The present work was submitted to Lehr- und Forschungsgebiet Educational Technologies at DIPF

Visual Analytics Dashboard for a Life-Long Learning portal

Bachelor-Thesis

Presented by

Burgstaller, David

6962029

First examiner:	Prof. Dr. Hendrik Drachsler
Second examiner:	Prof. Dr. Detlef Krömker
Supervisor(s):	M.Sc. Atezaz Ahmad

Frankfurt, 28. September 2023

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed.

Frankfurt, 28. September 2023 Full Name

Acknowledgments

Abstract

Diese Arbeit beschäftigt sich mit der Suche und Implementierung einer alternativen Darstellungsform für die Metasuchmaschine ”InfoWeb Weiterbildung” (IWWB). Statt einer Auflistung der Kurse wird ein Dashboard als Lösungsansatz gewählt. Dabei bildet eine Karte den Kern der Anwendung, auf der die Standorte aller Kurse dargestellt werden. Darüber hinaus geben weitere Graphen einen Überblick über den Gesamtdatensatz. Außerdem wird durch eine interaktive Gestaltung eine gute Balance zwischen Informationsdichte und Übersichtlichkeit angestrebt.

Insgesamt wird dadurch erhofft, dass es sowohl Nutzer als auch Forschern möglich ist, interessante Informationen aus dem Datensatz zu ziehen und sie in Entscheidungsprozessen bestmöglichst unterstützt werden. Um dies optimal zu bewerkstelligen, wurden die Themengebiete der explorativen Datenanalyse und der Datenvisualisierung betrachtet und daraus wertvolle Erkenntnisse herangezogen.

Inhaltsverzeichnis

1 Einleitung

Laut Han et al. (2011) lebt unsere Gesellschaft nicht mehr im Informationszeitalter, sondern im Datenzeitalter. Als Grund nennt er hierfür den Fortschritt der Technologie, der es ermöglicht täglich Daten im Tera- oder Petabyte-Bereich zu sammeln. Sei es im Internet, in Firmen, in der Wissenschaft oder der Medizin, in fast jedem Bereich werden Daten gesammelt und gespeichert (Why Data Mining?, Kap. 1.1). Duarte (2023) bekräftigt diese Aussage und zeigt durch Statistiken, dass die täglich erstellte Datenmenge weltweit auf 328,77 Millionen Terrabyte oder 328.77 Exabyte geschätzt wird. Für die nächsten Jahre soll diese voraussichtlich noch steigen.

1.1 Problemstellung

Mit dem Fortschritt können zwar Daten gesammelt werden, doch für das menschliche Gehirn ist es nicht möglich den riesigen Umfang an Rohdaten zu erfassen und zu nutzen. Solange aus den Daten keine verwertbaren Erkenntnisse erlangt werden, sind sie nutzlos. Aus diesem Grund wird nach Möglichkeiten gesucht, das volle Potenzial der Daten auszuschöpfen (Sinar, 2015).

Vor demselben Problem steht auch die Meta Suchmaschine InfoWeb Weiterbildung (IWWB) [1]. Bei IWWB handelt es sich hierbei um eine Metasuchmaschine, die mehrere Datenbanken auf Weiterbildungskurse in ganz Deutschland durchsucht (Deutscher Bildungsserver, 2013). Diese Kurse werden für den Nutzer mit knappen Informationen in einer Liste präsentiert. Hier ist es ihm möglich einen einzelnen Kurs auszuwählen, woraufhin er sofort zur Quelle des Kurses weitergeleitet wird. Gegebenenfalls kann er sich bei diesem Kurs anmelden und teilnehmen.

Mit den Daten von ca. 3,4 Millionen Weiterbildungskursen in ganz Deutschland bietet IWWB einen großen Datensatz an mit viel Potenzial für Wissenserkenntnisse. Durch den Umfang lässt sich annehmen, dass die gesamten Daten die Bildungssituation in Deutschland gut widerspiegeln könnten und somit für Wissenschaftler von großem Wert sein könnten. Daraus erlangte Erkenntnisse könnten demnach entscheidend für die weitere Ausrichtung der Bildungsentwicklung in Deutschland sein. Jedoch erschwert der Datenumfang gleichermaßen dessen Erforschung und Analyse. Durch die Darstellungsform der Daten in einer Liste ist es, trotz eingebauter Filterfunktion, menschlich unmöglich, Information über ein paar Kurse hinaus zu erlangen. Somit ist das vermeintliche Potenzial nicht in Gänze ausgeschöpft. Aus diesem Grund befasst sich die vorliegende Arbeit mit der Informationsgewinnung aus den Daten der IWWB, sowie einer alternativen Darstellungsform, um sie für den Menschen greifbar und anwendbar

[1] https://www.iwwb.de/kurssuche/startseite.html

zu machen. Zum einen um Beispielsweise Wissenschaftler einen Überblick über die Weiterbildungskurse zu gewähren, sodass sie hilfreiche Erkenntnisse und Einschätzungen über die Bildungssituation Deutschlands erlangen können. Zum anderen, um Internetnutzern auf der Suche nach Weiterbildungsmöglichkeiten in Entscheidungsprozessen bestmöglichst zu unterstützen.

1.2 Forschungsfragen

Die Beantwortung folgender übergeordneten Forschungsfrage bildet somit das Ziel dieser Bachelorarbeit:

Wie kann das volle Potenzial des Datensatzes ausgeschöpft werden und die darin verborgenen Informationen aufgedeckt werden?

Aufgrund der Tatsache, dass nach keinen spezifischen Informationen gesucht wird, sollte die Darstellungsform einen Überblick über die Daten schaffen, wodurch sowohl Fragen aufgeworfen werden als auch mögliche Antworten präsentiert werden. Es gilt infolgedessen die Daten zu analysieren. Hierfür bietet sich, als vielversprechender Lösungsansatz, die Entwicklung eines Dashboards an. Darin sollen mehrere Visualisierungen Aufschluss über die Daten geben und somit verwendbare Informationen vermitteln. Die Entscheidung für die Wahl des Lösungsansatzes gründet auf Erkenntnissen, welche im Grundlagenkapitel erlangt wurden und wird dort weiter ausgeführt.

In Anbetracht der spezifischen Daten von IWWB ist das Ziel der Entwicklung, die dort aufgelisteten Kurse in Themengebiete zu klassifizieren, sowie deren Standorte auf einer Karte zu visualisieren. Mit diesem Lösungsansatz werden folgende untergeordnete Forschungsfragen im Rahmen dieser Bachelorarbeit aufgegriffen und beantwortet.

1. Wie lassen sich Kursinformationen mithilfe von Markierungen auf einer Karte visualisieren?
    a) Ist es möglich eine Menge Markierungen auf der Karte ohne Leistungsprobleme zu erzeugen? Wie lässt sich das Problem der Menge an Markierungen lösen?
    b) Wie lassen sich mehrere Kurse von einem Anbieter und demselben Standort übersichtlich darstellen?
2. Wie können die Kurse in unterschiedlichen Themen eingeteilt und visualisiert werden?

In all diesen Herausforderungen steht die Informationsvermittlung an den Nutzer im Vordergrund, weshalb eine geeignete, übersichtliche Darstellung nicht vernachlässigt werden darf. Aus diesem Grund werden im zweiten Kapitel die Grundlagen der relevanten Themen aus der Datenwissenschaft für eine effektive und effiziente Informationsvermittlung thematisiert.

1.3 Überblick der Arbeit

Im folgenden 2. Kapitel werden die theoretischen Grundlagen besprochen, welche den Entscheidungs- und Entwicklungsprozess der Anwendung nachvollziehbar machen. Anschließend werden im

??. Hierfür werden insbesondere die Themengebiete der explorativen Datenanalyse und die Datenvisualiserung betrachtet. Im anschließenden Kapitel 3 wird die Implementierung der Anwendung erläutert und ihre Funktionalität erklärt. Zusammenfassend werden in Kapitel 4 die aus dieser Arbeit erlangten Erkenntnisse vorgestellt und die entwickelte Anwendung anhand der Forschungsfragen kritisch bewertet. Im vorletzten Kapitel 5 wird das Potenzial der Anwendung durch Zukünftige Forschung oder Modifikationen geschildert und auf Limitationen der Arbeit hingewiesen. Zu Letzt werden in Kapitel 6 die Erkenntnisse zusammengetragen und abschließend auf die Forschungsfragen eingegangen.

2 Grundlagen und Verwandte Arbeiten

Dieses Kapitel befasst sich mit Themen, welche dem Verständnis der folgenden Arbeit zugrunde liegen. Hierfür wird zunächst das Themengebiet der Datenwissenschaft vorgestellt und die Problemstellung in den spezifischeren Bereich der explorativen Datenanalyse eingegrenzt. Des weiteren wird das damit zusammenhängende Gebiet der Datenvisualisierung aufgegriffen und erläutert. Abschließend wird die Grundstruktur einer Webanwendung, sowie die technischen Grundlagen hierfür erklärt.

2.1 Datenwissenschaft

Ozdemir (2016) definiert *Data Science* als die Kunst und Wissenschaft der Gewinnung von Wissen durch Daten. Die Definition mag zwar kurz erscheinen, inhaltlich umfasst die Datenwissenschaft jedoch eine riesige Menge an Themen, welche im Rahmen der vorliegenden Arbeit nicht alle vorgestellt werden können. Das Ziel von *Data Science* wird mit der Definition gut hervorgehoben. Das Themengebiet beschäftigt sich laut Ozdemir damit, wie aus Daten Wissen erlangt werden kann um dieses vor allem für Folgendes zu nutzen (How to Sound Like a Data Scientist, Kap. 1):

1. Entscheidungen zu treffen
2. Die Zukunft zu prognostizieren
3. Die Gegenwart und Vergangenheit zu verstehen
4. Neue Produkte und Industrien zu erschaffen

Doch was sind Daten überhaupt und was ist der Unterschied zu Informationen und Wissen? Diese Fragen gilt es zu beantworten um die Definition und Zielsetzung der Datenwissenschaft zu verstehen.

Bellinger et al. (2004) definieren Daten als Symbole, als isolierte, nicht interpretierte Fakten, welche keinerlei Relationen zu anderen Daten besitzen. Sie haben keine Relevanz und auch keine Bedeutung in sich selbst. Information hingegen sind Daten, denen durch eine Interpretation und durch Relationen von einer Person Bedeutung gegeben werden. Dabei muss die Bedeutung nicht zwingend nützlich und nicht für jeden gleich sein. Nach Bellinger et al. ist *Wissen* die Anhäufung von Informationen mit dem Ziel einen Nutzen daraus zu ziehen. Das Wissen kann zwar infolgedessen abgerufen und genutzt werden, ermöglicht aber an sich keinen weiteren Wissenszuwachs. Den Prozess von der Verknüpfung von Wissen zur Erschaffung neues Wissen wird als das Verstehen beschrieben. Während das Verstehen als ”lernen” beschrieben

werden kann, besteht das Wissen aus ”auswendig lernen”. Wie in der von Bellinger et al. herangezogenen Abbildung 2.1 dargestellt, funktioniert das Verstehen nicht abgekoppelt, sondern führt zu den Übergängen in die nächsten Stufen.

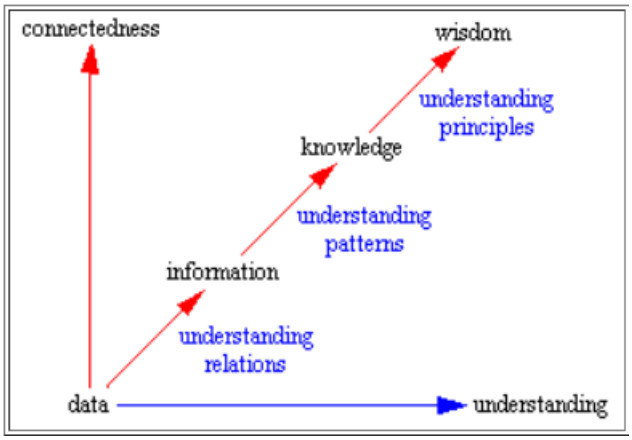


Abbildung 2.1: Daten, Information, Wissen
Quelle: (Bellinger et al., 2004)

Weisheit geht geht weit über das Verständnis hinaus und ist die Essenz des philosophischen Forschens. Sie stellt Fragen auf die es momentan keine Antworten gibt und womöglich auch keine menschliche Antwort geben kann (Bellinger et al., 2004).

Die Weisheit geht aber über die notwendigen Grundlagen für die vorliegenden Arbeit hinaus, weshalb im Folgenden nicht näher darauf eingegangen wird. Die einfache Definition von Daten, Information und Wissen als Ansammlung und Gebrauch von nützlichen Informationen ist für diese Arbeit ausreichend.

Um die Vorgehensweise im *Data Science* nachvollziehen zu können, führen Lau et al. (2021) den sogenannten *Data Science Lifecycle* ein und unterteilen den Prozess Informationsgewinnung in einzelne Schritte. In der folgenden Abbildung 2.2 stellen sie ihre Unterteilung vor (The Data Science Lifecycle, Kap.1).

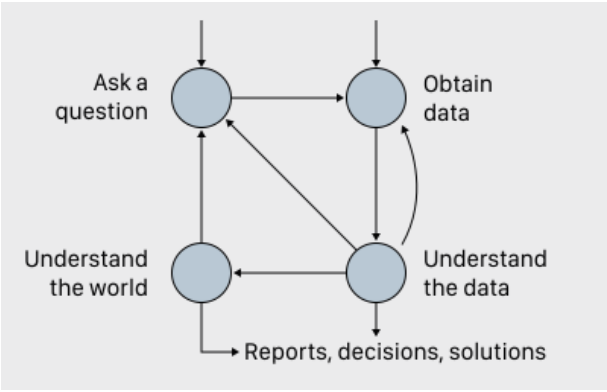


Abbildung 2.2: Data Science Lifecycle
Quelle: (Lau et al., 2021, The Data Science Lifecycle, Kap.1)

Sich Fragen zu stellen und Antworten auf diese Fragen zu finden ist das Fundament von *Da-*

ta Science. Mit den Fragen (”*Ask a question*”) beginnt schließlich die Suche nach bestimmten Daten, welche möglicherweise Antworten geben könnten (vgl. Abbildung 2.2) . Dies führt zu dem zweiten Punkt des Lebenszyklus (”*Obtain data*”), das Sammeln von Daten. Die Autoren erklären an dieser Stelle, dass man von zwei Ausgangspunkten hierhin gelangen kann. Zum einen können die Fragen im Hinterkopf zu der Suche und dem Sammeln von Daten animieren. Zum anderen kann bereits eine große Datenmenge bereitstehen, durch deren Analyse sich erst Forschungsfragen herauskristallisieren. In beiden Fällen müssen die Daten für die Analyse vorbereitet werden, indem möglicherweise Umstrukturierungen nötig sind, falsche Werte bereinigt werden oder Daten in das korrekte Format transformiert werden müssen. Trotz unterschiedlicher Ausgangssituation wird im nächsten Schritt die Datenanalyse (”*Understand the data*”) vollzogen, um die Daten besser zu verstehen. Hier ist laut den Autoren oft das Themengebiet der *exploratory data analysis* (explorative Datenanalyse zu Deutsch) der Schlüssel. An diesem Punkt des Lebenszyklus werden die Daten meist visualisiert, um Besonderheiten und Muster erkenntlich zu machen. Generell ist die Arbeit an dieser Stelle häufig sehr interaktiv, da neue Fragen aufkommen, Fehler im Datensatz deutlich werden oder neue Potenziale der Sammlung entdeckt werden. Dadurch landet man immer wieder an den Knotenpunkten zuvor und durchläuft den Lebenszyklus erneut. Falls das gewählte Ziel lediglich das Beschreiben und Entdecken eines Datensatzes ist, so kann der Zyklus hier enden und Lösungen mit der Analyse der Daten können präsentiert werden. In einigen Fällen wird hier jedoch noch ein Schritt weitergegangen und im nächsten Knotenpunkt (”*Understand the World*”) versucht, die Daten in Relation mit der Welt zu stellen, um die Funktionsweise dieser besser zu verstehen, oder sogar Vorhersagen über deren Zukunft treffen zu können (Lau et al., 2021, The Data Science Lifecycle, Kap.1).

In dieser Bachelorarbeit wird der letzte Punkt *”Understand the world”* nicht weiter betrachtet. Stattdessen entspricht der *”Understand the date”* Schritt bereits ziemlich genau der in der Einleitung vorgestellten Zielsetzung, weshalb im folgenden Abschnitt das, laut den Autoren Lau et al., mit diesem Punkt zusammenhängende Themengebiet ”*exploratory data analysis*” genauer betrachtet wird.

2.2 Exploratory Data Analysis

Mit seinem Buch ”Exploratory Data Analysis” aus dem Jahre 1977 gilt John W. Tukey als der Gründer des Themengebiets der explorativen Datenanlyse. Darin präsentiert er seine Erkenntnisse, dass durch einzelne simple Graphen und einfache statistische Methoden, wie die Berechnung des Durchschnitts, des Median, oder der Quantile, ausreichen, um sich ein Überblick über einen Datensatz zu verschaffen. Mit dem Fortschritt der Technologie und damit einhergehende Möglichkeiten große Datensätze verarbeiten zu können, hat sich auch die explorative Datenanalyse weiterentwickelt (nach Bruce et al., 2020, Exploratory Data Analysis, Kap. 1).

