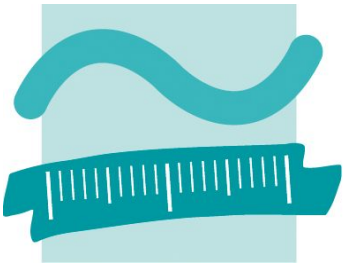


BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN

University of Applied Sciences

Fachbereich VI - Informatik und Medien

Studienperformanz - interaktive Visualisierung der Daten

Bachelorarbeit
zur Erlangung des akademischen Grades Bachelor of Science

Studiengang:	Medieninformatik Bachelor
Bearbeiter:	Anita Kusnierz
Matrikelnummer:	837730
Eingereicht am:	28.07.2020
Betreuer:	Prof. Dr. Agathe Merceron
Gutachter:	Prof. Dr. Petra Sauer

Inhaltsverzeichnis

Glossar

API	API bezeichnet eine Schnittstelle, die eine Software anderen Anwendungen zur Verfügung stellt, um auf sie zugreifen oder sie steuern zu können. Seite *Programmierschnittstelle*. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 14.06.2020.URL: https://de.wikipedia.org/wiki/Programmierschnittstelle [22.07.2020]
Ausreißer	In der Statistik spricht man von einem Ausreißer, wenn ein Beobachtungswert scheinbar nicht in eine erhobene Messreihe passt, also den Erwartungen widerspricht. Grundsätzlich handelt es sich dabei um besonders große oder kleine Messwerte. Seite *Ausreißer*. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 14.06.2020. URL: https://de.wikipedia.org/wiki/Ausrei%C3%9Fer [29.05.2020]
Bibliothek	Eine Programmbibliothek ist eine Sammlung von Unterprogrammen/ Routinen, die Lösungswege für thematisch verwandte Probleme anbieten. Dabei kann die Wahl der richtigen Programmierbibliothek die Implementierung fast jeder Aufgabe erleichtern. Seite *Programmbibliothek*. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 06.05.2020. URL: https://de.wikipedia.org/wiki/Programmbibliothek [29.05.2020]
Bimodale Verteilung	Es handelt sich um eine Häufigkeitsverteilung eines Merkmals oder Wahrscheinlichkeitsverteilung einer Zufallsvariablen mit zwei Modalwerten. Durch Bimodale Verteilung lassen sich die zugrundeliegenden Daten sehr gut in zwei Klassen einteilen. Quelle: [RS94] S. 255
GTK	GTK wurde ursprünglich für das Grafikprogramm *Gimp* geschrieben und ist ein freies GUI-Toolkit und enthält viele Steuerelemente, mit denen sich grafische Benutzeroberflächen für Software erstellen lassen. Seite *GTK_(Programmbibliothek)*. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 28.06.2020.URL: https://de.wikipedia.org/wiki/GTK_(Programmbibliothek) [15.05.2020]
JSON	JSON (*engl.* JavaScript Object Notation) ist ein Datenformat in einer einfachlesbaren Textform und ermöglicht den Datenaustausch zwischen Anwendungen. Grundsätzlich besteht JSON-Struktur aus Name-Wert-Paaren. Seite *JSON*. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 20.07.2020. URL: https://en.wikipedia.org/wiki/JSON [22.07.2020]
Kontingenztabelle	Kontingenztabelle ist eine Tabelle, in der die relativen oder absoluten Häufigkeiten von Kombinationen bestimmter Merkmalsausprägungen

	verzeichnet sind. Dabei hat Kontingenz die Bedeutung des gemeinsamen Auftretens von zwei Merkmalen. Diese Häufigkeiten werden durch deren Randsummen ergänzt. Prof. Dr. Udo Kamps. Seite *Merkmal*. In: Gabler Wirtschaftslexikon. URL: https://wirtschaftslexikon.gabler.de/definition/kontingenztabelle-40194 [22.07.2020]
Merkmal	Merkmal steht in der Statistik für eine Bezeichnung für eine an den Elementen einer Gesamtheit interessierende Eigenschaft, die in unterschiedlichen Ausprägungen vorkommt. Prof. Dr. Udo Kamps. Seite *Merkmal*. In: Gabler Wirtschaftslexikon. URL: https://wirtschaftslexikon.gabler.de/definition/merkmal-40152 [22.07.2020]
Modul	Ein Modul steht bei Bachelor- und Master-Studiengängen an Hochschulen für eine Lehreinheit und besteht aus einer oder mehreren Lehrveranstaltungen mit einem gemeinsamen Lernziel. Die Begriffe Modul und Lehrveranstaltung werden in der Arbeit synonym verwendet. Seite *Modul_(Hochschule)*. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 30.06.2020. URL: https://de.wikipedia.org/wiki/Modul_(Hochschule) [22.07.2020]
qt	qt ist ein Anwendungsframework und GUI-Toolkit zur plattformübergreifenden Entwicklung von Programmen und grafischen Benutzeroberflächen. Das Framework ist in in C++ geschrieben und verwendet einen Präprozessor, genannt *moc* (meta object compiler). Es gibt auch Anbindungen für andere Programmiersprachen, unter anderem für Python (PyQt, PySide), Ruby (QtRuby), C# (Qyoto-Projekt, QtSharp) und Java (Qt Jambi). Das Framework unterstützt unter anderem Windows-, UNIX- und Mac-Systeme. Seite *Qt_(Bibliothek)*. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 30.06.2020. URL: https://de.wikipedia.org/wiki/Qt_(Bibliothek) [11.06.2020]
Regressionsanalyse	Bei der Regressionsanalyse handelt es sich um ein statistisches Analyseverfahren, das die Beziehungen einer abhängigen zu einer oder mehreren unabhängigen Variablen zu modellieren versucht. Seite *Regressionsanalyse*. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 21.06.2020. URL: https://de.wikipedia.org/wiki/Regressionsanalyse [11.06.2020]

Abkürzungen

API	Anwendungsprogrammierschnittstelle (*engl. application programming interface*)
DF	DataFrame
EDA	Explorative Datenanalyse
IQR	Interquartilsabstand (*engl. interquartile range*)

Abbildungsverzeichnis

Listingverzeichnis

Tabellenverzeichnis

1. Einleitung

Viele Universitäten wollen die Studierenden beim Studienabschluss noch wirksamer unterstützen. Laut einer Studie aus dem Jahr 2014 liegt die Abbruchquote bei einem Bachelorstudium sowohl an Universitäten als auch an Fachhochschulen bei etwa 28%.[1] An dieser Stelle kann die Studienberatung beim gezielten Beratung den Studierenden helfen. In der Regel wissen aber die Berater nicht, wie sich die Studierenden in ihrem Studium verhalten. Sie wissen etwa nicht, wie viele Semester sie gewöhnlich im jeweiligen Studiengang zum Studienabschluss bzw. zum Studienabbruch benötigen und welche Leistungen sie in jeweiligen Fächern erzielen. Die Universitäten verwenden digitale Angebote wie Learning-Management-Systems (*LMS*) und bieten Online Kurse an, die den Lernprozess der Studierenden verbessern. Dabei fallen kontinuierlich Daten an, die einer Analyse unterzogen werden können und einen Einblick in das Lernverhalten der Studierenden geben können.
Die vorliegende Arbeit widmet sich der Entwicklung von interaktiven Visualisierungen über Studienperformanz. Das Ziel ist dabei, die Studienberatung durch die Implementierung von verschiedenen Diagrammtypen zu unterstützen. Infolgedessen können die Berater den Überblick über die Lernfortschritte der Studierenden erhalten und eine zielgenaue Beratung anbieten.

Die Arbeit entstand im Rahmen des Projekts *Students Advice* der *Beuth Hochschule.*[2] Die Datenquellen gehören dem Projekt und umfassen die Daten eines sechssemestrigen Bachelor-Studium für drei Studiengänge: *Medieninformatik*, *Druck- und Medientechnik* sowie *Architektur*.
Der Schwerpunkt der Arbeit liegt in der Recherche und Exploration geeigneter Visualisierungsformen sowie einer visuellen Exploration der Daten. Die erstellten Visualisierungen sollen die Studienperformanz für drei Studiengänge *Medieninformatik*, *Druck- und Medientechnik* sowie *Architektur* darstellen und dem Ansatz der explorativen Datenanalyse folgen. Im Rahmen des Projekts wird zur Visualisierung von Daten die Programmiersprache *Python* eingesetzt und das Vorgehen mit Hilfe von *Jupyter Notebooks* dokumentiert.

Die Arbeit setzt sich aus zwei Bereichen zusammen, einem theoretischen und einem praktischen Teil. Im theoretischen Teil werden zunächst die Grundlagen zum Thema Visualisierung erläutert sowie der Ansatz der explorativen Datenanalyse und deren typischen Visualisierungsformen vorgestellt. Im Anschluss wird der Überblick über die *Python* Visualisierungbibliotheken und die im Rahmen des Projekts verwendeten Technologien verschafft, insbesondere *Plotly* als zentrale *Python*-Bibliothek in dieser Arbeit. Abschließend

[1] [DZ]
[2] https://projekt.beuth-hochschule.de/students-advice/

folgt diesem Teil eine Beschreibung der Anwendungsfälle für die zu erstellenden Visualisierungen. Der praktische Teil beginnt mit einer Beschreibung der verwendeten Datenquelle und der Datenvorbereitung. Anschließend werden die Umsetzungsschritte stellvertretend für eine Visualisierung beleuchtet. Den Abschluss der Arbeit bilden eine Zusammenfassung und ein Ausblick, in der die Überlegungen zu implementierten Visualisierungen angestellt werden.

2. Fachliches Umfeld

In diesem Kapitel werden die theoretischen Konzepte dieser Bachelorarbeit erläutert. Zunächst wird in das Thema Visualisierung eingeführt. Danach wird der Ansatz der explorativen Datenanalyse vorgestellt und eine Übersicht über die Visualisierungsformen der explorativen Datenanalyse gegeben. Im Anschluss wird die Übersicht über die vorhandenen *Python* Bibliotheken und die Werkzeuge vorgestellt, die im Rahmen dieser Arbeit verwendet werden.

2.1 Grundlagen der Visualisierung

Die Visualisierung ist ein vielfältiges Forschungsgebiet und umfasst Aspekte aus zahlreichen Disziplinen wie Human Computer Interaction, Wahrnehmungspsychologie, Datenbanken, Statistik und Data Mining. Die Visualisierung wird oft als die Kommunikation von Informationen mit Hilfe von graphischer Repräsentation definiert. Zweifelsohne können die Bilder komplexe Zusammenhänge erheblich besser als die Inhalte in Textform verdeutlichen. Das liegt daran, dass die Wahrnehmung von Bildern innerhalb des menschlichen Begreifen parallel durchläuft, wobei die Inhalte in Textform durch einen sequentiellen Lesevorgang beschränkt sind, so dass der Mensch bei der Textanalyse mehr Zeit braucht, um die Zusammenhänge aufzudecken. Das Besondere daran ist auch die Tatsache, dass die Bilder in Form von Karte oder Diagramm unabhängig von der lokalen Sprache, durch Menschen aus einer anderen sprachlichen Umfeld verstanden werden.[3] Daher kann die Visualisierung als universelles Mittel für Präsentation von Fakten und Informationen fungieren.[4]
Die Voraussetzung jeder Visualisierung sind die darzustellenden Informationen, aufgrund dessen wird die Visualisierung auch als eine Berechnungsmethode definiert.[5] Dabei werden symbolische Informationen in geometrische transformiert und dadurch können die Wissenschaftler die Untersuchung der Ergebnisse ihrer Berechnungen und Simulationen in Form von Bildern darlegen.

Grundsätzlich wird der Bereich Visualisierung in die Unterbereiche *Informationsvisualisierung* (InfoViz), *wissenschaftliche Visualisierung* (SciViz) *(engl. scientific visualization)* sowie *Visual Analytics* aufgeteilt. Dabei werden oft Informationsvisualisierung und wissenschaftliche Visualisierung hinsichtlich ihrer gleichen Zielsetzung - einer Darstellung von Daten - als gleichrangig betrachtet. Der Hauptunterschied besteht darin, andere Daten als

[3] vgl. [WGK15] S.3
[4] vgl. [SM00] S. 3
[5] vgl. [SM00] S.1

Basis zur Visualisierung zu verwenden.[6] Mit der Informationsvisualisierung wird der Fokus auf die Visualisierung von abstrakten Daten gelegt, die keinen physikalischen und direkten räumlichen Bezug besitzen.[7] Dabei handelt sich etwa um textuelle oder numerische Daten wie Börsenkurse, Finanzdaten, Beziehungsnetzwerke im Internet. Währenddessen unterliegen bei wissenschaftlichen Visualisierung die Daten der räumlichen Struktur und werden oft mit einem konkreten mentalen Bild beim Nutzer assoziiert. Daher liegt der Fokus auf einer realitätsnahen Darstellung von Volumen, Oberflächenstrukturen, Beleuchtungsquellen oder Strömungsdaten.[8] Einen weiteren Unterschied bilden auch die Nutzergruppen. Während die wissenschaftliche Visualisierung primär durch Experten interpretiert wird, können die Informationsvisualisierungen durch Nutzergruppen ohne jeglichen naturwissenschaftlichen Hintergrund bewertet werden.[9]

Visual Analytics ist wiederum ein relativ neuer interdisziplinärer Ansatz, der sich aus den Erkenntnissen der Informationsvisualisierung und der wissenschaftlichen Visualisierung heraus entwickelt hat.[10] Visual Analytics entstand als Antwort auf die dynamische Ausbreitung der Informationsmengen, die sich zwar besser speichern und sammeln lassen, jedoch ist ihre Analyse problematisch.[11] Aus diesem Anlass befasst sich der Ansatz mit der Analyse von sehr großen Datenmengen, die durch interaktive visuelle Darstellungen unterstützt wird.

