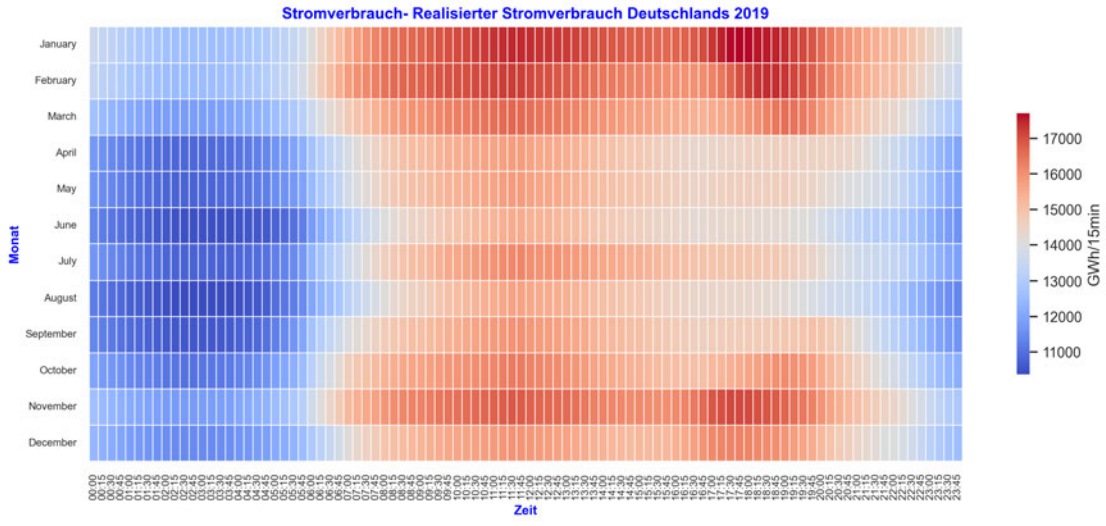

Abbildung 5.24: Heatmap Betrachtung Variante 1 mit Seaborn

Betrachtung variante 2

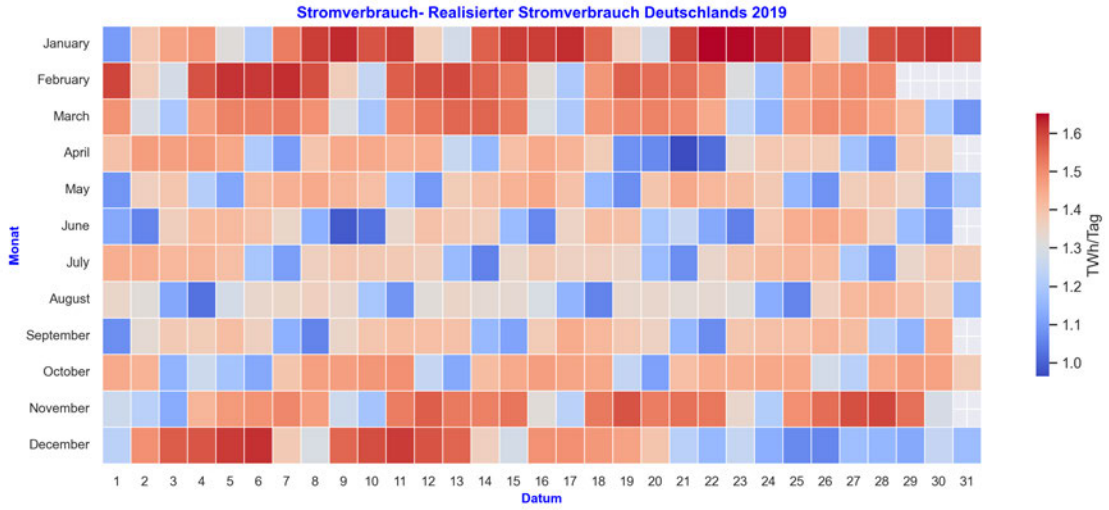

Abbildung 5.25: Heatmap Betrachtung Variante 2 mit Seaborn

Betrachtung variante 3

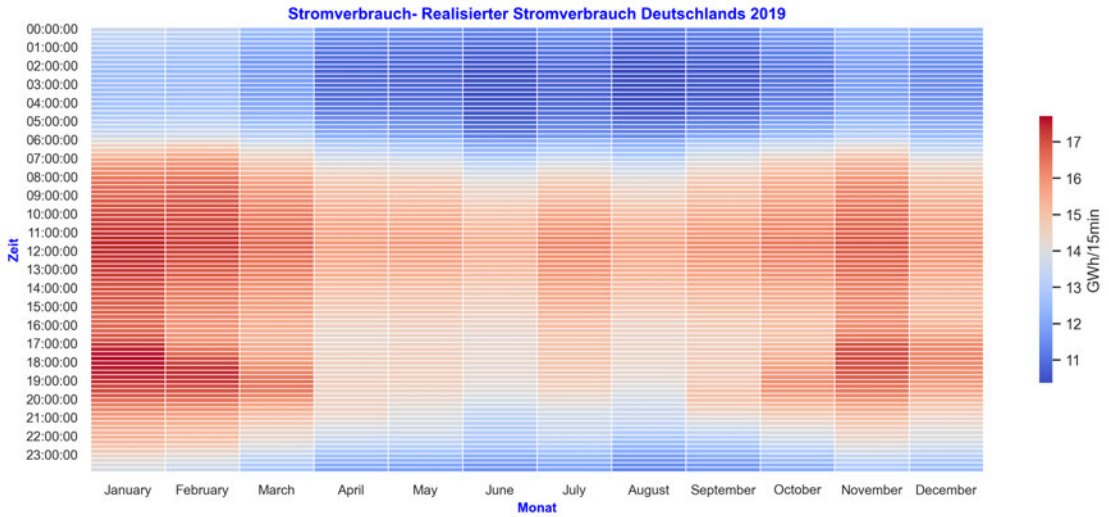

Abbildung 5.26: Heatmap Betrachtung variante 3 mit Seaborn

Betrachtung variante 4

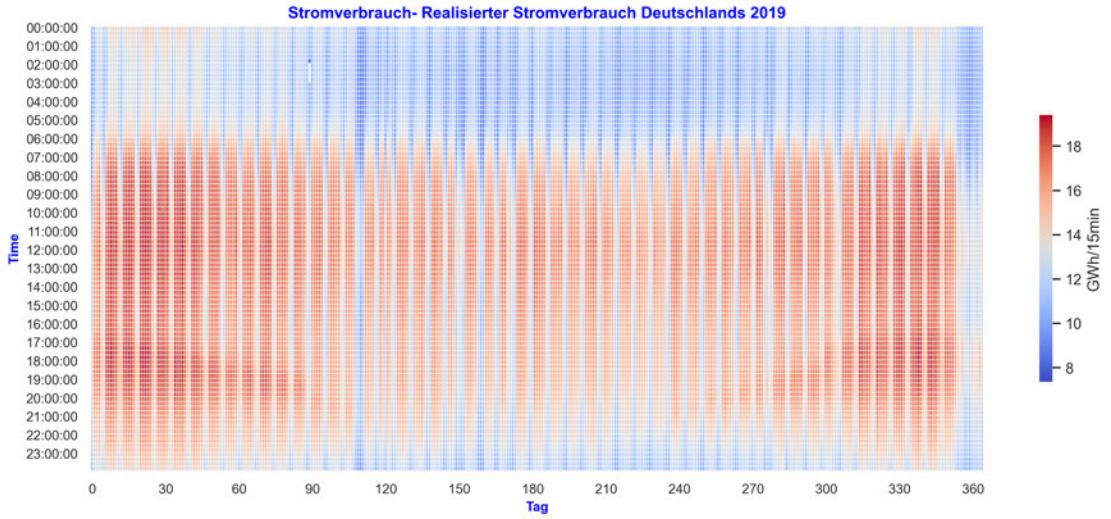

Abbildung 5.27: Heatmap Betrachtung Variante 4 mit seaborn

Mit Seaborn können alle grundlegenden Plots und komplexe Visualisierungen wie Heatmaps und deren Varianten erstellt werden. Matplotlib ist sehr anpassungsfähig, aber es kann schwierig sein zu wissen, welche Einstellungen optimiert werden müssen, um eine ansprechende Darstellung zu erhalten. Seaborn kommt mit einer Reihe von angepassten Themen und einer High-Level-Schnittstelle zur Steuerung des Aussehens von Matplotlib-Grafiken. Seaborn erleichtert die Verwendung von Farben, die gut zu Dateneigenschaften und Visualisierungszielen passen. Einige Diagramme, wie z. B. Kreisdiagramm und Flächendiagramm, haben keine eigenen eingebauten Funktionen. Daher wird Matplotlib verwendet, um das Diagramm zu erzeugen, und die Seaborn-Funktionen werden verwendet, um ansprechende Diagramme zu erstellen. Die Daten von 2019 werden mit Seaborn visualisiert und die Ergebnisse werden ähnlich wie die mit Matplotlib erstellten Diagramme analysiert [48].