Tukey betont (nach Lau et al., 2021, Exploratory Data Analysis, Kap. 10), dass die Explorative Datenanalyse nicht passiv beschreibend, sondern vielmehr aktiv einschneidend sein soll, wobei der Fokus auf der Entdeckung des Unerwarteten liegen soll. Dadurch lassen sich laut Lau et al. (2021) die Daten mithilfe der Analyse nicht nur besser verstehen, sondern im weiteren Verlauf der Forschung auch die Ergebnisse kontrollieren und bestätigen. Die Explorative Datenanalyse sollte daher nicht nur als Anfang einer Forschungsarbeit zum Verständnis der

Daten verwendet werden, sondern die Forschung in jeder Lebenszyklusphase begleiten, um Ergebnisse zu kontrollieren und neue interessante Merkmale der Daten zu entdecken. Jedoch wird von den Autoren zeitgleich vor einer Überinterpretation durch die Analyse gewarnt. Mit einer Menge an Daten und genauem Begutachten lassen sich schnell interessante Zusammenhänge erkennen, die jedoch völlig falsch sein könnten. Aus diesem Grund empfiehlt sich die Veröffentlichung der Analyse und des Quellcodes, um die Forschungsergebnisse nachvollziehbar zu machen.

Laut Lau et al. (2021) spielt die Visualisierung der Daten in Form von Graphen eine zentrale Rolle in der explorativen Datenanalyse. Die Gründe dafür werden in einem späteren Kapitel näher erläutert. Für die Wahl passender Visualisierungen ist es aber zunächst hilfreich zu verstehen, welche Arten von Daten es gibt und wie sie kategorisiert werden können.

Mount (2021) betont in diesem Zusammenhang, dass die Kategorisierung von Daten in der Datenanalyse nicht strikt festgelegt ist, sondern sich je nach Autor und Problemstellung leicht unterscheiden kann. Er selbst stellt folgende Unterteilung wie in Abbildung 2.3 vor. Darüber hinaus gibt es noch spezifischere Aufteilungen mit weiteren Datentypen, welche für das Gesamtverständnis der vorliegenden Arbeit jedoch nicht relevant sind. In diesem Fall genügen die klassischen Typen, welche Mount anhand der Abbildung 2.3 im Folgenden vorstellt (Foundations of Exploratory Data Analysis, Kap. 1).

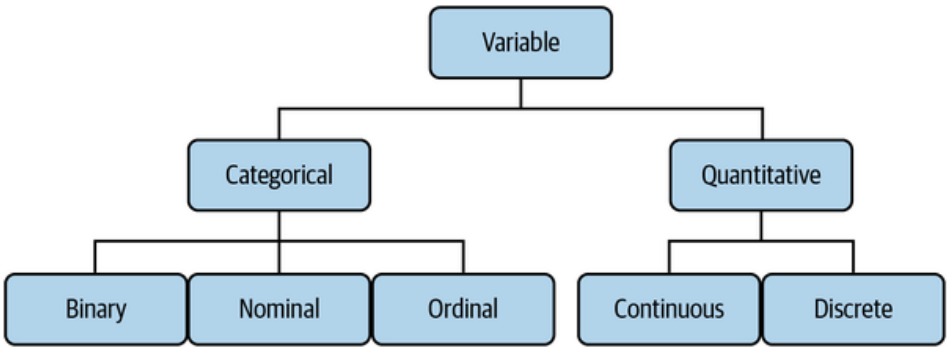


Abbildung 2.3: Daten Kategorisierung
Quelle: (Mount, 2021, Foundations of Exploratory Data Analysis, Kap. 1)

Zu den *kategorischen Datentypen* (”*Categorical*”), auch *qualitative Datentypen* genannt, werden Daten zugeordnet, die eine Charakteristik beschreiben. Die Werte sind in den meisten Fällen nicht numerisch, sondern kommen in Textform vor. Die Variabel des Herkunftslandes entspricht beispielsweise einer solchen Kategorie, da ihr Inhalt einen beschreibenden Wert annimmt. Der Wert könnte somit ”Deutschland” oder ”Österreich” sein, jedoch lässt er sich nicht quantitativ mit anderen Werten vergleichen. Aus diesem Grund sind einige Berechnungen, zum Beispiel die Berechnung des Durchschnittswertes, nicht möglich. Stattdessen bieten sich für diesen Datentyp andere Berechnungen an, wie die des am meisten vorkommenden Wertes oder die generelle Häufigkeit jedes Wertes. Generell ist es möglich, qualitative Datentypen hinsichtlich der Menge an Werten, welche sie annehmen können, zu unterteilen. Während **binäre** (”*Binary*”) Typen nur einen von ausschließlich zwei möglichen Werten annehmen, können die **nominalen** (”Nominal”) Typen einen von beliebig viele Werten annehmen. Darüber hinaus be-

achten die *ordinalen* (”*ordinal*”) Typen, neben einer beliebigen Wertannahme, außerdem eine bestimmte Reihenfolge der Werte. Ein gutes Beispiel dafür gibt der Autor mit der Größe eines Getränkes an, welche den Wert ”klein”, ”mittel” oder ”groß” annehmen kann. Oder als weiteres Beispiel nennt er die Werte der Wochentage (”Montag”, ”Dienstag”, ”Mittwoch”, etc.) . Intuitiv lassen sich die Werte in eine Reihenfolge setzen und vergleichen, auch wenn dies quantitativ nicht möglich ist (Mount, 2021, Foundations of Exploratory Data Analysis, Kap. 1).

Als *quantitative Daten* werden Daten beschrieben, die messbar sind und konkrete Werte annehmen. Sie werden fast ausschließlich durch Zahlen repräsentiert und lassen sich daher miteinander vergleichen. Hierbei wird unter *diskreten* (”*Discrete*”) und *kontinuierlichen* (”*Continuous*”) Werten unterscheiden. Diskrete Variablen können nur einen festen Wert aus einer abzählbaren Menge zwischen zwei Werten annehmen, während kontinuierliche Daten einen beliebigen Wert aus einer unendlichen Menge zwischen zwei Werten annehmen können (Mount, 2021, Foundations of Exploratory Data Analysis, Kap. 1).

Mount (2021) beschreibt die Kategorisierung der Datentypen als notwendig, um zu verstehen, welche Berechnungen mit den Daten überhaupt möglich sind und wie diese visualisiert werden können. Hierfür nutzt er ein Beispiel, worin er erklärt, dass sich kategorische Datentypen generell nicht quantitativ miteinander vergleichen lassen. Stattdessen lässt sich aber die Häufigkeit von Werten, welche dem kategorischen Datentyp angehören, quantitativ miteinander vergleichen. Durch ein solches Verständnis über die Datentypen, lassen sich diese entsprechend manipulieren, um anschließend an Informationen zu gelangen (Foundations of Exploratory Data Analysis, Kap. 1).

Laut Lau et al. (2021) ist der Datentyp nicht nur für Berechnungen relevant, sondern auch für die Wahl der Visualisierung. Die Autoren zeigen mit folgender Tabelle 2.4 den Zusammenhang von einem Datentyp und einer möglichen Wahl des Darstellungsgraphen auf (Exploratory Data Analysis, Kap. 10).

Feature type	Dimension	Plot
Quantitative	One feature	Rug plot, histogram, density curve, box plot, violin plot
Qualitative	One feature	Bar plot, dot plot, line plot, pie chart
Quantitative	Two features	Scatterplot, smooth curve, contour plot, heat map, quantile-quantile plot
Qualitative	Two features	Side-by-side bar plots, mosaic plot, overlaid lines
Mixed	Two features	Overlaid density curves, side-by-side box plots, overlaid smooth curves, quantile-quantile plot

Abbildung 2.4: Zusammenhang von Datentyp und Graph
Quelle: (Lau et al., 2021, Exploratory Data Analysis, Kap. 10)

Aus der Tabelle (siehe Abbildung 2.4) werden im Folgenden einige, für die Bachelorarbeit

relevante, Graphen genauer beschrieben.

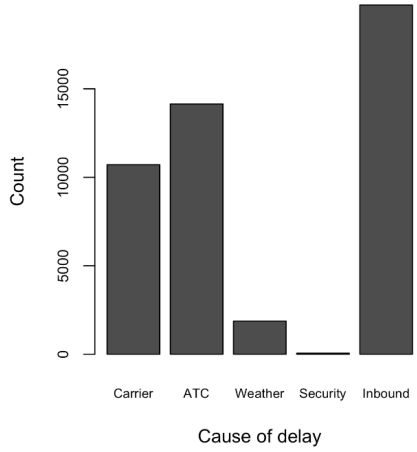


Abbildung 2.5: Balkendiagram Beispiel
Quelle:(Bruce et al., 2020, Foundations of Exploratory Data Analysis, Kap. 1)

Das *Balkendiagramm* kann sowohl horizontal als auch vertikal ausgerichtet sein und nutzt die Länge eines Balkens um den entsprechenden Wert zu repräsentieren. Je länger der Balken, desto höher der Wert. Somit können Werte von verschiedene Kategorien mit einem Blick verglichen werden (siehe Abbildung 2.5). Diese Art der Darstellung bietet sich für kategorische Datentypen an. Hierbei lässt sich die Häufigkeit, oder der prozentuale Anteil einzelner Kategorien im Datensatz berechnen und im Balkendiagramm darstellen. Die verschiedenen Kategorien werden auf einer Achse auflisten, während die andere Achse die Häufigkeit oder den Anteil der jeweiligen Kategorie, mit der Länge des Balkens, symbolisiert (Bruce et al., 2020; Lau et al., 2021; Wexler et al., 2017).

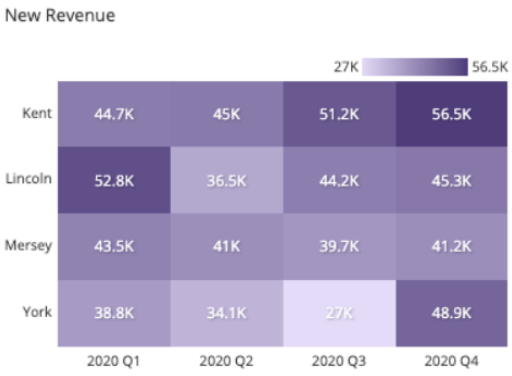


Abbildung 2.6: Wärmekarte Beispiel
Quelle: (Yi & Sapountzis, n. d.)

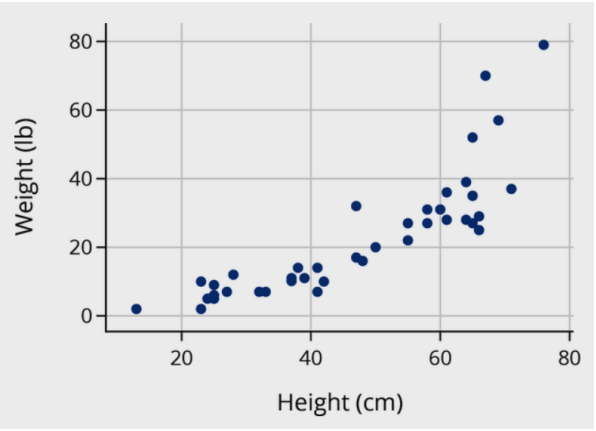


Abbildung 2.7: Scatterplot Beispiel
Quelle: (Lau et al., 2021, Data Visualization, Kap. 11)

Eine *Wärmekarte*, geläufiger als *heatmap*, zeigt ein Raster an, das durch zwei Werte gebildet wird. Die Werte werden jeweils von einer Achse repräsentiert. Die Farbe einer Zelle innerhalb des Rasters gibt den Wert an, wobei die Sättigung der Farbe steigt, je höher der Wert ist (siehe

Abbildung 2.6). Nach der Tabelle 2.4 bietet sich die Darstellung für quantitative Datentypen an, weil es hierbei möglich ist, den Zusammenhang zweier Variablen zu präsentieren. Demselben Prinzip folgt auch der bekanntere *Scatterplot*-Graph (siehe Abbildung 2.7), jedoch kann dieser bei großen Datensätzen schnell unübersichtlich werden, weshalb bei großen Datenmengen die Wärmekarte als Alternative empfohlen wird (Bruce et al., 2020; Lau et al., 2021; Yi & Sapountzis, n. d.).

Sollten die Daten geographische Informationen in Form von Längengrad und Breitengrad beinhalten, raten Lau et al. (2021) zu der Darstellung einer *Karte*, auf der die Daten präsentiert werden können. Als Beispiel zeigen die Autoren folgende Visualisierung (siehe Abbildung 2.8), in der sie die Standorte von US-Luftqualitätssensoren markieren (Data Visualization, Kap. 11).

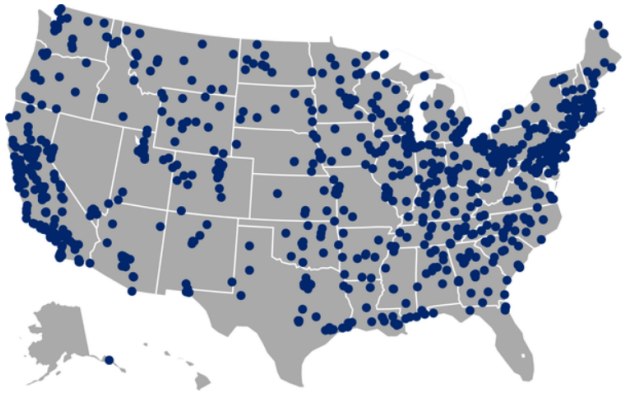


Abbildung 2.8: Kartendarstellung Beispiel
Quelle: (Lau et al., 2021, Data Visualization, Kap. 11)

Eine weitere Darstellungsform von Daten, die in der Tabelle nicht vorgestellt wird, ist die *Wortwolke*, die verwendet wird, um die Worthäufigkeit von Wörtern innerhalb eines Textes erkenntlich zu machen (vgl. Abbildung 2.9). Je größer ein Wort im Graphen geschrieben ist, desto häufiger kommt es im Text vor („Wordcloud: Definition und Funktion“, 2023).

Abbildung 2.9: Wortwolke Beispiel
Quelle: (Zhou, 2020)

2.3 Data Visualization

Da die Datenvisualisierung eine zentrale Rolle in der Explorativen Datenanalyse spielt (Lau et al., 2021), wird in diesem Kapitel das Themengebiet der Datenvisualisierung aufgegriffen und vorgestellt.

Card et al. (1999) definieren Visualisierung als das Nutzen von computerunterstützter, interaktiver, visualisierter Repräsentation von Daten zur Verstärkung der Kognition. Unter Kognition ist dabei das Erfassen und Nutzen von Wissen zu verstehen. Des Weiteren verdeutlichen die Autoren, dass das Ziel von Visualisierung nicht die Bilder als solche sind, sondern vielmehr darin besteht, einen Einblick in die Daten zu bekommen. Das Ziel dieses Einblicks ist wiederum das Entdecken, sowie die Entscheidungs- und Erklärungsfindung.

Kirk (2012) wählt einen ähnlichen Ansatz und definiert Datenvisualisierung als die Repräsentation und Präsentation von Daten, die unsere visuelle Wahrnehmungsfähigkeiten ausnutzen, um die Kognition zu verstärken. Kognition beschreibt er dabei als den Prozess, Information in Gedanken, Erkenntnisse und Wissen zu verarbeiten. Außerdem unterscheidet er zwischen Repräsentation und Präsentation der Daten. Seiner Auffassung nach stellt die Repräsentation die Art und Weise dar, wie die Daten visualisiert werden, beispielsweise welche Graphen genutzt werden, um Daten darzustellen. Die Präsentation geht einen Schritt weiter und befasst sich mit der Einbettung der Datendarstellung in die gesamte, kommunizierte Arbeit. Hierbei betont er, dass es bei Visualisierung um eine Botschaft geht, die man den Nutzer übermitteln will (Defining data visualization, Kap 1.1).

Nach Unwin (2020) ist das Ziel der Datenvisualisierung die Informationsgewinnung aus der Visualisierung. Durch diese Informationen könne Wissen angehäuft und Entscheidungen getroffen werden. Um dies möglichst effizient und effektiv zu ermöglichen, ist das Design der Visualisierung ausschlaggebend, betonen Card et al. (1999) und Kirk (2012). Kirk (2012) behauptet darüber hinaus, man kann Datenvisualisierung als eine Überschneidung der Gebiete Kunst und Wissenschaft bezeichnen. Der künstlerische Teil kommt dabei zum Tragen, wenn es um die Freiheit im Design der Visualisierung geht, in der dieses auf innovative Art und Weise als Kommunikationsmittel verwendet wird und somit auf emotionaler Ebene im Gedächtnis bleibt (The bedrock of visualization, Kap. 1.3).

Andere Autoren wie Few (n. d.) meinen wiederum, es handelt sich bei der Datenvisualisierung mehr um Wissenschaft als um Kunst, da die wissenschaftlichen Erkenntnisse über die menschlichen Wahrnehmung genutzt werden, um die Botschaft der Visualisierung auf effektive und effiziente Weise zu transportieren.

Ohne diese Meinungsverschiedenheit weiter auszuführen, beziehen sich dennoch beide Autoren auf das gleiche Konzept der visuellen Wahrnehmung. Beide betonen, dass die visuelle Auffassungsgabe des Menschen die kognitive Auffassungsgabe deutlich übertrifft. Dabei beziehen sie sich auf die präattentive, visuelle Verarbeitung. Darunter wird das unterbewusste Verarbeiten von visuellen, präattentiven Eigenschaften verstanden, das deutlich schneller abläuft als unsere bewusste Wahrnehmung. Zu diesen Eigenschaften können die Unterschiede in Länge, Größe, Farbton, Farbintensität, Winkel, Texturen, Form etc. gehören. Entsprechend ermöglicht eine kluge Einbindung dieser Eigenschaften in die Visualisierung dem Menschen, ein schnelleres und einfacheres Verständnis über die Darstellung zu erlangen, da sie nicht bewusst verarbeitet werden müssen (Few, n. d.; Kirk, 2012; „Preattentive Visual Properties and

How to Use Them in Information Visualization“, 2019).

Kirk (2012) und Few (n. d.) greifen ein weiteres Konzept der visuellen Wahrnehmung auf, in welchem die ”Gestalt School of Psychology” in den 1900er Jahre einige Gestaltprinzipien festlegt hat, welche auch heute noch anerkannt werden.