2.1.1 Ziele der Visualisierung

Trotz der im Kapitel 2.1 genannten Unterschiede zwischen *wissenschaftlicher Visualisierung* und *Informationsvisualisierung* lassen sich die Ziele der Visualisierung auf beide Forschungsgebiete übertragen. In diesem Zusammenhang ist nach Schumann und Müller das Ziel von Visualisierung, eine effektive Auswertung von vorhandenen Daten zu ermöglichen.[12] Dabei soll das Verständnis und die Kommunikation über die Daten und daraus resultierenden Konzepten und Modellen nachvollziehbar veranschaulicht werden.[13] Eine geeignete Visualisierungsform hilft ferner dem Anwender selbst die Aussage zu erkennen, die Zusammenhänge zu bewerten und zu verstehen. Und die daraus gewonnenen Erkenntnisse können den Austausch von Arbeitsergebnissen unterstützen sowie bisher verborgene Zusammenhänge aufdecken.

---

[6] vgl. [WGK15] S. 27
[7] vgl. [SM00] S. 346
[8] vgl. [DatVis]
[9] vgl. [PD10] S. 441
[10] vgl. [KMS+08] S.2
[11] vgl. ebenda S.2
[12] vgl. [SM00] S. 2
[13] vgl. [SM00] S.2

Die Visualisierung dient laut Schumann und Müller sowohl den Analysezwecken, als auch Präsentationszwecken.[14] Die Analyse umfasst *die explorative Analyse* und *die konfirmative Analyse*. Mit der *explorativen Analyse* wird die interaktive, oft ungerichtete Suche nach Informationen und Strukturen angestrebt, die eine Grundlage zur Formulierung von Hypothesen über die Daten und ihren Hintergrund bildet. *Die konfirmative Analyse* besteht wiederum darin, die Hypothesen auf ihre Richtigkeit mit Hilfe einer geeigneten Visualisierungsform zu verifizieren. Das Ergebnis kann entweder eine Bestätigung oder ein Verwerfen der aufgestellten Hypothese sein.
Nach der Datenanalyse wird die Präsentation und Kommunikation der erzielten Ergebnisse aufgeführt. Anschließend ist die Visualisierung so entworfen, dass die relevanten Aussagen leicht zu erkennen sind. Daher ist es einleuchtend, dass die Ergebnisse durch Dritte identifiziert und verstanden werden können.[15]

2.1.2 Anforderungen an die Visualisierung

Eine geeignete Visualisierung unterliegt unterschiedlicher Einflussgrößen wie Vorwissen des Anwenders, die Art und Struktur der Daten oder die Bearbeitungszeit.[16] Zu diesen Einflussgrößen zählen auch allgemeine Ziele, die die Qualität einer guten Visualisierung bestimmen. Diese Eigenschaften werden durch drei Bewertungskriterien: die Expressivität, Effektivität und Angemessenheit verwirklicht[17].

- *Expressivität* - besagt, dass die Daten möglichst unverfälscht wiedergegeben werden sollten. Dabei spielt die richtige Wahl der Darstellungsart eine entscheidende Rolle. Ferner wird es bezweckt, lediglich in den Daten enthaltenen Informationen darzustellen.
- *Effektivität* - richtet sich nach der eigentlichen Zielsetzung und dem Anwendungskontext der Visualisierung. Dabei steht der Betrachter im Mittelpunkt und es wird bemüht, dem Betrachter die enthaltenen Informationen intuitiv zu vermitteln. Im Grunde soll aus einer Vielzahl von expressiven Darstellungsformen für eine bestimmte Datenmenge eine passende Visualisierungsform für einen bestimmten Sachverhalt gewählt werden.
- *Angemessenheit* - umfasst den Rechen- und Ressourcenaufwand, der mit der Generierung einer visuellen Darstellung verbunden ist und zum Ergebnis in einem angemessenen Verhältnis steht.[18]

[14]vgl. [SM00] S.4
[15] vgl. [SM00] S.6
[16]vgl. [SM00] S. 9f
[17]vgl. [SM00] S. 9ff
[18]vgl. [SM00] S. 10-12

2.1.3 Grundlegende Konzepte

Visualisierungspipeline

Für das Grundverständnis des Visualisierungsprozesses ist die *Visualisierungspipeline* von Bedeutung (vgl. Abbildung 2.1). Die *Visualisierungspipeline* ist im Grunde die Generierung einer visuellen Darstellung aus abstrakten Daten und besteht aus einer Abfolge von drei Schritten: der Datenaufbereitung (*Filtering*), der Datenabbildung (*Mapping*) und der Bildgenerierung (*Rendering*), die den Weg von gegebenen Rohdaten zu einer visuellen Repräsentation beschreiben.[19]

Abbildung 2.1: Die Visualisierungspipeline nach Schumann, Quelle:[20].

Der Prozess beginnt mit der Datenaufbereitung, in der die Rohdaten nach bestimmten Kriterien wie zum Beispiel die Umstrukturierung reduziert und gefiltert werden. Das Filtern der Daten umfasst unter anderem die Datenkonvertierung und Entfernung bestimmter Werte.[21]
Im *Mapping* Schritt werden die einzelnen Punkte auf geometrische Primitive, wie beispielsweise Linien, Punkte und Kreise abgebildet. Dabei werden die geometrischen Primitive durch das Setzen entsprechender Attribute (Position, Größe) beeinflusst.[22] Im letzten Schritt der Bildgenerierung erfolgt die Umwandlung der geometrischen Primitive in Bilddaten. Die Visualisierungspipeline wird noch mal in der Abbildung 2.2 aus Sicht des Datenflusses zusammengefasst.

Abbildung 2.2: Datenfluß in der Visualisierungspipeline, Quelle: [23].

[19]vgl. [SM00] S. 15ff
[20]vgl. [SM00] S. 15
[21] vgl. ebenda S.15f
[22] vgl. Ebenda S.15f
[23] vgl. [SM00] S. 17

Dabei kann der Prozess der Visualisierung durch die menschliche Interaktion unterstützt werden und wird als *Referenzmodell* der Visualisierung beschrieben.[24] Der Nutzer kann die Kontrolle über die einzelnen Schritte des Visualisierungsprozess übernehmen und somit wird der Erkenntnisgewinn gefordert.

Visuelle Variablen

Das *Mapping* - der zweite Schritt des Visualisierungsprozesses beschäftigt sich mit der Auswahl einer geeigneten Darstellungsform und der Wahl der graphischen Elemente sowie Attribute. Um die unterschiedlichen Aspekte der Daten darzustellen, erfolgt die Visualisierung von Daten durch geeignete Abbildung auf visuelle Elemente sog. *visuelle Variablen* (vgl. Abbildung 2.3). Diese Elemente beschreiben, mit welchen visuellen Merkmalen bestimmte Dateneigenschaften dargestellt und verdeutlicht werden sollen.[25] Dazu zählen Position, Größe, Helligkeitswert, Musterung oder Textur, Farbe, Richtung, Orientierung, Form des Elements.

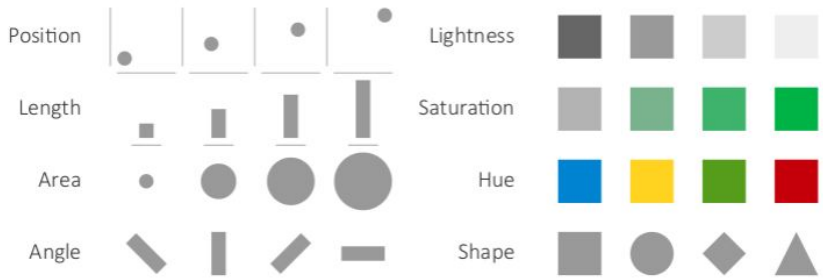


Abbildung 2.3: Visuelle Variablen, Quelle:[26]

Diese Variablen können verschiedene Eigenschaften erfüllen und lassen sich in drei Formen unterscheiden.[27]

- *selektiv*: Sie werden auch als trennend genannt und ermöglichen dem Betrachter unterschiedliche Datenwerte spontan in Gruppen aufzuteilen und zu unterscheiden. Sie dienen der Darstellung von nominalen Daten. Zu diesen zählen Größe (Länge, Fläche/Volumen), Helligkeit, Textur, Farbe, Orientierung.
- *ordinal*: Der Betrachter kann unterschiedliche Datenwerte in Ordnung bringen. Sie dienen der Darstellung von ordinalen Daten. Zu diesen zählen Größe, Helligkeit, Textur

[24] vgl. [SM00] S. 21f
[25] vgl. [SM00] S. 126
[26] vgl. [TS20] S.54
[27] vgl. [SM00] S. 127f.

- *proportional*: Neben spontaner Aufteilung der Daten in eine Ordnung kann der Betrachter die Assoziationen zwischen Ausprägungen der visuellen Variablen mit der verknüpften Meßgröße bilden. Sie dienen der Darstellung von ordinalen und quantitativen Daten. (Größe, Orientierung, Helligkeit)

Datentypen

Nach Schumann bildet ein Datenmodell den Ausgangspunkt jeder Visualisierungen, der für einen Ausschnitt der realen Welt steht.[28] Das Datenmodell besteht aus Informationsobjekten mit ihren Attributen bzw. Merkmalen und Relationen untereinander. Dabei werden die Informationsobjekte durch ein oder mehrere Attribute beschrieben. Grundsätzlich lassen sich zwei Attributtypen unterschieden. Hierzu zählen die *quantitativen* und *kategorischen* Daten.[29]
Die quantitativen Daten verwenden metrische Skalen. Die *kategorischen* Daten, die auch als *qualitativen* Daten bezeichnet werden, verwenden nicht metrische Skalen und dienen der Beschreibung, Gruppierung und Ordnung.[30] Intern werden die *kategorischen* Daten oft als Zahlenwerte repräsentiert, weil sie sich in Zahlenwerte umwandeln (z.B durch Indexierung) lassen. Zusätzlich zählen zu dieser Gruppe die *nominalen Daten und ordinalen Daten.* Bei den nominalen Daten handelt es sich um eine ungeordnete Menge von Namen.[31] Beispiel hierfür wäre die Variable *Pflanzennamen* mit der möglichen Ausprägung {Kamille, Wegwarte, Rittersporn, Augentrost, Margerite}. Als mögliche Operation gilt hier die Festlegung auf Gleichheit bzw. Ungleichheit. Die *ordinalen Daten* dagegen beschreiben eine geordnete Menge von nicht messbaren Werten. Zusätzlich zum Gleichheits- und Ungleichheitstest kommt auch eine Ordnungsrelation hinzu. Beispiel hierfür wäre eine alphabetische Sortierung der *Pflanzennamen* {Augentrost, Kamille, Margerite, Rittersporn, Wegwarte}.

2.2 Explorative Datenanalyse

In den Wissenschaften werden Visualisierungen oft als eine statische Darstellung von gewonnener Erkenntnis präsentiert. Sie werden als *Presentation Graphics* (Präsentationsgrafik) genannt und stellen die gewonnenen Ergebnisse etwa in der Form von Balken-, Streu-, oder Liniendiagrammen dar.[32] Dabei liefern die Präsentationsgrafiken keine Hinweise darauf, auf

[28] vgl. [SM00] S. 35f
[29] vgl. [PD10] S.449
[30] vgl. [PD10] S. 448f
[31] vgl. [PD10] S. 450
[32] vgl. [CHU08] S.4

welchem Wege die Ergebnisse erzielt wurden, sie liefern vielmehr ein Beweis eines mathematischen Theorems. Im Gegensatz zu einer Darstellung von gewonnener Erkenntnis sind die Visualisierungen oft als Mittel zum Erkenntnisgewinn eingesetzt. Diese Art von Visualisierungen wird als *Exploratory Graphics* (explorative Visualisierung) bezeichnet.[33] Durch Transformation, Gewichtung, Filterung komplexer Daten und eine geeignete Repräsentationsform stellen sie die Informationen dar, die sowohl leicht interpretierbar sind als auch Zusammenhänge ausfindig machen. Darüber hinaus eignen sich explorative Visualisierungen als Unterstützung in der Datenuntersuchung. Dementsprechend wird explorative Visualisierung in der Literatur nicht als Endprodukt, sondern als Zwischenprodukt und Mittel definiert.[34]

In diesem Zusammenhang stellt die explorative Datenanalyse Aussagen über Daten aus den Visualisierungen dar, ohne konkrete Fragestellungen vorzuformulieren. Die explorative Datenanalyse wird als Teilgebiet der deskriptiven Statistik betrachtet und um 1977 durch John Tukey als Begriff geprägt.[35] Der Ansatz wird als Bestandteil der Datenanalyse betrachtet und basiert auf der Annahme, dass wissenschaftliche Fortschritte durch zufällige Entdeckungen von Fragen und Hypothesen erzielt werden, die als unmöglich gehalten sind.[36] Dabei werden die statistischen Modellierungsmethoden verwendet, die jedoch das Ziel haben, die Ideen zu finden, anstatt sie zu bestätigen. Der von Tukey definierte Ansatz wird auch oft mit der deskriptiven Statistik in Verbindung gebracht. Der Unterschied in ihrer Einordnung ergibt sich nach Polasek aus Beantwortung von zwei Fragen. Während in der deskriptiven Statistik wird folgende Frage gestellt *Wie kann man eine Verteilung eines Merkmals beschreiben?*, wird in EDA folgende Fragestellung *Was ist an einer Verteilung eines Merkmals bemerkenswert, bzw. explanativ?* relevant.[37]

Ergänzend zu diesen theoretischen Vorüberlegungen werden nun traditionelle Visualisierungsformen der explorativen Datenanalyse, sowie ihre Varianten dargestellt. Diese sollen erste Anregungen für eine Umsetzung der Visualisierungen im praktischen Teil dieser Arbeit geben.