5.3 Daten Visualisierung mit Vega-Altair

5.3.1 Liniendiagramm

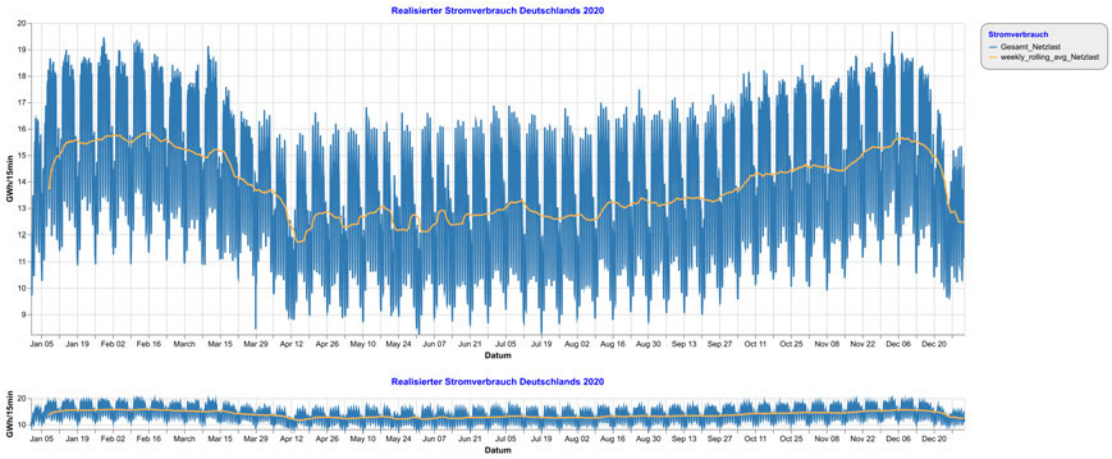

Abbildung 5.28: Liniendiagramm mit Vega-Altair

5.3.2 Balkendiagramm

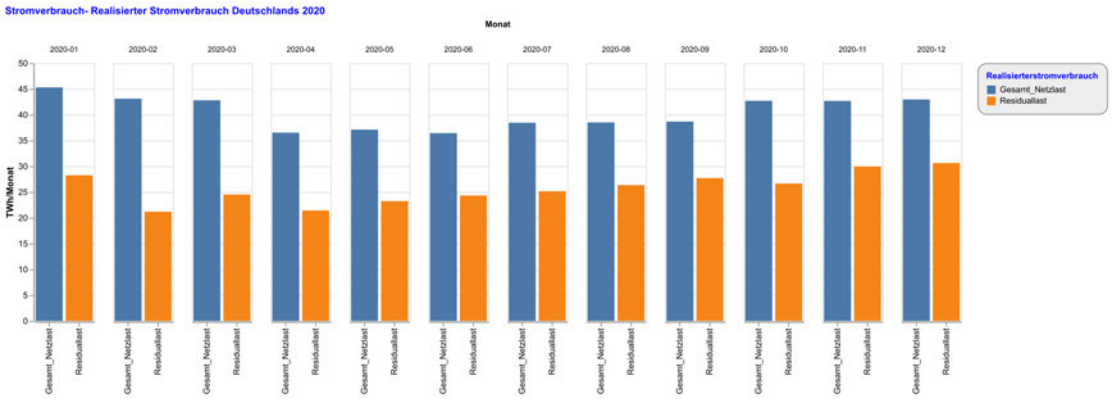

Abbildung 5.29: Balkendiagramm mit Vega-Altair

5.3.3 Boxplot

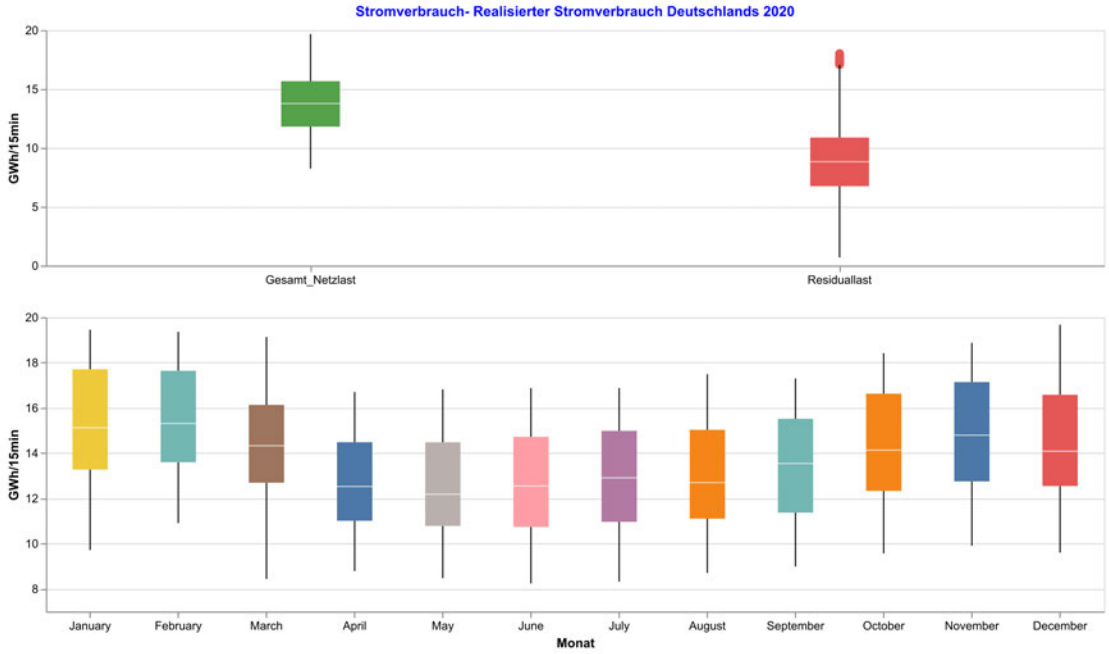

Abbildung 5.30: Boxplot mit Vega-Altair

5.3.4 Kreisdiagramm

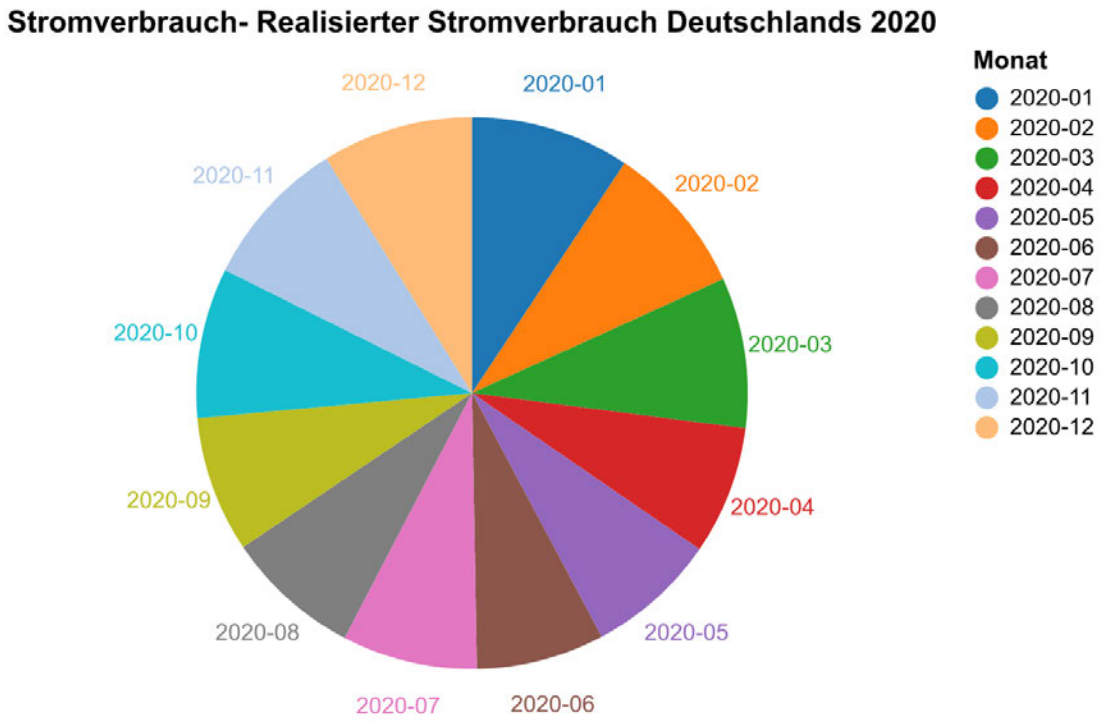

Abbildung 5.31: Kreisdiagramm mit Vega-Altair

5.3.5 Flächendiagramm

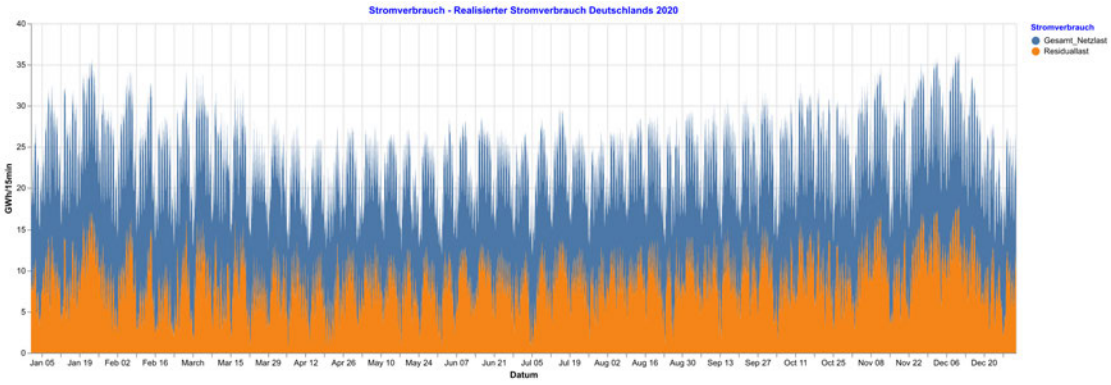

Abbildung 5.32: Flächendiagramm mit Vega-Altair

5.3.6 Histogramm

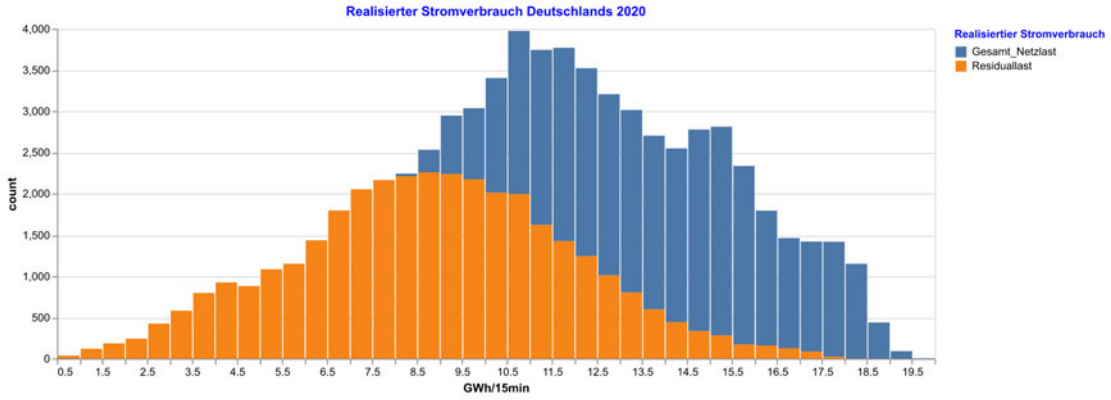

Abbildung 5.33: Histogramm mit Vega-Altair

5.3.7 Violinplot

Abbildung 5.34: Violinplot mit Vega-Altair

5.3.8 Wärmekarte

Betrachtung variante 1

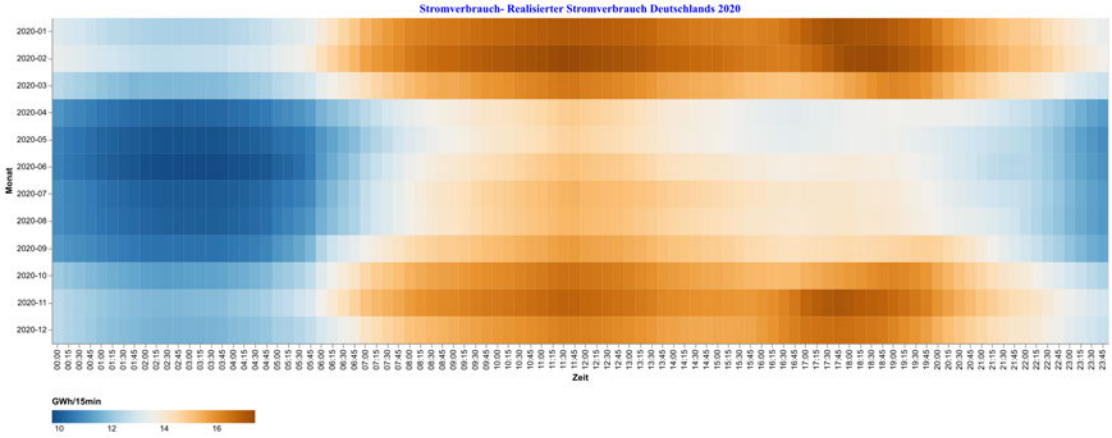

Abbildung 5.35: Heatmap Betrachtung Variante 1 mit Vega-Altair

Betrachtung variante 2

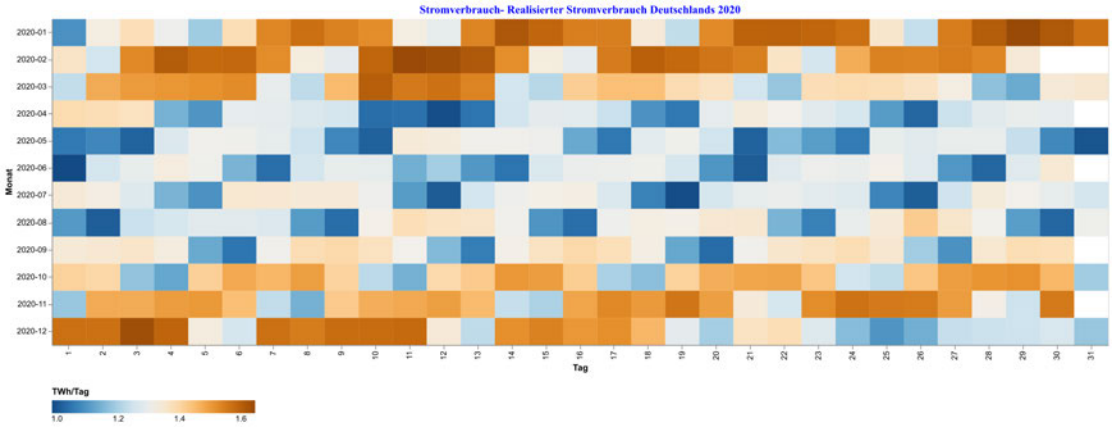

Abbildung 5.36: Heatmap Betrachtung Variante 2 mit Vega-Altair

Betrachtung variante 3

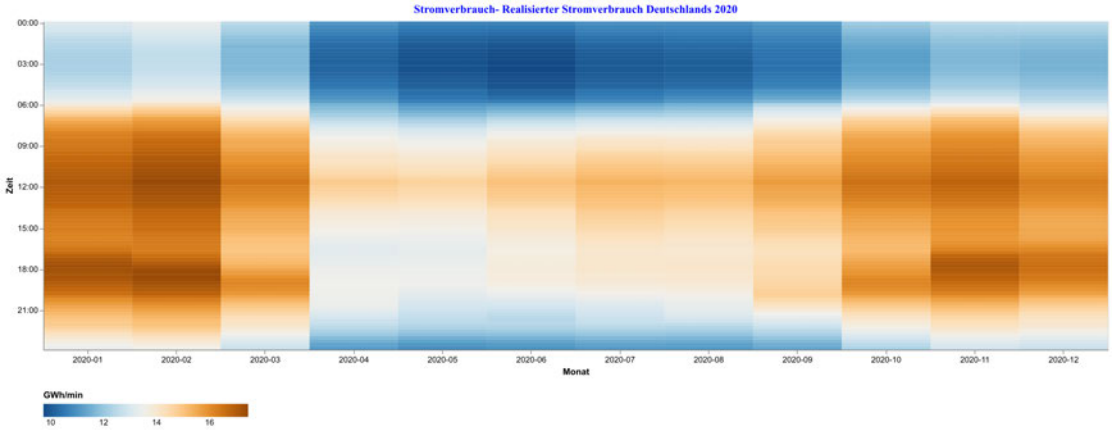

Abbildung 5.37: Heatmap Betrachtung Variante 3 mit Vega-Altair

Betrachtung variante 4

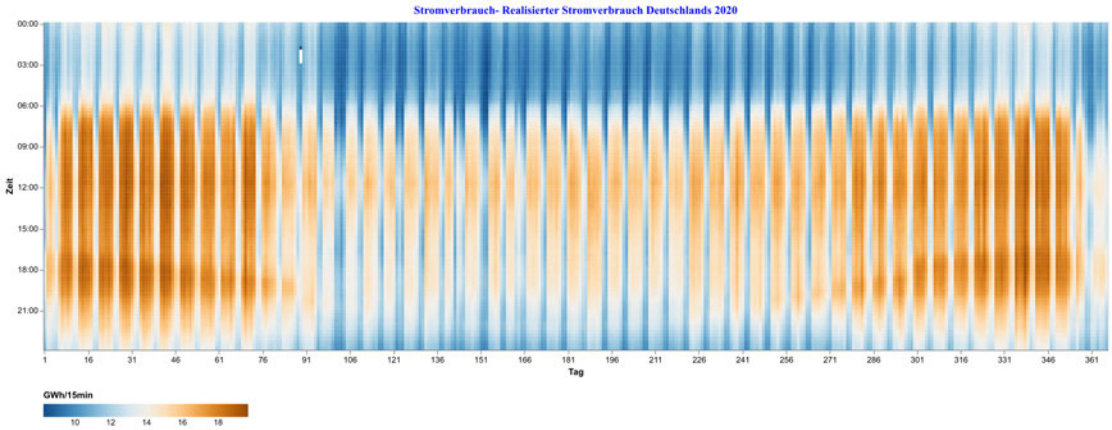

Abbildung 5.38: Heatmap Betrachtung Variante 4 mit Vega-Altair

Vega-Altair ist eine Python-Bibliothek für deklarative statistische Visualisierung, basierend auf Vega und Vega-Lite. Bei der Erstellung eines Diagramms in Altair erfolgt automatisch eine Übersetzung des Codes in eine Vega-Lite-Spezifikation im JSON-Format. Diese Spezifikation beschreibt die Struktur des Diagramms, einschließlich der darzustellenden Daten, der verwendeten Marker (wie Balken, Linien und Punkte) und der Kodierungen, die Datenvariablen auf visuelle Eigenschaften abbilden (wie die x-Achse, y-Achse und Farbe). Dies sollte auch alle weiteren Anpassungen wie Achsenbeschriftungen, Titel

und Legenden umfassen. Der Kerngedanke besteht darin, Verbindungen zwischen den Datenspalten und den visuellen Kodierungskanälen zu erklären [46].

Altair implementiert sowohl einfache Plots als auch komplexere Visualisierungen wie Heatmaps und ihre Varianten mit Vega-Lite. Altair verwendet eine deklarative Grammatik für die Visualisierung und Interaktion, die es ermöglicht, den Plot durch Aktionen wie das Erhöhen oder Verringern der Deckkraft und die Auswahl von Intervallen durch Klicken und Ziehen zu verändern. Widgets sind an den Plot gebunden, um datengesteuerte Lookups zur Anzeige ausgewählter Variablen zu ermöglichen. Für logikgesteuerte Vergleiche wird dem Diagramm ein Farbwähler hinzugefügt, mit dem der Benutzer interaktiv Diagrammfarben auswählen kann. Außerdem sind zwei Ansichten derselben Daten vertikal miteinander verbunden, wobei die Auswahl im unteren Diagramm die Anzeige im oberen Diagramm aktualisiert. Grundlegende Interaktionen wie Schwenken und Zoomen sind nicht hinzugefügt, da dies die Leistung beeinträchtigt. Stattdessen ist nur Tooltips implementiert, die es dem Benutzer ermöglichen, den Mauszeiger über die Daten zu bewegen, um Informationen zu erhalten. Diagramme können in JSON, HTML, PNG, SVG und PDF gespeichert werden [47].

5.4 Daten Visualisierung mit Bokeh

5.4.1 Liniendiagramm

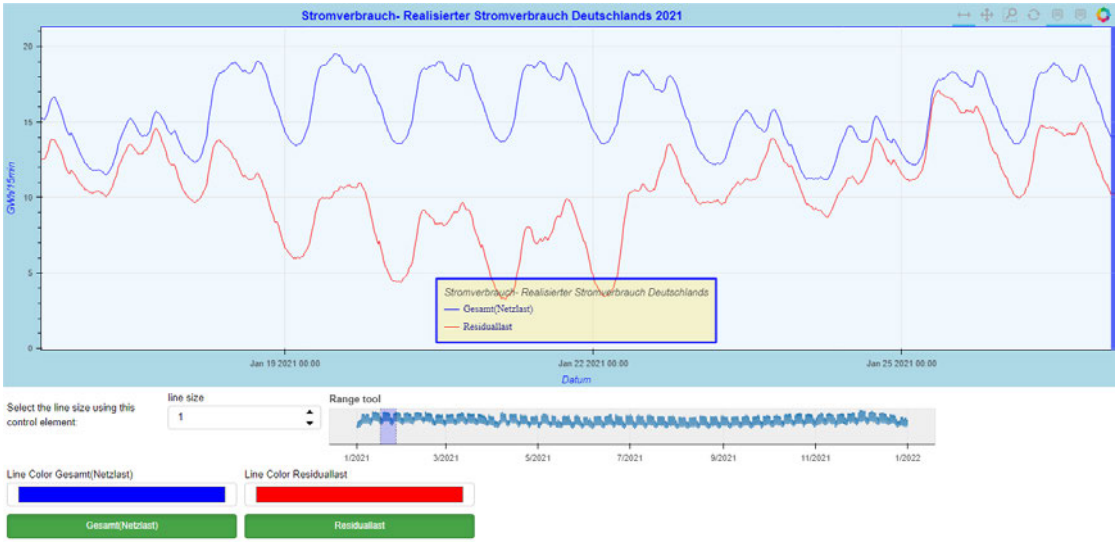

Abbildung 5.39: Liniendiagramm mit Bokeh

5.4.2 Balkendiagramm

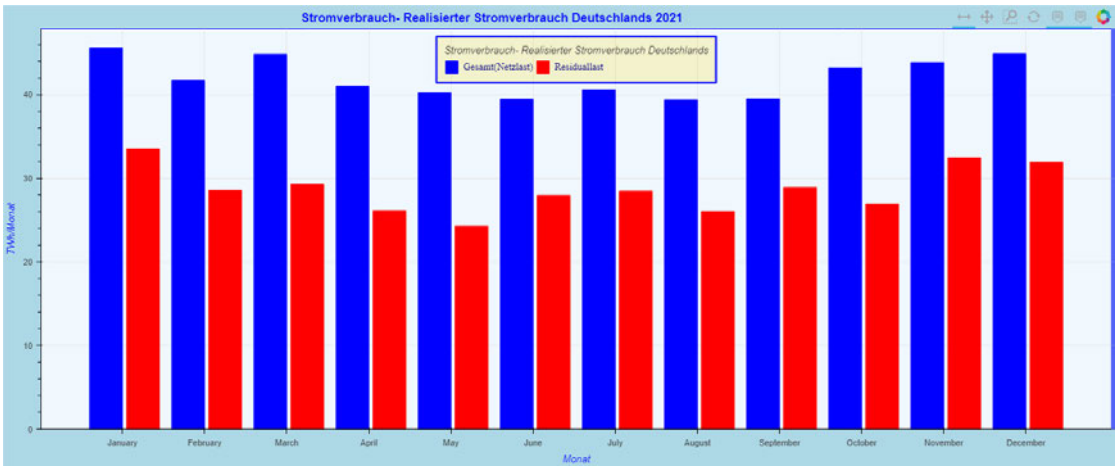

Abbildung 5.40: Balkendiagramm mit Bokeh

5.4.3 Boxplot

Bokeh bietet keine integrierte Funktion, um Boxplots zu erstellen.

5.4.4 Kreisdiagramm

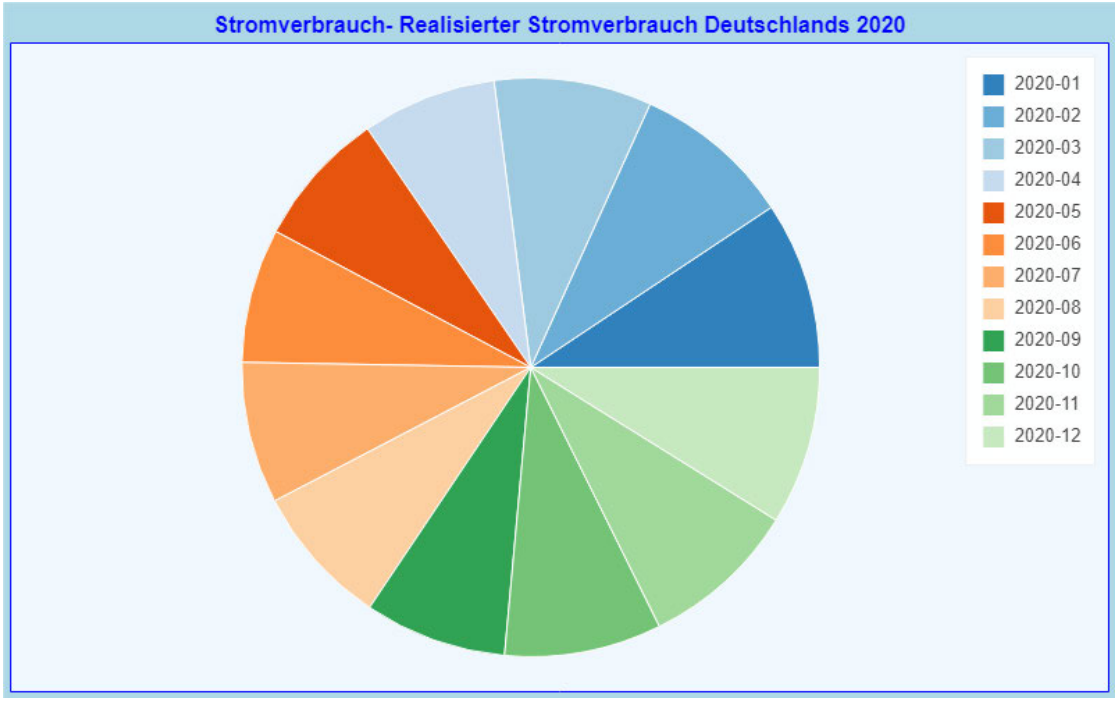

Abbildung 5.41: Kreisdiagramm mit Bokeh

5.4.5 Flächendiagramm

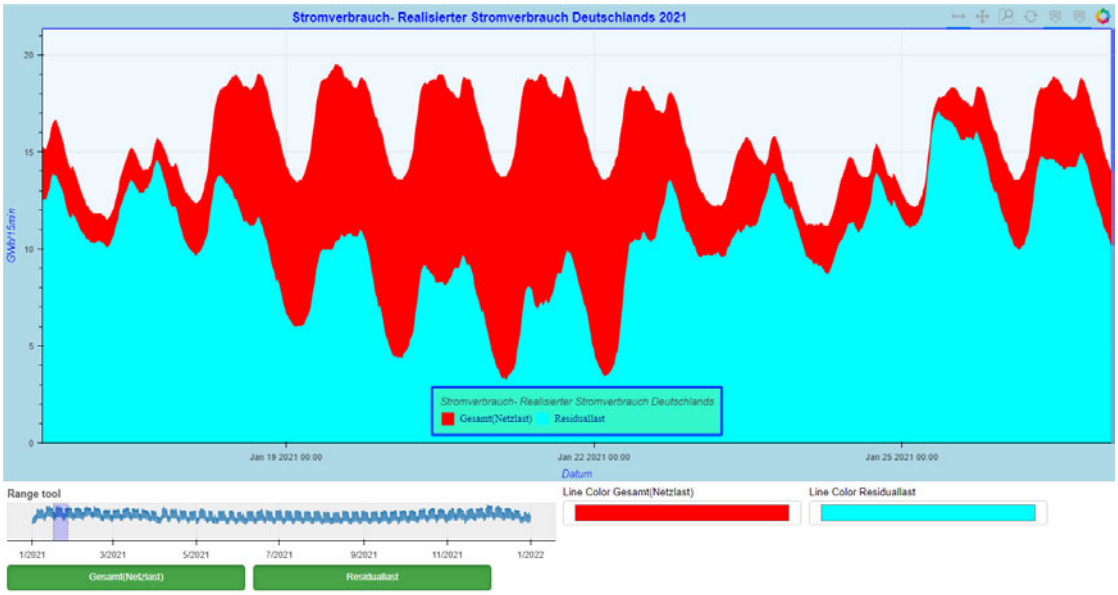

Abbildung 5.42: Flächendiagramm mit Bokeh

5.4.6 Histogramm

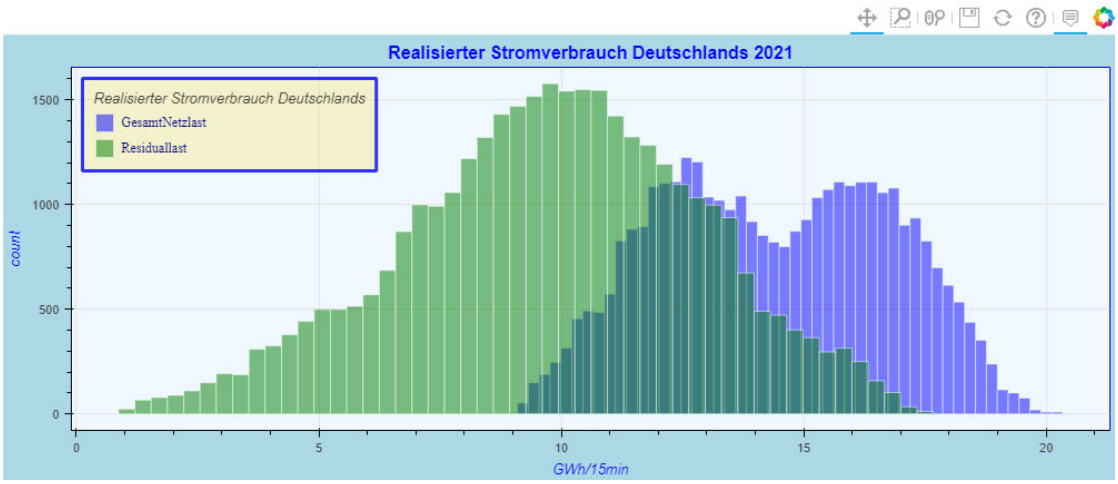

Abbildung 5.43: Histogramm mit Bokeh

5.4.7 Violinplot

Ähnlich wie Boxplot bietet Bokeh keine integrierte Funktion, um Violinplot zu erstellen.