Diese Prinzipien zeigen auf, in welcher Weise unser Verstand visuelle Wahrnehmungen strukturiert und sortiert. Aus diesem Grund finden sie heute, in so gut wie allen Bereichen des Designs, praktische Anwendung, da der Designer sein Werk auf die Wahrnehmung des Betrachters abstimmen kann und somit eine gute Kommunikation garantieren kann (Riva, 2023). In der folgenden Abbildung 2.10 werden die fünf Gestaltprinzipien kurz erläutert.

Proximity	Objects that are close together are perceived as a group.	
Similarity	Objects that share similar attributes (e.g., color or shape) are perceived as a group.	
Enclosure	Objects that appear to have a boundary around them (e.g., formed by a line or area of common color) are perceived as a group.	
Closure	Open structures are perceived as closed, complete, and regular whenever there is a way that they can be reasonably interpreted as such.	
Continuity	Objects that are aligned together or appear to be a continuation of one another are perceived as a group.	
Connection	Objects that are connected (e.g., by a line) are perceived as a group.	

Abbildung 2.10: Gestaltprinzipien
Quelle: (Few, n. d.)

Aufgrund der Tatsache, dass es sich bei dieser Bachelorarbeit um ein zeitlich begrenztes Projekt handelt, wird nicht weiter auf die visuellen präattentiven Eigenschaften und die Gestaltprinzipien eingegangen. Außerdem würde dies neue Themengebiete der Psychologie aufgreifen, die über den Rahmen dieser Arbeit hinausgehen. Der kleine Einblick in diese Themen sollte lediglich zeigen, dass die Visualisierung eine sinnvolle Wahl zur Informationsvermittlung ist.

Das *Dashboard* ist im Bereich der Datenvisualisierung ein typisches Design und wird laut Smith (2013) gerne für analytische Aspekte, unter anderem von Datenwissenschaftlern, ge-

nutzt (S.27). Dabei lehnt sich Smith, in seiner Erklärung des Begriffs Dashboard, an die Definition von Stephen Few an, welcher unter dem Dashboard eine visuelle Darstellung der wichtigsten Informationen versteht, die zum Erreichen eines oder mehrerer Ziele benötigt wird. Alle Informationen sind dabei auf einem einzigen Bildschirm zusammengefasst, so dass sie auf einen Blick überwacht und verstanden werden können (S. 22). Des weiteren empfiehlt Smith für die Analyse ein interaktives Dashboard zu nutzen, da es die Untersuchung der Daten und die Entdeckung von Informationen fördert (S.27).

Die Implementierung und das Design einer Interaktivität wird durch einen weit verbreiteten und viel zitierten Artikel von Shneiderman (1996) unterstützt. Schneiderman erklärt sein Designprinzip, das ”Visuelle, Informationssuchende Mantra” mit folgendem Satz: ”Overview first, zoom and filter, then details-ondemand” (Shneiderman, 1996). Auf deutsch lässt es sich mit ”zuerst Übersicht, dann Zoom und Filter, anschließend Details auf Anfrage” übersetzen.

Hierbei erklärt Craft (2005), dass mit ”*Overview First*” ein Überblick über die Daten, als generellen Kontext und zum besseren Verständnis, geschaffen werden soll. Anschließend soll mit ”*zooming and filter*” ermöglicht werden, die Komplexität der Daten zu reduzieren und einzelne Elemente zu entfernen, beziehungsweise Andere hervorzuheben. Hierdurch kann der Fokus bewusst auf interessante Elemente gerichtet werden. Andere Elemente können wiederum aus dem Blickfeld gefiltert werden. Der Datensatz wird indirekt, oder eventuell auch direkt, verkleinert und der Überblick erhöht. Mit ”*details-ondemand*” wird anschließend versucht in der Masse an Daten einzelne Datenpunkte genauer zu investigieren, damit die Relationen zu einzelnen anderen Datenpunkte oder zum gesamten Datensatz, verstanden werden können. Die Details von Anfang an sichtbar zu machen, würde bei großen Datensätzen die Präsentation sprengen und hätte den gegenteiligen Effekt einer unübersichtlichen, schwer verständlichen Visualisierung. ”*Details-ondemand*” hilft dagegen detaillierte Informationen zu übermitteln ohne der Übersicht der Visualisierung zu schaden (Craft, 2005).

2.4 Webanwendung

Da es sich bei der vorliegenden Bachelorarbeit um die Entwicklung einer Webanwendung handelt, werden im folgenden Kapitel einige Grundlagen zum Verständnis der Webentwicklung vorgestellt, auch wenn die Webanwendung als solche thematisch nicht im Vordergrund der Arbeit steht.

Eine typische Webanwendung besteht aus drei Kernkomponenten, dem *Webbrowser*, auch *Frontend* oder *client-side* genannt, dem *Webserver*, der häufiger unter den Namen *Backend* oder *server-side* zu finden ist und zuletzt dem *Datenbankserver*. Im Rahmen dieser Bachelorarbeit wurde eine *Zwei-Schichten-Architektur* implementiert, bei der es keinen eigenen Datenbankserver gibt, sondern die Daten entweder im Frontend oder Backend verarbeitet werden. Aus diesem Grund werden in den folgenden Abschnitten die ersten beiden Kernbestandteile genauer ausgeführt, während auf den Datenbankserver nicht weiter eingegangen wird (William, 2022).

2.4.1 Frontend

Der Webbrowser ist für die Kommunikation mit dem Nutzer zuständig. Über den Browser wird ihm eine visualisierte Benutzeroberfläche dargeboten, mit der er interagieren kann. Die Logik der spezifischen Datenrepräsentation findet hier ihren Platz (William, 2022). Dabei sind *HTML* (*HyperText Markup Language*) und *Javascript* der Grundbaustein für nahezu jedes Frontend („Usage statistics of HTML for websites“, 2023; „Usage statistics of JavaScript“, 2023).

Bei HTML handelt es sich um eine von Tim Berners-Lee entwickelte Auszeichnungssprache (*markup language*), die schon zu Beginn des Internets dazu diente eine Webseite zu strukturieren und zu formatieren. Hierfür werden Abschnitte der Webseite mit sogenannten ”*Tags*” gekennzeichnet, wodurch ihr einer entsprechende Struktur zugewiesen wird. Meist kommt HTML in Kombination mit *CSS* (*Cascading Style Sheets*) vor, wodurch der Struktur der Webseite zusätzlich eine visuell gestaltete Ansicht beigefügt wird (Beal, 2021).

Javascript fügt der ganzen Webseite eine logische Komponente hinzu und macht das Frontend dynamisch anpassbar. Dadurch können auf Anfragen des Nutzers reagiert werden und der HTML/CSS angepasst werden. Aufgrund der Tatsache, dass Javascript keiner Firma gehört, existiert eine Vielzahl an ”*open source*” Projekten. Durch den Packet Manager *Node Package Manager*[1] (NPM) kann jeder sogenannte Pakete erstellen, veröffentlichen und auf andere bereits Veröffentlichte zugreifen. Die Pakete können von kleineren Bibliotheken mit einzelnen Funktionen bis zu großen ”Frameworks” mit zahlreichen Bibliotheken reichen, die eine bestimme Aufgabe erfüllen (Alshaikhly, 2020; Gupta, 2020; Zacharias, 2022).

Die Kommunikation zwischen Frontend und Backend findet durch das *HyperText Transfer Protocol* (HTTP) statt. Hierfür initiiert der Nutzer über das Nutzen des Frontends im Webbrowser einen *HTTP-Request*. Die Anfrage wird von dem Server empfangen, bearbeitet und die Ergebnisse in einer *HTTP-Response* zurückgeschickt. Die Anfrage besteht aus folgenden Komponenten (Mustapha, 2023):

1. URL
2. HTTP-Methode
3. Liste von *Headers*
4. *Request-Body*

Bei *Uniform Resource Locator* (URL) handelt es sich um eine exakte Adresse zu einer Ressource im Internet. Hauptsächlich besteht diese aus einer *Domain*, die die Adresse des Webservers angibt (z.B. ”www.example.com”), sowie einem *Pfad*, der den genauen Standort der Ressource auf dem Webserver anzeigt (MDN contributors, 2023).

Die HTTP-Methode gibt darüber hinaus an, welche Operation mit der Anfrage auf dem Backend ausgeführt werden soll. In Abbildung 2.11 werden alle vorhanden Methoden aufgelistet und kurz erläutert. Des Weiteren kann ein *HTTP-Request* optional auch eine Liste von *Headers* mit zusätzliche Informationen über die Anfrage enthalten. Ein Header besteht dabei immer aus einem Paar von Namen und Wert, die durch ein Doppelpunkt getrennt sind. Ebenfalls optional ist der *Request-Body*, dessen Inhalt die Daten sind, die der Nutzer an den Server sendet (Mustapha, 2023).

[1] https://www.npmjs.com/

HTTP METHODS	DEFINITION
HEAD	Asks the server for status (size, availability) of a resource.
GET	Asks the server to retrieve a resource.
POST	Asks the server to create a new resource.
PUT	Asks the server to edit/update an existing resource.
DELETE	Asks the server to delete a resource.

Abbildung 2.11: HTTP Methoden
Quelle (Mustapha, 2023)

Mit diesen vier Komponenten kann ein *HTTP-Request* erstellt werden und sieht nach dem Beispiel von Mustapha (2023) folgendermaßen aus:

```
1 GET /watch?v=8PoQpnlBXD0 HTTP/1.1
2 Host: www.youtube.com
3 Cookie: GPS=1; VISITOR_INFO1_LIVE=kOe2UTUyPmw; YSC=Jt6s9YVWMd4
```

GET stellt dabei die HTTP-Methode dar, gefolgt von dem Pfad, der den Webserver nach einer Ressource auf dem entsprechenden Pfad fragt. ”HTTP/1.1” gibt zusätzlich an, um welche Version des Protokolls es sich handelt. Anschließend wird die Adresse des Zielservers, auch als Host bezeichnet, genannt (”www.youtube.com”). In diesem Beispiel gibt es zusätzlich ein *Cookie-Header*, der weitere Informationen enthält (Mustapha, 2023).

Für das Erstellen und Versenden solcher HTTP-Anfragen bietet Javascript die Funktion *”fetch()”* an, die von allen modernen Webbrowsern unterstützt wird. Es muss mindestens ein Argument enthalten, welches der URL Adresse, das Ziel der Anfrage, entspricht. Des Weiteren können Optionen als zweites Argument angegeben werden, wodurch sich die zuvor vorgestellten Komponenten der HTTP-Anfrage genauer spezifizieren lassen. Sind keine weiteren Optionen genannt, so handelt es sich um eine HTTP-Anfrage der GET-Methode, die dazu aufruft den Inhalt der URL herunterzuladen und zurückzugeben („Fetch“, 2022; Ubah, 2021).

Bei *fetch()* handelt es sich um eine asynchrone Funktion, die ein *Promise* als Rückgabewert enthält. Asynchrone Funktionen werden parallel zum eigentlichen Programmcode ausgeführt, sodass das Programm weiter exekutiert wird ohne auf die Fertigstellung der asynchronen Funktion zu warten. Aus diesem Grund ist der *Promise* hilfreich. Konkret handelt es sich bei dem *Promise* um ein Objekt, das noch auf einen Wert wartet. Dabei kann es drei Zustände einnehmen: *Pending*, *Fulfilled* und *Rejected*. *Pending* gibt bekannt, dass weiterhin auf die Fertigstellung der Anfrage gewartet wird. Der *Fulfilled*- oder der *Rejected*-Zustand gibt an, ob die Anfrage entsprechend erfolgreich oder fehlgeschlagen ist (Mustapha, 2023; Ubah, 2021; Zlatkov, 2017).

Letztendlich ist nicht bekannt wie lange die Anfrage dauert und ab wann es möglich ist auf

den Rückgabewert vom Server zuzugreifen. In diesem Zusammenhang gibt es zwei Optionen. Zu einem ist es möglich der asynchronen Funktionen ein *Await*-Ausdruck hinzuzufügen. Dadurch wird das Programm gezwungen auf die Durchführung und der Lieferung der Ergebnisse der asynchronen Funktion zu warten, bevor der weitere Quellcode ausgeführt wird. Streng genommen handelt es sich dabei nicht mehr um eine asynchrone Funktion, da sie gezwungen wird synchron ausgeführt zu werden. Zum anderen kann die Funktion asynchron bleiben und stattdessen durch den Ausdruck *.then()* der Funktion mitteilen was ausgeführt werden soll, sobald der *Promise* den Zustand *Fulfilled* annimmt. Analog dazu wird mit *.catch()* der Umgang mit einem *Rejected*-Zustand mitgeteilt (Mustapha, 2023; Zlatkov, 2017).

Sobald der *Promise* erfolgreich war, ist es möglich die Daten abzugreifen. Der Inhalt eines *HTTP-Response* enthält mehr Informationen als nur die gesuchten Daten, jedoch sind diese zusätzlichen Informationen für die vorliegende Arbeit nicht relevant. Um auf die angefragten Daten zuzugreifen bietet Javascript einige Funktionen an, wodurch die Resultate in unterschiedlichen Formaten herausgegeben werden können. Relevant sind im Rahmen der Arbeit die Funktionen *response.text()* und *response.json()*, welche die Daten des *Promise-Response* im Text-Format oder im *JavaScript Object Notation (JSON)*-Format abgreifen („Fetch“, 2022).

Bei JSON handelt es sich um ein programmiersprachenunabhängiges Datenformat, das für den Datenaustausch gedacht ist. Es ist für Menschen gut lesbar und folgt zeitgleich einer Konvention, die von vielen Programmiersprachen verwendet wird. Ein JSON-Objekt besteht aus einer Menge von Paaren, die jeweils aus einem Namen und einem Wert bestehen. Der Name ist immer in Form eines *Strings* gegeben, während der Wert alle gängigen Typen einer gewöhnlichen Programmiersprache annehmen kann. Zudem kann der Wert ebenfalls ein weiteres JSON-Objekt sein. Hierbei folgt dem Namen ein Doppelpunkt und anschließend der zugewiesene Wert. Generell werden die einzelnen Paare in der Menge durch ein Komma getrennt. Die Struktur von Paaren aus Name und Wert lässt sich in vielen Programmiersprachen gleichermaßen finden und ist daher nicht unbekannt, weshalb sich JSON als Datenaustauschformat eignet. Im Quellcode lassen sich die Werte mithilfe des zugewiesenen Namens abrufen und operieren („Einführung in JSON“, n. d.).

2.4.2 Backend

Im vorangehenden Abschnitt wurde das Frontend und deren Funktion betrachtet. Des Weiteren wurde auf die Kommunikation des Frontend mit dem Server eingegangen und darauf, wie der Nutzer Anfragen an das Backend verschickt. Daran anschließend wird im folgenden Abschnitt betrachtet wie der Server diese Anfragen empfängt, bearbeitet und beantwortet.

Wie bereits erwähnt ist das Backend für Funktion der Anwendung zuständig. Hier werden Daten empfangen und verwaltet, sowie Operationen ausgeführt und an das Frontend zurückgeschickt. Der Server kann in einer beliebigen Programmiersprache entwickelt werden. Doch um den Aufwand zu reduzieren und um nicht alle Funktionen von Grund auf neu zu entwickeln, besitzen viele Programmiersprachen *Frameworks*, die bereits eine Vielzahl an Werkzeugen und Strukturen anbieten, um die Webserver-Entwicklung drastisch zu vereinfachen (Kiefer, 2023; William, 2022).

Für die Kommunikation mit dem Frontend benötigt der Backend-Server ein *application programming interface* (API). Die API regelt, welche *requests* empfangen werden dürfen und wie

sie beantwortet werden sollen. Hierfür werden von der API sogenannte *endpoints* zur Verfügung gestellt, die explizit einen Pfad des Servers angeben auf den zugegriffen werden darf. Das Frontend kann, wie zuvor erklärt, durch eine URL auf einen solchen *endpoint* zugreifen und eventuell nach einer Ressource fragen. Anschließend bestimmt die API wie auf den *request* reagiert wird und sendet eine *HTTP-response* als Antwort. Die Anfrage kann dabei angenommen oder auch abgelehnt werden. Im Fall eines erfolgreichen *requests* wird dieser bearbeitet und gegebenenfalls eine Ressource zurückgesendet (Juviler, 2023).

Im diesem Kapitel wurden die Problemläsung auf das Themengebiet der Datenvisualisierung und der explorativen Datenanalyse eingegrenzt. Im folgenden Kapitel wird die Implementierung zur Beantwortung der Forschungsfrage und Lösung des Problems vorgestellt

3 Implementation

In diesem Kapitel wird auf die Implementierung des Dashboardes als alternative Darstellungsform des Datensatzes der Metasuchmaschine IWWB vorgestellt. Hierfür wird im Vorfeld das angestrebte Design und Ziel erklärt und anschließend die Architektur der Anwendung beschrieben. Anschließend werden verwendete Technologien vorgestellt und deren Grundlagen zum Verständnis der Funktionsweise des Quellcodes dargelegt. Daraufhin wird das bereitgestellt Datensatz von IWWB analysiert und dessen Struktur offengelegt. Letztendlich werden relevante Implementierung der Webanwendung dokumentiert.

3.1 Design und Ziel

Mit der Implementierung des Dashboards wurde der Versuch unternommen, die im vorangehenden Kapitel erlangten wissenschaftlichen Erkenntnisse umzusetzen, um den Nutzern eine möglichst effektive und effiziente Informationsvermittlung der Daten zu bieten. Durch eine simple Gestaltung und passende Beschreibungen wurde angestrebt, dass die Inhalte sich selbst erklären und somit jeder Nutzer eigene Schlüsse aus den Daten ziehen kann.

Eine Karte, auf der die Standorte aller Kurse präsentiert werden, bildet dabei den Kern der Seite. Gleichzeitig wurde versucht die Kurse in Kategorien zu klassifizieren und diese farblich zu kennzeichnen. Zuletzt sollten einige simple Graphen, die in der explorativen Datenanalyse gängig sind, implementiert werden, um einen allgemeinen Überblick über die Daten zu geben (siehe Abbildung 3.1).