[33] vgl. Ebenda S.4
[34] vgl. [BK18] S. 7
[35] vgl. [P94] S. 1
[36] vgl. [LW17] S.8
[37] vgl. [P94] S.4

2.2.1 Visualisierungsformen

Im Wesentlichen gelten *Diagramme* als eine allgemeine Form der graphischen Darstellung von Daten.[38] Zu den gängigen Diagrammarten der explorativen Analyse zählt u.a. *Boxplot*, *Histogramm*, *QQ-Diagramm*, *Streudiagramm* und *Mosaikplot*. Allerdings müssen explorative Visualisierungen nicht unbedingt dem Verfahren der explorativen Statistik entsprechen. Die häufigsten Visualisierungsformen wie *Punktdiagramme*, *Linien-* und *Kurvendiagramme*, *Säulen* und *Balkendiagramme* sowie *Kreisdiagramme* können auch explorative Zwecke erfüllen. In diesem Sinne werden demzufolge die für die Arbeit und das Verfahren der *EDA* relevanten Diagrammtypen beleuchtet.

Histogramm ist eine graphische Darstellungsform der Häufigkeitsverteilung metrisch skalierter Merkmale (vgl. Abb. 2.4a).[39] Die Daten werden in Klassen (engl. *bins*) eingeteilt. Dabei wird jede Klasse durch ein Rechteck dargestellt. Die Höhe des Rechtecks hj wird durch die relative Häufigkeit der Klasse fj und der Klassenbreite dj bestimmt. Zur Berechnung wird folgende Formel verwendet.[40]

$$hj = \frac{fj}{dj} \qquad (1.1)$$

Dementsprechend werden auf der x-Achse die Klassengrenzen und auf der y-Achse die Häufigkeitsdichten abgetragen.[41] Dabei hängt die Anzahl der verwendeten Datenklassen von der Anwendung ab. Darüber hinaus lassen es sich bei Histogrammen typische Konfigurationen feststellen. Zusätzlich können die Häufigkeitsverteilungen als *kumulative Histogramme* dargestellt werden.[42] Sie stellen die Summe aller Datenwerte dieser und der vorangegangenen Klassen dar. Dabei ermöglichen sie, bestimmte Fälle einzelner Klasse mit der Gesamtpopulation zu vergleichen (vgl. Abb. 2.4 b).

[38] vgl. [SM00] S.126
[39] vgl. [RS94] S. 157
[40] vgl. [TH06] S. 40f
[41] vgl. [RS94] S. 157
[42] vgl. [SM00] S. 136

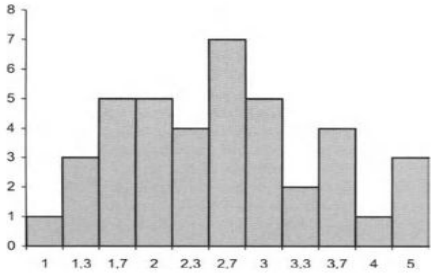


Abbildung 2.4a: Beispiel eines Histogramms: Klausurnotenverteilung (Quelle:[43]).

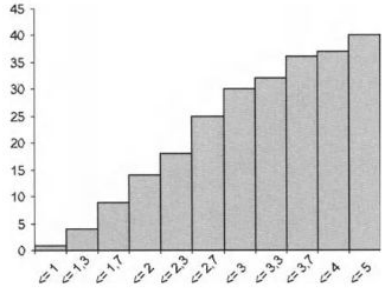


Abbildung 2.4b: Kumulatives Histogramm für die Klausurnotenverteilung aus Abb. 2.4a (Quelle:[44]).

Eine weitere Vergleichsmöglichkeit der Häufigkeitsverteilungen, besteht darin verschiedene *Histogramme* in einem Diagramm zu kombinieren. Dabei kann es zu Verdeckungen kommen, wobei eine Häufigkeitsverteilung transparent dargestellt werden kann.

Dichtediagramm (*engl. distplot*) ist eine Variante des *Histogramms*. Mit dieser Darstellungsform wird die zugrunde liegende Wahrscheinlichkeitsverteilung der Daten über eine entsprechend kontinuierliche Kurve gezeichnet.[45] (vgl. Abb 2.5). Diese Kurve entsteht durch Schätzung der Daten. Die Kerndichteschätzung (*eng. kernel density estimation*) ist die am häufigsten verwendete Methode für dieses Schätzverfahren. Dabei wird am Ort jedes Datenpunktes eine kontinuierliche Kurve (der *Kernel*) mit einer durch den Parameter *Bandbreite* gesteuerten Breite gezeichnet.[46] Anschließend werden diese Kurven addiert und als Resultat eine endgültige Dichteschätzung geliefert. Dazu weist die Bandbreite die Ähnlichkeiten mit der Binbreite im *Histogramm*. Ist sie zu klein, erscheint die Dichteschätzung zu hoch und optisch überfüllt; ist sie zu groß, so verschwinden mögliche Merkmale bei der Verteilung der Daten. Durch den Kernelwahl wird die Form der Dichtekurve bestimmt, die in der Regel so skaliert ist, dass der Bereich unter der Kurve Eins entspricht. (vgl. Abb. 2.5). Dabei ist der am meist verwendete Kernel ein Gaußscher Kernel.[47]

[43] vgl. [SM00] S. 135
[44] Vgl. [SM00] 138
[45] vgl. [W20] S. 60f.
[46] vgl. ebenda
[47] vgl. [W20] S. 61

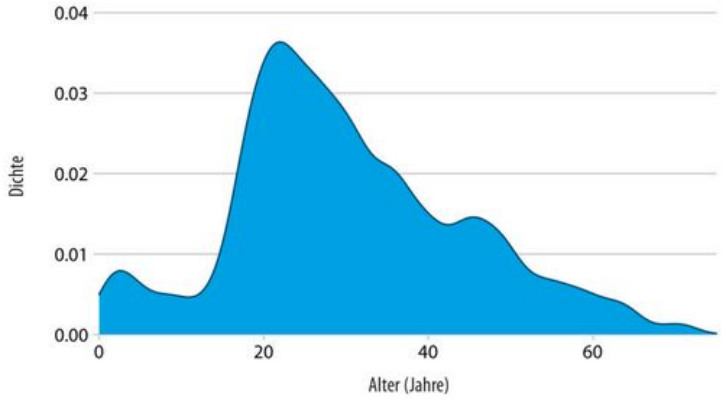


Abbildung 2.5: Dichtediagramm der Altersverteilung der Passagiere auf der Titanic, Quelle: [W20][48]
Datenquelle: *Encyclopedia Titanica* (www.*encyclopedia-titanica*.org).

Boxplot dient der grafischen Darstellung eines mindestens ordinalskalierten Merkmals und charakterisiert im Allgemeinen die Streuung der Daten.[49] Dabei können *Boxplots* mehrere Verteilungen gleichzeitig visualisieren. Im Wesentlichen besteht ein *Boxplot* aus einem Rechteck, genannt Box und zwei Linien (sog. *Antenne*, o. *Whisker*), die mit kurzen Endstrichen enden. Das Schema eines *Boxplots* umfasst den kleinsten x_1 und den größten x_n Beobachtungswert, sowie drei Quartile $x_{0.25}$, $x_{0.5}$ und $x_{0.75}$.[50]

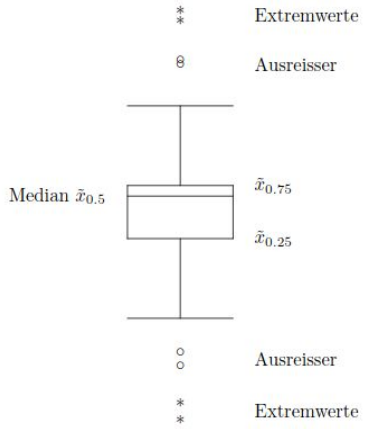


Abbildung 2.6: Komponenten eines Boxplot (Quelle: [51]).

Um die Quartilen zu berechnen, wird die zu untersuchende Beobachtungsreihe geordnet $x_1 \leq x_2 \leq ... \leq x_n$. Dabei ist α eine Zahl zwischen Null und Eins und bezeichnet die Größe des Quantils. Anschließend wird das Produkt aus der Gesamtmenge α und n berechnet. Ist das Produkt

[48] vgl. [W20] S. 60f.
[49] vgl. [RS94] S. 60
[50] vgl. [RS94] S. 60
[51] vgl. [TH06] S. 60

ganzzahlig so gilt die Formel 1.2. Ist das Produkt nicht ganzzahlig, gilt die Formel 1.3, wobei das Produkt (*k)* auf die kleinste ganze Zahl >*nα* aufgerundet wird.[52]

$$\bar{x}_\alpha = \frac{1}{2}\left(x_{(n\alpha)} + x_{(n\alpha)+1}\right) \quad (1.2)$$

$$\bar{x}_\alpha = x_{(k)} \quad (1.3)$$

Die Box umfasst drei Quartile und die Länge der Box entspricht dem Interquartilsabstand (IQR). [53] (vgl. Abb 2.6). Der Strich in der Box kennzeichnet den Medianwert ($x_{0.5}$). Die Strichlage gibt Auskunft über die zugrunde liegende Verteilung der Daten. Die Werte, die sich außerhalb der Whisker befinden, werden als separate Punkte dargestellt. Hier wird zwischen Ausreißern, welche innerhalb des 1,5-fachen und 3-fachen IQR liegen, und Extremwerten, welche außerhalb des 3-fachen IQR liegen, unterschieden.[54] In der Abbildung 2.7 wird ein Boxplot-Beispiel dargestellt. Dabei werden die Verteilungen von Abschlussnoten in Hinsicht auf die Studiendauer (in Semester) im Studiengang *Medieninformatik* präsentiert.

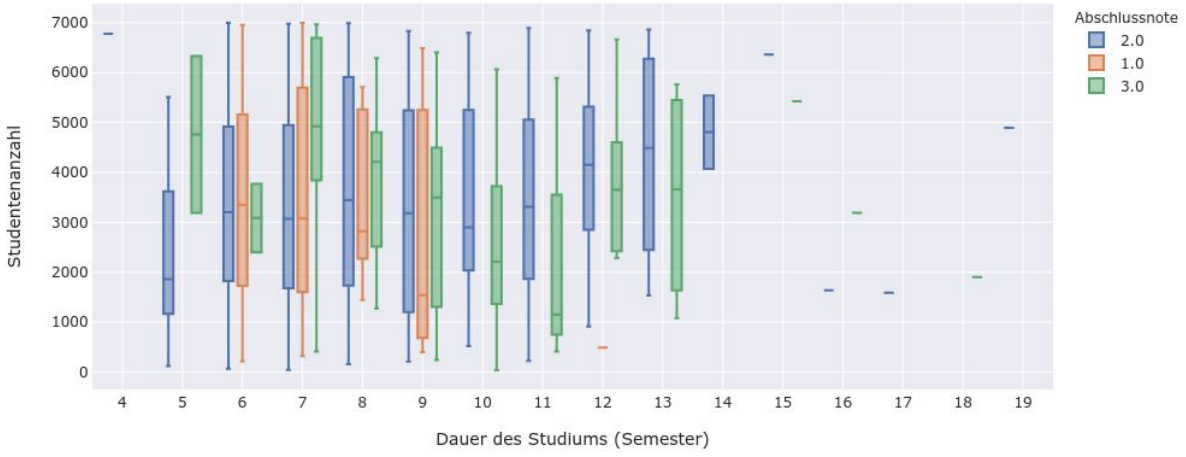


Abbildung 2.7: Beispiel eines Boxplots, Datenquelle: *Students Advice,* Visualisierung: eigenes Werk

Violinplot ist eine Art von Hybrid-plot. Es kombiniert gespiegelte Dichteschätzung (siehe *Dichtediagramm*) mit den Informationen vom Typ *Boxplot* also der Median, die zwei Quartile und die beiden Extremwerte. Dabei entspricht der breiteste Teil der Violine der höchsten Punktdichte im Datensatz.[55] Die Abbildung 2.8 zeigt die Anatomie eines *Violinplots*, die aus einer Darstellung als Punktewolke und dazugehörigem *Violinplot* besteht.

[52] vgl. [TH06] S. 57
[53] vgl. [RS94] S. 60
[54] vgl. [TH06] S. 85
[55] vgl. [W20] S. 78f

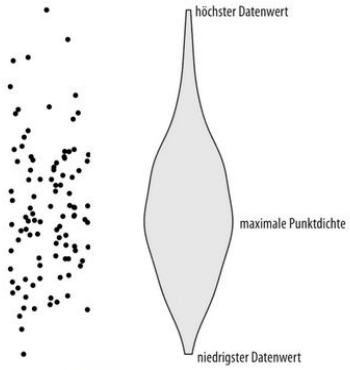


Abbildung 2.8: Anatomie eines Violinplots als Punktewolke (links) und der dazugehörige Violinplot (Quelle:[56]).

Im Unterschied zu *Boxplot* werden bei *Violinplot* bimodale Daten wiedergegeben, die bei bimodalen Datenverteilung auf das Vorhandensein verschiedener Peaks, deren Position und relative Amplitude hinweisen.[57] In der Abbildung 2.9 ist ein Beispiel eines *Violinplots* zu sehen. Dabei wird die Semesteranzahl bis zum Studienabbruch in Hinsicht auf das Geschlecht im Studiengang *Medieninformatik* dargestellt.