5.4.8 Wärmekarte

Betrachtung variante 1

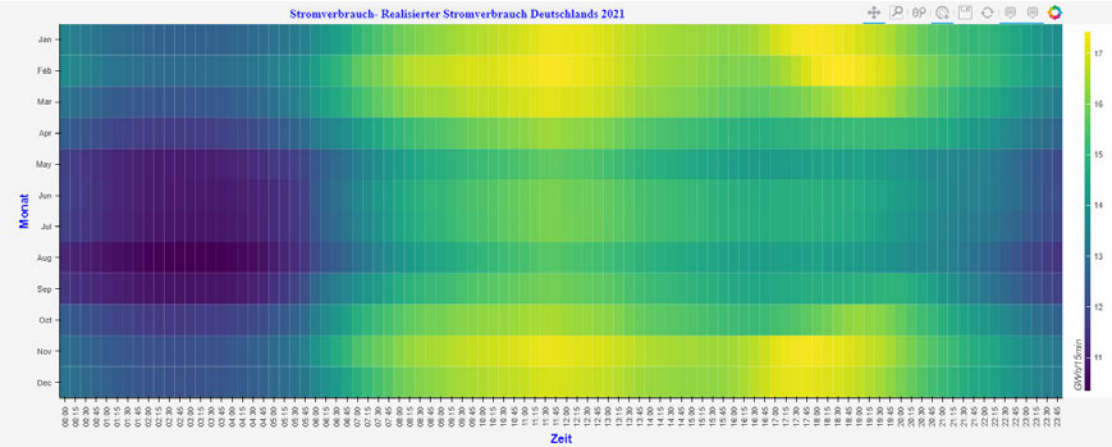

Abbildung 5.44: Heatmap Betrachtung Variante 1 mit Bokeh

Betrachtung variante 2

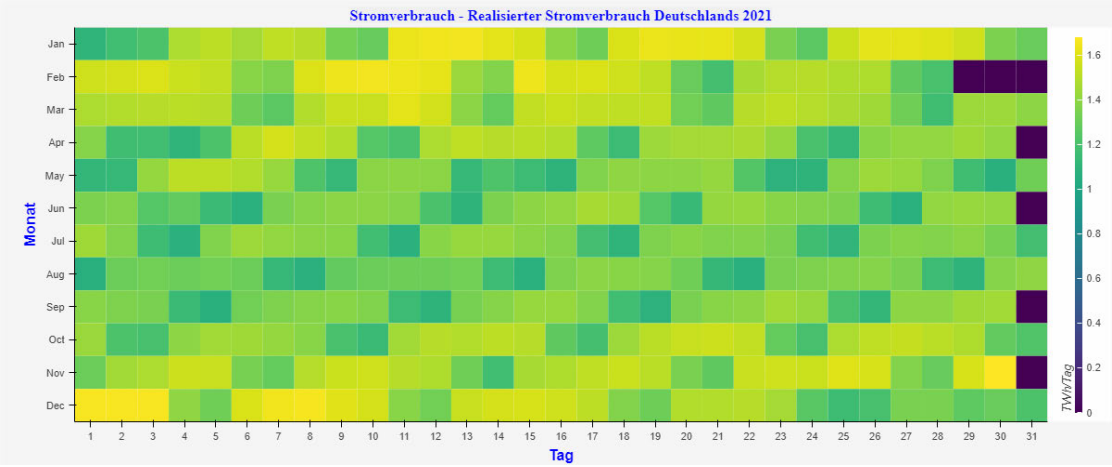

Abbildung 5.45: Heatmap Betrachtung Variante 2 mit Bokeh

Betrachtung variante 3

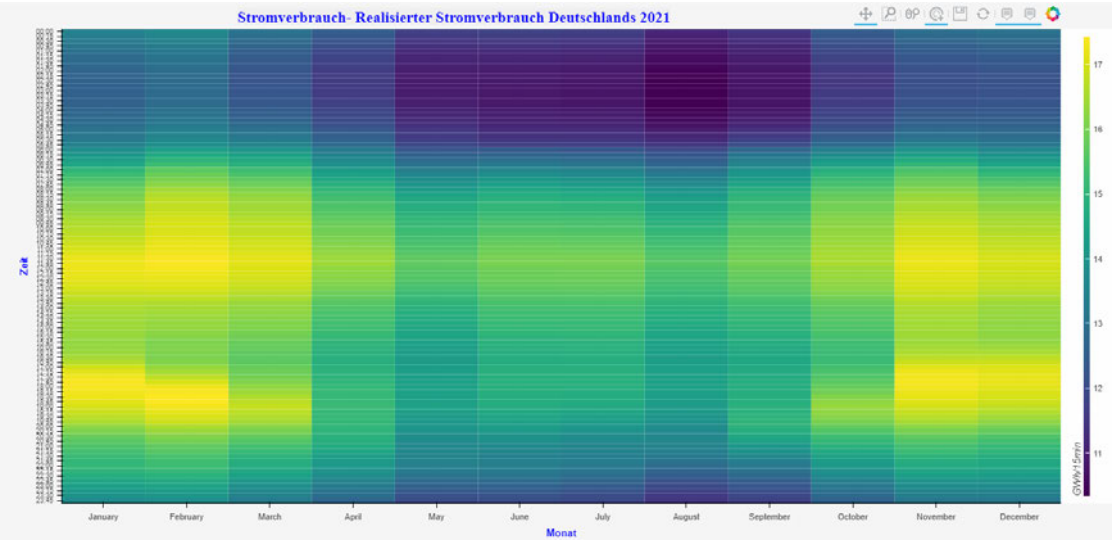

Abbildung 5.46: Heatmap Betrachtung Variante 3 mit Bokeh

Bokeh bietet keine integrierte Funktion zur direkten Erstellung von Boxplots und Violinplots. Alle anderen Visualisierungstypen werden jedoch von Bokeh unterstützt. Es gibt mehrere Möglichkeiten, auf browserbasierte Benutzerinteraktionen zu reagieren. Plottools wie xpan, pan, box_zoom, lasso_select, zoom_in, crosshair, save und reset erleichtern das Hinzufügen bestimmter Interaktionsarten zwischen Plots. Hovertool wird verwendet, um Tooltips anzuzeigen, wenn der Mauszeiger über bestimmte Bereiche des Plots bewegt wird. Legenden, die zu Bokeh-Plots hinzugefügt werden, sind interaktiv, d. h. auf die Legendeneinträge zu klicken oder zu tippen, blendet die entsprechende Glyphe in einem Plot aus. Die Widgets von Bokeh bieten eine Reihe von interaktiven Funktionen, um eine Front-End-Benutzeroberfläche für eine Visualisierung bereitzustellen. Zur Verarbeitung dieser Interaktionen werden Java-Script-Callbacks verwendet. Es gibt einen Color-Picker zur Auswahl einer Farbe für das Diagramm, ein Div-Element zur Anzeige von Text, das HTML unterstützt, einen Toggle-Button, um ein bestimmtes Diagramm ein- oder auszublenden, und ein Range-Tool, um den Bereich eines anderen Diagramms zu steuern. Bokeh kann Bilder im RGBA-Format für Portable Network Graphics (PNG) mit der Funktion export_png() und Bilder für Scalable Vector Graphics (SVG) mit der Funktion export_svg() aus Layouts erzeugen [7]. Bokeh bietet Werkzeuge zur Erstellung komplexer, interaktiver und webfähiger Diagramme. Die Bibliothek ist gut dokumentiert und ermöglicht es, das Aussehen des Diagramms mit Bokeh-Themen anzupassen.

Darüber hinaus können Legenden, Texte und Anmerkungen hinzugefügt sowie Achsen, Gitter und Symbolleisten angepasst werden.

5.5 Daten Visualisierung mit Plotly

5.5.1 Liniendiagramm

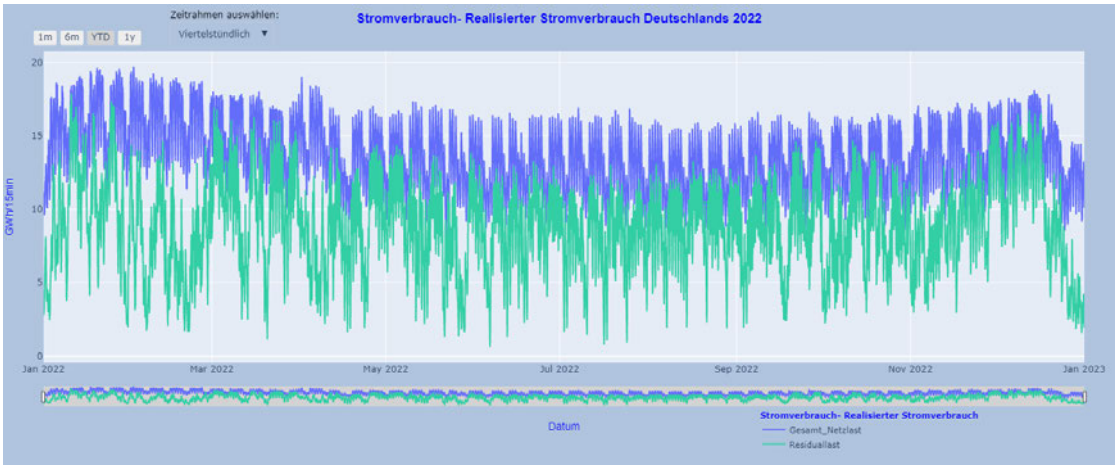

Abbildung 5.47: Liniendiagramm mit Plotly

Die Daten für 2020 werden mit Plotly visualisiert. Der Bereichsschieberegler ermöglicht die Interaktion mit dem Diagramm, um Daten eines bestimmten Wertebereichs zu filtern. Die Datenpunkte können mit der Maus überfahren und die Darstellung kann gezoomt, geschwenkt und zurückgesetzt werden. Mit Hilfe des Dropdown-Menüs können auch Viertelstundenwerte, Stundenwerte und Wochenwerte für die Trendanzeige ausgewählt werden.

5.5.2 Balkendiagramm

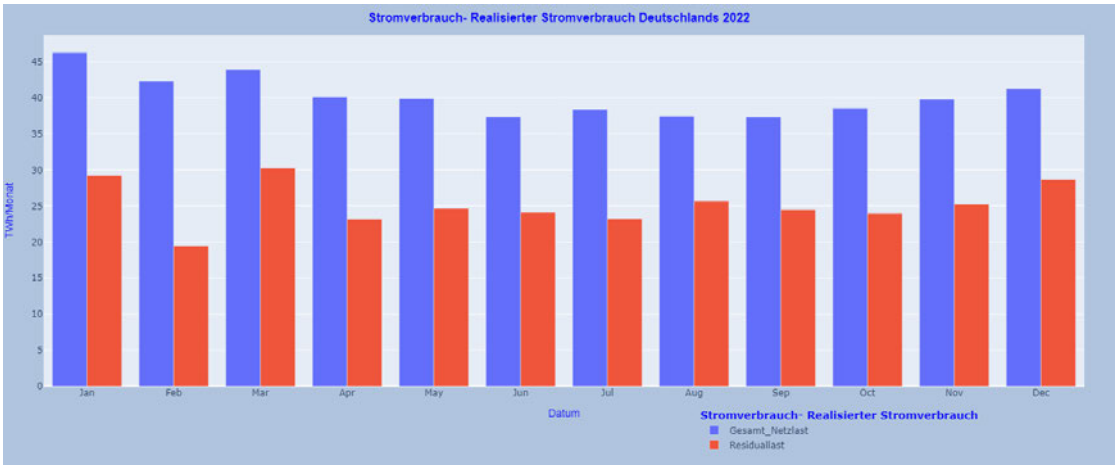

Abbildung 5.48: Balkendiagramm mit Plotly

5.5.3 Boxplot

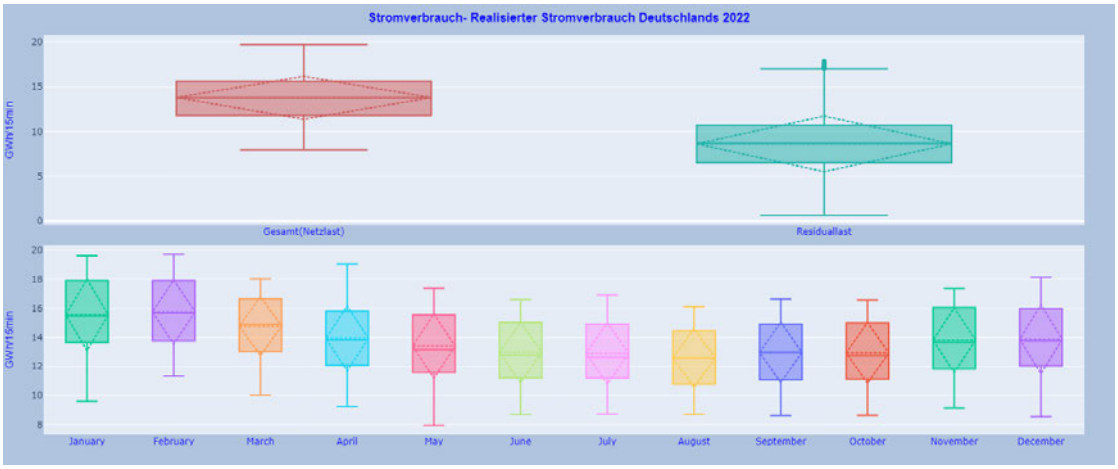

Abbildung 5.49: Boxplot mit Plotly

Wie bei den statischen Diagrammen sind Median, Mittelwert, oberes/unteres Quartil, Maximum/Minimum und Ausreißer sichtbar. Beim Überfahren mit der Maus werden die entsprechenden Werte angezeigt.

5.5.4 Kreisdiagramm

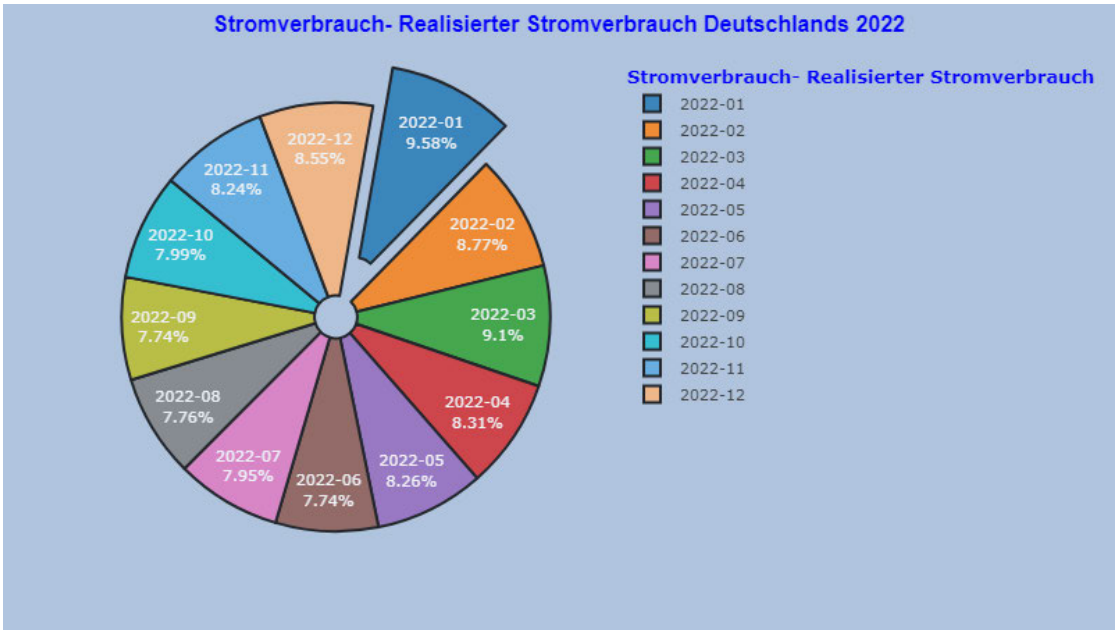

Abbildung 5.50: Kreisdiagramm mit Plotly

5.5.5 Flächendiagramm

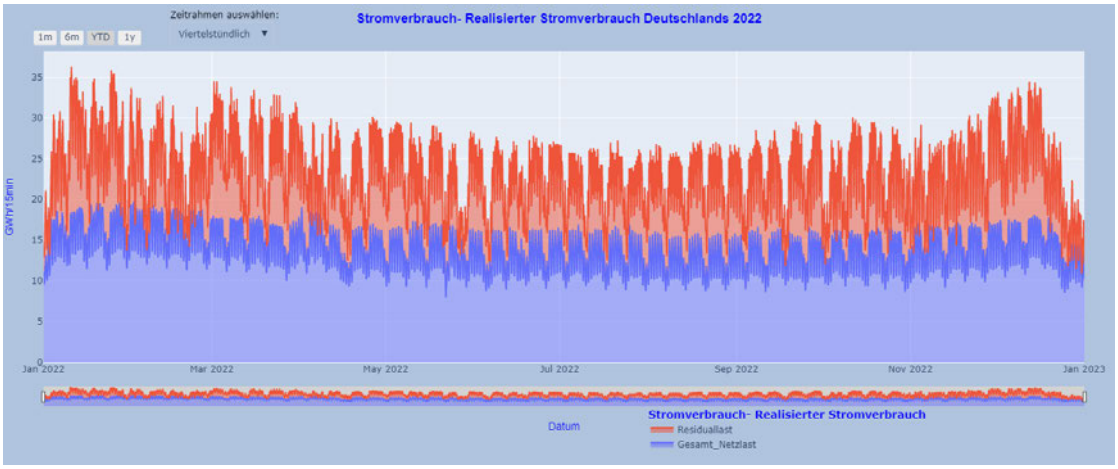

Abbildung 5.51: Flächendiagramm mit Plotly

Ähnlich wie beim Liniendiagramm gibt es verschiedene Funktionen, um mit den Daten zu interagieren, z. B. das Hovern über die Daten, den Bereichsschieber, das Dropdown-Menü und benutzerdefinierte Schaltflächen als Selektoren.

5.5.6 Histogramm

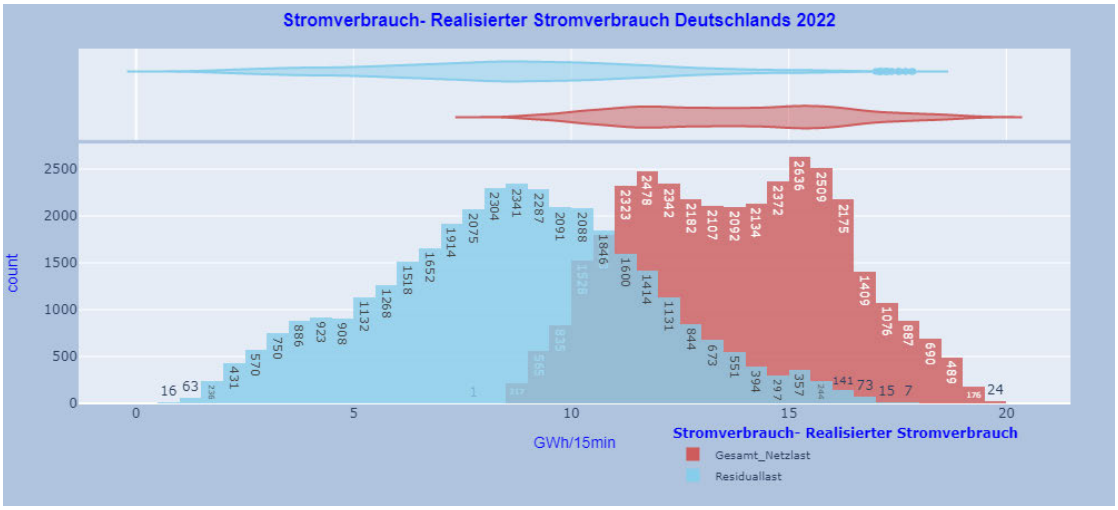

Abbildung 5.52: Histogramm mit Plotly

Das Histogramm zeigt die Häufigkeitsverteilung der Gesamtnetz- und Residuallast, wobei die Anzahl in den Bins sichtbar ist. Zur Darstellung der Wahrscheinlichkeitsdichte der Daten bei verschiedenen Werten wird ein Violindiagramm verwendet. Beim Hovern über die Daten werden die Größe des Bins und die Häufigkeit des Wertes im Bin angezeigt.

5.5.7 Violinplot

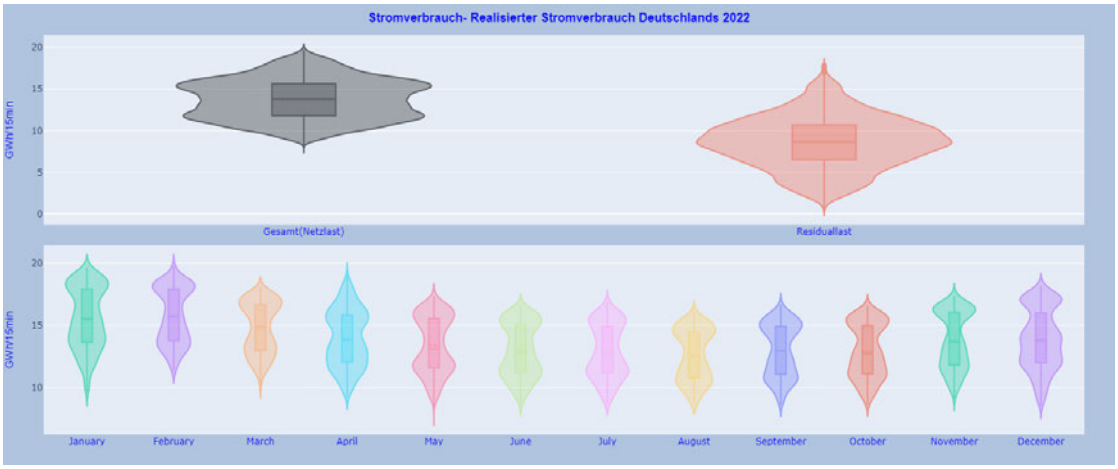

Abbildung 5.53: Violinplot mit Plotly

Boxplot und eine gedrehte Kerneldichte auf jeder Seite, die die Wahrscheinlichkeitsdichte der Daten bei verschiedenen Werten zeigt. Beim Hovern über die Daten werden der Interquartilsbereich, das Maximum, das Minimum, der Mittelwert, der Median und der Ausreißer angezeigt.