Das Dashboard wurde interaktiv gestaltet, sodass der Informationsfluss kontrolliert werden kann und die Übersicht nicht verloren geht. Zum einen soll es dahingehend möglich sein ausschließlich die Kursdaten aus einzelnen Jahren anzuzeigen. Zum anderen werden durch Klicken und Anvisieren der Graphen zusätzliche Informationen angezeigt, die sonst verborgen bleiben.

3.2 Architektur

Die Webanwendung wurde wie üblich in Frontend und Backend unterteilt. Das Backend beinhaltet den Datensatz, führt die Berechnungen der Daten aus und speichert diese, bereits für die Visualisierungen präpariert, ab. Im Frontend werden die benötigten Resultate der jeweiligen Graphen abgerufen und visualisiert. Somit findet die Berechnung eines Datensatzes im Ba-

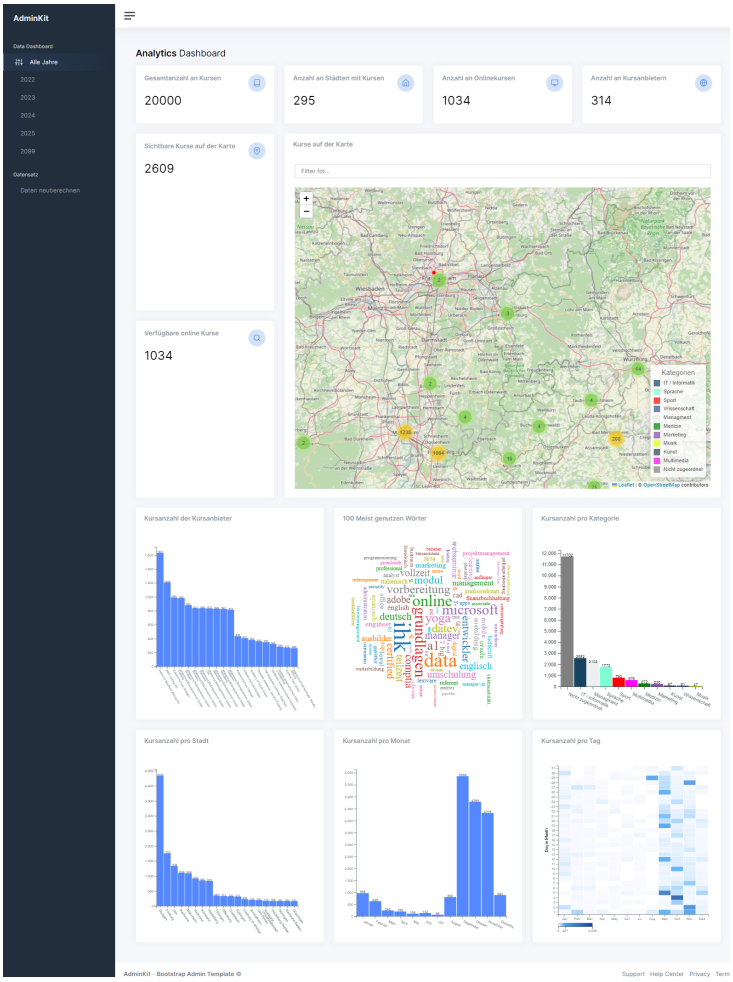


Abbildung 3.1: Gesamtbild des entwickelten Dashboards.

ckend einmalig statt, wodurch Resultate in einzelne Dateien abgespeichert werden. Anschließend greift die Webseite lediglich auf die bereits berechneten Ergebnisse zu. Durch die beschriebene Architektur lässt sich die Ladezeit drastisch reduzieren und ermöglicht auf diesem Weg eine normale Nutzung der Webseite. Aufgrund der Tatsache, dass es sich um statische Daten handelt, die sich während der Laufzeit nicht ändern, stellte sich diese Vorgehensweise als unproblematisch heraus. Allerdings muss dem Betreiber der Webseite bewusst sein, dass eine Neuberechnung notwendig ist, sobald sich der Datensatz ändert.

3.3 Verwendete Technologien

3.3.1 Frontend

Das Frontend basiert auf einem Dashboard Template [1], das nicht der eigenen Arbeit entspringt. Da das Template lediglich der visuellen Strukturierung der Webseite dient und keine Funktionalität beinhaltet wird darauf nicht weiter eingegangen. Alle relevanten Implementierungen, die in das Template eingebettet werden, werden im Laufe dieses Kapitels erläutert.

3.3.2 React

React[2] ist eine JavaScript-Bibliothek zur Erstellung einer Benutzeroberfläche, dessen Konzept das Implementieren von einzelnen Komponenten fördert. Diese Komponenten kombinieren JavaScript-Logik, HTML-Strukturen und in manchen Fällen sogar CSS-Stile. Vereint werden die einzelnen Komponenten schlussendlich zu einer gesamten Webseite (Schwarzmüller, 2022, What is a React Component, Kap. 2).

Weil es sich bei React um eine große Bibliothek mit zahlreichen Funktionen handelt, ist es im Rahmen dieser Arbeit nicht möglich die komplette Funktionsweise zu erläutern. Stattdessen werden im Folgenden einige Konzepte von React erläutert, die für diese Arbeit relevant sind. Für einen tieferen Einblick gibt es eine beträchtliche Anzahl an Dokumentationen und Anleitungen im Internet.

Grundsätzlich kann eine React Komponente auf zwei Arten realisiert werden. Zum einem als ein *Class Component* und zum anderen als ein *Function Component*. Für die vorliegende Arbeit von Bedeutung ist jedoch nur das letztere. *Function Components* bestehen aus einem Funktionsblock in der die Logik der Komponente implementiert wird, sowie aus einem HTML-Rückgabe Block, das bestimmt wie die Komponente letztendlich auf der Webseite angezeigt werden soll („React Components“, n. d.). Die folgende Beispielfunktion gibt React mit ihrer Anleitung auf ihrer Webseite an („Add React to an Existing Project“, n. d.).

```
function NavigationBar() {
  // TODO: Actually implement a navigation bar
  return <h1>Hello from React!</h1>;
}
```

Außerdem lassen sich die Komponenten einfach in den HTML-Quellcode einer Webseite einbetten, wie es React in derselben Anleitung zeigt.

```
import { createRoot } from 'react-dom/client';

function NavigationBar() {
  // TODO: Actually implement a navigation bar
  return <h1>Hello from React!</h1>;
}

const domNode = document.getElementById('navigation');
const root = createRoot(domNode);
```

[1] Quelle des Templates: https://adminkit.io/
[2] https://react.dev/

```
10 root.render(<NavigationBar />);
```

Mit den letzten drei Zeilen wird ein HTML-Element mit der ID "navigation"der React-Komponente "NavigationBar()ßugewiesen. Anschließend lässt sich die Komponente durch folgenden Quellcode an einer beliebigen Stelle im HTML einfügen. Der HTML-Teil der Komponente wird letztendlich innerhalb des HTML-Blocks mit der entsprechenden ID angezeigt („Add React to an Existing Project“, n. d.).

```
1 <!-- ... somewhere in your html ... -->
2 <nav id="navigation"></nav>
3 <!-- ... more html ... -->
```

Zu einem weiteren wichtigen Konzept von React gehören die sogenannten *Hooks*. Es sind spezielle Funktionen, die nur innerhalb von React-Komponenten zur Verfügung stehen. Zwei *Hooks* werden in diesem Rahmen genauer betrachtet. Zu einem der *useState() Hook* und zum anderen der *üseEffect() Hook"*. *useState()* ermöglicht es einer Variabel Daten zuzuweisen, welche React dazu bringen die Komponente zu *re-rendern*, sobald sich diese Daten ändern. *Re-rendern* bedeutet in dem Zusammenhang, dass der HTML-Block der Komponente neu erstellt wird, falls notwendig. Die Anzeige im Webbrowser wird dementsprechend an die veränderten Daten angepasst. Zusammengefasst heißt das, dass sich die Seite Aktualisiert sobald sich die *useState*-Variable ändert (Schwarzmüller, 2022, Working with Events and State, Kap. 4).

Darüber hinaus zeigt Maximilian Schwarzmüller Schwarzmüller (2022) ein häufig vorkommendes Problem auf, welches sich mit dem *useEffect() Hook* lösen lässt. Im Frontend werden üblicherweise Daten von unterschiedlichen Quellen abgerufen, die anschließend visualisiert werden sollen. Wenn hierfür ausschließlich der *useState() Hook* genutzt wird, kommt es zu dem Problem, dass die beschafften Daten der *useState()* Variabel zugewiesen werden und somit ein *re-render* auslösen. Die Neuberechnung der Komponente führt aber zu einem erneuten Abruf der Daten, welche schließlich wieder der *useState()* Variabel zugewiesen werden. Dadurch entsteht eine unendliche Schleife, die andauernd Daten abruft und die Komponente neu berechnet. Mithilfe vom *”useEffect() Hook”* lässt sich das verhindern (Handling Side Effects, Kap. 8).

useEffect() erhält wie im folgenden Quellcode zwei Argumente. Zum einen eine Funktion, die von *useEffect()* ausgeführt werden soll und zum anderen ein optionales Argument, das aus einer Liste von Variablen besteht (Schwarzmüller, 2022, Handling Side Effects, Kap. 8):

```
1 useEffect(function () {
2
3     // Funktions Block
4
5 }, []);
```

Die Funktion im ersten Argument wird vom *”useEffect() Hook”* nach jedem Render-Zyklus von React durchgeführt. Ist jedoch eine Liste im zweiten Argument vorhanden, so wird die *”useEffect()”* Funktion nicht nach jedem Rendern ausgeführt, sondern nur beim ersten Renderzyklus und bei jedem Weiteren, wenn mindestens eine Variabel innerhalb der Liste verändert wurde. Ist die Liste leer, so wird die Funktion nur ein einziges mal bei dem ersten Render-Zyklus der Webseite durchgeführt. Durch diese Eigenschaft können Daten innerhalb der *”useEffect()”* abgerufen werden und anschließend mit *”useState()”* visualisiert werden. Durch eine geeignete Wahl an Variablen als zweites Argument oder einer leeren Liste lässt sich eine unendliche

Schleife verhindern (Schwarzmüller, 2022, Handling Side Effects, Kap. 8).

3.3.3 Leaflet

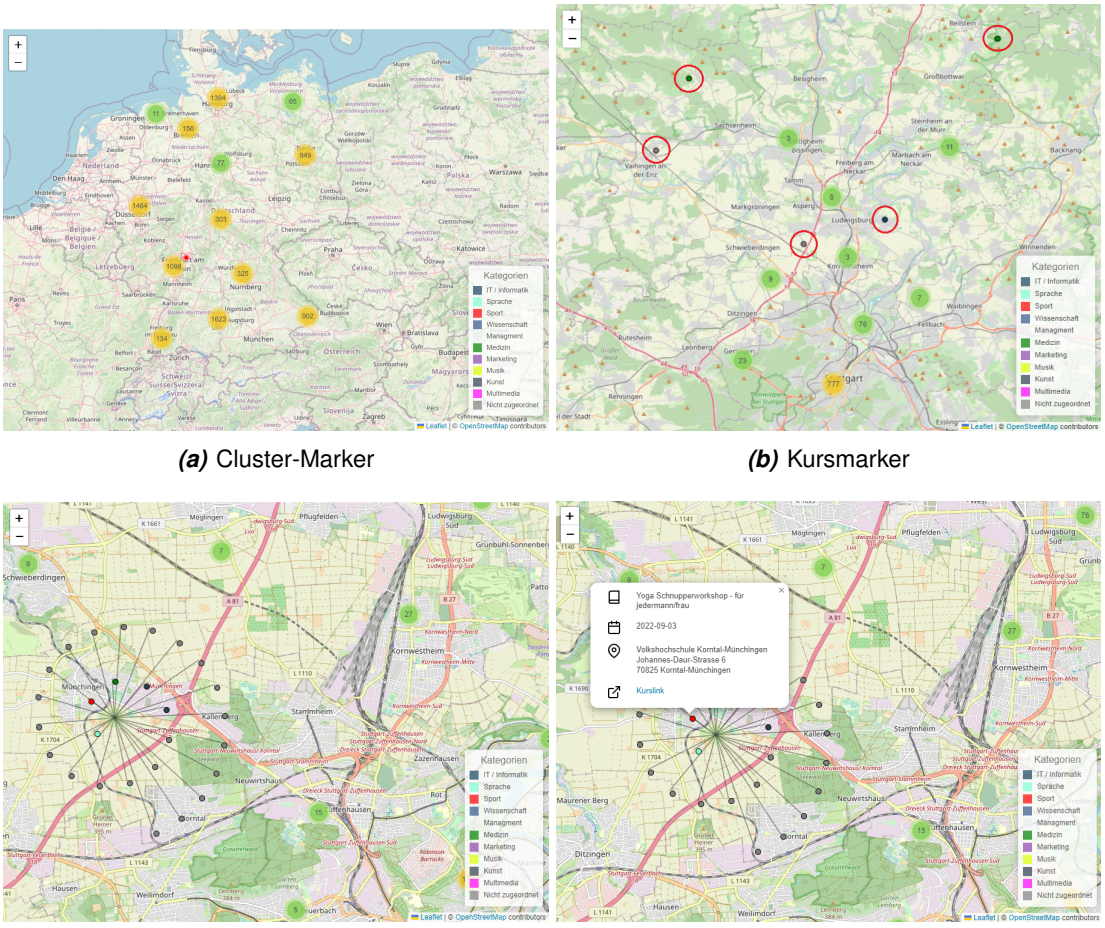
Die Karte wurde mithilfe der JavaScript-Bibliothek *Leaflet* [3] erstellt. Wie sie auf ihrer Website schreiben, handelt es sich dabei um eine *Open Source*-Bibliothek für die Implementierung einer interaktiven Weltkarte. Leaflet bietet zahlreiche Möglichkeiten die Karte zu modifizieren und ist zudem gut dokumentiert [4]. Aus diesem Grund wird auf die Erstellung und Konfigurierung der Karte nicht ins Detail eingegangen. Stattdessen wird lediglich an gegebener Stell ein Überblick gegeben, um die Gesamtfunktion der Anwendung und die Modifikation der Karte nachvollziehen zu können.

Neben Leaflet wurde das *Plugin*-Packet "*leaflet.markercluster*[5]" verwendet. Während es Leaflet ermöglicht Markierungen auf der Karte hinzuzufügen, ist es mit dem Plugin möglich die Marker in sogenannte *Cluster* zu unterteilen. Hierbei werden nahe beieinander liegende Marker zu einem einzigen *Cluster*-Marker zusammengefügt. Auf diesen *Cluster* wird die Anzahl an zusammengefügten Markern angezeigt. Des weiteren verändert sich die Farbe und Größe des *Cluster*-Markers mit zunehmender Anzahl. In Abbildung 3.2(a) sind mehrere solcher *Cluster* zu erkennen. Dem Nutzer ist es möglich einen solchen Marker anzuklicken, wodurch er sich in mehrere kleinere Cluster-Marker unterteilt. Dieser Prozess kann so oft wiederholt werden bis letztendlich die, mit Leaflet erstellten, Kursmarker angezeigt werden. In Abbildung 3.2(b) wurden einige solcher Leaflet-Kursmarker mit roten Kreisen markiert. Befinden sich jedoch mehrere Kurse auf der exakt selben Position, so expandiert der *Cluster*-Marker beim Anklicken spiralförmig und zeigt auf diese Weise, wie in Abbildung 3.2(c) zu sehen ist, alle einzelnen Kursmarker. Zuletzt ist es auch möglich einen einzelnen Kursmarker anzuklicken, wodurch ein mit Leaflet erstelltes *Popup* erscheint, der exakte Informationen über diesen Kurs preisgibt (siehe Abbildung 3.2(d)).

3.3.4 D3js

Die restlichen Graphen wurden mithilfe der *Open-Source* JavaScript-Bibliothek **D3js** [6] erstellt. Der Zweck von D3 ist die Visualisierung von Daten auf Internetseiten („What is D3?“, n. d.). Da D3js ebenfalls gut dokumentiert ist, wird auch hier nicht ins Detail, bezüglich der Erstellung von Graphen, eingegangen, sondern auf Ressourcen im Internet verwiesen. Hierfür bietet sich beispielsweise die Webseite "D3.js graph gallery" [7], als gute Anlaufstelle für die Erstellung unterschiedlicher Diagrammtypen an. Die Implementierung der Balkendiagramme und der Wärmekarte basieren ebenfalls auf diesen Anleitungen.

[3] https://leafletjs.com/
[4] https://leafletjs.com/reference.html
[5] https://www.npmjs.com/package/leaflet.markercluster
[6] https://d3js.org/
[7] https://d3-graph-gallery.com/index.html



(a) Cluster-Marker ***(b)*** Kursmarker

(c) Spirale Expansion ***(d)*** Kursinformation

Abbildung 3.2: Funktion vom ”leaflet.markercluster” Paket

3.3.5 Backend

Für die Implementierung des Servers wurde sich für das Framework *Flask*[8] mit der Programmiersprache *Python*[9] entschieden. Bei Flask handelt es sich um einen *Mikro-Framework*. Es ist minimalistisch aufgebaut und besonders für kleine Webanwendungen geeignet (Kiefer, 2023).

Python ist eine *Open Source*-Programmiersprache, die 1991 von Guido Van Rossum entwickelt wurde. Gerade wegen ihrer einfachen Syntax, ist Python schnell verständlich und gut lesbar. Zusätzlich bietet Python eine große Standardbibliothek [10] an, die eine Vielzahl an Funktionen anbietet und für jeden zugänglich und verwendbar ist. Des weiteren ist Python in dem Bereich der Datenanalyse und *Data Science* weit verbreitet (Scarlett, 2023).