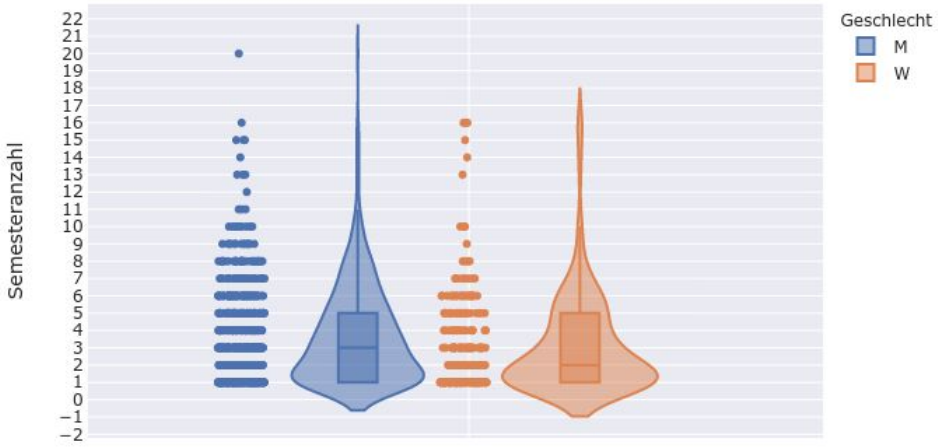


Abbildung 2.9: Beispiel eines Violinplots als Punktewolke und Boxplot. Semesteranzahl bis zum Abbruch in Hinsicht auf das Geschlecht, Datenquelle: Students Advice, Visualisierung: eigenes Werk.

Quantil-Quantil-Diagramm bezeichnet ein graphisches Verfahren zum Vergleich von Verteilungen.[58] Mit dem Ansatz können die Beobachtungswerte zweier Merkmale verglichen werden, dh. man kann die Aussage treffen, ob die Daten einer bestimmten Verteilung unterliegen.[59] Dabei werden die Quantile des einen Datensatzes über denen des anderen Datensatzes abgetragen. Alternativ können die Quantile eines Datensatzes über denen einer

[56] vgl. ebenda S. 85
[57] vgl. ebenda S. 85
[58] vgl. [RS94] S. 294
[59] vgl. [RS94] S. 294

angenommenen Verteilung abgetragen werden. Diese werden in aufsteigender Reihenfolge geordnet zu Wertepaaren zusammengefasst und in einem Koordinatensystem abgetragen. Liegen die Punkte auf der *45*°-Linie, so liegt den beiden Merkmalen die gleiche Verteilung zugrunde (vgl. Abbildung 2.10 a), andernfalls existieren systematische Unterschiede (vgl. Abbildung 2.10 b, c).

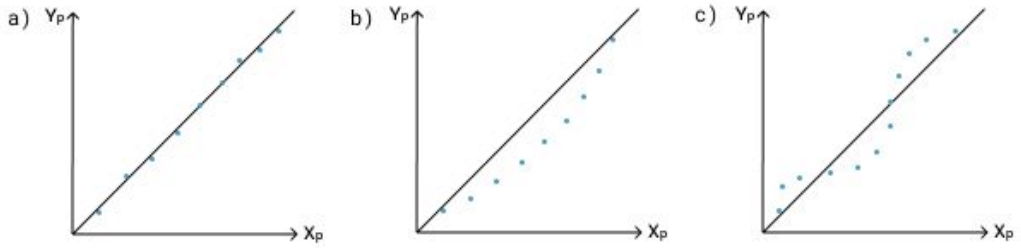


Abbildung 2.10: Typische Quantil-Quantil Diagramme, Quelle [60]

Streudiagramm (*engl. scatterplot*) bezeichnet eine graphische Darstellung der Beobachtungswerte zweier metrisch skalierter Merkmale *X* und *Y* in einem kartesischen Koordinatensystem.[61]Entsprechend werden die Werte des Merkmals *X* auf der *x*-Achse und die Werte des Merkmals *Y* auf der *y*-Achse abgetragen. Dabei wird jedes Paar von Beobachtungswerten (x_1, y_1) als Punkt angezeigt.[62] Daß mehrere Datensätze auf einen gemeinsamen Punkt abgebildet werden können, kann man als Nachteil ansehen. Hierzu bietet sich an, die Zahl der Wertepaare, die auf diesem Punkt liegen, mittels Größe oder Form des dargestellten Punktes zu kodieren. Als Variante des Streudiagramms gilt *Blasendiagramm*, in dem zusätzlich zu den zwei metrischen Merkmalen ein drittes hinzu kommt, das mittels der Größe der Blasen dargestellt wird.[63] Darüber hinaus liefern die Streudiagramme Hinweise auf die Form der Abhängigkeit (*Regressionsanalyse*) und die Stärke der Merkmale (*Korrelationsanalyse*).[64] Die Streudiagramme eigenen sich gut zur Entdeckung von Korrelationen zwischen zwei numerischen Merkmalen. Dabei wird die Stärke der Korrelation durch die Streuung ausgedrückt, mit der die Punktwolke die Linie oder Kurve annähert.[65] In der Abbildung 2.11 werden diese Zusammenhänge verdeutlicht. So spricht man von einer *linearen Korrelation*, wenn sich im Streudiagramm monoton steigende oder fallende Linien bilden. (vgl. Abbildung 2.11 g) Handelt es sich um monoton steigende oder fallende Kurven, so spricht man

[60] vgl. [RS94] S. 294
[61] vgl. [RS94] S. 353
[62] vgl. [RS94] S. 353
[63] vgl. [W20] S. 118
[64] vgl. [SM00] S. 131
[65] vgl. ebenda S.131

von einer *exponentiellen Korrelation* (vgl. Abbildung 2.11 h), andererseits hat man mit komplexen Zusammenhängen zu tun.[66]

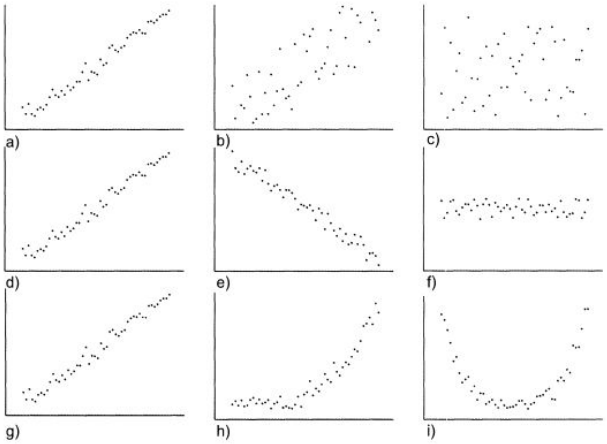


Abbildung 2.11: Formen der Korrelation. Quelle [67]
Unterschiedliche Stärken der Korrelation: (a) stark, (b) schwach, (c) keine,
Ausprägungen der Korrelation: (d) direkt/positiv, (e) indirekt/negativ, (f) keine,
Formen der Korrelation: (g) linear, (h) exponentiell, (i) komplex

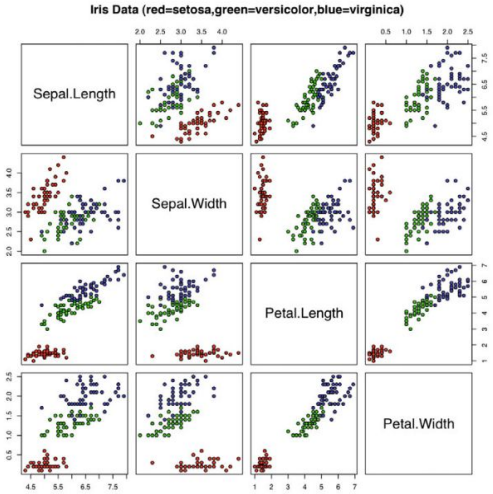


Abbildung 2.12: Beispiel eines Streudiagramm Matrix, Quelle[68]

Streudiagramm-matrix bezeichnet ein graphisches Verfahren zur Veranschaulichung von paarweisen Zusammenhängen zwischen mehr als zwei Merkmalen.[69] Es stellt eine Anordnung von *Streudiagrammen* als Matrix, die alle möglichen paarweisen Kombinationen anzeigen. Bei n-dimensionalen Daten ergeben sich $n(n-1)/2$ Streudiagramme, wobei die Hauptdiagonale leer

[66] vgl. [SM00] S. 131f
[67] vgl. [SM00] S. 132
[68] vgl. [S13] S. 348
[69] vgl. [RS94] S.320f

bleibt, da ein Merkmal nicht gegen sich selbst abgetragen wird. Die Abbildung 2.12 zeigt das Beispiel einer Streudiagramms-Matrix. Dabei werden vier Merkmale: Länge und Breite des Kelchblattes [Sepalum] und des Kronblattes [Petalum] von drei Schwertlilienarten dargestellt.

Mosaic-plot dient der Visualisierung von Datensätzen mit zwei oder mehreren qualitativen Merkmalen.[70] Es wurde anfangs eingeführt, um die *Kontingenztabellen* zu visualisieren. Damit bietet es einen Überblick über die Daten. Die *Mosaik*-Diagramme bestehen aus Gruppen rechteckiger Kacheln.[71] Dabei entspricht jede Kachel einer Zelle aus einer Kontingenztabelle und ihre Fläche ist proportional zur Größe der Zelle. Die Abbildung 2.13 stellt ein Beispiel eines *Mosaic-plots* dar. Im Folgenden werden abteilungsübergreifende Zulassung und Ablehnung von Männern und Frauen an der University of California, Berkeley dargestellt.

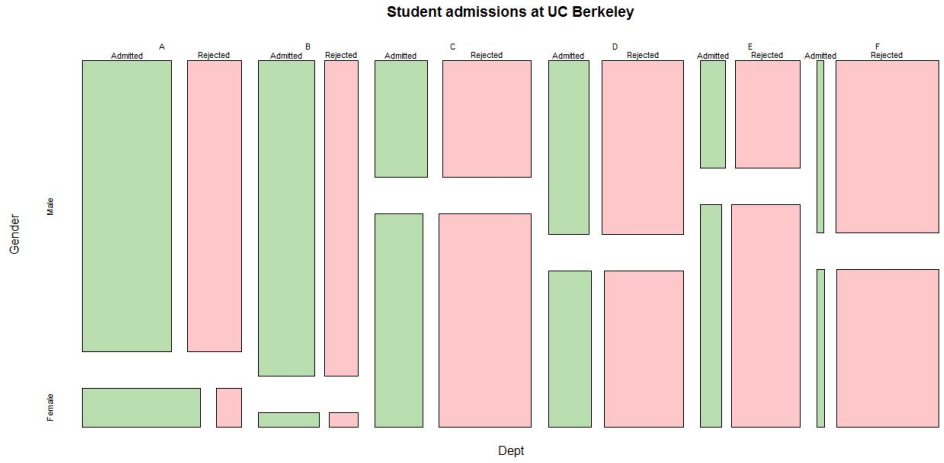


Abbildung 2.13: Beispiel eines Mosaic-Plots, Zulassung und Ablehnung in Hinsicht auf das Geschlecht, University of California, Berkeley, Quelle[72]

Parallele Koordinaten bezeichnen eine Methode zur Visualisierung von multivariater Daten.[73] Die senkrechten Linien zeigen die Achsen des Koordinatensystems. Im Gegensatz zum Streudiagramm, wo die zwei Koordinatenachsen rechtwinklig zueinander angeordnet sind, verlaufen sie hier parallel und im äquidistanten Abstand. Der Unterschied zu einem Kartesischen Koordinatensystem wird in der Abbildung 2.14 verdeutlicht.

[70] vgl. [BK18] S. 238f
[71] vgl. [CHU08] S. 619
[72] vgl. [AM]
[73] vgl. [SM00] S.186

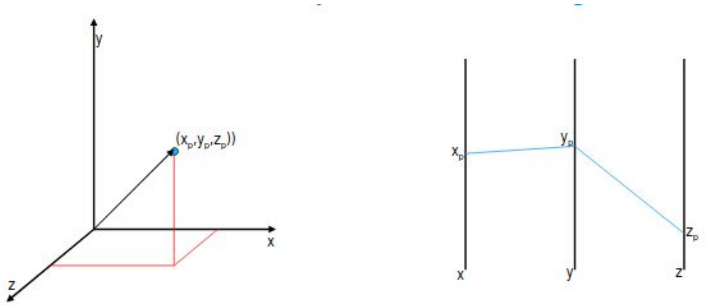


Abbildung 2.14: Unterschied Kartesisches Koordinatensystem und Parallele Koordinaten, Quelle [74].

Jede Linie von links nach rechts entspricht einem Datenpunkt und wird durch einen Polygonzug mit Ecken auf den parallelen Achsen dargestellt.[75] Parallele Koordinaten ermöglichen einen guten Überblick von Werteverteilungen und vereinfachen die Korrelationen zwischen benachbarten Achsen zu erkennen. Nachteilig dabei ist die Tatsache, dass man den Zusammenhang zwischen weit entfernten Achsen ohne direkte Interaktionsunterstützung schwer erkennen kann.[76] Die Abbildung 2.15 zeigt eine in *Plotly* umgesetzte Variante von Parallelen Koordinaten. Dabei wird die Dauer des Studiums in Bezug auf die Semesteranzahl und das Geschlecht von abgebrochenen Studierenden im Studiengang *Medieninformatik* dargestellt.