5.5.8 Wärmekarte

Betrachtung variante 1

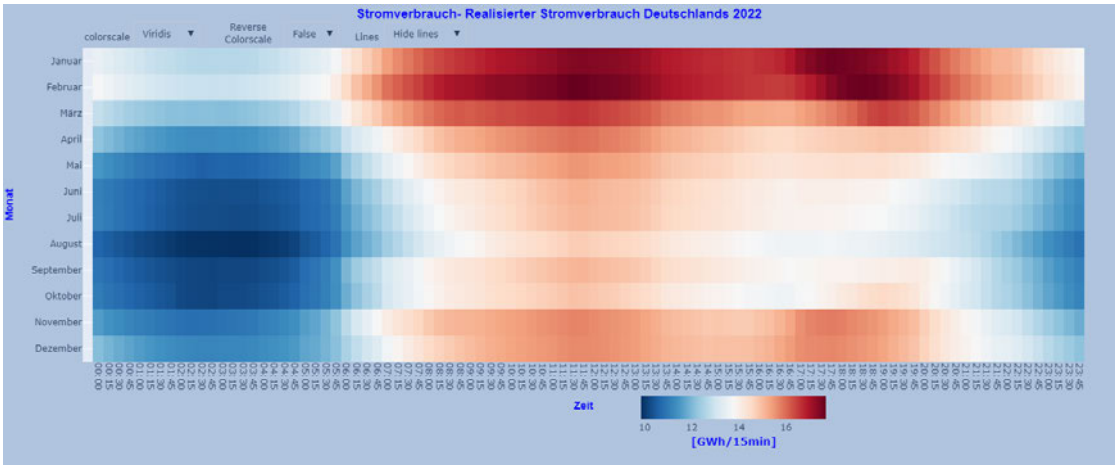

Abbildung 5.54: Heatmap Betrachtung Variante 1 mit Plotly

Diese Heatmap zeigt den durchschnittlichen Stromverbrauch über die Stunden des Tages für jeden Monat. Mit dem Mauszeiger kann über die Heatmap interagiert werden, um Informationen zu erhalten. Es wird auch gezeigt, wie mehrere Datenattribute aktualisiert werden können: Farbskala, Richtung der Farbskala und Liniendarstellung.

Betrachtung variante 2

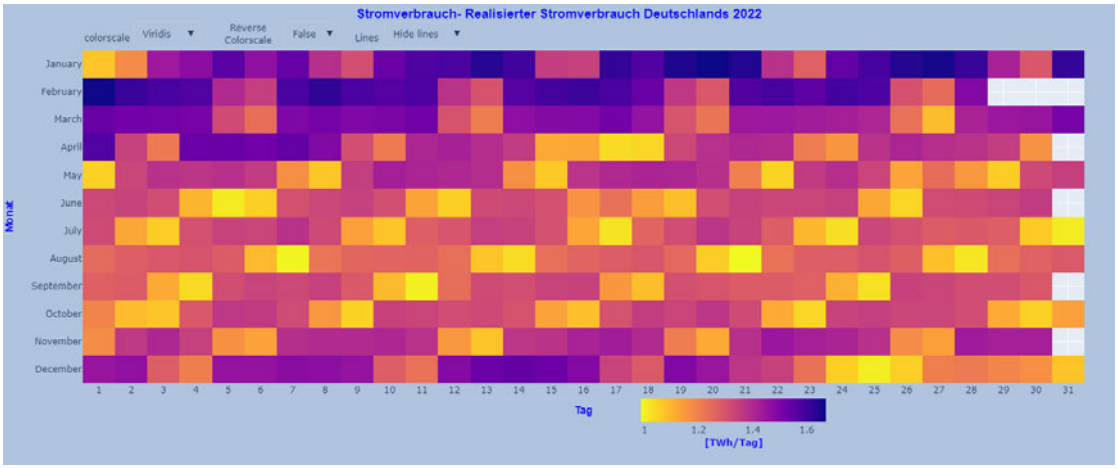

Abbildung 5.55: Heatmap Betrachtung Variante 2 mit Plotly

Betrachtung variante 3

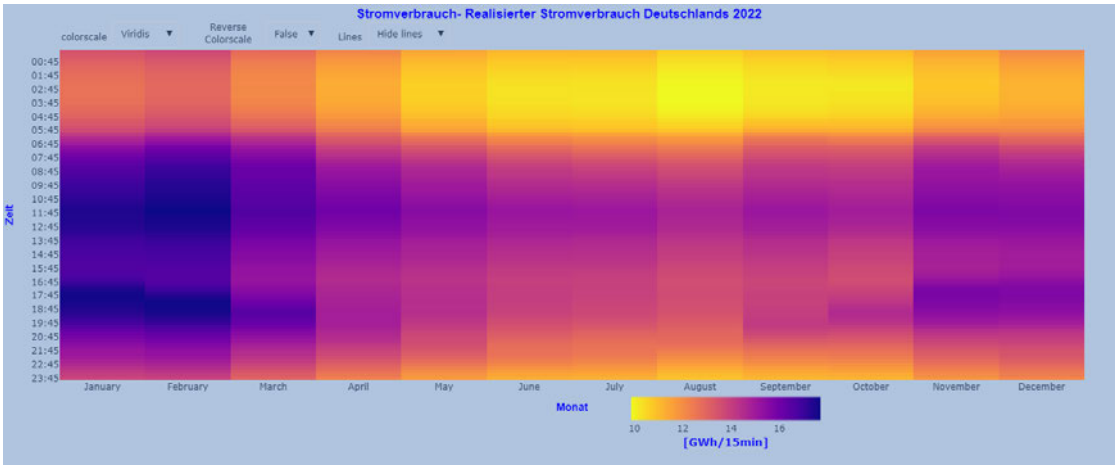

Abbildung 5.56: Heatmap Betrachtung Variante 3 mit Plotly

Betrachtung variante 4

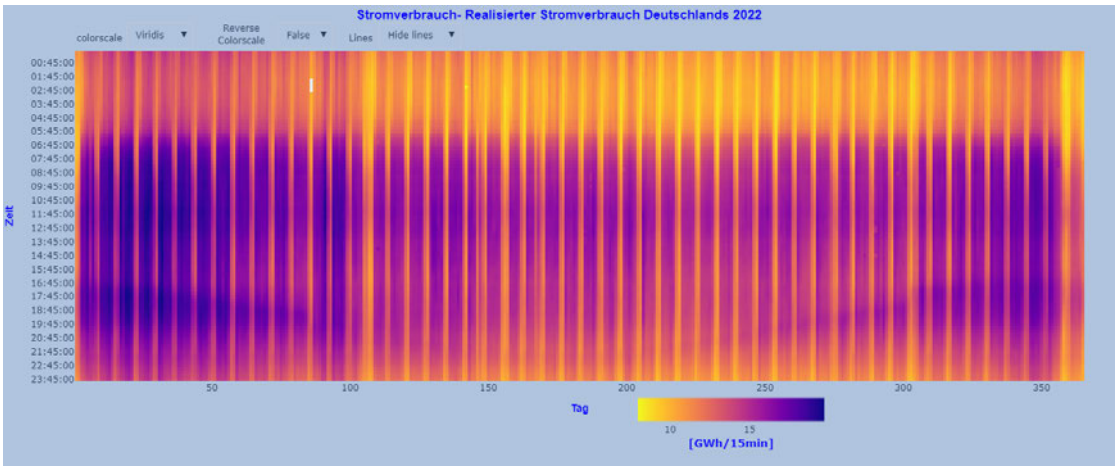

Abbildung 5.57: Heatmap Betrachtung Variante 4 mit Plotly

Plotly basiert auf der Plotly-JavaScript-Bibliothek (plotly.js) und ermöglicht es Python Benutzern, ansprechende interaktive Webvisualisierungen zu erstellen, die in Jupyter-Notizbüchern angezeigt, als eigenständige HTML-Dateien gespeichert oder als Teil von reinen Python-Webanwendungen mit Dash bereitgestellt werden können. Die Kernfunktionalität ist in JavaScript geschrieben und bietet Anbindungen an verschiedene Sprachen wie Python, R, Julia, JavaScript, ggplot2, F# und Matlab. Mit Plotly werden alle Diagramme erstellt, die im Rahmen der vorliegenden Arbeit benötigt werden. Plotly bietet

eine vollständige Dokumentation für alle Diagramme. Dies erhöht die Benutzerfreundlichkeit. Standardmäßig bietet Plotly grundlegende Interaktionsfunktionen wie die Auswahl von Intervallen zur Auswahl von Diagrammelementen durch Klicken und Ziehen, Zoom, Pan, Autoscale, Hover-Informationen und eine interaktive Legende zur Anzeige der ausgewählten Variablen. Der Bereichsschieber ermöglicht die Auswahl eines Wertebereichs innerhalb eines definierten Minimal- und Maximalbereichs. Der Bereichsselektor ist ein Werkzeug zur Auswahl der im Diagramm darzustellenden Bereiche. Es stehen Schaltflächen zur Verfügung, um vorkonfigurierte Bereiche im Diagramm auszuwählen. Ein Dropdown-Menü für die Auswahl von Viertelstunden-, Stunden- und Wochenwerten im Liniendiagramm wurde hinzugefügt, damit Trend und Saisonalität erkannt und verschiedene Datenattribute wie Farbskala, Richtung der Farbskala und Linienanzeige in Heatmaps aktualisiert werden können [36].

5.6 Sankey Diagramm

Sankey-Diagramme sind eine spezielle Art von Flussdiagrammen, bei denen die Breite der Pfeile proportional zum Fluss ist. Sie werden normalerweise zur Visualisierung von Energieübertragungen zwischen Prozessen verwendet. Für das Verständnis der Sankey-Diagramme und der Flüsse sind Quelle (Ausgangsknoten), Ziel (zu dem die Quelle eine Verbindung herstellt) und der Wert für das Volumen des Verbindungsflusses grundlegend.

Der Eurostat-Datensatz für das Sankey-Diagramm enthält die Energiebilanzflüsse für verschiedene Energieträger wie feste Brennstoffe, erneuerbare Energien usw. In dieser Arbeit wird nur der Fluss der festen Brennstoffe betrachtet. Um den Durchfluss in einem Datensatz darzustellen, sind manuelle Eingaben erforderlich. Es müssen der Quellknoten, der Zielknoten, sowie ein Wert zur Darstellung des Durchflussvolumens definiert werden. Der Datensatz für feste Brennstoffe ist komplex und erfordert eine Vorverarbeitung der Daten. Die visuelle Darstellung des Sankey-Plots auf der Webseite von Eurostat dient zur Bestimmung der Quelle und des Ziels. Diese werden anschließend aus der Excel-Datei gefiltert und ihr jeweiliger Wert wird im Code zugewiesen. Die Beschriftung sowie die Farbe der Knoten helfen bei der Differenzierung zwischen Knoten- und Quellennamen. Im Vergleich zu anderen Visualisierungstypen, bei denen die eingebauten Funktionen die wesentliche Grundlage bilden, erfordert die Erarbeitung des Sankey-Plots einen beträchtlichen zeitlichen Aufwand. Die sorgfältige Definition der Quelle und des Ziels sowie die

Suche des Wertes in einer komplexen Excel-Tabelle sind zeitintensiv und manuell durchzuführen. Im Vergleich zu anderen Bibliotheken für die Datenvisualisierung bietet Plotly ausführliche Dokumentation und hilfreiche Beispiele zur Erstellung komplexer Sankey-Diagramme, Aus diesem Grund ist Plotly die stärkste Bibliothek für die Darstellung von Sankeydiagrammen [37].

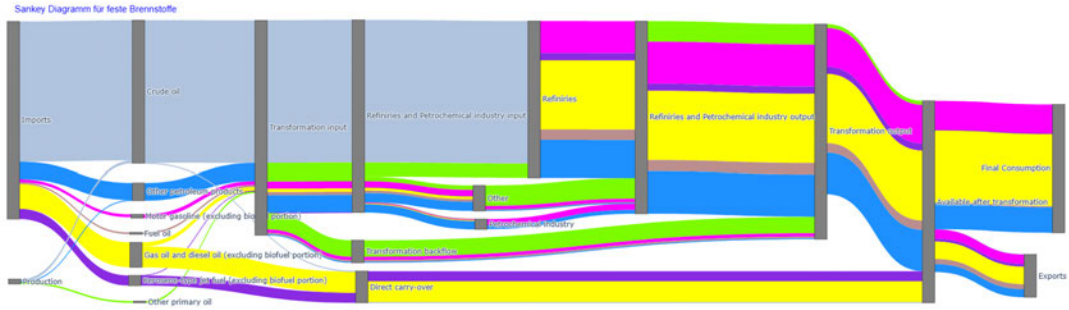

Abbildung 5.58: Sankeydiagramm für feste Brennstoff

Die Abbildung 5.58 zeigt das Energieflussdiagramm für feste Brennstoffe. Es lässt sich feststellen, dass ein erheblicher Teil dieser Brennstoffe importiert wird, wobei Erdöl den größten Anteil ausmacht. Es zeigt auch die Umwandlungs- und Raffinerieprozesse, die diese Brennstoffe durchlaufen, bevor sie den Endverbraucher erreichen. Beim Hovern über das Diagramm können zusätzliche Informationen über Menge, Quelle und Ziel angezeigt werden. Anhand verschiedener Farben lassen sich die Art der Brenstoffe feststellen. Darüber hinaus ist es möglich, die Knoten zu verschieben. Um die Standardansicht zu optimieren, werden die Knoten verschoben und eine Bildschirmaufnahme zur Dokumentation gespeichert.

5.7 Dashboard Erstellung

Praktisch jede Python-Bibliothek kann verwendet werden, um eine statische PNG, SVG, HTML oder andere Ausgabe zu erzeugen, die in eine Präsentation eingefügt, per EMail verschickt oder als Abbildung in einem Dokument veröffentlicht werden kann. Es ist auch möglich, auf Python basierende Live-Anwendungen oder Dashboards zu erstellen, die Benutzer verwenden können, um Daten zu untersuchen oder zu analysieren. Python bietet mehrere Bibliotheken für die Erstellung webbasierter Dashboards. Die vier wichtigsten hierfür entwickelten Tools sind Dash, Streamlit, Panel und Voila. Das Dashboard mit

Dash und Streamlit wird im Rahmen dieser Arbeit implementiert, weil sie mehr Github-Sterne haben, was bedeutet, dass sie bei den Benutzern beliebt sind. Dash arbeitet auch mit Plotly zusammen und verbessert die Leistung dieser Bibliothek durch zusätzliche Funktionen. Streamlit ist strukturierter und auf Einfachheit ausgerichtet [18].

5.7.1 Dashboard mit Dash

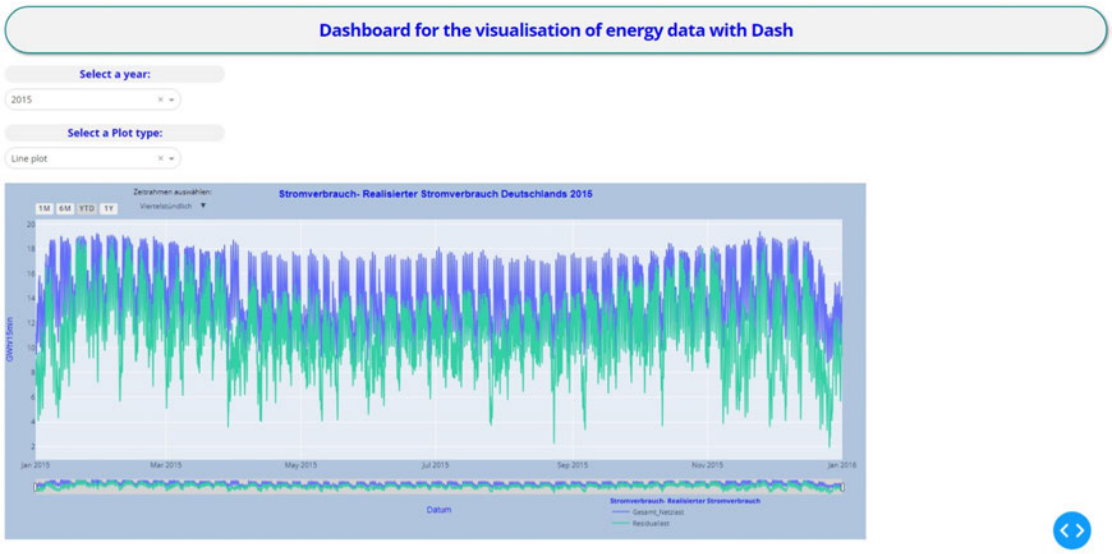

Abbildung 5.59: Dashboard mit Dash

Dash ist ein Low-Code-Framework, um schnelle Datenanwendungen in Python zu entwickeln. Es wird mit der Grafikbibliothek Plotly geliefert. Mit Dash kann eine Anwendung mit nur wenigen Zeilen Code erstellt werden. Dash-Anwendungen setzen sich aus zwei Teilen zusammen: dem Layout, das beschreibt, wie die Anwendung aussieht, und den Callbacks, die beschreiben, wie Dash-Anwendungen mit Hilfe von Callback-Funktionen erstellt werden. Das Layout besteht aus einer hierarchischen Baumstruktur von Komponenten. Dash HTML Components (dash.html) stellt Klassen für alle HTML-Tags zur Verfügung. Die Schlüsselwort-Argumente beschreiben die HTML-Attribute wie z. B. style, class und id. Dash-Core-Components (dash.dcc) erzeugen übergeordnete Komponenten wie Controls und Diagramme. Callback-Funktionen werden automatisch aufgerufen, wenn sich eine Eigenschaft einer Eingabekomponente ändert. Dadurch wird eine Eigenschaft in einer anderen Komponente (der Ausgabe) aktualisiert [2].

Nach dem Import der Pakete und der Initialisierung der Anwendung mit dem Dash-Konstruktor wird das Layout definiert, um die Komponenten der Anwendung darzustellen, die im Webbrowser angezeigt werden, normalerweise in einer html.div. Dash-HTML und Dash-Core-Components werden verwendet, um das Layout des Dashboards anzupassen. Die Pandas-Bibliothek wird verwendet, um ein CSV-Datenblatt in einen Pandas-Datenrahmen einzulesen und um die Daten in die Anwendung einzufügen. Alle Plots aus der Plotly-Bibliothek werden von Dash unterstützt. Um die Daten zu visualisieren, wird das DCC-Modul importiert und dcc.Graph verwendet, um interaktive Diagramme zu erstellen. Die Figure-Eigenschaft von dcc.Graph wird verwendet, um die Diagramme in unserer Anwendung anzuzeigen. Um dem Benutzer mehr Freiheit bei der Interaktion mit der Anwendung zu geben und die Daten detaillierter zu untersuchen, werden zwei Dropdown-Menüs implementiert. Diese ermöglichen die Auswahl des Jahres und des Visualisierungstyps. Um dies zu implementieren, muss die Callback-Funktion zur Anwendung hinzugefügt werden. Zuerst werden zwei Dropdown-Komponenten hinzugefügt, um dem Benutzer ein erweiterbares Dropdown-Menü zu bieten. Dann werden das Callback-Modul und die beiden Argumente Output und Input importiert, die normalerweise im Callback verwendet werden, um die Interaktion zwischen den Dropdowns und der Graph-Komponente zu ermöglichen. Beide erhalten eine ID, die vom Callback zur Identifizierung der Komponenten verwendet wird. Jedes Mal, wenn der Benutzer ein anderes Jahr oder eine andere Darstellung auswählt, wird der Graph aktualisiert.

5.7.2 Dashboard mit Streamlit

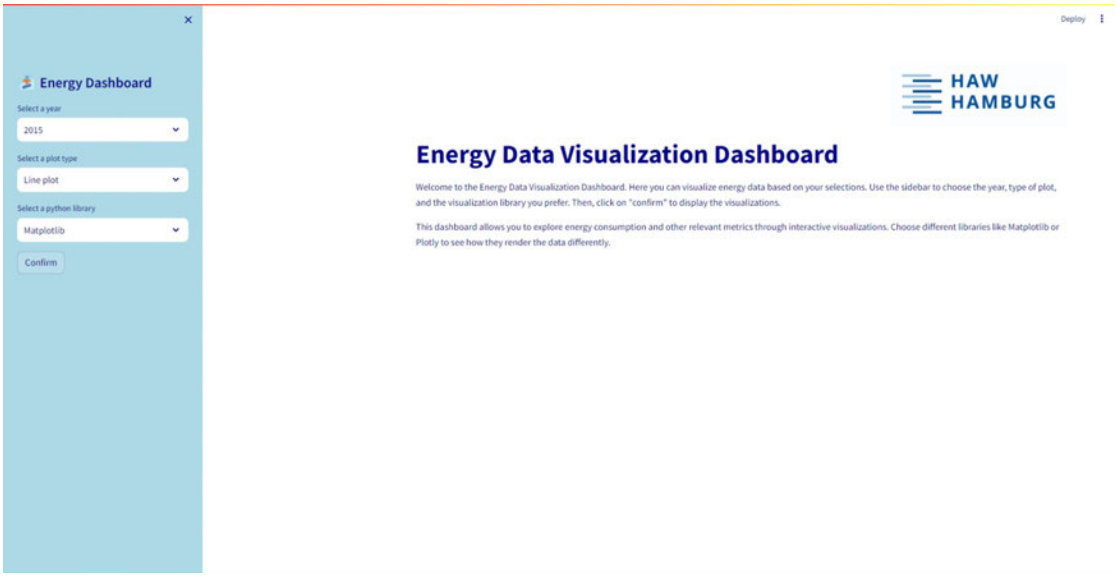

Abbildung 5.60: Dashboard mit Streamlit

Streamlit ist eine Open-Source-Python-Bibliothek, mit der benutzerdefinierte Webanwendungen einfach erstellt und gemeinsam genutzt werden können. Nach der Ausführung des Skripts wird lokal ein Streamlit-Server gestartet, der die Anwendung in einem neuen Tab des Webbrowsers öffnet. Es können Diagramme, Text, Widgets und Tabellen erstellt werden. Für weitere Informationen wird auf die API-Dokumentation verwiesen. Die Anwendung in Abbildung 5.60 verwendet st.write zur Anzeige von Argumenten. Verschiedene andere Textelemente wie z. B. st.markdown werden für die Anzeige von Strings im Markdown-Format verwendet. st.title wird für die Anzeige von Text im Titelformat verwendet. Datenelemente wie st.dataframe werden verwendet, um einen Datensatz als interaktives Element darzustellen. Streamlit unterstützt alle unsere Python-Bibliotheken, nämlich Matplotlib, Seaborn, Plotly, Bokeh und Vega-Altair. Aus diesem Grund können alle implementierten Visualisierungen von jeweiligen Bibliotheken in Streamlit angezeigt werden. Eingabewidgets wie Buttons und Selectboxen sind implementiert. Sie ermöglichen die Auswahl mehrerer Jahre an Datensätzen sowie verschiedener Visualisierungstypen und -bibliotheken zur Visualisierung der Daten [43].