Im vorherigen Kapitel wurde bereits die Funktionsweise einer API und der bereitgestellten *endpoints* präsentiert. Flask stellt für die Erstellung solcher *endpoints* den Ausdruck ”@app.route(/

[8] https://flask.palletsprojects.com/en/2.3.x/
[9] https://www.python.org/
[10] https://docs.python.org/3/library/

")" zur Verfügung. Dadurch wird der Pfad innerhalb der Anführungszeichen auf dem Webserver bereitgestellt. Anschließend wird unterhalb dieses Ausdrucks eine Funktion definiert, die bestimmt was getan und zurückgeschickt werden soll, sobald der entsprechende Pfad ein *HTTP-Request* empfängt. Insgesamt sieht das wie im folgenden Beispiel aus (Birchard, 2020):

```
@app.route("/pfad", methods=['GET', 'POST'])
def hello():
    return "Some Data!"
```

Mit einem *HTTP-Request*, adressiert an den Webserver, und dem angegebenen Pfad, wird die Funktion *hello()* aufgerufen. Zusätzlich lässt sich angeben für welche HTTP-Methoden die Anfragen zugelassen werden sollen. Ist keine HTTP-Methode angegeben so werden standardgemäß lediglich *GET-Requests* zugelassen (Birchard, 2020).

3.4 Datensatz

Bevor auf relevante Implementierungen eingegangen wird, wird der gegebene Datensatz vorgestellt. Bei dem Datensatz handelt es sich um eine JSON-Datei. Innerhalb dieser Datei gibt es eine Liste mit JSON-Objekten, die jeweils einen Kurs repräsentieren. Ein Kurs-Objekt sieht folgendermaßen aus und bietet folgende Daten an:

```
{
    "ID": "",
    "Zulieferername": "",
    "ZuliefererAnbieterid": "",
    "Zuliefererid": "",
    "Kurslink": "",
    "Anbieterid": "",
    "Veranstaltungstyp": "",
    "Kurstitel": "",
    "Kursbeschreibung": "",
    "Kursbeginn": "",
    "Schlagwort": "",
    "KursPLZ": "",
    "Kursstadt": "",
    "Longitude": "",
    "Latitude": "",
    "Kursstrasse": "",
    "Veranstaltername": "",
    "Anbieterlink": "",
    "Zuliefererlink": "",
    "Anbietername": "",
    "Anbieterstrasse": "",
    "AnbieterPLZ": "",
    "Anbieterstadt": "",
    "Bundesland": "",
    "Anbieterland": "",
    "Anbietertelefon1": "",
    "Anbietertelefon2": "",
    "Anbieterfax": "",
    "Anbieteremail1": "",
    "Anbieteremail2": "",
    "Anbieterurl1": "",
```

```
33     "Anbieterurl2": "",
34     "Anbieterkommentar": "",
35     "Anbieterkontakt": "",
36     "istLehrerfortbildung": "",
37     "istBildungsurlaub": ""
38 }
```

In der Auflistung oben handelt es sich um ein Beispiel bei dem alle Werte jedes Kursattributes einemleeren *String* entsprechen. Im Normalfall sind die Werte jedoch mit sinnvollen angaben gefüllt, doch in manchen Fällen sind einige Attribute leer oder mit einem Nullwert angegeben. Die leeren Werte werden bei den in den Berechnungen herausgefiltert, da es ansonsten zu einer Fehlermeldung kommt.

3.5 Implementierung des Frontend

In diesem Abschnitt wird die Implementierung des Frontends erläutert. Es ist zuständig für die Visualisierung der Daten in verschiedenen Graphen.

Alle relevanten JavaScript-Dateien für die Implementierung der Graphen befinden sich in dem Ordner unter dem Pfad ”frontend\src\js\components” (siehe Abbildung 3.3). Jede Datei entspricht einer React-Komponente, die ein Element des Dashboards visualisiert.

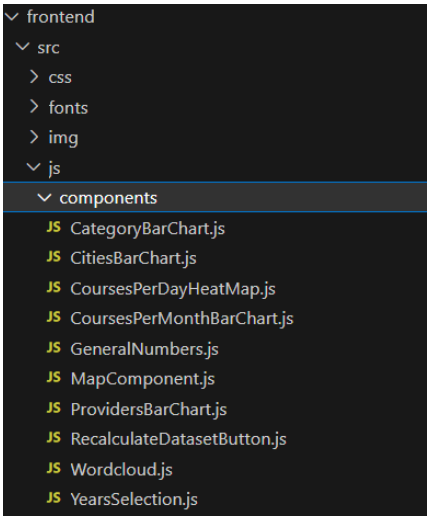


Abbildung 3.3: Pfad der Implementierten Komponenten.

Um die Erläuterung der Entwicklung übersichtlich zu gestalten, werden die Komponenten aus Abbildung 3.3 nacheinander betrachtet und erklärt.

3.5.1 Jahresauswahl

Die Komponente ”YearsSelection.js’ ist für die Jahresauswahl im Navigationsmenü zuständig (siehe Abbildung 3.4). Um das zu visualisierende Jahr mit den anderen Graphen zu kommunizieren, wird die Auswahl im *Session Storage* gespeichert, sodass sie von den anderen Kompo-

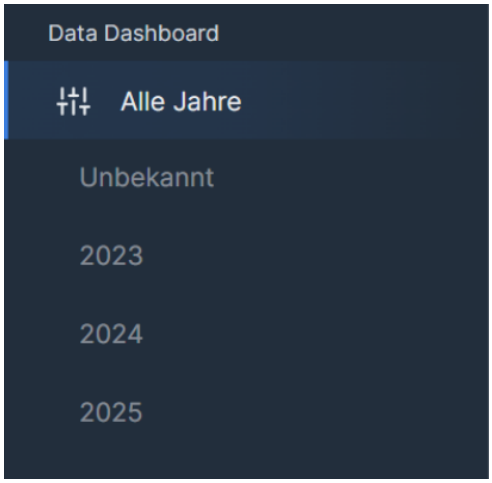


Abbildung 3.4: YearsSelection.js

nenten aus dem *Storage* abgerufen werden kann. Als Startwert beim Aufrufen der Seite wird das aktuelle Jahr festgelegt und im *Storage* gespeichert

Um die Jahreszahlen dynamisch an den Datensatz anzupassen, wird zu Beginn eine Anfrage an den Server gesendet, der eine Liste zurückgibt, worin die im Datensatz vorkommenden Jahreszahlen aufgelistet werden. Eine solche Liste könnte folgendermaßen aussehen.

```
["2019", "2022", "2023", "2024"]
```

Kein Jahr kommt dabei doppelt vor und im Datensatz nicht vorhandene Jahreszahlen kommen gar nicht vor. Anschließend wird in dem HTML-Rückgabeblock von React für jedes Element innerhalb dieser Liste ein Knopf erstellt. Hierfür wird, wie im folgenden Quellcode beispielhaft gezeigt, die ”map”-Funktion einer Liste genutzt.

```
return (
    ...

  { datasetsYear.map(element => window.year == element ?
        <li className="sidebar-item active" key={element}>
            <a className="sidebar-link" onClick={() => {
                    window.sessionStorage.setItem('year', element);
                    window.year = element;
                    location.reload();
                }}>
                <i className="align-middle" data-feather="sliders"></i>
                <span>{element < 2000? "Unbekannt": element}</span>
            </a>
        </li>
    :
        <li className="sidebar-item" key={element}>
            <a className="sidebar-link" onClick={() => {
                    window.sessionStorage.setItem('year', element);
                    window.year = element;
                    location.reload();
                }}>
```

```
                <i className="align-middle" data-feather="sliders"></i>
                <span>{element < 2000? "Unbekannt": element}</span>
            </a>
        </li>)
  }
    ...
)
```

Für jedes Element in der Liste wird vorerst geprüft, ob es sich bei dem Element um das im *Session Storage* gespeicherte Jahr handelt. Ist dies der Fall wird ein Knopf erstellt, der wie in Abbildung 3.4 durch eine leichte blaue Färbung hervorgehoben wird. Im anderen Fall wird ein simpler Knopf ohne Hervorhebung erstellt. In beiden Fällen werden beim Anklicken des Knopfes die gleichen Funktionen aufgerufen. Zuerst wird das neue Jahr im *Session Storage* gespeichert und anschließend die Seite neu geladen. Alle Graphen greifen damit auf das neue Jahr im *Storage* zu und zeigen infolgedessen ausschließlich die Daten der neuen Auswahl an.

3.5.2 Generelle Daten

Bei ”GeneralNumbers.js” handelt es sich um die zweite Komponente, die nun betrachtet wird. Wie in Abbildung 3.5 zu sehen ist, visualisiert diese Komponente vier verschiedene Zahlen, die jeweils eine Auskunft über den gesamten Datensatz geben.

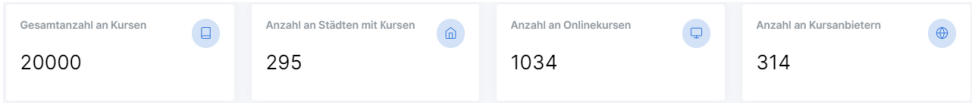


Abbildung 3.5: GeneralNumbers.js

Der Ausgangspunkt der Komponente ist folgende ”useEffect()”-Funktion, die eine leere Liste als zweites Argument hat. Es wurde bereits erklärt, dass die die Funktion dadurch im ersten Argument nur ein einziges mal durchgeführt wird. Innerhalb dieser Funktion werden die benötigten Daten vom Server angefragt, wie im folgenden Quellcode dargestellt.

```
const fetchGeneralData = async () => {
    const response = await fetch(`${API_URL}/generalData/{window.year}`);
    setGeneralNumbers(await response.json());
}
```

Bei der Variabel *window.year* handelt es sich dabei um das aktuell ausgewählte Jahr, welches aus dem *Session Storage* abgerufen wird. Wie zu sehen ist ändert sich der URL-Pfad für die angefragte Datei entsprechend der Auswahl des Jahres. Wenn die Variabel *window.year* dem leeren Wert ”” entspricht, wird die Resultat-Datei für alle Jahre gefetcht. Somit ist im Server für jedes vorhandene Jahr eine Datei hinterlegt und zusätzlich noch eine Datei, in der die Ergebnisse von allen Jahren gespeichert sind. Abschließend werden die erlangen Daten in einer ”useState”-Variabel gespeichert, wodurch sich die Ansicht der Seite Aktualisiert. Auf dieselbe Art und Weise funktioniert die *Fetch*-Funktion jeder weiteren Komponente und wird aus diesem Grund im weiteren Verlauf nicht erneut erklärt.

3.5.3 Karte

Die Komponente ”MapComponent.js” bildet den Großteil der Webseite und ist für die Implementierung der Karte verantwortlich. Hierfür werden vorerst die Daten für das ausgewählte Jahr vom Server angefragt. Das Backend gibt dafür eine JSON-Datei zurück, in der die Kursinformation jedes Kurses bereits, in Kategorien unterteilt, abgespeichert ist. Dies sieht folgendermaßen aus:

```
{
     "IT_KEYWORDS": [Kursinformation1, Kursinformation2, ... ],
     "LANGUAGE_KEYWORDS": [Kursinformation1, Kursinformation2, ...],
     "SPORT_KEYWORDS": [...],
     "SCIENCE_KEYWORDS": [...],
     "MANAGMENT_KEYWORDS": [...],
     "MEDICINE_KEYWORDS": [...],
     "MARKETING_KEYWORDS": [...],
     "MUSIC_KEYWORDS": [...],
     "ART_KEYWORDS": [...],
     "MEDIA_KEYWORDS": [...],
     "UNKNOWN_KEYWORDS": [...]
}
```

Bei den Kursinformationen innerhalb der Liste handelt es sich ebenfalls um eine Liste, in der die Informationen inklusive Standorte aufgelistet werden. Diese sind zur Erstellung des Popups eines Kursmarkers, wie in Abbildung 3.2(d) gezeigt, nötig.

Mithilfe dieser Informationen werden, wie in der Leaflet-Dokumentation beschrieben, ein Marker für jeden Kurs mit Popup[11] mit selbst kreiertem *Icon*[12] erstellt. Allerdings werden die Marker nicht direkt der Karte angebunden. Stattdessen wird der Dokumentation des ”leaflet.markercluster”-Plugins[13] gefolgt, wonach die Marker an eine dafür bereitgestellte Klasse angebunden werden sollen. Diese Klasse wurde für das Projekt entsprechend konfiguriert und letztendlich an die Leaflet-Karte angebunden, wodurch die Implementierung des Features abgeschlossen wird.

All das wird mit folgender ”useEffect()”-Funktion realisiert, die aufgerufen wird, sobald die Leaflet-Karte ”mapInstance” erstellt wurde. Die vom Plugin bereitgestellte Klasse wird in dem Quellcode als ”markerClusterGroup”-Variabel abgespeichert.

```
// Sobald die Leaflet Karte erstellt wurde, wird diese 'useEffect()' Funktion aufgerufen
useEffect(() => {
    if (!mapInstance) return;

    if (mapInstance) {

        // Die Karte wird auf den Standort des Webseitennutzern herangezoomt
        navigator.geolocation.getCurrentPosition(function(position) {
            currentPositon_lat = position.coords.latitude;
            currentPositon_long = position.coords.longitude;
            mapInstance.flyTo([currentPositon_lat,currentPositon_long], center_zoom + 4 );
```

[11] https://leafletjs.com/examples/quick-start/
[12] https://leafletjs.com/examples/custom-icons/
[13] https://www.npmjs.com/package/leaflet.markercluster

```
12          });
13
14          fetchOnlineCourseData();
15          fetchData(); // Für jeden Kurs wird nach dem Fetchen ein Marker erstellt,
16          // die alle der 'markerClusterGroup' Klasse hinzugefügt werden
17
18          // Die Plugin Klasse 'markerClusterGroup' wird der Leaflet Karte hinzugefügt
19          mapInstance.addLayer(markerClusterGroup);
20
21          // Die Legende für die Kurskategorien wird erstellt
22          addCategoryLegendToMap(mapInstance);
23      }
24  }, [mapInstance]);
```

In der Funktion wird zuerst das Blickfeld der Karte auf die Position des Nutzers gerichtet. Anschließend werden die Daten der Kurse aus dem Backend-Server *ge-fetcht*. Alle Kursmarker werden innerhalb der ”fetchData()”-Funktion erstellt und der Cluster-Klasse ”markerClusterGroup” beigefügt. Anschließend wird in Zeile 17 das Plugin mit den Markern der Karte hinzugefügt und schließt somit die Implementierung des ”leaflet.markercluster”-Plugins ab, wie in Abbildung 3.2 beschrieben. Zuletzt wird für die Karte eine Legende der verschiedenen Kategorien erstellt.

Um die Kategorienbildung der Kurse für die gesamte Webseite einheitlich zu gestalten, wurden zwei externe JSON-Dateien erstellt. Sie sind in dem JSON-Ordner unter dem Pfad zu finden, der in der Abbildung 3.6 angezeigt wird.

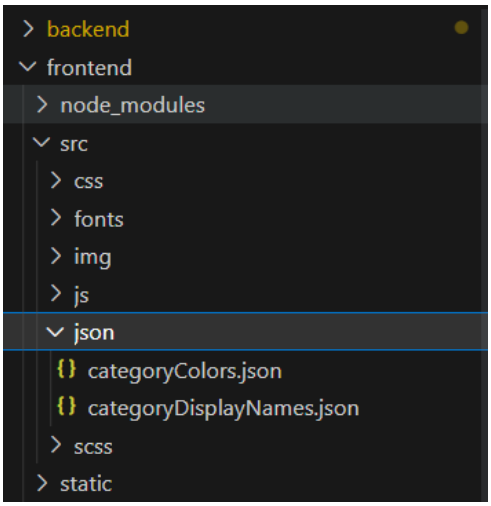


Abbildung 3.6: Dateipfad für die Gestaltung der Kategorien

Die Datei ”categoryColors.json” setzt dabei für jede Kategorie eine Farbe fest, mit der diese repräsentiert werden soll, während die Datei ”categoryDisplayNames.json” den Namen der Kategorie festsetzt und diesen in der Legende repräsentiert. Die JSON-Dateien werden in folgender Tabelle 3.1 dargestellt.

Die Farbe der Kursmarker, die Legende, sowie die Farben und Namen der Kategorien Balkendiagramms werden mithilfe der dargestellten Dateien einheitlich abgestimmt. Darüber hin-

”categoryDisplayNames.json”

```
{
    "IT_KEYWORDS": "IT / Informatik",
    "LANGUAGE_KEYWORDS": "Sprache",
    "SPORT_KEYWORDS": "Sport",
    "SCIENCE_KEYWORDS": "Wissenschaft",
    "MANAGMENT_KEYWORDS": "Managment",
    "MEDICINE_KEYWORDS": "Medizin",
    "MARKETING_KEYWORDS": "Marketing",
    "MUSIC_KEYWORDS": "Musik",
    "ART_KEYWORDS": "Kunst",
    "MEDIA_KEYWORDS": "Multimedia",
    "UNKNOWN_KEYWORDS": "Nicht zugeordnet"
}
```

”categoryColors.json”

```
{
    "IT_KEYWORDS": "#154360",
    "LANGUAGE_KEYWORDS": "#7FFFD4",
    "SPORT_KEYWORDS": "#FF0000",
    "SCIENCE_KEYWORDS": "#34568B",
    "MANAGMENT_KEYWORDS": "#ECF0F1",
    "MEDICINE_KEYWORDS": "#008000",
    "MARKETING_KEYWORDS": "#8E44AD",
    "MUSIC_KEYWORDS": "#DFFF00",
    "ART_KEYWORDS": "#2C3E50",
    "MEDIA_KEYWORDS": "#FF00FF",
    "UNKNOWN_KEYWORDS": "#808080"
}
```

Tabelle 3.1: Dateien zur Einheitlichen Kategorie Gestaltung

aus wäre es möglich weitere Kategorien einzufügen, wodurch die Graphen automatisch angepasst werden würden. Hierfür müsste allerdings auch die entsprechende JSON-Datei im Backend erweitert werden, auf die im späteren Teil des Kapitels eingegangen wird.