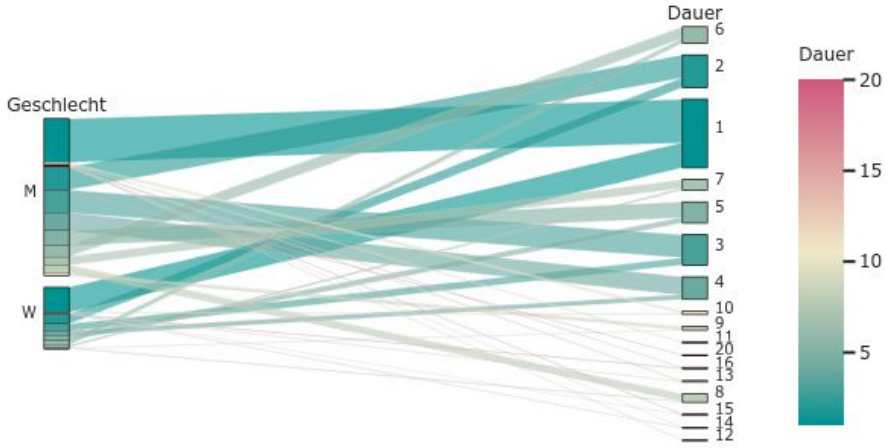


Abbildung 2.15: Beispiel von Parallelen Kategorien. Anzahl der Semester zum Studienabbruch in Hinsicht auf das Geschlecht, Datenquelle: Students Advice, Visualisierung: eigenes Werk.

Ein weiteres Beispiel (vgl. Abb 2.16) präsentiert eine Darstellung des Notenverlaufs von zwei Studentengruppen, in der die erste Gruppe (*Rot*) für Studenten steht, die die Prüfung 3 mit der Note C bestanden haben und die Abschlussprüfung mit Note A bestanden haben. Die zweite Gruppe (*Blau*) bestand sowohl die Prüfung 3 als auch die Abschlussprüfung mit Note C.

[74] vgl. [IL]
[75] vgl. [PD10] S. 460
[76] vgl. [PD10] S. 460

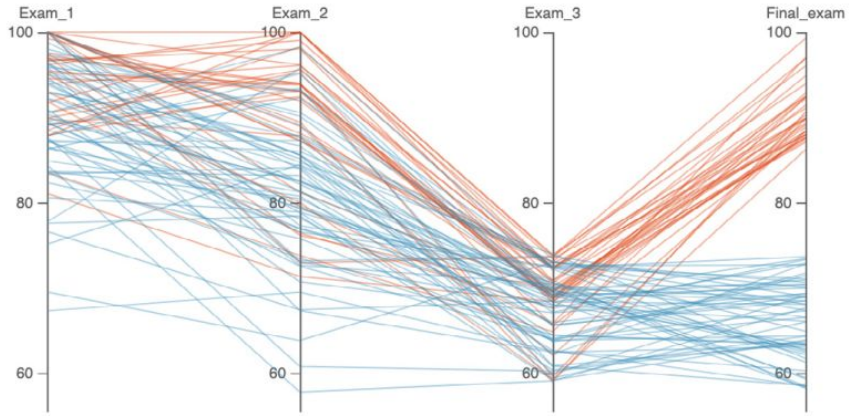


Abbildung 2.16: Parallele Koordinaten. Darstellung von Notenverläufen. Quelle [77]

2.3 Visualisierungsbibliotheken

Der Fokus dieser Arbeit liegt in der explorativen Visualisierung der Daten. Dabei sollen die Visualisierungen mit Hilfe von der Programmiersprache *Python* umgesetzt werden. *Python* verfügt über eine Vielzahl von Visualisierungsbibliotheken, die im Folgenden beleuchtet und diskutiert werden.

Die umfangreiche Landschaft von *Python* Bibliotheken und deren Relationen zueinander wird in Abbildung 2.17 dargestellt.[78] Hierzu werden die *Python* Bibliotheken nach ihrem Ursprung, Fokus und ihrer Geschichte gruppiert und zueinander in Verbindung gebracht. Dabei bildet eine klar getrennte, orangefarbene Gruppe die *SciVis*-Visualisieungsbibliotheken, die im wissenschaftlichen Kontext verwendet werden. Dazu gehören *VisPy*[79] und *Glumpy*[80], die auf dem OpenGL-Standard *(Open Graphics Library)* basieren, der sich als Standard für graphische *API* durchgesetzt hat. *SciVis* Bibliotheken eignen sich für dynamische Visualisierungen von physikalischen Prozessen sehr großer Datenmengen in drei oder vier Dimensionen.

Die restlichen Visualisierungsbibliotheken werden in der Regel im Kontext der Informationsvisualisierung (*InfoVis*) mit Schwerpunkt auf die Visualisierung von Informationen in beliebigen Räumen verwendet. Diese Gruppe lässt sich ferner in weitere Untergruppen gliedern. Die stellvertretenden Bibliotheken einzelner Untergruppen werden unten beleuchtet und mit den Beispieldiagrammen und dem Quellcode versehen.

---

[77] vgl. [H19] S. 171
[78] vgl. [CD19]
[79] vgl. [VI]
[80] vgl. [GL]

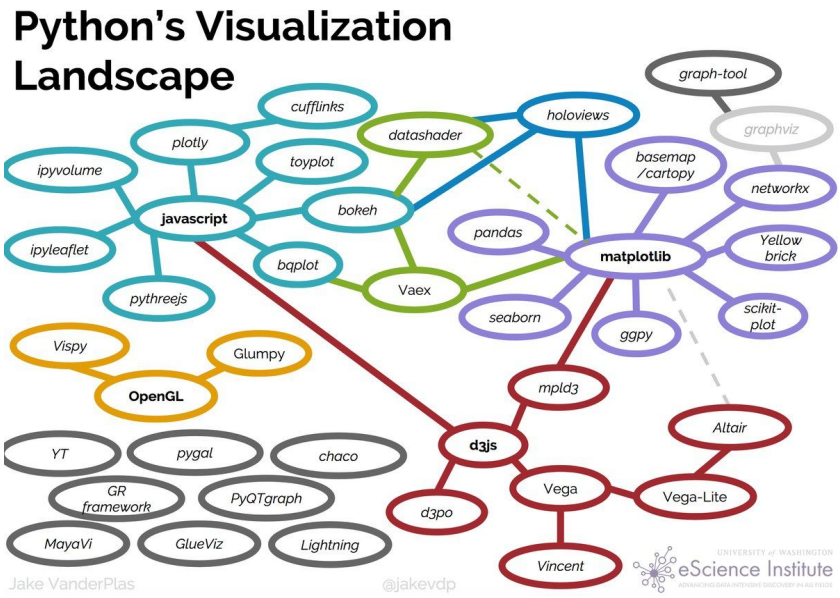
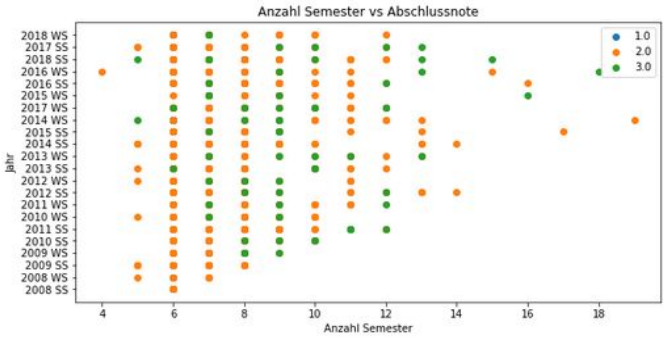


Abbildung 2.17: Python Visualisierungslandschaft (Quelle:[81]).

matplotlib[82] erschien 2003 und ist eine der ältesten und bekanntesten *Python*-Bibliotheken für Datenvisualisierung. *Matplotlib* wurde inspiriert durch *MATLAB*, einer leistungsfähigen Hochsprache für numerische Simulationen, die die Berechnung, Visualisierung und die Programmierung in einer Umgebung integriert.[83] Des weiteren bietet *matplotlib* eine große Auswahl an *2D* Diagrammtypen und Ausgabeformaten, wobei die *3D* Diagrammtypen im geringen Umfang bereitgestellt werden. Die durch *matplotlib* erzeugten Grafiken lassen sich auch in GUI-Bibliotheken wie *Qt*[84] und *GTK*[85] verwenden. Darüber hinaus verfügt *matplotlib* über eine prozedurale Schnittstelle namens *PyLab*[86], die mehrere Bibliotheken für die wissenschaftliche Arbeit wie beispielsweise *NumPy*[87] zusammenfasst. Ein Beispieldiagramm mit Quellcode ist in Listing 2.1 zu sehen.

```
fig, ax = plt.subplots(figsize=(10,5))
for c, df in graduate_students.groupby('Graduation_Grade'):
        ax.scatter(df['Duration_Start_Graduation'], df['Graduation_Semester'],label=c)
ax.legend()
ax.set_title('Anzahl Semester vs Abschlussnote')
ax.set_xlabel('Anzahl Semester')
ax.set_ylabel('Jahr')
```

[81] vgl. [PV18]
[82] vgl. [MA]
[83] vgl. [HQ07] S. 329
[84] vgl. [QT]
[85] vgl. [GT]
[86] vgl. [PYL]
[87] vgl. Abschnitt 2.4.4



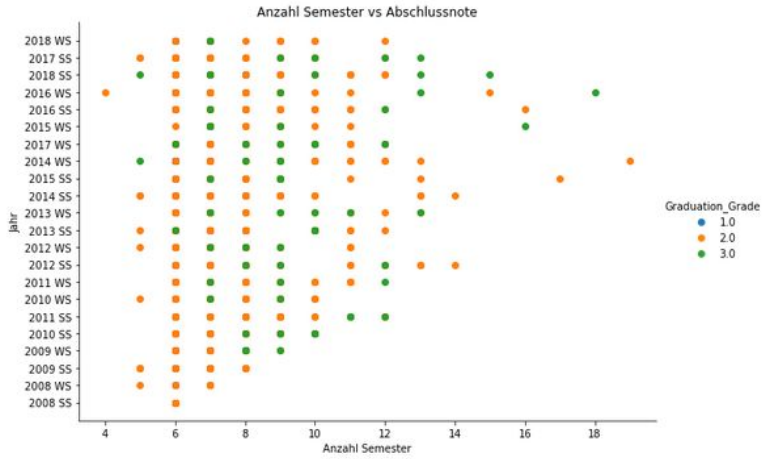
Listing 2.1: Beispiel eines Streudiagramms mit Quellcode umgesetzt mit *matplotlib*. Anzahl Semester vs Abschlussnote, Datenquelle: Students Advice, Visualisierung: eigenes Werk.

Eine weitere Untergruppe der Visualisierungsbibliotheken bilden *matplotlib-basierten* Bibliotheken, die sich auf de Grundlage von *matplotlib* ausgebildet haben. Sie verwenden *matplotlib* als Rendering-Engine für einen bestimmten Datentyp oder in einer bestimmten Domäne wie *Pandas*. Für Bibliotheken wie *ggplot*, *plotnine* stellt *matplotlib* eine *API* bereit, um den Prozess der Erstellung von Plots zu vereinfachen. Eine Erweiterung der *matplotlib*-Bibliothek um weitere Arten von Plots realisiert die Bibliothek *seaborn.* Im Folgenden werden die repräsentativsten Bibliotheken dieser Untergruppe ausführlicher beschrieben.

seaborn[88] ist eine Bibliothek vorzugsweise für statistische Grafiken. Sie soll als eine Ergänzung der Bibliothek *matplotlib* dienen. Ähnlich wie *matplotlib* bietet sie die Unterstützung von *Numpy*- und *Pandas*-Datenstrukturen. Darüber hinaus stellt die Bibliothek die Visualisierung statistischer Zeitreihendaten sowie eingebaute Themen für das Styling von *matplotlib* Grafiken bereit. Zum Vergleich mit dem Diagramm aus dem Listing 2.2 wird das gleiche Diagramm im Listing 2.1 mit Hilfe von *seaborn* geplottet.

```
sns.FacetGrid(graduate_students, hue='Graduation_Grade',height=6, aspect=1.5)
                  .map(plt.scatter, 'Duration_Start_Graduation','Graduation_Semester')
                   .add_legend().set(title='Anzahl Semester vs Abschlussnote',
                                     xlabel='Anzahl Semester', ylabel='Jahr')
```

[88] vgl. [SE]

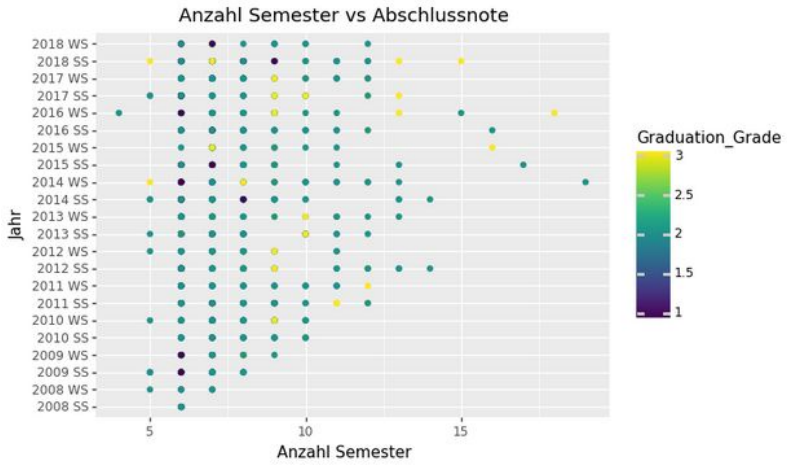


Listing 2.2: Beispiel eines Streudiagramms mit Quellcode umgesetzt mit *seaborn*. Anzahl Semester vs Abschlussnote über die Zeit, Datenquelle: StudentsAdvice, Visualisierung: eigenes Werk.

plotnine[89] ist ein Äquivalent zu *ggplot2*, einem Plotting System für die Statistiksprache *R*. *Plotnine* entstand als Folge der ebenfalls von *ggplot* inspirierten Bibliothek namens *ggpy,* formal bekannt als *ggplot for python*. Mit Hilfe von *plotnine* und *ggplot* können Visualisierungen in *Python* nach demselben *Grammar of Graphics* Prinzip wie in *R* erstellt werden.[90] Die Grundidee der *Grammar of Graphics* besteht darin, dass alle Diagramme aus verschiedenen Grundelementen bestehen. Dazu gehören die Daten, ein Koordinatensystem, statische Transformationen und Objekte innerhalb der Diagramme (Punkte, Säulen).[91] Im Listing 2.3 ist das mit *plotnine* umgesetzte Beispieldiagramm mit Quellcode vom Listing 2.1 zu sehen.

```
(ggplot(graduate_students) +
    aes(x = 'Duration_Start_Graduation',y = 'Graduation_Semester', color = 'Graduation_Grade') +
    geom_point() +
    ggtitle('Anzahl Semester vs Abschlussnote') +
    xlab('Anzahl Semester') +
    ylab('Jahr'))
```

[89] vgl. [GG]
[90] vgl. [IN]
[91] vgl. [PD]



Listing 2.3: Beispiel eines Streudiagramms mit Quellcode umgesetzt mit *plotnine*. Anzahl Semester vs Abschlussnote, Datenquelle: StudentsAdvice, Visualisierung: eigenes Werk.