5.8 Karten Visualisierung

5.8.1 Karten mit Folium

Folium ist eine leistungsstarke Python-Bibliothek, mit der verschiedene Arten von Leaflet-Karten erstellt werden können. Folium nutzt die Stärken des Python-Ökosystems bei der Datenverarbeitung und der Leaflet.js-Bibliothek beim Mapping. Die Bibliothek bietet eine Vielzahl von integrierten Tilesets von OpenStreetMap, Mapbox und Stamen und unterstützt benutzerdefinierte Tilesets mit Mapbox- oder Cloudmade-API-Schlüsseln. Folium unterstützt die Verwendung von GeoJSON- und TopoJSON-Overlays sowie die Bindung von Daten an diese Overlays, um Choroplethen-Karten mit Farbschemata zu erstellen und die Platzierung von Markern auf der Karte. Die interaktiven Ergebnisse machen diese Bibliothek besonders nützlich für die Erstellung von Dashboards [39].

Elektroladestationen in Deutschland werden auf der Karte visualisiert. Nach Installation und Import von Folium kann mit nur einer Zeile Code eine einfache Beispielkarte erstellt werden, indem ein Tileset wie z. B. OpenStreetMap ausgewählt wird und Marker mit Popup- und Tooltip-Funktionalität hinzugefügt werden. Außerdem können Choroplethen durch die Verknüpfung von Daten zwischen Pandas DataFrames/Series und Geo/TopoJSON-Geometrien erstellt werden. Folgende Funktionen sind implementiert: Skalierung am unteren Kartenrand, Zoom-Steuerung, Setzen von Grenzen, so dass die Karte nicht über diese Grenzen hinausscrollt. UI-Elemente wie die LayerControl zum Ein- und Ausblenden von Layern sowie einfache und HTML-Popups zur Anzeige von Informationen wurden der Karte hinzugefügt. Vektorebenen wie Polygone werden verwendet, um die Grenzen von Bundesländern und Landkreisen zu markieren, die in einer Geometriespalte unserer JSON-Datei enthalten sind. Die Groupby-Operation gruppiert alle Ladestationen nach Bundesland und zählt, wie viele Ladestationen jedes Bundesland hat. Für Geojson-Popup und Tooltip werden json und csv zusammengeführt, sodass die Geometriespalte, in der das Polygon für das Bundesland steht, eine gemeinsame Spalte mit dem Namen des Bundeslandes und der Zählspalte für die Anzahl der Ladestationen in jedem Bundesland enthält. Es sind Funktionen wie FitOverlays implementiert, die das Schwenken und Zoomen ermöglichen, um die Overlays anzuzeigen. Außerdem sind Plugins wie Draw, Minimap, Fullscreen, Mouseposition, ScrollZoomToggler, MarkerCluster und Search implementiert, um zusätzliche Interaktion zu ermöglichen. Folium verfügt nicht über eine integrierte Methode zum Speichern einer Karte als PNG-Datei. Stattdes-

sen wird die Karte als HTML-Datei gespeichert und anschließend ein Bildschirmfoto als PNG-Datei gespeichert [40].

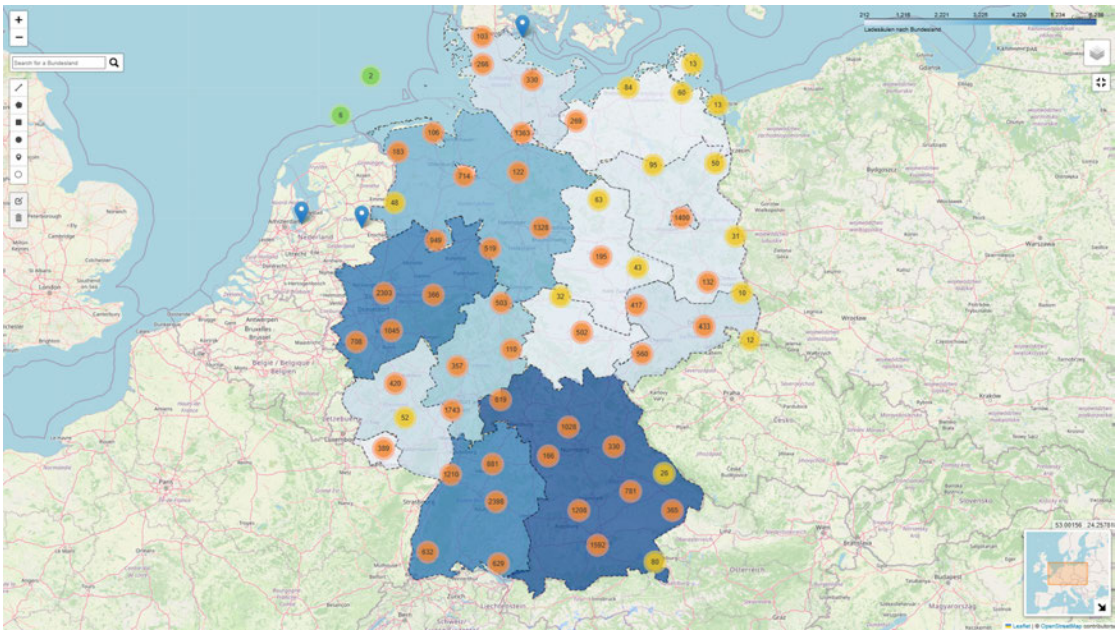

Abbildung 5.61: Kartenvisualisierung auf Bundeslandebene

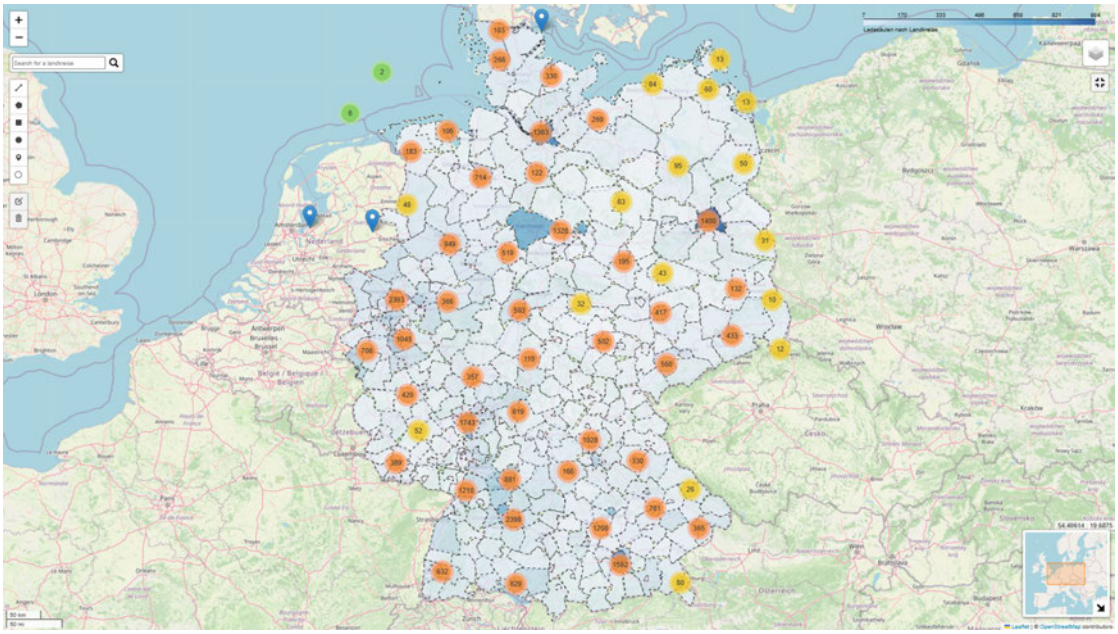

Abbildung 5.62: Kartenvisualisierung auf Landkreisebene

Abbildung 5.61 zeigt die Visualisierung von Elektroladesäulen auf Bundeslandebene und Abbildung 5.62 auf Landkreisebene. Alle oben genannten interaktiven Funktionen sind integriert. Einige Punkte liegen außerhalb Deutschlands. Dies liegt daran, dass die Längen- und Breitengrade in der CSV-Datei der Bundesnetzagentur nicht korrekt sind. Bei der Überprüfung hat sich gezeigt, dass die Elektroladesäule zwar existiert, aber nicht an den geographischen Koordinaten. Choroplethmap zeigt die Visualisierung, bei der die Bundesländer oder Landkreise im Verhältnis zur Anzahl der Ladesäuledichte eingefärbt sind. Beim Hovern über die Bundesländer/Landkreise und Anklicken werden die Informationen zum Bundesland bzw. Landkreis und die Anzahl der Ladesäulen angezeigt. Ein Marker mit Popup- und Tooltip-Funktion zeigt den Standort der Ladesäulen an. Durch die Verwendung von Marker-Clustern werden die Marker je nach Zoom-Stufe angezeigt, was die Leistung und Nutzbarkeit von Karten mit einer großen Anzahl von Markern verbessert. Die Darstellung erfolgt dynamisch, d. h. beim Herauszoomen werden die Cluster aufgelöst, um einzelne Marker oder kleinere Cluster anzuzeigen, und beim Hineinzoomen wieder zu Clustern zusammengeführt. Die drei Bundesländer mit der höchsten Dichte an Elektroladesäulen sind Bayern, Nordrhein-Westfalen und Baden-Württemberg. Die acht Landkreise mit der größten Anzahl an Elektroladestationen sind Berlin, Hamburg, München, Hannover, Wolfsburg, Stuttgart, Groß-Gerau und Ingolstadt.

5.8.2 Karten mit Geopandas

GeoPandas erweitert die Data-Science-Bibliothek pandas um die Unterstützung von Geodaten. Die zentrale Datenstruktur in GeoPandas ist der geopandas.GeoDataFrame, eine Unterklasse von pandas.DataFrame, der Geometriespalten speichert und spatiale Operationen ausführen kann. Der geopandas.GeoSeries ist eine Unterklasse von pandas.Series. Er verarbeitet die Geometrien. Ein GeoDataFrame ist eine Kombination aus einem pandas.Series mit traditionellen Daten (numerisch, boolesch, textuell usw.) und einem geopandas.GeoSeries mit Geometrien (Punkte, Polygone usw.) [21].

Die GeoJSON-Daten enthalten sowohl Daten als auch Geometrie. Diese können mit der Funktion geopandas.read_file() gelesen werden. Die Funktion erkennt automatisch den Dateityp und erzeugt einen GeoDataFrame. Anschließend muss der GeoDataFrame mit der Funktion GeoDataFrame.to_file() wieder zurück in die Datei geschrieben werden. GeoPandas erleichtert die Erstellung von Choroplethenkarten. Dazu wird einfach der Befehl ‚plot' mit dem Argument ‚column' verwendet, das die Spalte angibt, deren Werte für die Farbzuweisung verwendet werden sollen. Sie können entweder eine statische Karte

mit mehreren Ebenen erstellen (Choropleth und Ladestationen als Punkte) oder eine interaktive Karte mit der Methode explore(). Die interaktive Darstellung bietet weitgehend die gleichen Anpassungsmöglichkeiten wie die statische und darüber hinaus noch einige zusätzliche Funktionen. Für das Choropleth wird die Spalte NAME_1 mit den Namen der Bundesländer als Eingabe verwendet und beim Überfahren mit dem Mauszeiger nur der Name im Tooltip angezeigt, beim Anklicken aber alle Werte. Die Methode explore() gibt ein folium.Map-Objekt zurück, auf dem die Folium-Funktionalität direkt genutzt werden kann, indem zwei GeoDataFrames auf derselben Karte dargestellt werden und Folium-Features hinzugefügt werden.

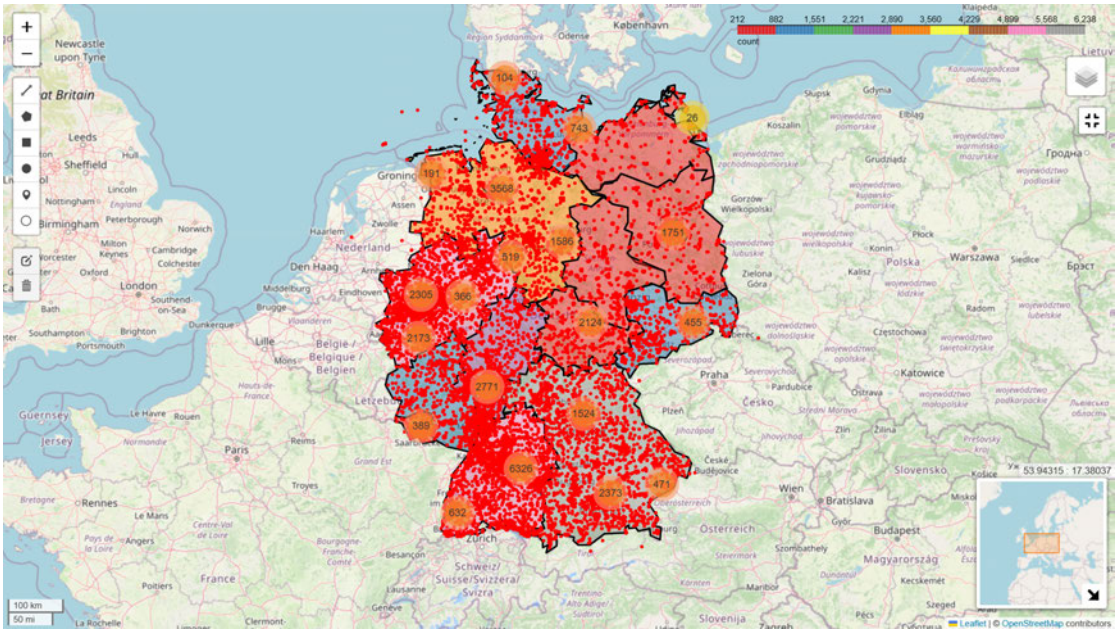

Abbildung 5.63: Kartenvisualisierung auf Bundeslandebene mit Geopandas

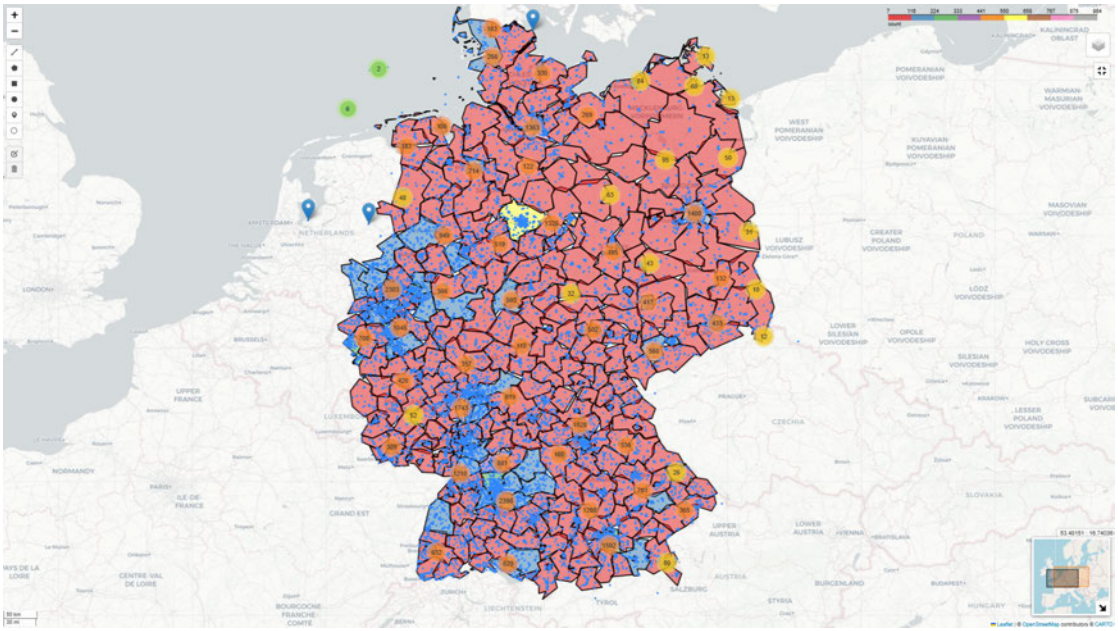

Abbildung 5.64: Kartenvisualisierung auf Landkreisebene mit Geopandas

Bei der Kartenvisualisierung mit Geopandas sind die Funktionen von Folium ebenfalls implementiert. Im Gegensatz zu Folium müssen Popups und Tooltips nicht programmiert werden, sondern werden standardmäßig angezeigt. Wichtige interaktive Funktionen, wie z. B. das Hovern über die Karte, um weitere Informationen zu erhalten, sowie das Hinein- und Herauszoomen, um einzelne Marker oder kleinere Cluster anzuzeigen oder zusammenzuführen, sind ebenfalls implementiert.

5.8.3 Karten mit Plotly

Die Python-Grafikbibliothek Plotly ermöglicht, interaktive Karten in Publikationsqualität online zu erstellen. Zur Erstellung von Choropleth-Karten mit Plotly sind zwei Arten von Eingaben erforderlich: GeoJSON-formatierte Geometriedaten, bei denen jedes Feature entweder ein id-Feld oder einen identifizierenden Wert in den Eigenschaften hat, sowie eine Liste von Werten, die durch den Feature-Identifier indiziert sind. Die GeoJSON-Daten werden dem geojson-Argument übergeben und die Daten werden in derselben Reihenfolge wie die IDs im location-Argument in das color-Argument von px.choropleth_mapbox übergeben [35][38]. Zuerst werden die GeoJSON-Datei und die CSV-Datei für die Ladesäulen geladen. Diese enthalten die Geometrieinformationen für

die Bundesländer und die Landkreise sowie die Längen- und Breitengrade für die Ladesäulen. Wie bei anderen Bibliotheken müssen die Bundesländer gruppiert und die Anzahl der Ladestationen gezählt werden. Die Funktion px.choropleth_mapbox stellt dann jede Zeile des DataFrames als eine Region des Choropleths dar.