Neben der Karte wurde die Anzahl an sichtbaren Kursmarkern im Kartenfenster präsentiert (vgl. Abbildung 3.1). Hierfür wurde eine Event-Funktion implementiert, die aufgerufen wird, sobald die Karte bewegt oder *gezoomt* wird. Der Quellcode dieser Implementierung wird im folgenden Code-Block erklärt.

```
1  if (mapInstance != null) {
2      // 'moveend' heißt das Event für das Bewegen oder Zoomen der Karte
3      mapInstance.on('moveend', function() {
4
5          // Der Zähler für die Marker wird vorerst auf 0 gesetzt
6          setVisibleMarkersCount(0);
7          var visibleMarkerCountTemp = 0;
8
9          // Die Funktion getBounds() schränk die Karte auf das Sichtfenster ein
10         var bounds = mapInstance.getBounds();
11
12         // Die Funktion 'eachLayer' ist eine For-Schleige für jeden Layer in der Cluster Klasse
13         markerClusterGroup.eachLayer(function(marker) {
14             // Diese if Funktion überprüft ob es sich bei dem Layer um ein Marker handelt und
15             // ob sich die Position des Markers innerhalb des Sichtfensters 'bounds' befindet
16             if (marker instanceof L.Marker &&   bounds.contains(marker.getLatLng())) {
17                 visibleMarkerCountTemp += 1; // zähler für die Anzahl an treffern
18             }
19         });
20
21         // Hier wird die 'useState()' Variabel neu gesetzt
22         // und somit die Anzeige der Sichtbaren Marker aktuallisiert
23         setVisibleMarkersCount(visibleMarkerCountTemp);
24     })
25 }
```

Eine zusätzliche Modifikation, die der Karte beigefügt wurde, ist die Filterfunktion. Hiermit ist es möglich die angezeigten Kursmarker nach Kurstitel zu filtern. Bei der verwendeten Funktion handelt es sich um eine Event-Funktion, die aufgerufen wird sobald etwas in der Suchzeile über der Karte eingegeben wird. Da es sich als sehr herausfordernd darstellt, spezifische Marker zu löschen oder auszublenden, wurde eine Herangehensweise verwendet, in der alle Marker gelöscht und anschließend neu erstellt werden.

```
// Die Funktion wird aufgerufen sobald etwas in das Suchfeld geschrieben wird
const filterData = (inputField) => {
    markerClusterGroup.clearLayers(); // alle Marker werden hier gelöscht

    // Falls das Suchfeld leer ist werden alle Marker erstellt
    if (inputField.target.value == "") {
      setFilteredOnlineCourses(onlineCourses.length);

      // in markerJson befinden sich alle gefetchten Kurse aus dem Backend
      createMarkers(markerJson);
    }
    else {
        // Groß- und Kleinschreibung wird ignoriert
        var lowercaseInput = inputField.target.value.toLowerCase()

        // Aus allen Kursdaten in ''markerJson'' wird eine neues
        // Datenobjekt ''filteredMarkers'' erstell, in der alle Kurse hinzugefügt werden
        // bei denen sich der Suchbegriff im Titel befindet
        var filteredMarkers = new Map();
        Object.entries(markerJson).map( ([keyword, array]) =>
            filteredMarkers.set(keyword, array.filter( data =>
                data[2].toLowerCase().search(lowercaseInput) > -1)
            )
        );

        // Marker werden mit der neuen Liste erstellt
        createMarkers(Object.fromEntries(filteredMarkers));

        // Die Anzahl an Online Kurse am linken Rand wird ebenfalls gefiltert
        setFilteredOnlineCourses(onlineCourses.filter(data =>
            data[0].toLowerCase().search(lowercaseInput) > -1).length)
    }
    mapInstance.fireEvent("moveend");
}
```

3.5.4 Balkendiagramme

Im weiteren Verlauf werden die Komponenten, für die Erstellung der Balkendiagramme, erläutert. Hierbei handelt es sich um die JavaScript-Dateien ”ProvidersBarChart.js”, ”CoursesPerMonthBarChart.js”, ”CitiesBarChart.js” und ”CategoryBarChart.js”, dessen Zugehörigkeiten im Anhang 6 dargestellt werden. Alle Balkendiagramme erhalten Daten in folgendem Listenformat aus dem Backend.

```
[[Name1, Häufigkeit1], [Name2, Häufigkeit2], ...]
```

Beim Namen handelt es sich dabei um die, auf der X-Achse aufzulistenden, Attribute, während der Begriff "Häufigkeit"die Anzahl an Vorkommnissen im Datensatz widerspiegelt und sich schließlich auf der Y-Achse des Graphen zeigt. Dabei werden die Werte eines Kursattributes aller Kurse aus dem Datensatz betrachtet und die Häufigkeit jedes Wertes gezählt. So wird beispielsweise für das Balkendiagramm der Kursstädte alle Werte des Kursattributs ”Anbieterstadt” betrachtet und gezählt. Das Ergebnis sieht in etwa wie folgt aus:

```
[["Stuttgart", 868], ["Freiburg", 283], ["Ulm", 133], ["Schopfheim", 100], ... ]
```

Für die anderen Balkendiagramme werden dementsprechend andere Kursattribute betrachtet. Da es jedoch nicht möglich ist alle Werte mit ihren Häufigkeiten im Diagramm darzustellen, werden jeweils nur die am häufigsten vorkommenden Werte visualisiert. Wie viele Werte sich im Balkendiagramm wiederfinden, wird durch eine konstante Variabel am Anfang jeder Komponente festgelegt und kann nach Belieben geändert werden.

Für jedes Diagramm sind bereits solche fertig berechnete Listen im Server hinterlegt, die nur *gefetcht* werden müssen. Nur das Balkendiagramm, das die Häufigkeit der Kategorien auflistet (siehe Abbildung 1(d) im Anhang), nutzt dabei dieselben Daten aus dem Server, welche auch für die Visualisierung der Karten gebraucht wurden. Der Grund hierfür ist, dass dort bereits alle Kurse in Kategorien eingeteilt und aufgelistet werden. Daher entspricht die Länge der Liste einer Kategorie, der Häufigkeit derselben. Mit folgender ”map”-Funktion wird dies ausgenutzt und das Format transformiert.

```
var jsonAsList = Object.keys(responseJson.current).map( key =>
    [key, responseJson.current[key].length]);
jsonAsList = jsonAsList.sort((a, b) => b[1] - a[1]);
```

Die Variabel ”responseJson.current” ist hierbei das JSON-Format, das bereits aus der Karten-Komponente bekannt ist (siehe Quellcode 3.5.3). Es wird schließlich in dasselbe Format wie das der anderen Balkendiagramme transformiert, so dass sich beispielsweise folgende Liste ergibt:

```
[["UNKNOWN_KEYWORDS", 5868], ["IT_KEYWORDS", 1283], ["MANAGMENT_KEYWORDS", 806] ... ]
```

3.5.5 Wortwolke

Die Wordwolke wurde mit einem, auf React zugeschnittenem, Paket innerhalb der Datei ”Wordcloud.js” erstellt (siehe Abbildung 3.7). Hierfür bietet das Paket ”react-d3-cloud” [14] eine mit D3js erstellte, fertige React-Komponente an, mit der sich eine Wortwolke erstellen lässt. Dadurch kann der Graph direkt im HTML-Code eingebunden und konfiguriert werden. Die Optionen für die Konfiguration sind gut dokumentiert und können auf der NPM-Seite des Packets[14] eingesehen werden. Die Daten aus dem Backend entsprechen demselben Format, wie dem

[14] https://www.npmjs.com/package/react-d3-cloud

Abbildung 3.7: Wordcloud.js

der Balkendiagramme (siehe Unterkapitel 3.5.4). Somit wurden in der React-Komponente ausschließlich die Daten vom Server *gefetcht*, diese Daten anschließend in das vorgegebene Format der Wortwolke transformiert und die, durch das Paket bereitgestellte, Klasse erstellt. Die verwendete Konfiguration kann im Quellcode eingesehen werden.

Damit sich die Wortwolke dynamisch an die Größe des Datensatzes anpassen kann, war es nötig die Häufigkeit der Wörter auf einen prozentualen Wert zu ändern. Ansonsten käme es zu dem Problem, dass die Häufigkeit von vorkommenden Wörtern je nach Jahresauswahl in einem Fall im Tausenderbereich liegen würde, während sie in anderen Fällen lediglich im Zehnerbereich liegen würde. Dementsprechend würden die Wörter deutlich zu groß oder zu klein dargestellt werden. Um eine konstante, lesbare Größe für alle Wörter zu garantieren, wurde zum einem eine minimale Größe von Wörtern festgesetzt und zum anderen die Worthäufigkeit zu einem prozentualen Wert verändert, in Abhängigkeit des am häufigsten vorkommenden Wortes.

```
1  function transformValueToPercentage(list) {
2      biggestValue.current = parseFloat(list[0][1]);
3      list.forEach(word => word[1] = (parseFloat((word[1]) / biggestValue.current) * 100));
4      return list
5  }
```

Der dargestellte Quellcode zeigt, wie die Häufigkeit der Wörter in der Liste ersetzt wird, sodass das am häufigsten vorkommende Wort den Wert Hundert einnimmt und alle anderen Wörter einen prozentualen Anteil von diesem erhalten. Der Häufigkeitswert repräsentiert letztendlich auch die Schriftgröße der Wörter.
Visiert man mit der Maus ein Wort an, so wird dennoch die exakte Häufigkeit eines Wortes und nicht der prozentuale Wert angezeigt. Das gelang, indem die genutzte Umrechnung für die Anzeige zurückgerechnet wurde.

3.5.6 Wärmekarte

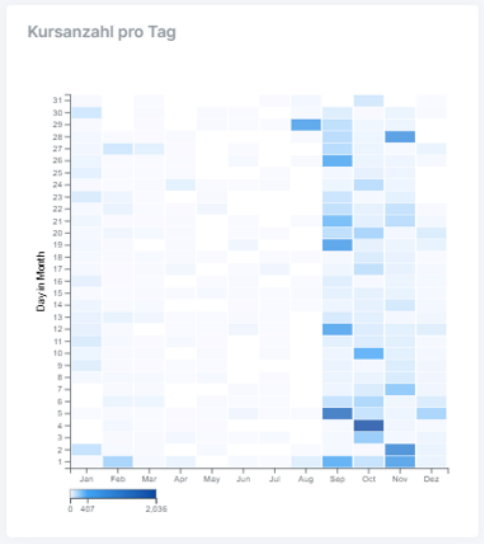


Abbildung 3.8: CoursesPerDayHeatMap.js

Die Wärmekarte wurde ebenfalls mit D3js in der JavaScript-Datei ”CoursesPerDayHeatMap.js” erstellt 3.8. Die Farbintensität spiegelt hier die Häufigkeit von Kursen an den jeweiligen Monaten und Tagen wieder. Es ist zu beachten, dass die Farbfunktion nicht linear verläuft. Während der Entwicklung wurde realisiert, dass sich in Bezug auf die Häufigkeit von Kursen ein großer Sprung im Datensatz befindet. So finden an bestimmten Tagen sehr wenige Kurse statt, an anderen wiederum sehr viele. Die Mittelwerte exestierten im Datensatz kaum. Das führte zu dem Problem, dass die Farbintensität keinen Verlauf aufwies, sondern nur zwei Werte, weiß oder blau, annahm. Als Lösung für dieses Problem wurde Funktion für die Farbintensität verändert, sodass sie nicht mehr linear verläuft. In Abbildung 3.9 wird die eingesetzte Funktion dargestellt.

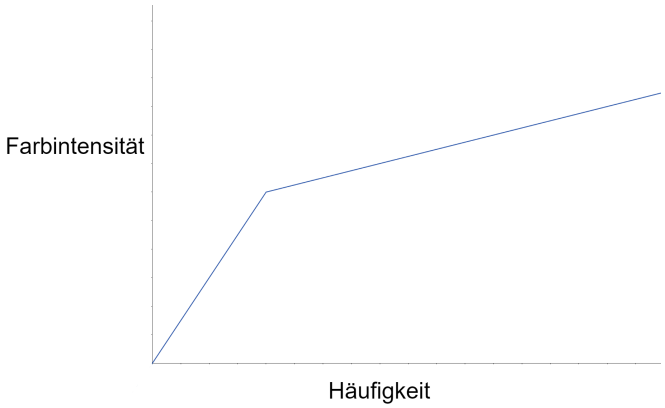


Abbildung 3.9: Farbintensitätsverlauf (Erstellt mit GeoGebra (https://www.geogebra.org/?lang=de)

3.6 Implementierung des Webservers

In diesem Abschnitt wird die Implementierung des Backends erläutert. Im Backend wird der Datensatz gelesen, die Daten für den entsprechenden Graphen berechnet und anschließend in separate JSON-Dateien abgespeichert.

Der Backendfolder besteht aus einer Python-Datei, in der alle Funktionen des Servers implementiert sind. Des Weiteren befindet sich im ”Database”-Ordner der Datensatz als JSON-Datei. In diesem Fall heißt er ”Top20k.json”. Zusätzlich beinhaltet er einen Ordner, in dem alle erstellten Resultat-Dateien abgespeichert werden (siehe Abbildung 3.10).

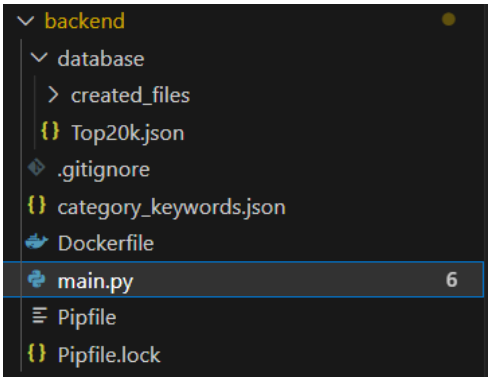


Abbildung 3.10: Backend Ordner

3.6.1 Implementation der Berechnungen

Alle Berechnungen basieren auf folgende Funktion, die den Datensatz liest, sowie alle Kurse in einer Liste zurückgibt, die in einem entsprechenden Jahr stattfinden.

```
1 def filtered_jsonlist_for_year(year):
2     if (year == ""):
3        return get_jsonlist_from_database()
4
5     return [json_object for json_object in get_jsonlist_from_database()
6         if json_object["Kursbeginn"][:4] == year]
```

Die Funktion ”get_jsonlist_from_database()” gibt eine Liste mit allen Kurse aus dem Datensatz zurück. Mithilfe der List Comprehension[15] von Python werden die Kurse aus der Liste gefiltert und zurückgegeben, so dass sich nur noch Kurse aus dem entsprechen Jahr in der neuen Liste befinden.

Wie bereits erwähnt wird für jeden Graphen eine Resultat-Datei pro Jahr und zusätzliche Eine für alle Jahre abgespeichert. Die Erstellung dieser Resultat-Dateien wurde Mithilfe der "run_database():Funktion implementiert, die alle Berechnungen für jedes Jahr und jeden Graphen einzeln startet und abspeichert. Der Ablauf dieser Funktion wird in folgendem Pseudocode dargestellt.

[15] https://www.w3schools.com/python/python_lists_comprehension.asp

```python
def run_database():

    # Alle Vorhanden Jahre im Datensatz werden hier gespeicher
    jahre_im_datensatz = get_existing_years_in_dataset()

    # das leere Element "" berechnet die Daten für alle Jahre
    jahre_im_datensatz.append("")

    for jahr in years_in_dataset:

        ergebnissliste1 = führe_Berechnung_Von_Graph1_Aus(jahr)
        erstelle_Datei_mit_Ergebniss(ergebnissliste1, jahr + "_name_von_graphen1.json")
        del ergebnissliste1

        ergebnissliste2 = führe_Berechnung_Von_Graph2_Aus(jahr)
        erstelle_Datei_mit_Ergebniss(ergebnissliste2, jahr + "_name_von_graphen2.json")
        del ergebnissliste2

        ...

    return "All Files are created"
```

Auf diese Weise wird für jeden Graphen pro Jahr eine Resultat-Datei erstellt und mit entsprechendem Jahr und Namen des Graphen benannt. Die Berechnung der einzelnen Graphen unterscheidet sich zwar etwas, aber das Prinzip der ”run_database()” bleibt dasselbe.

Im folgenden Abschnitt werden die einzelnen Berechnungen erläutert.

3.6.2 Häufigkeitsberechnung

Wie bereits im Kapitel über die Grundlagen vorgestellt, bieten qualitative Daten hilfreiche Informationen, wenn die Häufigkeit ihrer Werte dargestellt werden. Genau das wird für Balkendiagramme und der Wärmekarte bewerkstelligt. In folgender Tabelle 3.2 ist zu sehen, welches Kursattribut für welchen Graphen verwendet wurde.

Graph	Kursattribut	Name der Komponentendatei
Anzahl an Kuranbieter	”Anbietername”	ProvidersBarChart.js
Anzahl an Kursstädten	”Kursstadt”	CitiesBarChart.js
Kursanzahl pro Monat	”Kursbeginn” (Nur Monat)	CoursesPerMonthBarChart.js
Kursanzahl pro Tag	”Kursbeginn” (Monat und Tag)	CoursesPerDayHeatMap.js

Tabelle 3.2: Zugehörigkeit von Graph und Kursattribut

Nur das Balkendiagramm, in welchem die Kurse verschiedenen Kategorien zugeordnet werden, bildet hier eine Ausnahme, da kein solches Kursattribut existiert. Wie bereits erwähnt nutzt dieser Graph die Daten, die für die Karte berechnet wurden. Diese Berechnung wird später betrachtet.