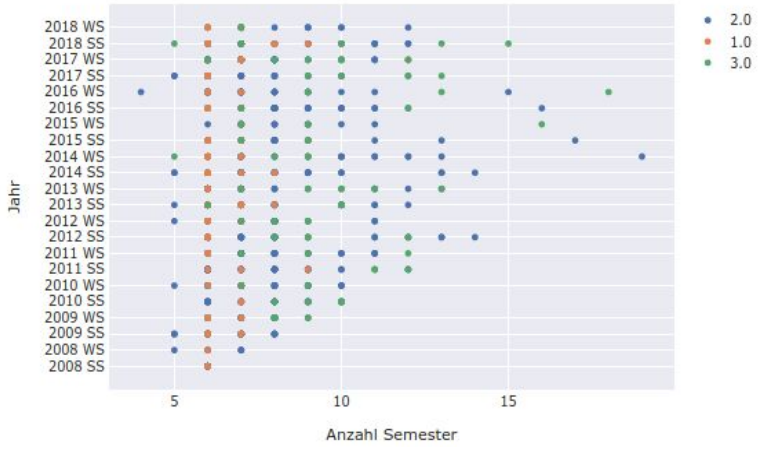
Eine weitere Gruppe der Python-Bibliotheken bilden *Javascript*-basierte Bibliotheken, die interaktive 2D Diagramme für Webseiten und *Jupyter Notebooks* bereitstellen. Dabei sind einige mit Verwendung von benutzerdefinierten *Javascript*, darunter *Bokeh* und *Toyplot*[92] entstanden.[93] Die anderen bildeten sich aufbauend auf der *Javascript* Bibliothek *d3.js*[94] wie etwa *Plotly* und *bgplot*[95] aus.

Bokeh[96] ermöglicht die Erstellung von webbasierten, interaktiven Diagrammen, Grafiken und Dashboards. Im Vergleich zu *seaborn* und *matplotlib* werden die Diagramme mit Hilfe von *HTML* und *Javascript* gerendert. Bokeh konvertiert die Datenquelle in eine JSON-Datei, die mit Hilfe von *Javascript* Bibliothek *BokehJS* ausgeführt wird.

Plotly[97] ist ein Analyse- und Datenvisualisierungstool. Plotly ermöglicht die Erstellung interaktiver Visualisierungen. Die Bibliothek entwickelte Open-Source *API* nicht nur für *Python* sondern unter anderem für *R*, *MATLAB*, *Javascript*. Die Grafiken werden im JSON-Format gespeichert, sodass sie mit Skripten anderer Programmiersprachen wie etwa *R*, *Julia*, *MATLAB* gelesen werden können.[98] Im Listing 2.4 ist ein *Plotly* Beispieldiagramm zu sehen. Die ausführliche Beschreibung von *Plotly.py* ist im Abschnitt 2.4.6 zu finden.

[92] vgl. [TP]
[93] vgl. [CD19]
[94] vgl. [D3]
[95] vgl. [BG]
[96] vgl. [BOK]
[97] vgl. [PL]
[98] vgl. [PLT]

```
traces = []
for g in graduate_students['Graduation_Grade'].unique():
    traces.append(
        go.Scatter(
           mode='markers',
           x=graduate_students.Duration_Start_Graduation[graduate_students['Graduation_Grade'] == g],
           y=graduate_students.Graduation_Semester[graduate_students['Graduation_Grade'] == g],
           name= g))
fig = go.Figure(
    layout=dict(
        title='Anzahl Semester vs Abschlussnote',
        xaxis={'title': 'Anzahl Semester'},
        yaxis={'title': 'Jahr'},
    ),
    data=traces
)
fig.show()
```



Listing 2.4: Beispiel eines Streudiagramms mit Quellcode umgesetzt mit *Plotly*. Anzahl Semester vs Abschlussnote, Datenquelle: Students Advice, Visualisierung: eigenes Werk.

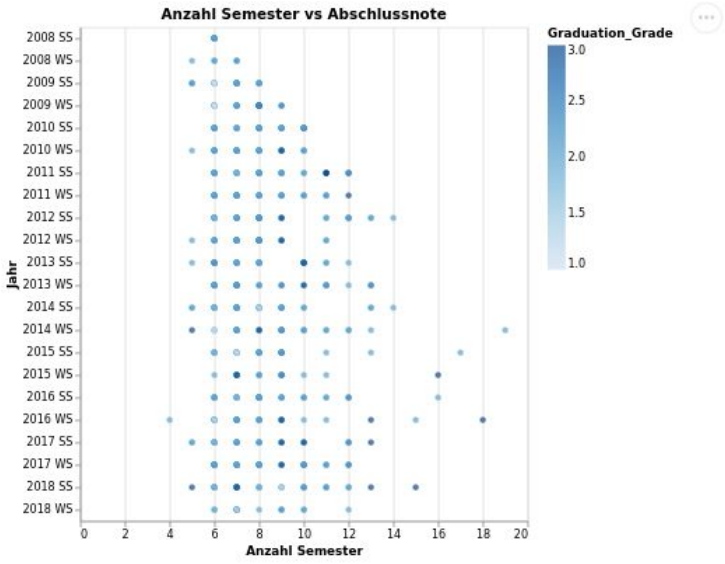
Eine weitere Gruppe bilden die deklarativen Bibliotheken wie *Altair* (zuvor *Vincent*) basierend auf *JSON*-Spezifikation. Sie setzen intern auf der *JavaScript*-Plotting-Bibliothek *Vega* [99] (bzw. *Vega-Lite*) auf, die das visuelle Erscheinungsbild als auch das interaktive Verhalten einer Visualisierung in einem JSON-Format beschreiben.[100]

Altair[101] ist eine statistische relativ neue Bibliothek zur Erstellung von interaktiven Grafiken. *Altair* ist ähnlich wie *plotnine* deklarativ, das bedeutet, dass der Fokus bei der Erstellung einer Visualisierung primär auf den Daten liegt, nicht wie im Falle von imperativer Bibliothek wie

[99] vgl. [VG]
[100] vgl. [AP]
[101] vgl. [ALT]

matplotlib auf dem Code.[102] Im Folgenden ist zum Vergleich das mit *Altair* umgesetzte Beispieldiagramm in Listing 2.5 zu sehen.

```
alt.Chart(graduate_students,
  title='Anzahl Semester vs Abschlussnote'
  ).mark_circle().encode(alt.X('Duration_Start_Graduation',
                          title='Anzahl Semester'
                          ), alt.Y('Graduation_Semester',
                          title='Jahr'),
                          color='Graduation_Grade')
```



Listing 2.5: Beispiel eines Streudiagramms mit Quellcode umgesetzt mit *altair*. Anzahl Semester vs Abschlussnote, Datenquelle: StudentsAdvice, Visualisierung: eigenes Werk.

2.4 Verwendete Technologien

In diesem Abschnitt werden die Technologien vorgestellt, die im Rahmen dieser Bachelorarbeit verwendet werden. Zunächst werden die Technologien in Bezug auf die Datenvorbereitung erläutert. Dazu wird die Programmiersprache *Python* und die Entwicklungsumgebung *Anaconda* sowie *Jupyter Notebooks* vorgestellt. Zusätzlich werden die für die Verwaltung von Daten konzipierten Bibliotheken *Pandas* und *NumPy* präsentiert. Anschließend wird die für die Erstellung der Visualisierungen eingesetzte Bibliothek *Plotly.py* beschrieben.

[102] vgl. [PD]

2.4.1 Python

Python[103] ist eine weit verbreitete, plattformübergreifende, objektorientierte Programmiersprache mit effizienten abstrakten Datenstrukturen (z.B. *Listen, Dictionaries, Tupel*) und als eine interpretierbare Sprache sowohl für Skripte als auch für schnelle Anwendungsentwicklung gut geeignet. Darüber hinaus lassen sich gut mit *Python* die Datenanalyseaufgaben dank guter Unterstützung der Bibliotheken wie etwa *Pandas* umsetzen.

2.4.2 Anaconda

Anaconda[104] ist eine Open-Source-Distribution für die Programmiersprachen *Python* und *R* und gleichzeitig ein Paket- und Umgebungsverwaltungssystem. *Anaconda* enthält die Entwicklungsumgebung *Spyder*, inklusive dem *IPython* und ein webbasiertes Frontend für *Jupyter*. Des weiteren besitzt *Anaconda* einen eigenen Paketmanager für Paketverwaltung namens *conda*. Zu dieser Version gehört eine Sammlung von mehr als 1.000 vorinstallierter Pakete. Diese Pakete belegen mehrere Gigabyte Speicherplatz auf der Festplatte, deshalb hat die Autorin der Arbeit die Installation von einer *Anaconda*-Variante namens *miniconda* vorgenommen und nachträglich zusätzliche Pakete installiert. Dabei ist *miniconda*[105] eine leichtgewichtige Version von *Anaconda* und enthält nur *conda* und *Python*.

2.4.3 Jupyter Notebook

Jupyter Notebook[106] ist eine webbasierte interaktive Umgebung, mit der die *Jupyter-Notebook* Dokumente erstellt werden können. *Jupyter Notebook* ist neben *JupyterHub* und *JupyterLab* ein Produkt einer Non-Profit-Organisation mit dem Namen *Project Jupyter*. Bis zur Version 3.x waren die Notebooks in *IPython* integriert.[107] Mit der Version 4.x sind diese ein eigenständiges Projekt von *Project Jupyter*. *IPython* ist dabei eine interaktive Shell zu *Python* und fungiert als ein Standard-Kernel für *Jupyter Notebooks*. Dabei kann Jupyter Notebook verschiedene sog. Kernel ausführen. In diesem Zusammenhang steht Kernel für Programme, die Anfragen (etwa nach Programmausführungen) beantworten und mit anderen *Jupyter*-Komponenten interagieren.[108] Des weiteren ist ein *Jupyter-Notebook* Dokument intern

[103] vgl. [PY]
[104] vgl. [AN]
[105] vgl. [MI]
[106] vgl. [JP]
[107] vgl. [IP]
[108] vgl. [LM]

ein JSON-Dokument und besteht aus einer Liste von Eingabe- und Ausgabezellen, die unter anderem ausführbaren Code, Text und statische und dynamische Visualisierungen enthalten können. Außerdem können die Dokumente in einem Format (HTML, PDF, LaTeX) gespeichert werden.

2.4.4 Numpy

NumPy[109] (*Numerical Python*) ist eine *Python*-Bibliothek, die *Python* um große, mehrdimensionale Arrays und Matrizen sowie um mathematische Funktionen für numerische Berechnungen wie etwa lineare Algebra erweitert. Der Datentyp *ndarray* von *NumPy* ist ein mehrdimensionales Array, das vektorisierte arithmetische Operationen besitzt.

2.4.5 Pandas

Pandas[110] ist eine Open-Source Bibliothek zur Datenanalyse und Datenmanipulation für *Python*. Pandas baut auf *NumPy* auf und bietet eine effiziente *DataFrame*-Implementierung. Darüber hinaus bietet *Pandas* Datenanalysewerkzeuge, Datenstrukturen und Operationen zur Manipulation von numerischen Tabellen und Zeitreihen. Die Bibliothek bietet einen sehr effizienten Umgang mit den Daten an, da sie auf *C* Programmiersprache basiert. Zu den Hauptmerkmalen von *Pandas* gehören: die einfache Handhabung fehlender Daten, das Einfügen und Löschen von Spalten, das intuitive Zusammenführen und Zusammenfügen von Datensätzen, sowie zeitreihenspezifische Funktionen wie die Erzeugung von Datenbereichen.
Zu den fundamentalen *Pandas*-Strukturen gehören: *Series*, *DataFrame* und *Index*.[111] Dabei ist eine Pandas-*Series* ein eindimensionales Array indizierter Daten. Bei *DataFrames* handelt es sich um mehrdimensionale Arrays mit Bezeichnungen für Zeilen und Spalten, wobei ein *DataFrame* schlichtweg einem Tabellenblatt mit vergleichbaren Funktionen wie *Excel* gleichzustellen sein kann. Die beiden Strukturen besitzen einen explizit definierten *Index*, über den der Zugriff und die Datenmodifikation möglich ist. Des weiteren besitzen die Spalten des *DataFrame* Objektes verschiedene Datentypen, deren Übersicht in der Abbildung 2.18 dargestellt wird.