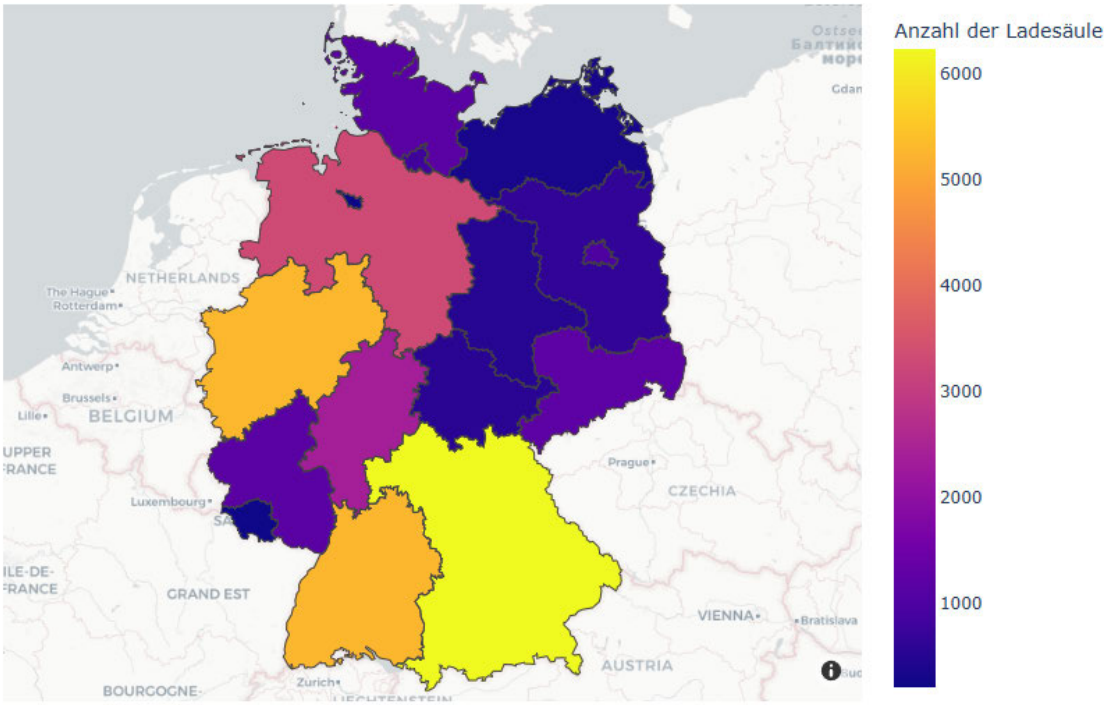

Abbildung 5.65: Choropleth Karten auf Bundeslandebene mit Plotly

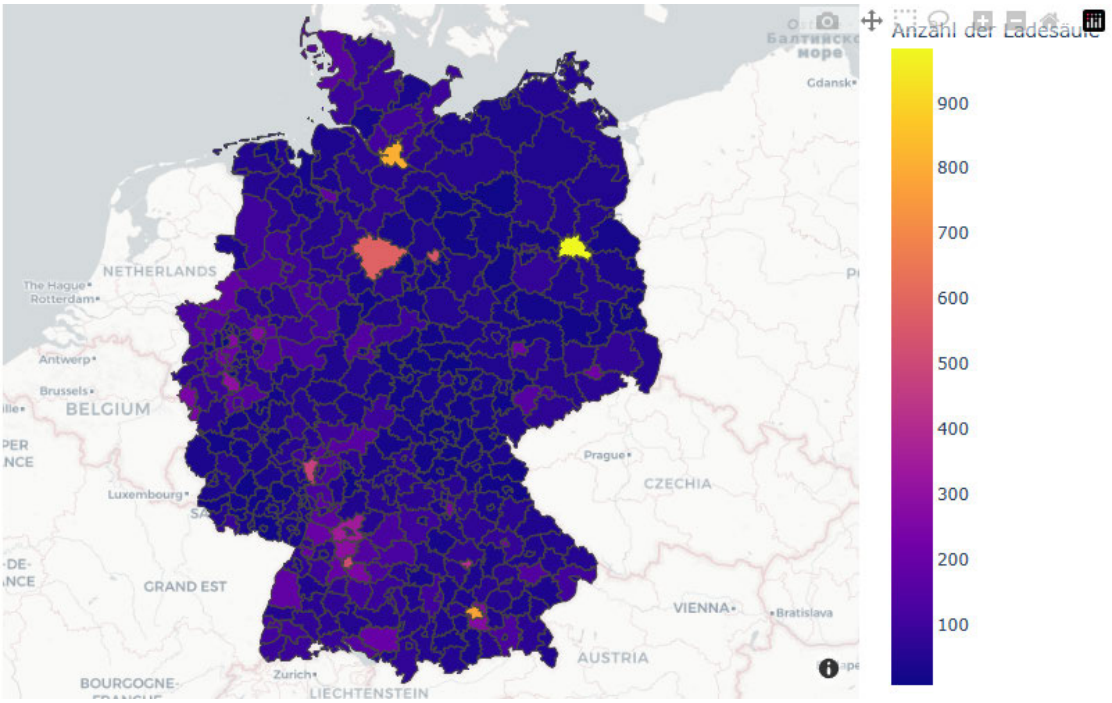

Abbildung 5.66: Choropleth Karten auf Landkreisebene mit Plotly

5.8.4 Zeitreihenanalyse

Eine Zeitreihe ist eine Datensammlung, die regelmäßig über einen bestimmten Zeitraum erhoben wird. Die Daten von SMARD.de werden von 2015 bis 2022 alle 15 Minuten als Zeitreihe erhoben. Bei der Analyse von Zeitreihendaten ist es wichtig, die beiden grundlegenden Konzepte Trend und Saisonalität zu verstehen. Ein Trend liegt vor, wenn die Daten langfristig zunehmen oder abnehmen. Saisonalität liegt vor, wenn die Zeitreihe ein regelmäßiges Muster aufweist, das mit dem Kalender zusammenhängt, z. B. ein tägliches, wöchentliches oder jährliches Muster. Wenn das Verhalten einer Zeitreihe periodisch vom Kalender beeinflusst wird, spricht man von Saisonalität.

Der erste Schritt jeder Datenanalyse ist die Darstellung der Daten. Dazu importieren wir den Datensatz in einen Pandas-Datenrahmen, benennen die Spalten um und verknüpfen alle Jahresdateien mit pandas.concat() zu einem einzigen Datenrahmen. Dann werden alle Spalten angezeigt, um ihre Kurven gleichzeitig zu betrachten. Die Abbildung 5.67 zeigt die Visualisierungen aller Spalten des Datensatzes.

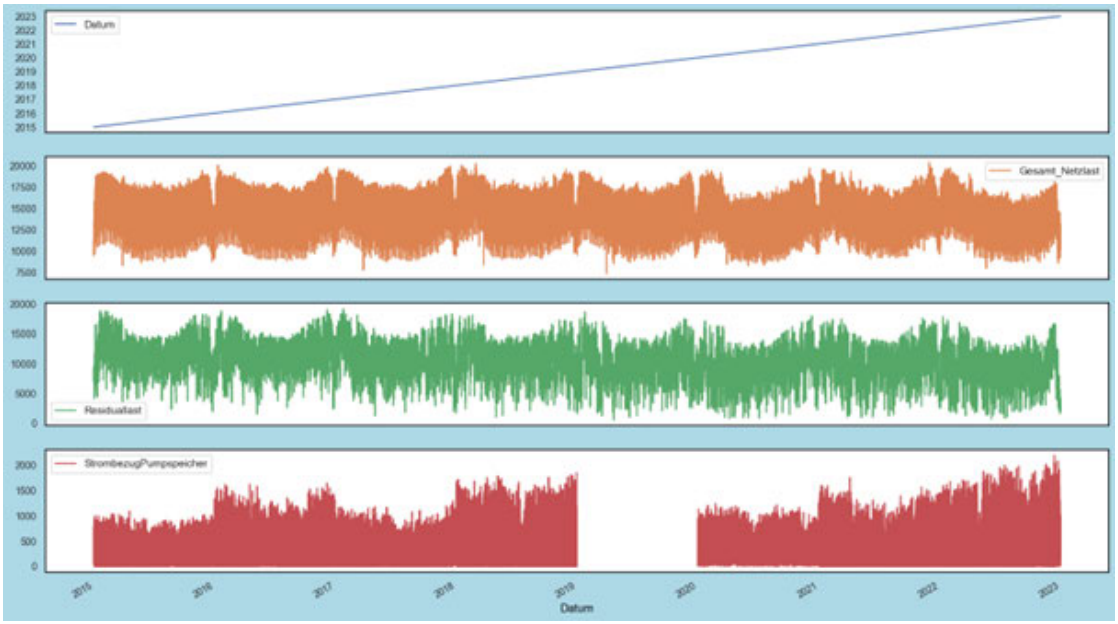

Abbildung 5.67: Visualisierung aller Spalten des Datensatzes

Die Gesamtnetzlastdaten im Zeitverlauf werden für die weitere Analyse betrachtet. Zeitreihendaten sind manchmal nicht im gewünschten Format verfügbar. In diesem Fall können grundlegende Zeitreihenoperationen wie Resampling, Shifting, Rolling und Differenzbildung in Pandas verwendet werden. Resampling ändert die Aggregationsebene einer Zeitreihe. In dieser Arbeit wird es verwendet, um 15-Minuten-Datensätze in tägliche, wöchentliche und monatliche Gesamtwerte umzuwandeln. Die Abbildung 5.68 zeigt die Daten für die Gesamtnetzlast nach dem Resampling. Shifting ist eine Technik, bei der die gesamte Reihe um eine bestimmte Anzahl von Perioden nach oben oder unten verschoben wird. Diese Technik ist nützlich, um die Autokorrelation der Daten mit Ihrem verzögerten Wert zu berechnen. Das Konzept der gleitenden Durchschnitte ist eine nützliche Technik zur Glättung von Zeitreihendaten. In Abbildung 5.69 sind die täglichen, wöchentlichen, monatlichen und jährlichen gleitenden Durchschnitte für unsere Daten dargestellt. Schließlich wird die Differenzierung verwendet, um die Zeitreihen stationär zu machen, was für die Erstellung von Prognosemodellen unerlässlich ist.

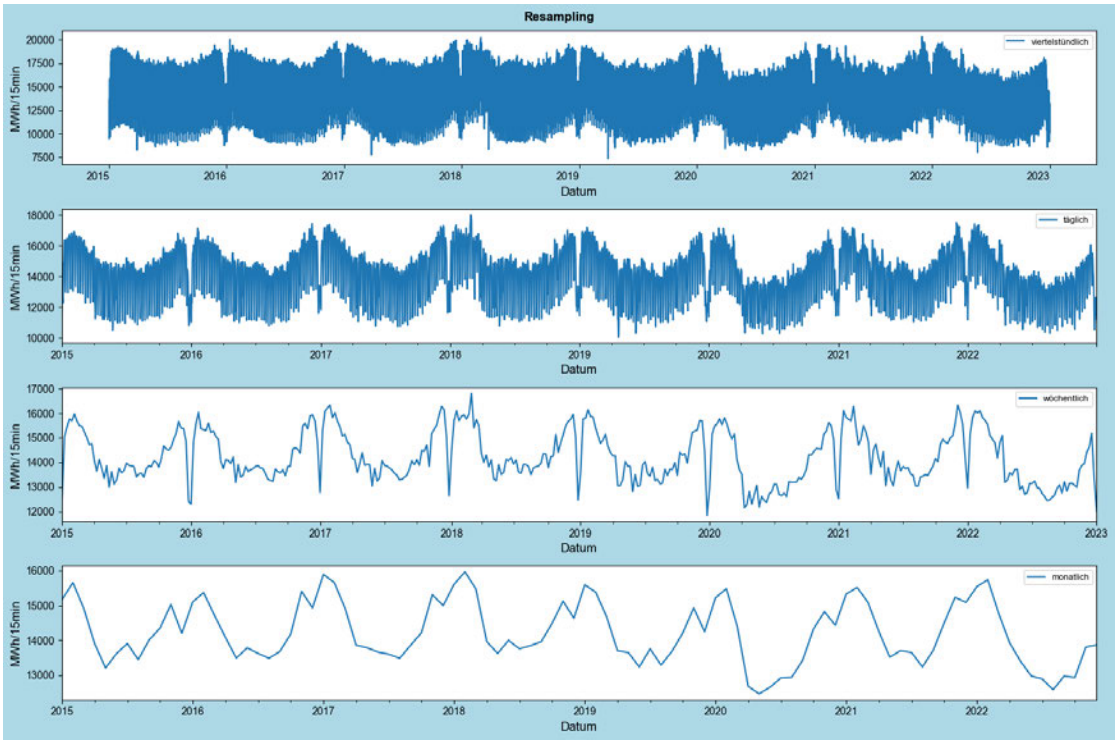

Abbildung 5.68: Resampling

Die Abbildung 5.68 zeigt die aggregierte Gesamtnetzlast auf verschiedenen Zeitebenen. Die viertelstündliche Aggregationsebene ist aufgrund der großen Anzahl von Datenpunkten sehr dicht und die Informationen sind schwer zu erkennen. Auf der täglichen Aggregationsebene sind die Tageshöchst- und -tiefstwerte erkennbar. Auf der wöchentlichen Ebene zeigt sich ein deutlicher Aufwärtstrend des Stromverbrauchs in den Wintermonaten und ein Abwärtstrend in den Sommermonaten. Darüber hinaus ist über die Jahre hinweg die Saisonalität des Stromverbrauchs sehr gut zu erkennen.

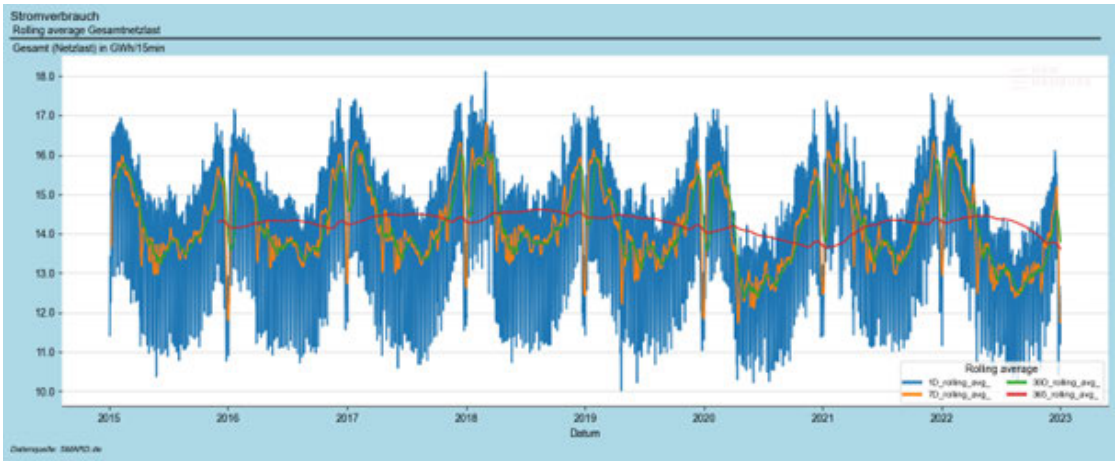

Abbildung 5.69: Rolling

Die wöchentlichen und monatlich gleitenden Mittelwerte zeigen wiederum einen Aufwärtstrend des Energieverbrauchs in den Wintermonaten und einen Abwärtstrend in den Sommermonaten. Der gleitende Jahresdurchschnitt zeigt zunächst einen stabilen Trend mit einem leichten Anstieg im Jahr 2017 und einem nahezu gleichbleibenden Wert im Jahr 2018 im Vergleich zu 2017. Im Jahr 2019 ist ein deutlicher Rückgang des Stromverbrauchs zu verzeichnen. Im Vergleich zu den Vorjahreswerten sinkt der Stromverbrauch im Jahr 2020 kontinuierlich, was auf die Maßnahmen zur Eindämmung der Coronapandemie zurückgeführt werden kann. Ab Anfang 2021 ist ein Aufwärtstrend zu beobachten und der Stromverbrauch erreicht wieder das Niveau von 2019, da die Maßnahmen zur Eindämmung der Coronapandemie geringer ausfallen. Im letzten Quartal 2022 ist ein Rückgang des Stromverbrauchs im Vergleich zum Vorjahresquartal zu beobachten. Die aus der Abbildung 5.69 extrahierten Informationen können durch Quellen wie SMARD, den Bericht von AGEB und BDEW bestätigt werden [10][5][3].

Zeitreihenzerlegung ist eine weitere statische Methode, um Trends und Saisonalitäten zu erkennen und die aus der Abbildung 5.69 gewonnene Informationen zu bestätigen, bei der eine Zeitreihe in mehrere Komponenten zerlegt wird, von denen jede ein zugrunde liegendes Muster repräsentiert. In der Regel setzt sich eine Zeitreihe aus den folgenden drei Komponenten zusammen:

- Trend: Auf- oder Abwärtstrend der Zeitreihe über einen längeren Zeitraum.
- Saisonalität: Der wiederkehrende kurzfristige Zyklus der Zeitreihe.

- Residual: Die zufällige Variation, die nach Abzug von Trend und Saisonalität übrig bleibt.

Die Abbildung 5.70 zeigt die Zeitreihenzerlegung unseres Datensatzes, wobei der Gesamtnetzlast wöchentlich aggregriert ist.

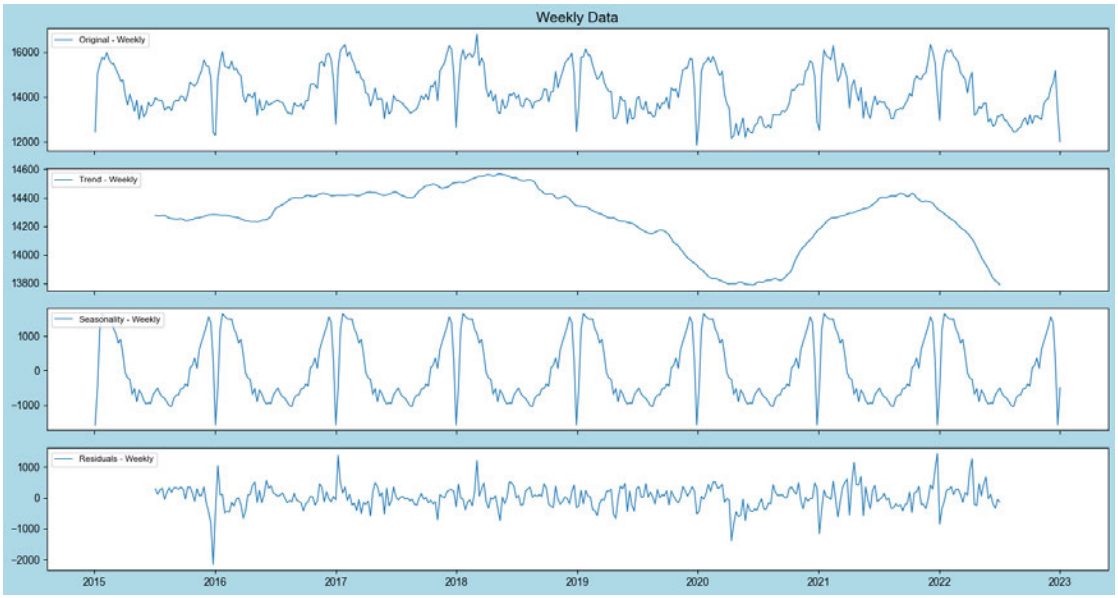

Abbildung 5.70: Zeitreihenzerlegung

Was in Abbildung 5.69 in Bezug auf den Trend und die Saisonalität zu erkennen ist, kann durch die in der Abbildung 5.70 dargestellte Trend- und Saisonalitätskomponente veranschaulicht werden. Dies bestätigt, dass der Datensatz ein saisonales Muster und einen Trend aufweisen. Dies bedeutet, dass Prognosen für die Zukunft möglich sind.

Zur Analyse von Trend und Saisonalität wird zusätzlich noch der Einfluss von Tageszeit, Wochentagen, Jahreszeiten und Feiertagen im Energieverbrauch untersucht. Für den Einfluss von Tageszeit wird Zeit-Spalte mit pandas gruppiert, den Mittelwert zu den einzelnen Zeitpunkten berechnet und visualisiert. Der Einfluss der Wochentage werden auf Basis eines monatlichen Datensatzes, der am ersten Montag beginnt, gefiltert und visualisiert. Der Einfluss der Jahreszeiten werden zunächst als wöchentlicher Durchschnittswert aus einem jährlichen Datensatz zusammengerechnet und dargestellt. Darüber hinaus wird mithilfe der Wärmekarte der Einfluss von Feiertagen bestimmt.

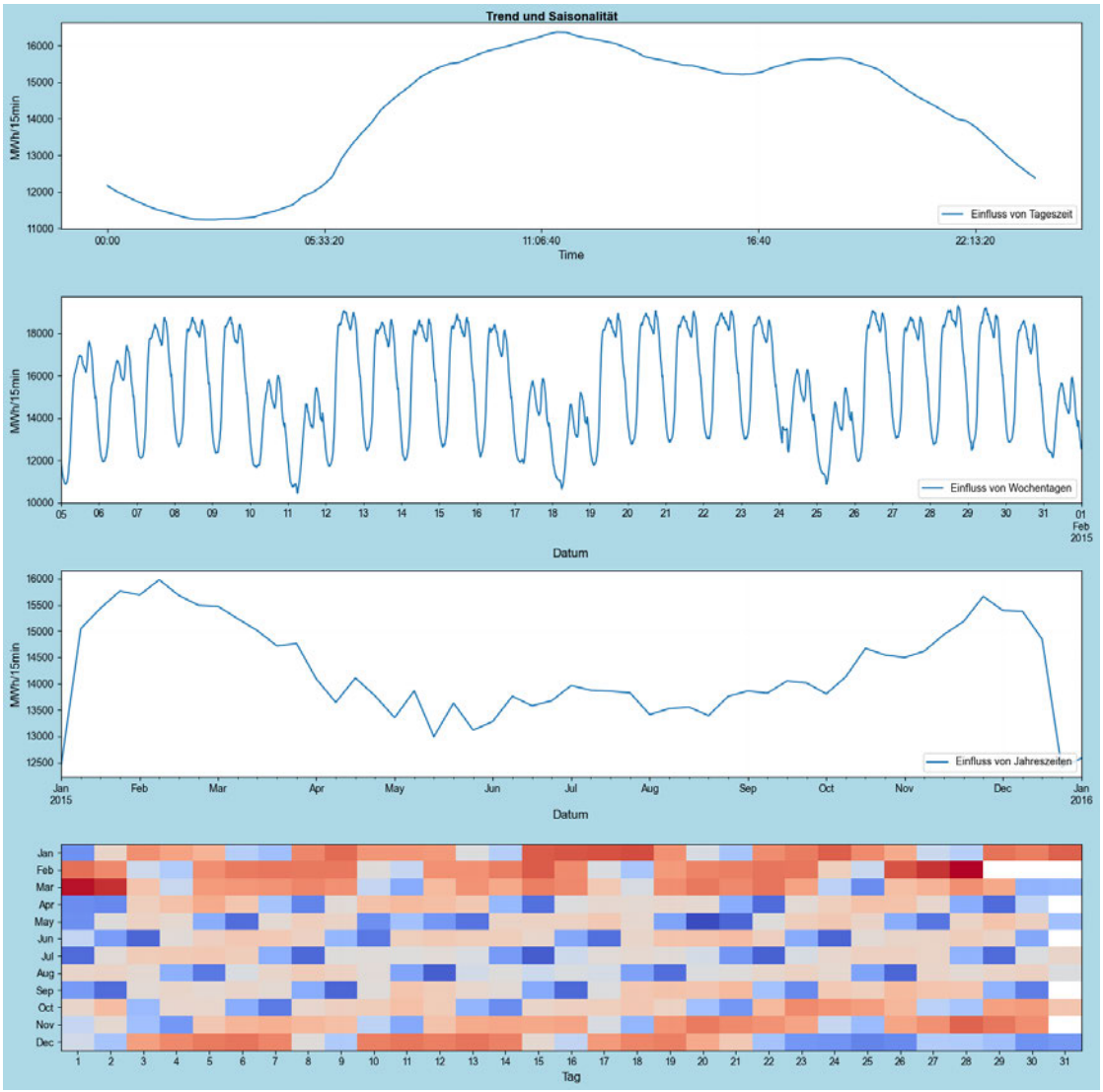

Abbildung 5.71: Trend und Saisonalität

Die Abbildung 5.71 verdeutlicht den Einfluss von Tageszeit, Wochentagen, Jahreszeiten und Feiertagen. Das erste Subplot zeigt einen typischen Tagesablauf mit zwei Höhepunkten am Mittag und am frühen Nachmittag. Im zweiten Subplot lässt sich der Wochenverlauf erkennen. Dabei wird am Wochenende im Vergleich zu Wochentagen weniger Strom verbraucht. Der jährliche Verlauf kann im dritten Subplot beobachtet werden. Dabei wird in Wintermonaten im Vergleich zu Sommermonaten eine höhere Strom verbraucht. Darüber hinaus lässt sich im vierten Subplot deutlich erkennen, wie sich Feiertage wie Ostern und Weihnachten auswirken.