Um die Häufigkeit der jeweiligen Kursattribute zu berechnen sind zwei Schritte notwendig. Zuerst wird eine Liste erstellt, in der der entsprechende Kursattribut-Wert jedes Kurses auf-

gelistet wird. Leere Werte werden zeitgleich herausgefiltert. Der folgende Quellcode listet beispielsweise alle Werte des Kursattributs ”Anbietername” auf.

```
provider_occurrences_list = [jsonObject["Anbietername"] for
    jsonObject in filtered_jsonlist_for_year(year)
    if jsonObject["Anbietername"] != ""]
```

Der zweite Schritt besteht nun darin die Anzahl jedes vorkommenden Anbieternamens in der Liste zu zählen. Zu diesem Zweck wurde folgende Hilfsfunktion implementiert

```
# Diese Funktion zählt die Häufigkeit jedes Element in einer Liste
# und gibt eine Liste mit zurück die Paarweise das Element und die Häufigkeit enthält
# Beispiel: [a,b,b,c,a,a,c,a,d] wird zu [[a,4],[b,2],[c,2],[d,1]]
def count_occurrences_of_each_elemt_in(list_elem):
    amount_dict = dict()
    for key in list_elem:
        # Ist das Element bereits im Dictonary enthalten
        # dann wird der Wert des Elemnts um 1 erhöht
        if (key in amount_dict):
            amount_dict[key] = (amount_dict[key]) + 1

        # Ist das Elemen noch nicht im Dictonary enthalten dann wird das Element
        # als neuer Schlüssel mit dem Wert 1 erstellt
        else:
            amount_dict[key] = 1

    # Das Dictonary wird in ein Listen in Liste Format transformiert und zurückgegeben
    return [list(tuple_elem) for tuple_elem in amount_dict.items()]
```

Wie bei JSON hat das Dictionary in Python ein Schlüssel, dem ein Wert zugeordnet ist. Für jedes Element in der gegebenen Liste wird geprüft, ob bereits ein Schlüssel existiert, das dem Element entspricht. Ist das nicht der Fall wird das Element als Schlüssel, mit dem Wert 1, erstellt. Existiert der Schlüssel bereits, so wird stattdessen der Wert um 1 erhöht. Der Rückgabewert kann nun in einer JSON-Datei abgespeichert werden und vom entsprechenden Graphen im Frontend abgerufen werden.

Auf diese Art und Weise wurden alle Berechnungen für die Graphen in der Tabelle 3.2 durchgeführt. Zusätzlich gibt die Länge dieser Liste Auskunft über alle unterschiedlich vorkommenden Werte. Diese Eigenschaft wurde genutzt um die Werte für die generellen Information ”Anzahl an Städten mit Kursen” und ”Anzahl an Kursanbietern” der ”GeneralNumbers.js”-Komponente zu berechnen.

Für die Wortwolke wurde die Häufigkeit aller Wörter in allen Kurstiteln und deren Schlagwörter gezählt. Hierfür wurde das Python-Paket ”Natural Language Toolkit” (NLTK)[16] genutzt. NLTK ist einer der führenden Plattformen für die Verarbeitung von natürlichen Sprachen. Es bietet dafür zahlreiche Funktionen und darüber hinaus vorgefertigte Datenkollektionen, sogenannte "corpora"(NLTK Team, 2023).

Die folgenden Funktion nutzt die von NLTK bereitgestellten Funktionen und Corpora, um damit die Häufigkeit jedes einzelnen Wortes zu berechnen.

[16] https://www.nltk.org/

```
def get_word_count_list(year):

    filter_words = set(stopwords.words('german'))

    # Hier werden Manuell Wörte hinzugefügt, die herausgefiltert werden sollen
    filter_words.update({"ca", ""m/w/d"", "sowie", ...})

    # Zahlen und Satzszeichen werden ebenfalls herausgefiltert
    filter_symbols = string.punctuation + "0123456789"

    # Klasse von nltk, die die Worthäufigkeit zählt
    word_occurrences = nltk.FreqDist('')

    # Es werden nur Kurse eines Jahres betrachtet
    for jsonObject in filtered_jsonlist_for_year(year):
        # Alle Wörter werden in einzelne Elemente unterteilt
        course_titel_words = nltk.word_tokenize(((jsonObject['Kurstitel'])
            + jsonObject['Schlagwort']).lower())

        # Alle Worthäufigkeiten werden gezählt, wenn es sich bei dem Wort
        # nicht um ein Wort/Symbol aus der Filterliste handelt
        word_occurrences.update([word for word in course_titel_words
            if word not in filter_words and word not in filter_symbols])

    # Die 100 am meisten Vorkommenden Wörter werden im Listen Format zurückgegeben
    return [list(tuple_elem) for tuple_elem in word_occurrences.most_common(100)]
]
```

In der Funktion wird zu Beginn ein Set aus Wörtern und Satzzeichen erstellt, die nicht berücksichtigt werden wollen. Dafür wird ein Corpus von NLTK verwenden, welches ein Set an Stoppwörtern der deutschen Sprache zu Verfügung stellt. Stoppwörter sind in einer Sprache häufig verkommene Wörter, die jedoch keine ausschlaggebende Aussage über das Thema des Textes geben. Deshalb werden diese Wörter in vielen Fällen nicht beachtet. Dadurch ist nicht nur die Berechnung schneller, sondern es wird auch verhindert, dass diese durch nichts-sagenden Wörter verfälscht wird (Ganesan, n. d.).

Zu den Stoppwörtern werden manuell zusätzlich einige weitere irrelevanten Wörter hinzugefügt. Anschließend erstellt die Funktion "nltk.word_tokenize("Text")eine Liste, in der jedes vorkommende Wort als Element aufgelistet wird. Diese kann schließlich durchlaufen und jedes Wort herausfiltern bei dem es sich um ein Stoppwort oder Symbol handelt. Nun zählt die bereitgestellte Klasse ”nltk.FreqDist(”)” mit der ”.update()”-Funktion die Häufigkeit jedes Wortes in der Liste. Abschließend werden die hundert meist vorkommenden Wörter mit ihrer Häufigkeit im Listen-Format zurückgegeben. Die Liste kann nun abgespeichert werden und vom Frontend abgerufen werden.

3.6.3 Kategorisierung der Kurse

Die Kategorisierung der Kurse wurde mithilfe von Stichwörtern für jede Kategorie durchgeführt. Hierfür wurde folgende JSON-Datei erstellt, in der für jede Kategorie mehrere Wörter aufgelistet werden.

```
{
    "IT_KEYWORDS": ["edv", "data", "excel", "programmier", "python", "sql", ...],
    "LANGUAGE_KEYWORDS": ["sprach", "a1", "a2", "englisch", "italienisch", ...],
    "SPORT_KEYWORDS": ["yoga", "fitness", "basketball", "athletik", ... ],
    "SCIENCE_KEYWORDS": ["pyhsik", "biologie", "wissenschaft", "forschung", ...],
    "MANAGMENT_KEYWORDS": ["steuer", "versicherung", "recht", "lexware", ...],
    "MEDICINE_KEYWORDS": ["pfleger", "medizin", "krank", "gesundheit", ...],
    "MARKETING_KEYWORDS": ["marketing"],
    "MUSIC_KEYWORDS": ["musik", "singen", "gitarre", "piano", "konzert", ...],
    "ART_KEYWORDS": ["kunst", "malen", "malerei", "zeichnen", "atelie"],
    "MEDIA_KEYWORDS": ["adobe", "photoshop", "video", "videobearbeitung", ...],
    "UNKNOWN_KEYWORDS": []
}
```

Für jeden Kurs wird überprüft ob sich ein Stichwort einer Kategorie im Kurstitel oder im Schlagwortattribut befindet. Ist das der Fall, so wird der Kurs der entsprechenden Kategorie zugeordnet. Falls keines der Stichwörter zu finden ist, wird der Kurs als unbekannt kategorisiert. Es ist zu erwähnen, dass eine Doppelzuweisung nicht verhindert wurde. So kann es sein, dass es Duplikate von Kursen gibt wenn sie mehreren Kategorien zugeordnet wurden.

Folgende Funktion ordnet die Kurse einer Kategorie zu und speichert die Zuordnung in einem Python Dictionary.

```
def create_courses_categorys_files(year):
    course_string = dict()

    # 'CATEGORY_KEYWORDS' entspricht der zuvor beschriebenden JSON Datei mit der Stichwörter
    # Jede Kategorie mit der entprechenden Stichworliste 'keyword_list' wird durchgegangen
    for keyword_key, keyword_list in CATEGORY_KEYWORDS.items():

        # Wenn wir beim der 'UNKNOWN_KEYWORDS' Kategorie angelangt sind wird der Kurs,
        # falls sich kein Stichwort aller anderen Kategorien im Titel befindet,
        # der 'UNKNOWN_KEYWORDS' Liste zugewiesen
        if (keyword_key == "UNKNOWN_KEYWORDS"):
            course_string[keyword_key] = [[jsonobject['Longitude'], ...]
                for jsonobject in filtered_jsonlist_for_year(year)
                if all(
                    word not in
                    jsonobject["Kurstitel"].lower() + jsonobject["Schlagwort"].lower()
                    for word
                    in functools.reduce(operator.iconcat, CATEGORY_KEYWORDS.values(), [])
                )]
        # Wenn wir noch nicht bei der 'UNKNOWN_KEYWORDS' Kategorie sind,
        # werden die Kurstitel/Schlagwörter auf die Stichwörter dieser Kategorie überprüft
        else:
            course_string[keyword_key] = [[jsonobject['Longitude'], ...]
                for jsonobject in filtered_jsonlist_for_year(year)
                if any(
                    word in
                    jsonobject["Kurstitel"].lower() + jsonobject["Schlagwort"].lower()
                    for word in keyword_list
                )]
    # Das Dictonary mit allen zugeordneten Kursen wird zurückgegeben
    return course_string
```

Das zurückgegebene Dictonary-Objekt befindet sich bereits in dem Format, das die Karte benötigt und kann anschließend als JSON-Datei abgespeichert werden.

4 Diskussion

Im vorangehenden Kapitel wurde versucht die vorgestellten, theoretischen Grundlagen umzusetzen und in die Implementierung der Anwendung einzubetten. In diesem Kapitel werden die Erkenntnisse und Probleme der Entwicklung erörtert. Des Weiteren wird Stellung zu den anfangs vorgestellten Forschungsfragen genommen, sowie deren Beantwortung diskutiert.

Diese Arbeit hatte als Ziel eine alternative Darstellungsform für die Metasuchmaschine IWWB zu suchen und zu implementieren. Statt einer Auflistung der Kurse, war hierbei ein Dashboard mit Karte als Lösungsansatz vorgesehen. Dadurch wurde erhofft einen besseren Überblick über die gesamten Daten zu schaffen und somit Nutzern wie auch Forschern in Entscheidungsprozessen zu unterstützen. Weiterhin wurde erhofft interessante Datenzusammenhänge oder neue Informationen zu entdecken, die einer normalen Liste nicht zu entnehmen sind. In dem Kapitel über die Grundlagen wurde der Lösungsansatz aufgegriffen und thematisch in den aktuellen Stand der Wissenschaft eingeordnet. Des Weiteren wurde an dieser Stelle aufgeführt warum die Visualisierung auf einem Dashboard eine effektive und effiziente Art der Wissensvermittlung ist. Auf diese Weise wurde der Lösungsansatz untermauert und im darauffolgenden Kapitel, über die Implementierung der Entwicklung, die Umsetzung dokumentiert. Durch eine interaktive Gestaltung wurde eine gute Balance zwischen Übersichtlichkeit und Informationsdichte gesucht.

In Anbetracht der übergeordneten Forschungsfrage der vorliegenden Arbeit und den Erkenntnissen aus den vorgestellten Themengebiete der Datenvisualisierung und der Explorativen Datenanalyse, ist das entwickelte Dashboard zwar gelungen, aber das volle Potenzial nicht in Gänze ausgeschöpft. Im Vergleich zu der Ausgangssituation, in der nur eine Liste mit allen Kursen, sowie eine Filterfunktion zur Verfügung standen, bietet das Dashboard einen deutlich umfassenderen Überblick über den Datensatz. Durch die Visualisierung werden die Informationen simpel, intuitiv und anschaulich vermittelt. Besonders hilfreich ist die interaktive Gestaltung der Karte, die übersichtlich ist und dennoch alle Informationen über einzelne Kurse enthält. Hier wurde auch das zuvor erwartete Problem der Leistungsschwankung bei der Vielzahl an Markierungen gut gelöst, sodass die Leistung der Karte die eigene Erwartung übertroffen hat. Gleichzeitig wurde durch die Implementierung des Clustering der Marker auch das Problem mit mehreren Kursen auf demselben Standort gelöst. Insgesamt ist die erhoffte Gesamtfunktion der Karte somit übertroffen worden. Das vorgestellte Mantra von Schneiderman ”Overview first, zoom and filter, then details-ondemand” (Shneiderman, 1996), wird im Fall der Karte außerdem gut umgesetzt und verdeutlicht beispielhaft seine Sinnhaftigkeit.

Die Kategorisierung funktioniert zwar und ordnet die Kurse in vielen Fällen richtig zu, ist aber dennoch deutlich ausbaufähig. Es bedarf weiterer Arbeit, da eine Vielzahl der Kurse bis-

lang keiner Kategorie zugeordnet werden. Da es sich bei dieser Arbeit jedoch um ein zeitlich begrenztes Projekt handelt, bietet die implementierte, einfach gestaltete Ergänzungsmöglichkeit für weitere Kategorien und Stichwörter eine gute Basis für zukünftige Forschungsarbeiten. Jedoch ist das Problem der Doppelzuweisung oder der falschen Zuweisung nicht gelöst beziehungsweise nicht eingeschränkt worden. Dementsprechend ist zu erwarten, dass einige Kurse fehlerhaft zugeordnet werden. In der Folge ist auch die Wahl der Stichwörter für die Kurse erschwert, da Wörter benötigt werden, die eine eindeutige Zuordnung garantieren und eine falsche Zuordnung ausschließen. Ansonsten verfälscht der Algorithmus die Daten durch eine Vielzahl an Doppelzuweisungen.

Neben der Karte bietet die Wortwolke einen gelungen Einblick in die Themengebiete der Kurse. Es wird schnell ersichtlich welche Themen sich häufig widerspiegeln. Bestätigt wird dies auch von dem Balkendiagramm, der die Anzahl an Kursen pro Kategorie darstellt. Jedoch könnten hier noch weitere Wörter manuell herausgefiltert werden, die keine aufschlussreiche Bedeutung haben.

Das Balkendiagramm, das die Anzahl der Kurse pro Stadt abbildet, bietet hingegen keine relevanten neuen Informationen, da in diesem Zusammenhang bereits die Karte durch das Clustering einen groben Überblick darüber verschafft. Bei den restlichen Balkendiagrammen handelt es sich um simple Darstellungen, die jeweils neue Informationen über den Datensatz preisgeben, welche sonst in keinen anderen Graphen ersichtlich sind und somit als gelungen angesehen werden können.

Die Visualisierung der Wärmekarte ist zwar gelungen, jedoch ist die Wahl der visualisierten Daten nicht aufschlussreich, da die Kursanzahl im Monat bereits einen guten Einblick in der Thematik gibt. Bei den Kursen pro Tag handelt es sich möglicherweise um irrelevante Informationen. Hier gäbe es deutlich interessantere Relationen zwischen den Kursen, die dargestellt werden könnten, um damit unerwartete Erkenntnisse zu erlangen.

Im Laufe der Entwicklung stellte die Größe und das Format des Datensatzes immer wieder ein Problem dar, weil es sich bei der JSON-Datei um keine effiziente Datenbank handelt. Für die Entwicklung wurde bereits eine kleinere Datei mit 20000 Kursen verwendet, aber führte dennoch zu langen Ladezeiten der Berechnungen. Aus diesem Grund wurde die Entscheidung getroffen die Berechnung einmalig durchzuführen und die Ergebnisse abzuspeichern. Damit wurde das Problem zwar erfolgreich behoben, ist aber keine effiziente Methode. Im Rahmen der Bachelorarbeit wurde kein größerer Datensatz getestet, aber die Vermutung liegt nahe, dass mit einem deutlich größeren Datensatz das Problem erneut auftritt und nicht mehr mit der angewandten Herangehensweise zu lösen ist. Eine bessere Alternative wäre wahrscheinlich die Verwendung einer leistungsstarken Datenbank.

Des Weiteren stellte der Inhalt einiger Kursdaten des Datensatzes ein Problem dar. Einige Kurse beinhalten Daten, die keine Aussagekraft haben und dadurch die Informationen des Dashboards verfälschen. Besonders ist hierbei ein Anbieter in Stuttgart aufgefallen, der ca. 1500 Kurse ohne aussagekräftige Kurstitel anbietet. Bei den Titeln handelt es sich meist um Buchstaben und Zahlenkombinationen, die keine sinnvollen Wörter ergeben. Auch bei der Jahresauswahl im Navigationsmenü ist zu sehen, dass dort Kurse im Jahr 2099 angeboten werden, was sich als sehr unrealistisch herausstellt. Da es sich bei der Arbeit um ein zeitlich begrenztes Projekt handelt, wurden ausschließlich Daten mit leerem Inhalt herausgefiltert. Somit blieben solche fehlerhaften Daten erhalten. Hier wäre möglicherweise eine bessere Bereinigung des

Datensatzes im Vorfeld eine hilfreiche Ergänzung.

5 Ausblick

Obwohl das entwickelte Dashboard bereits einen Einblick in die Daten gibt und die entdeckten Informationen auf eine verständliche und überschaubare Art und Weise darstellt, existieren noch einige Probleme, die es zu bewältigen gilt.

Vor allem die Leistungsfähigkeit bei einer größeren Datenmenge bedarf weiterer Forschung. Im Rahmen der Entwicklung wurde lediglich ein Satz aus 20000 Kursen verwendet, was gerade mal ca. 0,6% der Gesamtzahl an Kursen entspricht, die IWWB [1] zur Verfügung stehen. Um den Datensatz zu erweitern ist es entsprechend nötig, sowohl die Leistung als auch die Darstellung der Anwendung zu optimieren. Insbesondere die Karte wird dabei ein Problem darstellen. Bereits bei dem kleineren Datensatz ist das Clustering zwar eine gute Lösung, stößt aber dennoch in seltenen Fällen auf Grenzen. Sobald sich mehr als ca. 500 Kurse auf der selben Position befinden, wird auch die verwendete spiralförmige Expansion des Cluster-Markers unübersichtlich und die Karte gerät ins Stocken.