[109] vgl. [NUM]
[110] vgl. [PAN]
[111] vgl. [VL18] S. 124f

Pandas dtype	Python type	NumPy type	Usage
object	str or mixed	string_, unicode_, mixed types	Text or mixed numeric and non-numeric values
int64	int	int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64	Integer numbers
float64	float	float_, float16, float32, float64	Floating point numbers
bool	bool	bool_	True/False values
datetime64	NA	datetime64[ns]	Date and time values
timedelta[ns]	NA	NA	Differences between two datetimes
category	NA	NA	Finite list of text values

Abbildung 2.18: Übersicht über Pandas Datentypen, Quelle[112].

2.4.6 Plotly

Plotly.py[113] ist eine interaktive Open-Source Visualisierungsbibliothek. Die Bibliothek ermöglicht das Erstellen von über 40 Diagrammtypen mit dem Fokus auf statische, finanzielle, wissenschaftliche, geographische und 3D Themen.[114] Aufbauend auf der *Javascript*-Bibliothek *Plotly.js* können mittels *Plotly.py* die Visualisierungen als HTML-Dateien gespeichert werden und erhalten weitere Funktionen wie beispielsweise Zoom, Speicherung als PNG. Des weiteren stellt die Bibliothek verschiedene Steuerelemente wie Buttons, Auswahllisten sowie Sliders zur Verfügung. Als Einstiegspunkt in das Erlernen der Bibliothek ist *Plotly Express* empfohlen.[115] *Plotly Express* ist dabei eine einfach zu bedienende *High-Level*-Schnittstelle zu *Plotly*, die die Erstellung der Diagramme vereinfacht.

Die durch *Plotly.py* erstellten Grafiken werden aufgrund baumartiger Datenstrukturen, sog. *Figures* genannt, beschrieben. Ferner werden sie als Bäume mit Knoten, sog. Attributen konstruiert. Der Wurzelknoten hat dabei drei Attribute: *data*, *layout* und *frames*.[116] Die Abbildung 2.19 verdeutlicht diese Zusammenhänge. Hierzu besteht das Attribut *data* aus einer Liste typisierter Objekte, die als *trace* bezeichnet werden. Jedes *trace* verfügt über mehr als 40 Arten der Visualisierung wie z.B *bar, scatter.*

[112] vgl. [PAT]
[113] vgl. [PL]
[114] vgl. [PL] Seite: *Getting Started with Plotly*
[115] vgl. [PL] Seite: *Plotly Express*
[116] vgl. [PL] Seite: *The Figure Data Structure*

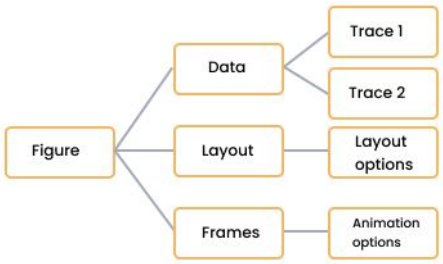


Abbildung 2.19: Die zugrunde liegende Struktur des *Figure* Objektes, Quelle[117]

Das Attribut *layout* wird durch ein *Dictionary* beschrieben, in dem die Werte das Aussehen der *Figure* wie z.B. Titel, Typographie, Achsen, Annotationen oder Legende beeinflussen. Das Attribut *frames* ist von der Struktur her eine Liste von *Dictionaries*, das im Allgemeinen die Animation der Visualisierung ermöglicht.
Anschließend werden die oben beschriebenen Attribute an ein *Figure* Objekt übergeben. Dabei kann *Figure* durch ein *Python Dictionary* (vgl. Listing 2.6 a) oder durch eine Instanz der Klasse `plotly.graph_objects.Figure` (vgl. Listing 2.6 b) beschrieben werden.

```
fig = dict({
    "data": [{"type": "bar",
              "x": [1, 2, 3],
              "y": [1, 3, 2]}],
    "layout": {"title": {"text": "A Figure Specified By Python Dictionary"}}
})
pio.show(fig)
```

Listing 2.6 a: Plotly Figure beschrieben durch ein *Dictionary*

```
fig = go.Figure(
    data=[go.Bar(x=[1, 2, 3], y=[1, 3, 2])],
    layout=go.Layout(
        title=go.layout.Title(text="A Figure Specified By A Graph Object")
    )
)
fig.show()
```

Listing 2.6 b: Plotly Figure beschrieben durch *Grafikobjekte*

Im Vergleich zur *Dictionary* basierten Vorgehensweise zur Erstellung von Visualisierungen mittels *Plotly.py* werden die *Figuren* mit Hilfe von Klassenhierarchie namens Grafikobjekte (engl. *graph objects)* konstruiert. Die Grafikobjekte gehören zum Modul

[117] vgl. [PLT]

`plotly.graph_objects`. Das Verwenden von Grafikobjekten kann als vorteilhafter gegenüber dem Verwenden von *Dictionary* basierter Vorgehensweise unter anderem aus folgenden Gründen angesehen werden:[118]

- *graph objects* stellen Funktionalitäten für die Datenvalidierung bereit.
- Die Eigenschaften von *graph objects* können sowohl durch einen *Dictionary* Wertefilter `fig["layout"]` als auch durch klassenortientierte Weise `fig.layout` zugegriffen werden
- *graph objects* stellen Funktionen zum Aktualisieren bereits erstellter *Figuren* bereit wie `.update_layout()` und `.add_trace()`
- *graph objects* Konstruktoren und Aktualisierungsmethoden akzeptieren den Einsatz von einer sogenannten `magic_underscores` Notation, die ermöglicht, die *Dictionary* basierte Schreibweise zu vereinfachen (z.B. `go.Figure (layout_title_text = "The Title")` anstelle von `(layout = dict (title = dict (text = "The Title")))`

Im Rahmen der Arbeit werden folgende Technologieversionen unter dem Betriebssystem *Ubuntu 18.04.4 LTS* verwendet:

Python	3.8.3
Jupyter Notebook	6.0.3
Pandas	1.0.3
NumPy	1.18.1
Plotly.py	*4.6.0*

Als Versionsverwaltungssystem wird GitLab benutzt. Der Link zum Repositorium: https://gitlab.beuth-hochschule.de/studentsadvice/studentsadvice_bt_kusnierz

[118] vgl. [PL] Seite: *Graph Objects*

3. Anwendungsfälle für Visualisierungen im Rahmen der Exploration

In diesem Kapitel werden die Anwendungsfälle für Visualisierungen vorgestellt. Die Tabelle 3.1 gibt einen Überblick über die wichtigsten Charakteristika der vorgestellten Anwendungsfälle. Diese verfolgen ein gemeinsames Ziel und zwar einen möglichst aussagekräftigen Überblick über die Leistungen der Studierenden zu gewährleisten. Dabei werden die unten vorgestellten Visualisierungen in zwei Bereiche unterteilt. Im ersten Bereich werden die Visualisierungen vorgestellt, die den Vergleich von Lehrveranstaltungen[119] in Hinsicht auf die Noten ermöglichen. Der zweite Bereich umfasst die Visualisierungen, die die Leistungen der Studierenden über die Zeit darstellen.

Nr.	Name	Visualisierungsform	Details
Vergleich von Lehrveranstaltungen			
1.	*GradesOverviewStatisticsHeatmap*	Heatmap	Übersicht über Durschnittsnoten
2.	*GradesOverallChart*	Gruppiertes Säulendiagramm Stapeldiagramm	Notenverteilung nach Fachsemester
Noten im zeitlichen Verlauf			
3.	*GradesLabelChart*	Liniendiagramm Stufendiagramm Gruppiertes Säulendiagramm	Noten nach Lehrveranstaltungergebnis
4.	*GradesTimeChart*	Stapeldiagramm Liniendiagramm Flächendiagramm	Notenverteilung im zeitlichen Verlauf
5.	*GradesStatisticsTimeBoxplot*	Boxplot Violinplot	Notenverteilung im zeitlichen Verlauf

Tabelle 3.1: Wichtige Merkmale der betrachteten Anwendungsfälle

Anzumerken ist an dieser Stelle, dass alle Visualisierungen in Form von einer interaktiven Visualisierung umgesetzt werden. Die Interaktivität wird durch die *Python* Bibliothek *Plotly* bereitgestellt. (vgl. Abschnitt 2.4.6) Dabei ist für jede Visualisierung bis auf *GradesOverviewStatisticsHeatmap* der Wechsel zwischen verschiedener Darstellungsformen

[119] Die Begriffe Modul und Lehrveranstaltung werden in der Arbeit synonym verwendet.

möglich, um den explorativen Charakter der Visualisierungen zu unterstützen und mögliche Nachteile einer bereits gewählten Darstellungsform auszugleichen. Des weiteren verfügt jede Visualisierung über eine Filterfunktion nach Studentenstatus. Diese wird durch einen Methodenaufruf ausgelöst. Das Ergebnis wird in jeder Visualisierung im Titelbereich unter einem Label *Studentenstatus* sichtbar.

3.1 Vergleich von Lehrveranstaltungen

GradesOverviewStatisticsHeatmap

Mit dem Ziel, eine Übersicht über die Noten zu verschaffen, wurde eine annotierte Heatmap entwickelt, im Folgenden *GradesOverviewStatisticsHeatmap* genannt. In Abbildung 3.1 ist eine beispielhafte Darstellung von dieser Abbildung zu sehen. Dabei wird der Zeitraum in Semester von 2005 bis zu 2019 oben und die Lehrveranstaltungen links angeordnet. Die Lehrveranstaltungen sind nach dem Fachsemester sortiert, in dem sie belegt werden sollten, wobei die Wahlpflichtmodule ein Präfix *WP* enthalten. Die Zellen stellen die Werte als arithmetischer Mittelwert oder als Median dar und werden in Abhängigkeit vom jeweiligen Wert eingefärbt. Die *null* Werte ergeben sich aus den fehlenden Daten im jeweiligen Semester und der Tatsache, dass einige Lehrveranstaltungen nicht im jeweiligen Semester angeboten wurden.

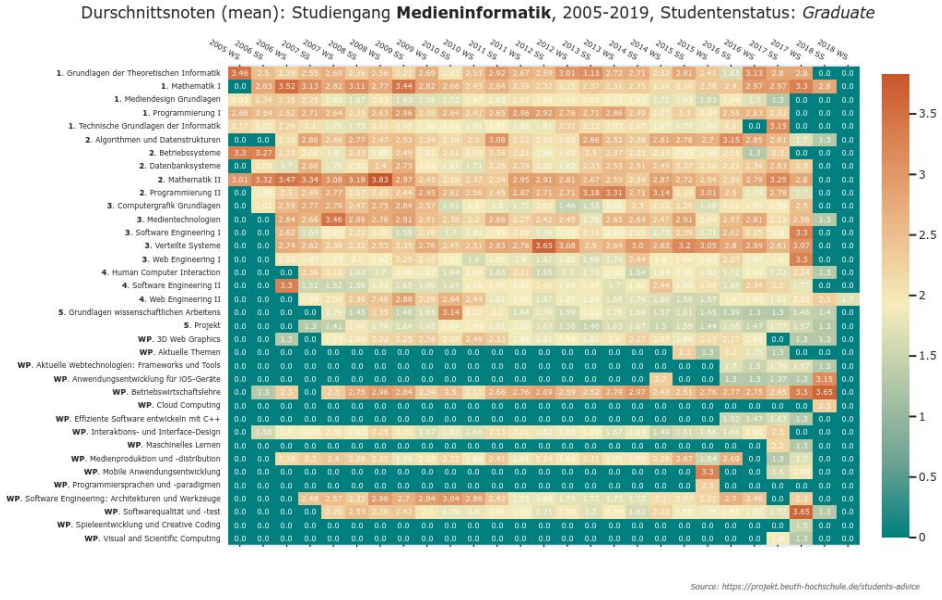


Abbildung 3.1: Beispiel für eine Heatmap von Durschnittsnoten.

Die Darstellung gibt offensichtliche visuelle Hinweise darauf, wie die Noten gruppiert werden

und wie sie sich über die Zeit ändern. Des weiteren können der Visualisierung die Informationen entnommen werden, wo sich die guten und schlechten Noten befinden. Der Nachteil ist jedoch, dass die Notenstufen nicht ersichtlich sind sowie die Noten nur über zwei Parameter, dem Median und dem Mittelwert beschrieben werden.

GradesOverallChart

Das Stapeldiagramm von Noten nach Fachsemester, im Folgenden *GradesOverallChart* genannt, liefert im Vergleich zu *GradesOverviewStatisticsHeatmap* einen vollständigen Überblick über die Noten jeweiliger Lehrveranstaltungen. Ein Beispiel einer solchen Visualisierung ist der Abbildung 3.2 zu entnehmen.

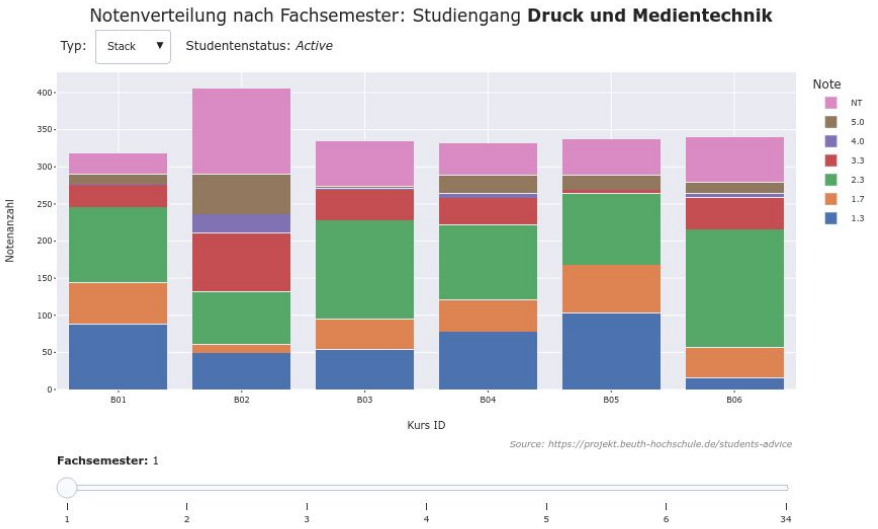


Abbildung 3.2: Beispiel für ein gestapeltes Säulendiagramm von Noten nach Fachsemester.