Ein weiteres Ziel der Zeitreihenanalyse ist die Prognose. Es ist festgestellt, dass der Datensatz ein saisonales Muster und einen Trend aufweist. Bevor eine Prognose erstellt wird, muss zudem geprüft werden, ob eine Autokorrelation besteht. Die Autokorrelation stellt eine Methode zur Ermittlung der Korrelation zwischen einer Zeitreihe und einer verzögerten Version ihrer selbst dar. Die Abbildung 5.72 präsentiert die Autokorrelationskurve für unterschiedliche Lags, darunter 96 Lags für den täglichen, 672 Lags für den wöchentlichen, 2688 Lags für den monatlichen und 32256 Lags für den jährlichen Zyklus.

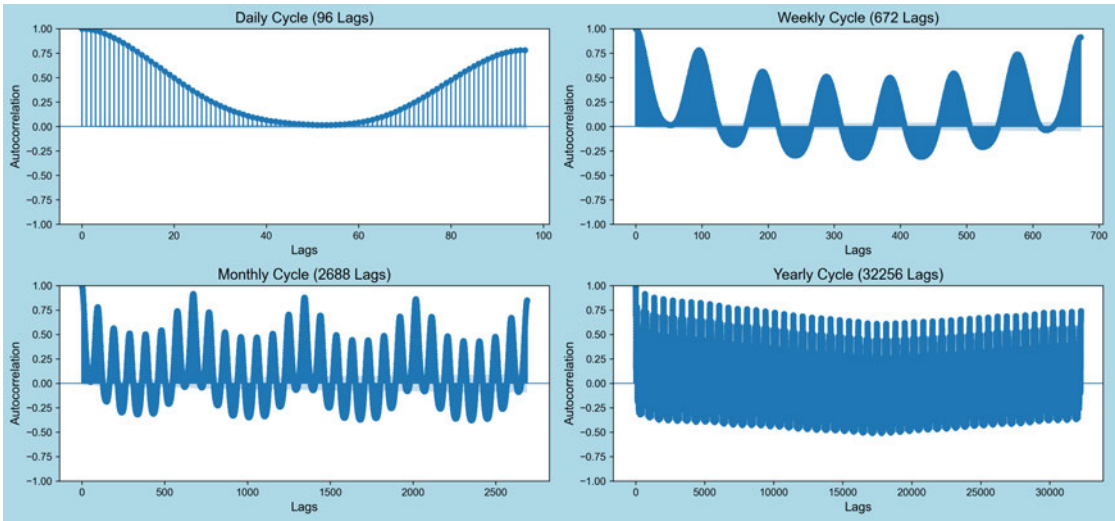

Abbildung 5.72: Autocorrelation

Das erste Sub-Plot zeigt den täglichen Zyklus. Die Verzögerung 0 ist der Mitternachtswert für Samstag, den 3. Januar 2015 und ist hoch korreliert, da er mit sich selbst korreliert. Mittags, d.h. genau 12,5 Stunden später (50 Lags), gibt es keine Korrelation, da der Stromverbrauch zu diesem Zeitpunkt seinen Höhepunkt erreicht. Bei Lag 96, also nach 24 Stunden, besteht wieder eine Korrelation von fast 75%, was auf einen ähnlichen Wert am nächsten Tag zur gleichen Zeit hindeutet.

Die zweite Sub-Plot zeigt die Autokorrelation für Wochenzyklen. Mit Tiefpunkten bei 50 Lags (12,5 Stunden) und Spitzenwerten bei 96 Lags (24 Stunden) wird in der zweiten Sub-Plot der typische tägliche Stromverbrauch untersucht, wobei die Spitzenwerte um Mitternacht und die Tiefpunkte um die Mittagszeit liegen.

Die dritte Sub-Plot zeigt den monatlichen Zyklus und macht deutlich, dass die Werte für jeden Tag stark mit den Werten der folgenden Woche korrelieren. Zum Beispiel

zeigt die Verzögerung 0 die Autokorrelation für Samstag um Mitternacht am 3. Januar 2015 und die Autokorrelation nach einem Intervall von 672 Verzögerungen, d.h. für die kommenden Wochen zur gleichen Zeit, ist fast 80%, was auf eine hohe Korrelation mit den Werten der Vergangenheit hinweist. Ähnliche Informationen können für alle Tage abgelesen werden.

Als nächster Schritt können verschiedene Prognoseverfahren wie Prophet, SARIMA usw. untersucht werden, die im Rahmen dieser Arbeit nicht berücksichtigt werden.

6 Evaluierung

Im Rahmen der Evaluierung werden Python-Bibliotheken für die Visualisierung von Energiedaten miteinander verglichen und anhand einer Reihe von Evaluierungskriterien bewertet. Folgende sind die Bewertungskriterien für die Evaluierung von Python-Bibliotheken:

1. Funktionalität

Können diese Bibliotheken alle in unseren Anforderungen aufgelisteten Visualisierungstypen darstellen und haben eigene eingebaute Funktionen dafür?

2. Dokumentation

Gibt es eine aktive Entwicklergemeinschaft und ausreichende Unterstützung (z. B. Dokumentation, Foren, Tutorials, Codebeispiele) für die Umsetzung der Visualisierungsaufgabe?

3. Leistung anhand Ausführungszeit

Wie lange hat die Ausführung des Codes gedauert?

4. Benutzerfreundlichkeit

Wie einfach ist es, die Bibliothek zu erlernen und zu verwenden, besonders für Anfänger?

5. Interaktivität

Bietet die Bibliothek zahlreiche Funktionen zur Interaktion mit den Visualisierungen?

6. Komplexität

Wie viele Codezeilen sind erforderlich, um die Plots ohne Kommentare und Leerzeilen zu reproduzieren

6.1 Statische und interaktive Visualisierung

6.1.1 Funktionalität

Es soll geprüft werden, ob die Bibliotheken die Muss-Anforderungen aus der Tabelle 3.3 und 3.4 erfüllen. Außerdem soll untersucht werden, mit welchen Soll- und Kann-Anforderungen die Visualisierungen erweitert werden können.

		Bibliotheken				
	Visualisierungs-typen	matplotlib	seaborn	vega altair	bokeh	plotly
Einfache Visualisierungen	Liniendiagramm	+	+	+	+	+
	Balkendiagramm	+	+	+	+	+
	Kreisdiagramm	+	-	+	+	+
	Flächendiagramm	+	+	+	+	+
	Punktediagramm	+	+	+	+	+
Visualisierungs-typen für die Datenanalyse	Boxplot	+	+	+	-	+
	Histogramm	+	+	+	+	+
	Violinenplot	+	+	+	-	+
Komplexe Visualisierungen	Wärmekarte	+	+	+	+	+
	Sankeydiagramm	-	-	-	-	+
	Karten	-	-	-	-	+
Interaktive Visualisierung	Interaktive Visualisierung	-	-	+	+	+

Tabelle 6.1: Vergleich von Visualisierungsbibliotheken

Tabelle 6.1 gibt einen Überblick über alle Diagrammtypen, die auf unseren Anforderungen basieren. Sie enthält statische und interaktive Visualisierungsbibliotheken. Das Plus-Zeichen zeigt an, dass diese Visualisierungstypen aus diesen Bibliotheken implementiert werden können, während das Minus-Zeichen anzeigt, dass für diese Visualisierungstypen keine direkte eingebaute Funktion zur Verfügung steht.

Alle Visualisierungstypen in der Tabelle 3.2 und die MUSS-Anforderungen in den Tabellen 3.3 und 3.4 werden erfüllt. Die einzige Ausnahme ist, dass Seaborn die Funktion Matplotlib zum Plotten von Kreisdiagrammen und Flächenplots verwendet und dass Bokeh keine eigene eingebaute Funktion für Box- und Violinplots hat. Seaborn als statische Visualisierungsbibliothek bietet zusätzlich integrierte Themen, Figurenästhetik und Farbpaletten für ansprechende Diagramme. Textanpassungen und Anmerkungen wie z. B. HAW-Logo und Quellenangabe, die als optional aufgelistet sind, werden mit Matplotlib

implementiert. Die Visualisierungen aller Bibliotheken lassen sich gut in ein Streamlit-Dashboard integrieren. Zu beachten ist, dass Streamlit nur die Bokeh-Version 2.4.3 unterstützt und einige interaktive Elemente von Bokeh nur mit höheren Bokeh-Versionen verfügbar sind. Interaktive Funktionen wie das Hinzufügen von benutzerdefinierten Steuerelementen wie Schaltflächen, Dropdown-Menüs, Schiebereglern und Selektoren gemäß den Anforderungen des Benutzers wurden ebenfalls implementiert.

6.1.2 Dokumentation

Tabelle 6.2 fasst die Struktur der Dokumentation für jede Bibliothek in der statischen und interaktiven Visualisierung zusammen und gibt nützliche Beispiele für die Umsetzung der Visualisierungsaufgabe.

Bibliotheken	**Documentation Struktur**	**Nützliche Webseiten**
Matplotlib	Dokumentation, Visualisierungstypen, Benutzerhandbuch, Anleitungen, Beispiele	Anpassung von Diagrammen, Gleitender Mittelwert, Tick-Formatierer, Achsen und Subplots, Diagramme speichern, Matplotlib anpassen mit rcParams, Arbeitsblätter
Seaborn	Offizielle Website, Visualisierungstypen, Anleitungen,	Figurästhetik, Arbeitsblätter
Vega-Altair	Offizielle Website, Benutzerhandbuch, Beispiele	Kodierungen, Markierungen, Interaktive Diagramme, Altair-Diagramme speichern, Anpassen von Visualisierungen
Bokeh	Offizielle Website, Dokumentation, Anleitungen	Grundlegendes Plotten, Visualisierung anpassen , Zeitreihendiagramme, Interaktion, Ausgabeoptionen
Plotly	Kurzübersicht, Beispiele	Zeitreihendiagramm , Statistische Diagramme, Karten

Tabelle 6.2: Überblick über Bibliotheksdokumentationsstruktur und mögliche Codebeispiele

Die Dokumentationsstruktur sowie die Beispiele, die in allen Bibliotheken zur Verfügung gestellt werden, ermöglichen es, verschiedene Visualisierungstypen leichter umzusetzen. Es gibt viele Möglichkeiten, die Darstellung mit Matplotlib zu verbessern, und es könnte zeitaufwendig sein, die passenden Beispiele dafür zu finden. Bokeh und Vega-Altair sind bekannt für ihre Interaktivität. Daher liegt der Fokus dieser Bibliotheken eher auf

der Interaktivität als auf der Darstellung der Diagramme allein. Daraus ergeben sich nur wenige Erläuterungen zur Implementierung der Visualisierungstypen und nur wenige Kommentare zum Code. Deshalb ist Plotly im Vergleich zu Bokeh und Vega-Altair benutzerfreundlicher, da es viele Beispiele und ausreichende Dokumentation enthält, um die Visualisierungstypen sowie die Interaktivität umzusetzen.

6.1.3 Leistung

Die Leistung der einzelnen Bibliotheken wird anhand der Anzahl der Codezeilen und der Ausführungszeit bewertet. Abbildung 6.1 zeigt ein Balkendiagramm zum Vergleich der Anzahl der Codezeilen und der Ausführungszeit der jeweiligen Bibliotheken für die Visualisierung von Liniendiagrammen. Das Liniendiagramm ist gewählt, da die anderen Visualisierungstypen keinen großen Unterschied in der Ausführungszeit des Codes aufweisen, während die Ausführungszeit des Liniendiagramms je nach Bibliothek variiert.

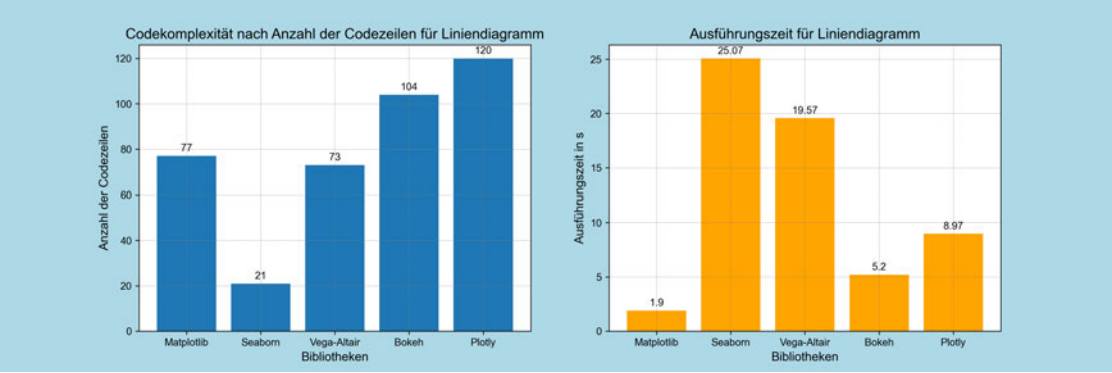

Abbildung 6.1: Codekomplexität und Ausführungszeit

Es ist deutlich zu erkennen, dass Seaborn zwar nur wenige Codezeilen zur Darstellung des Liniendiagramms benötigt, aber sehr lange für die Ausführung braucht. Alle anderen Visualisierungstypen mit Seaborn werden jedoch innerhalb von 10 Sekunden ausgeführt. An zweiter Stelle folgt Vega-Altair, das die längste Ausführungszeit hat. Nicht nur für Liniendiagramme, sondern auch für andere Visualisierungstypen wie Flächendiagramm (ca. 2 min 50 sec) benötigt Vega-Altair viel mehr Zeit. Die Anzahl der Codezeilen steigt, je nachdem, wie der Code erweitert wird, z. B. durch zusätzliche Anpassung von Diagrammen wie in Matplotlib oder durch die Verwendung interaktiver Funktionen wie in Vega-Altair, Ploty und Bokeh. Matplotlib hat eine schnellere Ausführungszeit für die

statische Visualisierung und Bokeh für die interaktive Visualisierung. Außerdem ist die Ausführungszeit von Plotly relativ schnell.

6.2 Dashboard Erstellung

Die gängigen Open-Source-Python-Bibliotheken Streamlit und Dash werden für die Erstellung des Dashboards verwendet. Die Hauptanforderung an das Dashboard ist, dass es dem Benutzer verschiedene Jahre unseres Datensatzes und verschiedene Visualisierungstypen aus den Anforderungen zur Auswahl als ein Dropdown-Menü anbietet. Nachdem eine Auswahl getroffen wurde, sollte ein entsprechendes Diagramm angezeigt werden. Diese werden nach den folgenden Kriterien bewertet.

6.2.1 Funktionalität

Streamlit unterstützt alle statischen und interaktiven Visualisierungsbibliotheken, die für die Arbeit aufgelistet sind. Benutzer können nicht nur die Visualisierungstypen, sondern auch die Bibliotheken in einem einzigen Dashboard auswählen, was hilfreich ist, um die Stärken und Schwächen der einzelnen Bibliotheken zu identifizieren. Dadurch wird es dem Benutzer ermöglicht, selbst zu entscheiden, welche Bibliothek für die Visualisierung ausgewählt werden soll. Streamlit unterstützt nur Python als Programmiersprache.

Mit Dash können nur von Plotly erstellte Visualisierungen im Dashboard angezeigt werden. Dash unterstützt nicht nur Python, sondern andere Programmiersprachen wie R, Julia und F#.

6.2.2 Benutzerfreundlichkeit

Dash ist ein Framework für Webanwendungen. Es bietet eine reine Python-Abstraktion um HTML, CSS und JavaScript herum. Sie benötigen keine tiefgreifenden Kenntnisse in HTML, CSS oder JavaScript, aber es kann etwas Zeit in Anspruch nehmen, sich mit den Dash-HTML-Komponenten wie html.div, html.label, den Dash-Kernkomponenten wie dcc.graph, dcc.dropdown und den Callback-Funktionen vertraut zu machen, um Interaktivität in Ihre Dash-Anwendung zu bringen.

Streamlit ist ein Open-Source-Python-Framework für Datenwissenschaftler und KI/ML-Ingenieure, das die Erstellung dynamischer Datenanwendungen mit nur wenigen Zeilen Code ermöglicht. Es folgt einem deklarativen Paradigma, bei dem zunächst Bibliotheken importiert werden, die Seite konfiguriert wird, Daten geladen werden, bei Bedarf eine Seitenleiste hinzugefügt wird, Visualisierungstypen und Bibliotheken definiert werden und schließlich alles zu einem Anwendungslayout zusammengefügt wird. Im Gegensatz zu Dash ist Streamlit für Nutzer ohne Vorkenntnisse deutlich benutzerfreundlicher sowie einfacher zu verstehen.

6.2.3 Interaktivität

In beiden Bibliotheken kann eine Auswahl verschiedener Jahre sowie verschiedener Visualisierungstypen als Drop-down-Menü vorgenommen werden, was die Anforderungen an die Interaktivität erfüllt. Alle interaktiven Funktionen, die in unterschiedlichen Bibliotheken vorhanden sind, mit Ausnahme von Bokeh, sind auch in Streamlit enthalten. Dash enthält auch Visualisierungstypen, die in Plotly enthalten sind. Da Streamlit nur Bokeh 2.4.3 Version unterstützt, können Fuktionen von höheren Versionen von Bokeh nicht in Streamlit angezeigt werden.

6.2.4 Komplexität

Da nur die Visualisierungstypen von Plotly in Dash implementiert werden können, ist die Anzahl der Codezeilen in Dash im Vergleich zu Streamlit sehr gering. Im Gegensatz dazu macht das Einfügen verschiedener Visualisierungstypen mit unterschiedlichen Bibliotheken den Code in Streamlit sehr komplex und die Anzahl der Codezeilen sehr groß. Aus diesem Grund ist die Implementierung von Dashboard mit Streamlit komplex und zeitintensiv, obwohl es im Vergleich zu Dash sehr einfach zu verstehen ist.

6.3 Karten Visualisierung

Im Rahmen dieser Arbeit sind die Karten mit den gängigen Bibliotheken in Python wie Plotly, Folium und Geopandas implementiert. Zur Evaluierung dieser Bibliotheken sind folgende Kriterien festgestellt.

6.3.1 Funktionalität

Jede Bibliothek ist einzigartig und bietet zahlreiche Funktionen und Erweiterungsmöglichkeiten für Visualisierungstypen. Nicht alle Funktionen können im Rahmen der Arbeit implementiert werden. Daher wird eine vorläufige Liste von Anforderungen erstellt, die die Bibliotheken mindestens erfüllen müssen. Eine Übersicht ist in Tabelle 3.5 zu finden. Es wird untersucht, ob die ausgewählten Bibliotheken zur Kartenvisualisierung die notwendigen Anforderungen erfüllen. Die »Muss«-Anforderungen werden von Folium und von Geopandas erfüllt. Plotly eignet sich nur für die Visualisierung von Choroplethenkarten auf Bundesland- und Landkreisebene und bietet keine Möglichkeit, zwei Geodataframes auf einer Karte zu plotten.

6.3.2 Interaktivität

Während Plotly nur grundlegende Interaktivität wie Hovern über die Karte, Ein- und Auszoomen, Speichern usw. bietet, verfügt Folium über zahlreiche interaktive Funktionen wie ClusterMarker, Search, LayerControl usw. Geopandas nutzt Folium-Funktionen, um die Interaktivität zu erweitern und ansprechende Karten anzuzeigen.

6.3.3 Dokumentation

Je besser die Dokumentation ist, desto einfacher ist es, die Schritte zur Kartenvisualisierung mit diesen Bibliotheken nachzuvollziehen. Tabelle 6.3 zeigt nützliche Beispiele, die ausreichende Informationen zur Kartenvisualisierung enthalten. Mit Hilfe der Community lässt sich feststellen, in wie weit diese Bibliotheken verbreitet sind.