Des Weiteren beweist sich die Kategorisierung der Kurse als vielversprechend. Jedoch bedarf es hier weiterer Forschung, um diese zu optimieren und Fehler zu verhindern. Möglicherweise könnte das Forschungsgebiet ”Data Mining” interessante Erkenntnisse und Algorithmen bieten, die zur Lösung hilfreich wären. Außerdem könnte das Gebiet weitere Algorithmen oder Informationen offenbaren, die mehr Relationen der Kurse aufdecken und somit die Informationsgewinnung maximieren. Im zeitlichen Rahmen dieser Arbeit wurde das Gebiert nicht ausgiebig untersucht, sondern vielmehr der Fokus auf die simple Analyse und Visualisierung gelegt.

Doch es gilt nicht nur die Anwendung zu optimieren, sondern auch die visualisierten Daten zu erforschen und auszuwerten. Wie im Einleitungskapitel erwähnt, könnte die Datenbank Aufschluss über die Bildungssituation in Deutschland geben. Hierfür bietet sich das Dashboard als ein geeignetes Hilfsmittel an, auf dem genauere Untersuchungen von Datensätze basieren könnten. Da es wahrscheinlich nicht möglich ist, den gesamten Datensatz des IWWB auf einmal mit dem Dashboard zu analysieren, wäre eine mögliche Alternative die bereits implementierte Filterfunktion des IWWB zu nutzen, um den Datensatz zu beschränken und anschließend mithilfe des entwickelten Dashboards zu visualisieren. Aufgrund der Tatsache, dass sich das Dashboard dynamisch an die Größe des Datensatzes anpasst, wäre es eine gute Möglichkeit die Bildungsforschung in Deutschland zu unterstützen.

Zusammenfassend kann man sagen, dass es sich bei dem Dashboard trotz einiger Probleme um einen guten Ausgangspunkt für weitere Forschungen handelt. Die Datenverarbeitung ist zwar nicht effizient aber trotzdem ausreichend, um eine gute Datenmenge zu analysieren.

[1] https://www.iwwb.de/kurssuche/startseite.html

Besonders in Kombination mit der Filterfunktion von IWWB hat das Dashboard das Potenzial ein hilfreiches Instrument für die Bildungsforschung zu werden.

6 Fazit

Im Rahmen dieser Bachelorarbeit wurde die Entwicklung eines Dashboards zur Datenanalyse präsentiert. Hierfür wurde eine Datensatz mit 20000 Kursen der Metasuchmaschine IWWB verwendet, mit dem Ziel Informationen zu erlangen und diese effizient und effektiv zu vermitteln. Auf Grundlage der Literatur, die im Grundlagenkapitel betrachtet wurde, konnte letztendlich festgestellt werden, dass die Explorative Datenanalyse nützliche Erkenntnisse und Vorgehensweisen anbietet, die dem Ziel dieser Arbeit zur Nutze kommen. Zum einen wurden die Unterteilung der Daten in quantitativ und qualitativ vorgestellt, sowie die damit einhergehenden möglichen Visualisierungsarten, was letztendlich im Implementationsteil auf den IWWB-Datensatz angewendet wurde. Außerdem verhalfen die Erkenntnisse der Datenvisualisierung zu einer simplen, übersichtlichen und dennoch informativen Gestaltung. Hier bot sich die Implementierung einer interaktiven Webseite als optimale Balance zwischen Informationsdichte und Übersicht an. Obwohl das Dashboard nicht optimal funktioniert und insbesondere bei deutlich größeren Datensätzen problematisch sein könnte, wurde die Entwicklung als gelungen und hilfreich befunden. Vor allem für die Implementation der Karte wurden gute Lösungswege, für die im Einleitungskapitel vorgestellten Herausforderungen, gefunden, die die erwartete Funktionalität übertrafen. Die Kategorisierung der Kurse stellte sich als sinnvolle und hilfreiche Ergänzung des Dashboards heraus, weist jedoch noch deutliche Schwächen auf, die es zu optimieren gilt. Um zusammenfassend auf die übergeordnete Forschungsfrage dieser Arbeit einzugehen, lässt sich abschließend sagen, dass das Dashboard einen hilfreichen Überblick über den Datensatz gewährt und Informationen aufdeckt, die zuvor in der Auflistung der Kurse nicht erkenntlich waren. Durch Visualisierungen werden die Information auf eine angenehme Art vermittelt, sodass ein einfacher Nutzer Wissen daraus ziehen kann. Durch weitere Optimierungen und Modifikationen lässt sich zuversichtlich hoffen, dass es sich als ein hilfreiches Werkzeug für die Bildungsforschung in Deutschland herausstellt.

Tabellenverzeichnis

Abbildungsverzeichnis

Literatur

Add React to an Existing Project. (n. d.). *React*. https://react.dev/learn/add-react-to-an-existing-project

Alshaikhly, A. (2020, 27. Juli). The Dominance of JavaScript. *Medium*. https://medium.com/@alshaikhly89/the-dominance-of-javascript-902b2bcd76ee

Beal, V. (2021, 30. September). HTML. *Webopedia*. https://www.webopedia.com/definitions/html/

Bellinger, G., Castro, D., & Mills, A. (2004). *Data, Information, Knowledge, and Wisdom*. http://www.systems-thinking.org/dikw/dikw.htm

Birchard, T. (2020, 16. September). The Art of Routing in Flask. *Hackers and Slackers*. https://hackersandslackers.com/flask-routes/

Bruce, P., Bruce, A., & Gedeck, P. (2020). Practical Statistics for Data Scientists, 2nd Edition. O'Reilly Media, Inc. https://learning.oreilly.com/library/view/practical-statistics-for/9781492072935/

Card, S., Mackinlay, J., & Shneiderman, B. (1999). *Readings in Information Visualization: Using Vision To Think*.

Craft, B. (2005). Beyond guidelines: What can we learn from the Visual Information Seeking Mantra? *2005*, 110–118. https://doi.org/10.1109/IV.2005.28

Deutscher Bildungsserver. (2013, 14. November). *InfoWeb Weiterbildung (IWWB) - Die Suchmaschine des Deutschen Bildungsservers für Weiterbildungskurse*. https://www.bildungsserver.de/onlineressource.html?onlineressourcen_id=20236

Duarte, F. (2023, 3. April). Amount of Data Created Daily (2023). *Exploding Topics*. https://explodingtopics.com/blog/data-generated-per-day

Einführung in JSON. (n. d.). *JSON*. https://www.json.org/json-de.html

Fetch. (2022, 14. April). *Javascript*. https://javascript.info/fetch

Few, S. (n. d.). Data Visualization for Human Perception. *Interaction Design Foundation*. https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/data-visualization-for-human-perception

Ganesan, K. (n. d.). What are Stop Words? *Kavita Ganesan*. https://kavita-ganesan.com/what-are-stop-words/

Gupta, M. (2020, 27. März). What are Libraries, Frameworks and Packages? *Madhuresh's Tech & Life Journal*. https://madhureshgupta.home.blog/2020/03/27/what-are-libraries-frameworks-and-packages/

Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques, 3rd Edition*. Morgan Kaufmann. https://learning.oreilly.com/library/view/data-mining-concepts/9780123814791/

Juviler, J. (2023, 14. August). What Is an API Endpoint? (And Why Are They So Important?) *HubSpot*. https://blog.hubspot.com/website/api-endpoint

Kiefer, M. (2023, 28. Juni). Beliebteste Frontend und Backend Web-Frameworks in 2023. *LAKdev*. https://www.lakdev.de/frameworks

Kirk, A. (2012). Data Visualization: a successful design process. Packt Publishing. https://learning.oreilly.com/library/view/data-visualization-a/9781849693462/

Lau, S., Gonzalez, J., & Nolan, D. (2021). *Learning Data Science*. O'Reilly Media, Inc. https://learning.oreilly.com/library/view/learning-data-science/9781098112998/

MDN contributors. (2023, 2. August). What is a URL? *MDN*. https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_URL

Mount, G. (2021). Advancing into Analytics. O'Reilly Media, Inc. https://learning.oreilly.com/library/view/advancing-into-analytics/9781492094333/

Mustapha, R. (2023, 6. April). What is HTTP? Protocol Overview for Beginners. *freeCodeCamp*. https://www.freecodecamp.org/news/what-is-http/

NLTK Team. (2023, 2. Januar). Natural Language Toolkit. *NLTK*. https://www.nltk.org/

Ozdemir, S. (2016). *Principles of Data Science*. Packt Publishing. https://learning.oreilly.com/library/view/principles-of-data/9781785887918/

Preattentive Visual Properties and How to Use Them in Information Visualization. (2019). *Interaction Design Foundation*. https://www.interaction-design.org/literature/article/preattentive-visual-properties-and-how-to-use-them-in-information-visualization

React Components. (n. d.). *W3Schools*. https://www.w3schools.com/react/react_components.asp

Riva, M. D. L. (2023, 11. Mai). What Are The 5 Gestalt Principles? *CareerFoundry*. https://careerfoundry.com/en/blog/ui-design/what-are-gestalt-principles/

Scarlett, R. (2023, 2. März). Why Python keeps growing, explained. *GitHub*. https://github.blog/2023-03-02-why-python-keeps-growing-explained/

Schwarzmüller, M. (2022). *React Key Concepts*. Packt Publishing. https://learning.oreilly.com/library/view/react-key-concepts/9781803234502/

Shneiderman, B. (1996). The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. *Proceedings 1996 IEEE Symposium on Visual Languages*.

Sinar, E. F. (2015). Data visualization. In S. Tonidandel, E. B. King & J. M. Cortina (Hrsg.), *Big Data at Work* (S. 115–157). Routledge.

Smith, V. S. (2013). Data Dashboard as Evaluation and Research Communication Tool. *New Directions for Evaluation*, *2013*(140), 21–45. https://doi.org/https://doi.org/10.1002/ev.20072

Ubah, K. (2021, 2. Juni). JavaScript Package Manager – Complete Guide to NPM and Yarn. *freeCodeCamp*. https://www.freecodecamp.org/news/how-to-make-api-calls-with-fetch/

Unwin, A. (2020). Why Is Data Visualization Important? What Is Important in Data Visualization? *Harvard Data Science Review*, *2*(1). https://hdsr.mitpress.mit.edu/pub/zok97i7p

Usage statistics of HTML for websites. (2023, 27. September). *W3Techs*. Verfügbar 19. August 2023 unter https://w3techs.com/technologies/details/ml-html_any

Usage statistics of JavaScript. (2023, 27. September). *W3Techs*. Verfügbar 19. August 2023 unter https://w3techs.com/technologies/details/cp-javascript/

Wexler, S., Shaffer, J., & Cotgreave, A. (2017). *The big book of dashboards : Visualizing your data using real-world business scenarios*. John Wiley & Sons, Incorporated.

What is D3? (n. d.). *D3js*. https://d3js.org/what-is-d3

William. (2022, 10. März). Web Application Architecture: The Latest Guide 2022. *ClickIT*. https://www.clickittech.com/devops/web-application-architecture/

Wordcloud: Definition und Funktion. (2023, 26. Februar). *DataScientest*. https://datascientest.com/de/wordcloud-definition

Yi, M., & Sapountzis, M. (n. d.). Essential Chart Types for Data Visualization. *Chartio*. https://chartio.com/learn/charts/essential-chart-types-for-data-visualization/

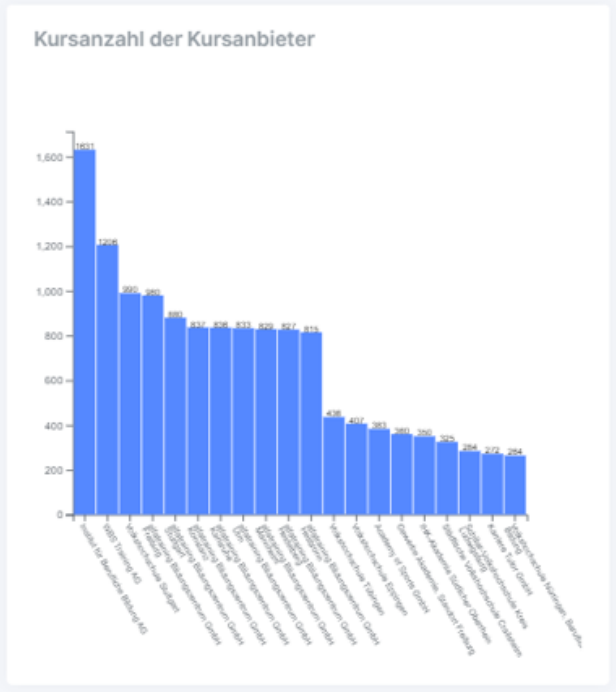
Zacharias, D. (2022, 30. Juni). Why Is JavaScript So Popular? *codepwr*. https://www.codepwr.com/why-is-javascript-so-popular/

Zhou, C. (2020). react-wordcloud. *npm*. https://www.npmjs.com/package/react-wordcloud

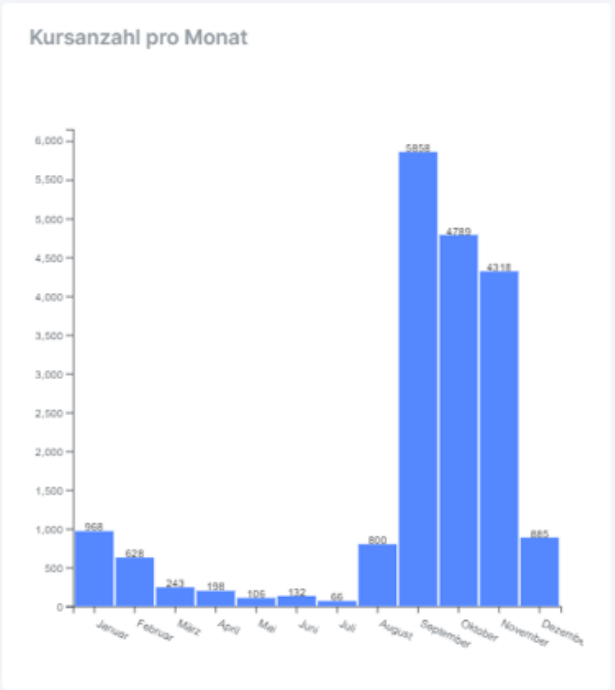
Zlatkov, A. (2017, 11. Oktober). How JavaScript works: Event loop and the rise of Async programming + 5 ways to better coding with async/await. *Medium*. https://medium.com/sessionstack-blog/how-javascript-works-event-loop-and-the-rise-of-async-programming-5-ways-to-better-coding-with-2f077c4438b5

Appendices

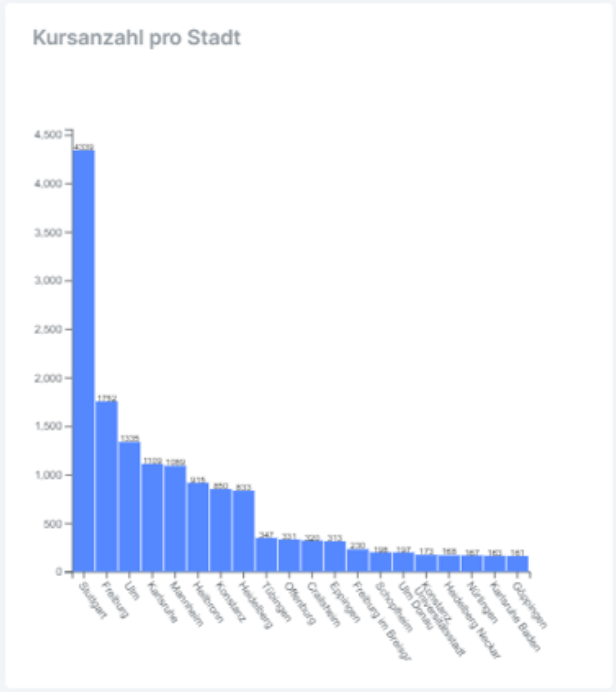
Anhang 1:Balkendiagramm Komponenten



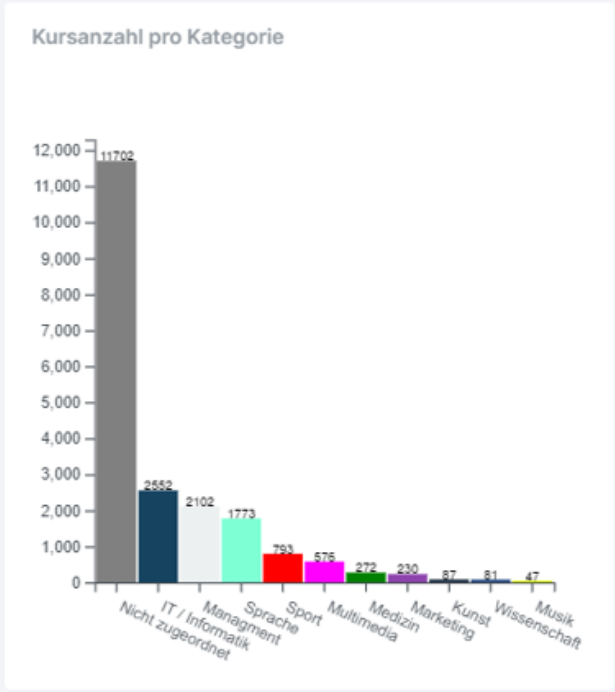
(a) ProvidersBarChart.js



(b) CoursesPerMonthBarChart.js



(c) CitiesBarChart.js



(d) CategoryBarChart.js

Abbildung 1: Balkendiagram Zugehörigkeit