Bibliotheken	Nützliche Beispiele	Community		
		Stars	Contributers	Downloads
Plotly	Choropleth Maps, Scatter Plots on Maps	15k	417	59M/Month
Folium	User guide, GeoJSON and choropleth, Features, MarkerCluster	6.6k	153	978k/Month
Geopandas	Mapping and plotting tools, Interactive mapping,	4.1k	209	6.4M/Month

Tabelle 6.3: Überblick über Codebeispiele und Community

Da sich Plotly nicht ausschließlich auf die Kartenvisualisierung konzentriert, bietet es nur eingeschränkte Funktionen. Im Vergleich zu Plotly und Geopandas bietet Folium eine bessere Dokumentation für die Visualisierung von Karten an. Allerdings ist auch

die Dokumentation von Plotly und Geopandas ausreichend. Plotly wird am häufigsten verwendet, da es für verschiedene Zwecke eingesetzt werden kann. Geopandas ist die am weitesten verbreitete Bibliothek für Kartenvisualisierung, da sie eine gute Grundlage für die Datenvorbereitung vor der Kartenvisualisierung bietet und über eine große Anzahl von Mitwirkenden und Downloads verfügt. Folium ist jedoch aufgrund seiner zahlreichen interaktiven Funktionen sehr bekannt, wenn es um die Visualisierung von Karten in Python geht.

7 Zusammenfassung

7.1 Überprüfung von Forschungszielen und -fragen

In diesem Abschnitt werden die in Kapitel 1.2 entwickelten Fragen daraufhin überprüft, ob sie im Rahmen der Arbeit beantwortet wurden.

RO1: Wie können Energiedaten am besten visualisiert werden und welche Visualisierungstypen sind dafür geeignet?

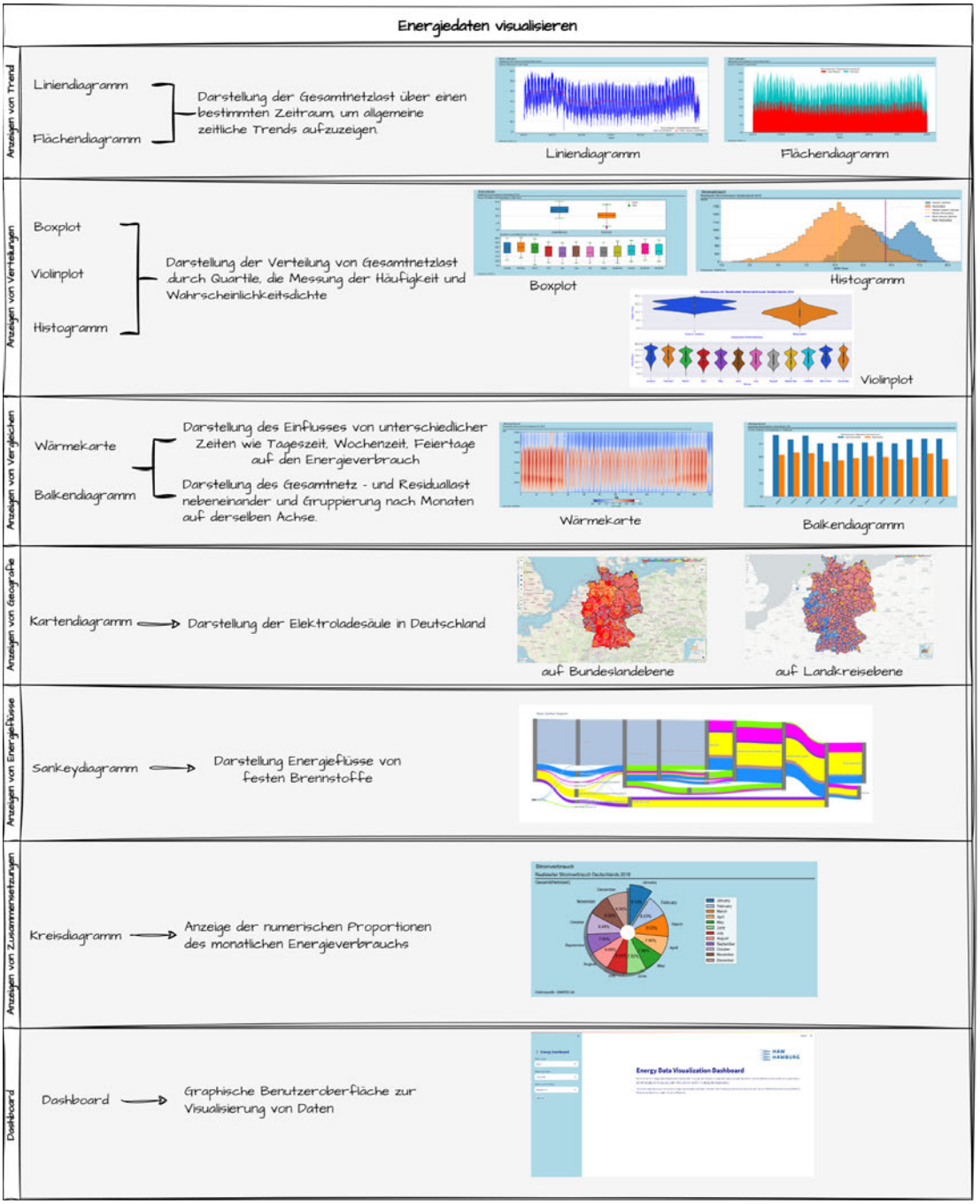

Abbildung 7.1: Zusammenfassung RO1

RO2: Welche Python-Bibliotheken sind gängig für die Datenvisualisierung?

RQ 2.1: Welche Python-Bibliotheken werden am häufigsten für die Datenvisualisierung verwendet?

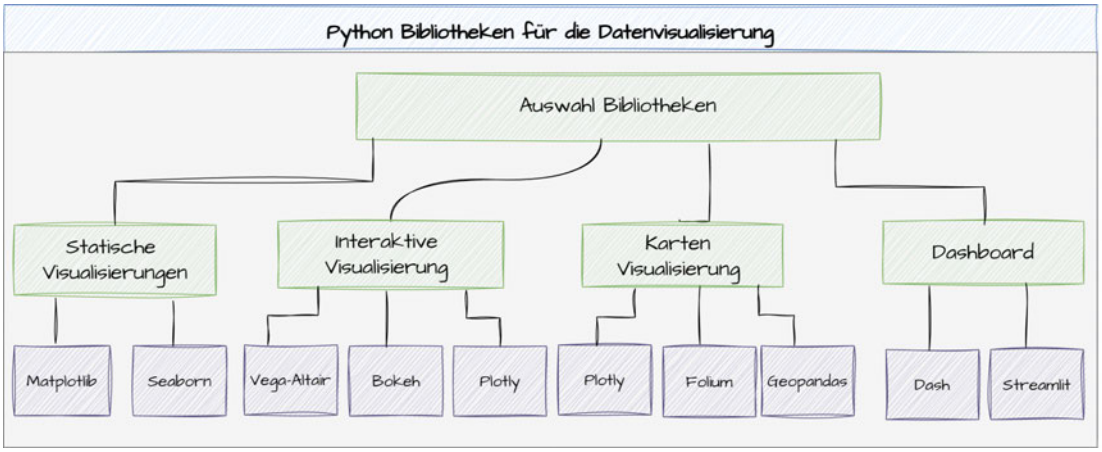

Abbildung 7.2: Python Bibliotheken für die Datenvisualisierung

Abbildung 7.2 zeigt einen Überblick über die Pythonbibliotheken, die am häufigsten für die Datenvisualisierung verwendet werden.

RQ 2.2: Können die in RQ1.3 genannten Visualisierungstypen mit diesen Bibliotheken realisiert werden?

Alle gängigen Python-Bibliotheken zur Datenvisualisierung sind leistungsfähig und erfüllen unsere Anforderungen. Jede dieser Bibliotheken hat ihre eigenen Stärken und Schwächen. Zum Beispiel können mit Matplotlib und Seaborn publikationsfähige statische Visualisierungen erstellt werden. Vega-Lite, Bokeh und Plotly mit ihren zusätzlichen interaktiven Funktionen helfen, ansprechende und interaktive Diagramme zu erstellen, wenn der Benutzer Interaktivität wünscht. Im Vergleich zu Bokeh und Vega-Altair verfügt Plotly über eingebaute Funktionen für alle Arten von Visualisierungen. Darüber hinaus verfügt Plotly über das Dash Framework, mit dem die von Plotly erstellten Visualisierungen in Form eines Dashboards dargestellt werden können. Bokeh erfüllt auch die Anforderungen an interaktive Visualisierungen und bietet zahlreiche Widgets zur Erweiterung der Interaktivität. Allerdings hat Bokeh keine eingebaute Funktion für einige statistische Diagramme, wie z.B. Boxplots und Violinplots, und für die Erfüllung dieser Anforderungen ist eine sehr komplexe Programmierung erforderlich. Im Gegensatz zu Plotly und

Bokeh ist Vega-Altair nicht standardmäßig interaktiv, sondern muss individuell programmiert werden. Folium bietet zahlreiche Funktionen, um mit Karten zu interagieren und ist daher sehr gut für die Kartenvisualisierung geeignet. Geopandas bildet die Grundlage für die Datenaufbereitung und integriert die Funktionen von Folium, um ansprechende Karten zu erstellen. Eine Kombination von Geopandas und Folium wird für die Kartenvisualisierung empfohlen. Plotly kann Choroplethen sehr gut visualisieren, bietet aber keine Möglichkeit, zwei Geodatenrahmen auf einer Karte zu plotten und erfüllt daher nicht die Anforderungen für die Kartenvisualisierung. Streamlit ist eine gute Wahl für die Erstellung von Dashboards mit Visualisierungstypen aus anderen Bibliotheken, da es mit allen anderen Bibliotheken, einschließlich Plotly, interagieren und die von diesen Bibliotheken erstellten Visualisierungen in das Dashboard integrieren kann.

RQ2.3: Welche Anforderungen müssen Bibliotheken erfüllen, um Energiedaten zu visualisieren?

Die Anforderungen an die Bibliotheken sind im Kapitel 3.4 in tabellarischer Form dokumentiert. Für jede Bibliothek sind die Muss-, Soll- und Kann-Anforderungen aufgelistet.

RO3: Wie können Python-Bibliotheken evaluiert werden?

Je nach Anwendungsfall sind verschiedene Kriterien wie Funktionalität, Dokumentation, Leistung, Interaktivität, Komplexität usw. für die Evaluation von Python-Bibliotheken verwendet. Für jede Bibliothek wird eine Tabelle mit Einzelbewertungen erstellt.

Bibliothek	Funktionalität	Benutzerfreundlichkeit	Leistung	Interaktivität	Dokumentation	Komplexität
Matplotlib	5	3	5	2	5	5
Seaborn	4	4	3	2	4	3
Vega-Altair	4	3	2	3	4	2
Bokeh	3	3	5	5	4	4
Plotly	5	5	5	5	5	5
Dash	5	3	5	4	5	4
Streamlit	5	5	5	5	5	5
Geopandas	4	4	4	3	5	5
Folium	5	4	4	5	5	5

Tabelle 7.1: Einzelbewertungen

7.2 Grenzen der Arbeit

Die Implementierung zahlreicher Visualisierungstypen mit Hilfe verschiedener Python-Bibliotheken ist zeitaufwändig und erfordert viel Geduld. Daher können im Rahmen dieser Arbeit einige Implementierungen nicht berücksichtigt werden. Die API's jeder Bibliothek haben sicherlich noch weitere interessante Funktionen. Diese können die Visualisierungen noch attraktiver und interaktiver machen. In dieser Arbeit wird nur auf die wichtigsten eingegangen, die ausreichen, um die Anforderungen zu erfüllen. Darüber hinaus gibt es viele weitere Visualisierungstypen zur Darstellung eines Datensatzes. In dieser Arbeit werden nur die gängigsten Visualisierungstypen für Energiedaten betrachtet.

7.3 Zukünftige Arbeiten

Falls möglich, sollte die Erweiterung des Dashboards um zusätzliche Funktionen untersucht werden. In der vorliegenden Arbeit liegt der Schwerpunkt auf der deskriptiven und der diagnostischen Analyse. Als nächster Schritt könnte die prädiktive Analyse, d.h. die Vorhersage der zukünftigen Stromverbrauchsentwicklung mit verschiedenen Prognoseverfahren durchgeführt werden.

Literaturverzeichnis

[1] 2019, PyViz authors. From data to viz | find the graphic you need. https://datavizproject.com/. Accessed: 2023-11-29.

[2] 2024 Plotly. Dash documentation & user guide | plotly. https://dash.plotly.com/. Accessed: 2023-11-15.

[3] AG Energiebilanzen e.V. Energieverbrauch in deutschland im jahr 2017. https://ag-energiebilanzen.de/wp-content/uploads/2021/02/ageb_jahresbericht2017_20180315-02_dt.pdf. Accessed: 2024-04-16.

[4] Louis Allen, Jack Atkinson, Dinusha Jayasundara Mudiyanselage, Joan Cordiner, and Peyman Moghadam. Data visualization for industry 4.0: A stepping-stone toward a digital future, bridging the gap between academia and industry. *Patterns*, 2:100266, 05 2021.

[5] BDEW. Bundesverband für energie- und wasserwirtschaft. https://www.bdew.de/presse/presseinformationen/zahl-der-woche-gesamtstromverbrauch-deutschland/. Accessed: 2024-04-16.

[6] bimanu. Datenvisualisierung: Definition, beispiele und vorteile. https://bimanu.de/blog/datenvisualisierung/. Accessed: 2023-11-20.

[7] Bokeh Contributors. Bokeh documentation. https://docs.bokeh.org/en/latest/index.html. Accessed: 2024-01-20.

[8] Bundesnetzagentur. Bundesnetzagentur - ladesäulenkarte. https://www.bundesnetzagentur.de/DE/Fachthemen/ElektrizitaetundGas/E-Mobilitaet/Ladesaeulenkarte/start.html. Accessed: 2023-02-15.

[9] Bundesnetzagentur. SMARD | SMARD - Strommarktdaten, Stromhandel und Stromerzeugung in Deutschland. https://www.smard.de/home. Accessed: 2023-11-15.

[10] Bundesnetzagentur. SMARD | SMARD - Strommarktdaten, Stromhandel und Stromerzeugung in Deutschland. https://www.smard.de/home/energiemarkt-aktuell/2020. Accessed: 2024-04-16.

[11] Thomas Cleff. *Deskriptive Statistik und Explorative Datenanalyse*, pages 4–5. Springer-Verlag, 3 2015.

[12] DataCamp. Florence nightingale: Pioneer of data visualization, 2021. Accessed: 2023-11-10.

[13] ETA-SOLUTIONS. Klare ziele für mehr energieeffizienz. https://www.energiewechsel.de/KAENEF/Redaktion/DE/Standardartikel/energieeffizienzgesetz.html#:~:text=Das%20Gesetz%20sieht%20vor%2C%20den,daf%C3%BCr%20notwendigen%20Ma%C3%9Fnahmen%20zu%20ergreifen. Accessed: 2024-04-11.

[14] ETA-SOLUTIONS. Leitfaden: Monitoring von energieeffizienzmaßnahmen. https://www.ptw.tu-darmstadt.de/media/fachgebietptw/dokumente_3/wissenssammlung_ptw/leitfaeden_2/Leitfaden_Energiemonitoring.pdf. Accessed: 2023-11-25.

[15] European Union. Eurostat. https://ec.europa.eu/eurostat/databrowser/view/nrg_bal_sd__custom_10511268/default/table?lang=en. Accessed: 2023-12-15.

[16] European Union. Eurostat. https://ec.europa.eu/eurostat/cache/sankey/energy/sankey.html?geos=EU27_2020&year=2022&unit=KTOE&fuels=RA000&highlight=_&nodeDisagg=0101000000000&flowDisagg=true&translateX=246.54633366907507&translateY=99.46374603487897&scale=0.6597539553864471&language=EN. Accessed: 2023-12-15.

[17] eurostat. Energy visualisation portal. https://ec.europa.eu/eurostat/cache/infographs/energy_portal/enviz.html?language=EN. Accessed: 2023-12-05.

[18] ferdio. Python tools for data visualization — pyviz 0.0.1 documentation. https://pyviz.org/index.html. Accessed: 2023-12-02.

[19] Financial-Times (F.). Chart-doctor/visual-vocabulary at main · financial-times/chart-doctor. https://github.com/Financial-Times/chart-doctor/tree/main/visual-vocabulary. Accessed: 2023-11-28.

[20] Fraunhofer-Institut für Solare Energiesysteme ISE. Energy-charts. https://www.energy-charts.info/index.html?l=de&c=DE. Accessed: 2023-12-05.

[21] GeoPandas developers. Introduction to geopandas. https://geopandas.org/en/stable/getting_started/introduction.html. Accessed: 2024-02-25.

[22] GfG. What is Data Analysis? https://www.geeksforgeeks.org/what-is-data-analysis/?ref=previous_article. Accessed: 2023-12-10.

[23] family=Rougier given i=N, given=Nicolas. *Scientific visualization*.

[24] International Energy Agency. Iea–international energy agency - iea. https://www.iea.org/data-and-statistics. Accessed: 2023-12-06.

[25] John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the Matplotlib development team. Customizing matplotlib with style sheets and rcparams — matplotlib 3.8.4 documentation. https://matplotlib.org/stable/users/explain/customizing.html#the-matplotlibrc-file. Accessed: 2024-12-05.

[26] John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the Matplotlib development team. External resources — matplotlib 3.8.4 documentation. https://matplotlib.org/stable/users/resources/index.html. Accessed: 2024-12-05.

[27] Jonath Jose. Introduction to time series analysis and its applications. 08 2022.

[28] Daniel A. Keim. *21 Datenvisualisierung und Data Mining*, pages 363–370. K. G. Saur, Berlin, Boston, 2004.

[29] Dirk Lehmann, Georgia Albuquerque, Martin Eisemann, Andrada Tatu, Daniel Keim, H. Schumann, Marcus Magnor, and Holger Theisel. Visualisierung und analyse multidimensionaler datensätze. *Informatik Spektrum*, 33:589–600, 12 2010.

[30] D. Nelson. *Data Visualization in Python: Explore and Manipulate Data and Create Engaging Interactive Plots with 9 Python Libraries*, pages 2–6. StackAbuse, 2021.

[31] M. Nurminen, A. Lindstedt, M. Saari, and P. Rantanen. The requirements and challenges of visualizing building data. In *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 968–972, 2021.

[32] Optenda. Energiemonitoring software energy monitor | optenda. https://ec.europa.eu/eurostat/cache/infographs/energy_portal/enviz.html?language=EN. Accessed: 2023-12-05.

[33] Patrick Planing. Statistik grundlagen. https://statistikgrundlagen.de/ebook/chapter/chapter-1-2/. Accessed: 2023-11-28.

[34] Quirine Philipsen. (explorative) datenvisualisierung. https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master-nm-rv-2015/philipsen.pdf. Accessed: 2023-02-20.

[35] Plotly. Choropleth maps in python. https://plotly.com/python/choropleth-maps/. Accessed: 2024-01-16.

[36] Plotly. Plotly open source graphing library for python. https://plotly.com/python/. Accessed: 2024-01-25.

[37] Plotly. Sankey. https://plotly.com/python/sankey-diagram/. Accessed: 2024-02-20.

[38] Plotly. Scatter plots on maps in python. https://plotly.com/python/scatter-plots-on-maps/. Accessed: 2024-01-16.

[39] Rob Story. Folium — folium 0.1.dev1+g3b79310 documentation. https://python-visualization.github.io/folium/latest/index.html. Accessed: 2024-01-17.

[40] Rob Story. User guide — folium 0.1.dev1+g3b79310 documentation. https://python-visualization.github.io/folium/latest/user_guide.html. Accessed: 2024-01-17.

[41] Jesus Rogel-Salazar. *Statistics and Data Visualisation with Python*, pages 2–3. CRC Press, 1 2023.

[42] Jesus Rogel-Salazar. *Statistics and Data Visualisation with Python*, page 145. CRC Press, 1 2023.

[43] Snowflake Inc. Streamlit • a faster way to build and share data apps. https://streamlit.io/. Accessed: 2024-01-10.

[44] Timotheos Frey. Grundlagen der datenvisualisierung. https://timfrey.files.wordpress.com/2018/09/intro_dataviz_201809.pdf. Accessed: 2023-11-10.

[45] Unicorn a.s. Entso-e transparency platform. https://transparency.entsoe.eu/. Accessed: 2023-12-06.

[46] Jacob VanderPlas, Brian Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. Altair: Interactive statistical visualizations for python. *Journal of Open Source Software*, 3(32):1057, 2018.

[47] Vega-Altair Developers. Interactive charts — vega-altair 5.3.0 documentation. https://altair-viz.github.io/user_guide/interactions.html. Accessed: 2024-01-15.

[48] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[49] Yan Holtz And Conor (Y. H. A. C.) Healy. From data to viz | find the graphic you need. https://www.data-to-viz.com/. Accessed: 2023-11-29.

A Anhang

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original