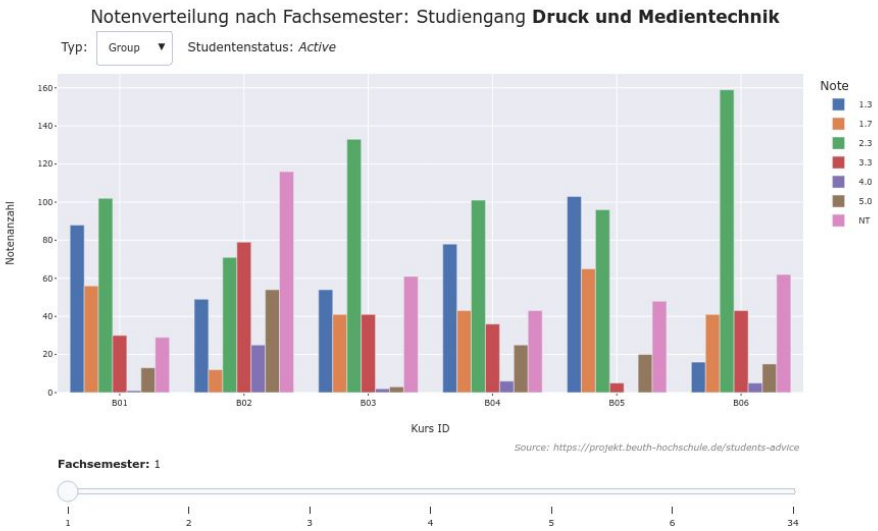


Abbildung 3.3: Beispiel für ein gruppiertes Säulendiagramm von Noten nach Fachsemester.

Dabei zeigt die *y*-Achse die absolute Anzahl der Studierenden nach erzielter Note, die *x*-Achse zeigt die Lehrveranstaltungen, die einem Fachsemester, in dem sie belegt werden sollten, zugeordnet sind. Dabei kann das gewünschte Fachsemester mit dem Schieberegler ausgewählt werden. Neben den Notenstufen wird zusätzlich eine Abkürzung *NT* verwendet, die für die Studierenden steht, die eine Lehrveranstaltung belegt haben. Die Namen der jeweiligen Lehrveranstaltungen sowie die Notenanzahl wird durch Tooltips angezeigt. Alternativ kann die Darstellungsform durch Auswahl in einem Drop-Down Menü gewechselt werden. Neben dem Stapeldiagramm steht ein gruppiertes Säulendiagramm zur Verfügung, mit dem die Lehrveranstaltungen innerhalb eines Semesters direkt miteinander verglichen werden können (vgl. Abb. 3.3).

3. 2 Noten im zeitlichen Verlauf

GradesLabelChart

Das Liniendiagramm, im Folgenden *GradesLabelChart* genannt, stellt die Leistungen der Studierenden nach übergreifendem Ergebnis einer Lehrveranstaltung, zusammengefasst als *bestanden*, *nicht bestanden* und *belegt*, gegenüber der Zeit. In Abbildung 3.4 ist eine beispielhafte Darstellung von diesem Diagramm zu sehen. Dabei zeigt die *y*-Achse die absolute Anzahl der Noten, die *x*-Achse zeigt den Zeitraum von 2005 bis zu 2019 aufgeteilt in Semester. Die Anteile der Ergebnisse einer Lehrveranstaltung sind mit der entsprechenden Farbwahl codiert, im Folgenden *Rot* für *nicht-bestanden*, *Grün* für *bestanden* und *Blau* für *belegt*. Somit können die Ergebnisse intuitiv einsehbar sein. Die Lehrveranstaltungen werden nach Fachsemester sortiert und können mit Hilfe von Schiebereglern gewählt werden.
Zusätzlich kann die Visualisierungsform gewechselt werden. Zur Auswahl steht eine Art vom Liniendiagramm - ein *Stufendiagramm* (vgl. Abb. 3.5), das die Daten als eine Reihe horizontaler und vertikaler Schritte anstelle einer Kurve zeigt, sowie ein gruppiertes Säulendiagramm.
Das Diagramm ermöglicht, Aussagen über die Leistungen der Studierenden im jeweiligen Semester zu treffen. Des weiteren kann das Diagramm weitere Erkenntnisse liefern wie beispielsweise den Zeitpunkt, an dem es zu vielen Belegungen von Lehrveranstaltungen im Vergleich zu anderen Lehrveranstaltungen bzw. zu einer starken Änderung der jeweiligen Ergebnisse gekommen ist.

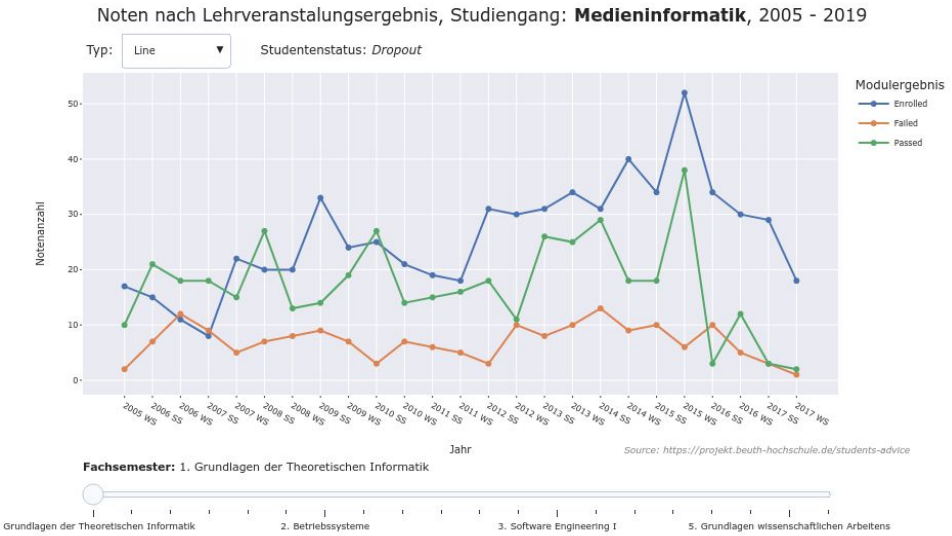


Abbildung 3.4: Beispiel eines Liniendiagramms nach Ergebnis der Lehrveranstaltung über die Zeit.

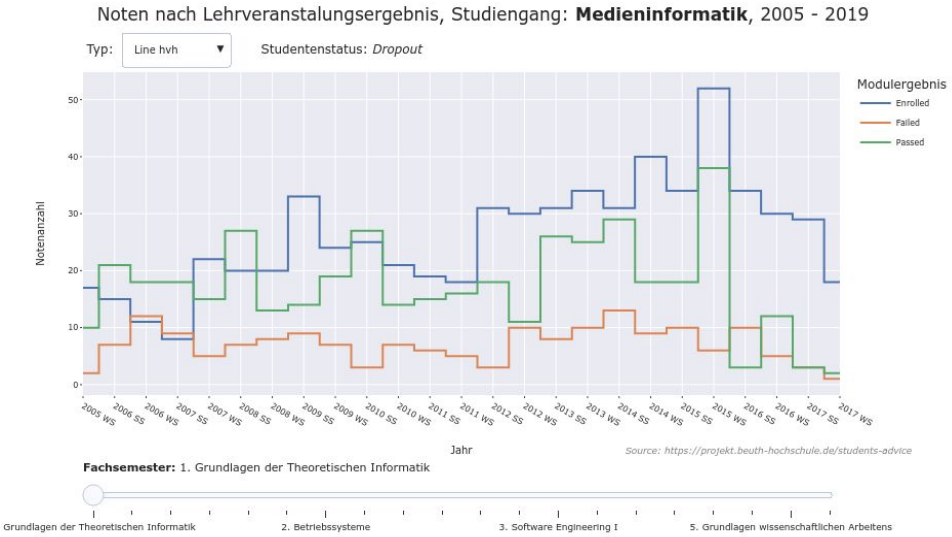


Abbildung 3.5: Beispiel eines Stufendiagramms nach Ergebnis der Lehrveranstaltung über die Zeit.

GradesTimeChart

Das Stapeldiagramm, im Folgenden *GradesTimeChart* genannt, basiert auf dem Liniendiagramm *GradesLabelChart* und zeigt die Noten einer mit Hilfe von einem Schieberegler ausgesuchten Lehrveranstaltung gegenüber der Zeit. Auf der *x*-Achse ist der Zeitraum von 2005 bis zu 2019 aufgeteilt in Semester aufgetragen, auf der *y*-Achse ist die absolute Anzahl der Studierenden gestapelt nach erzielter Note zu sehen. In Abbildung 3.6 ist eine beispielhafte

Visualisierung von *GradesTimeChart* dargestellt. Zusätzlich kann in dieser Darstellung die Visualisierungsform gewechselt werden. Zur Auswahl steht ein Flächendiagramm (vgl. Abb. 3.7) sowie ein Stufendiagramm.

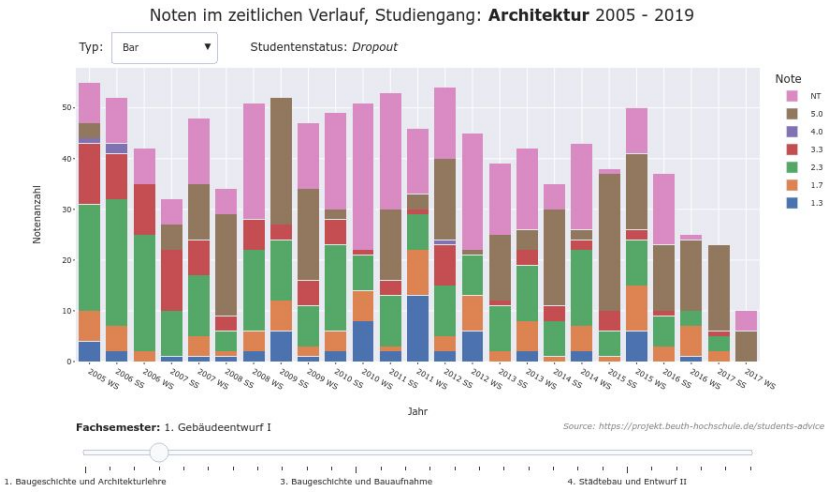


Abbildung 3.6: Beispiel für ein Säulendiagramm nach Notenskala gegenüber der Zeit.

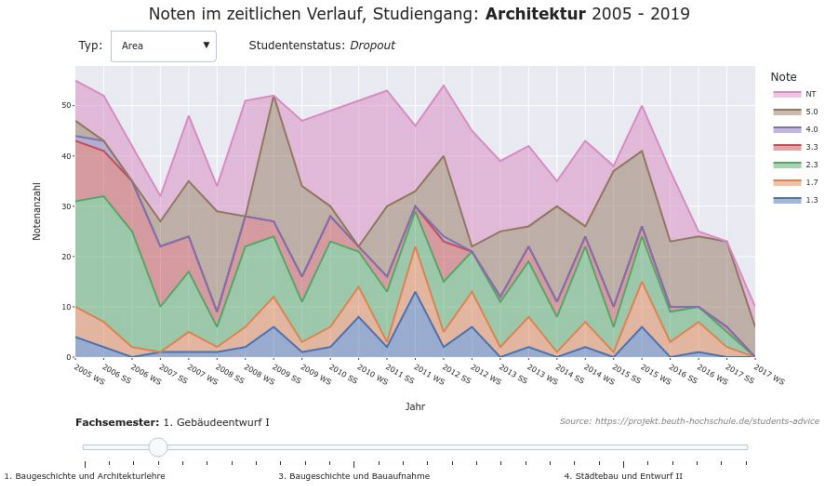


Abbildung 3.7: Beispiel für ein Flächendiagramm nach Notenskala gegenüber der Zeit.

GradesStatisticsTimeBoxplot

Um eine individuelle Analyse der Notenverteilungen jeweiliger Lehrveranstaltungen gegenüber der Zeit zu ermöglichen, wurde ein Boxplot-Diagramm im Folgenden *GradesStatisticsTimeBoxplot* genannt, entwickelt. In der Abbildung 3.8 ist eine beispielhafte Abbildung von *GradesStatisticsTimeBoxplot* zu sehen. Auf der *x*-Achse ist der Zeitraum in Semester von 2005 bis zu 2019 aufgetragen, auf der y-Achse sind die Noten aufgetragen. Des

weiteren ermöglicht das Diagramm die Aussagen über die Schiefe der Verteilung und Ausreißer. Dieses Diagramm wurde mit *Whiskern* bis zu einer Länge des 1,5-fachen Interquartilsabstands erstellt, die an dem größten bzw. kleinsten Datenwert enden. Zudem wurden dieser Visualisierung folgende Mittel der deskriptiven Statistik hinzugefügt: die Standardabweichung und der arithmetische Mittelwert. Außerdem kann die Auswahl des Diagrammtyps gewechselt werden. Dabei kann die Anzeige zu Violinplot erfolgen, die die Interpretation der Verteilung durch Dichtekurven verbessert.

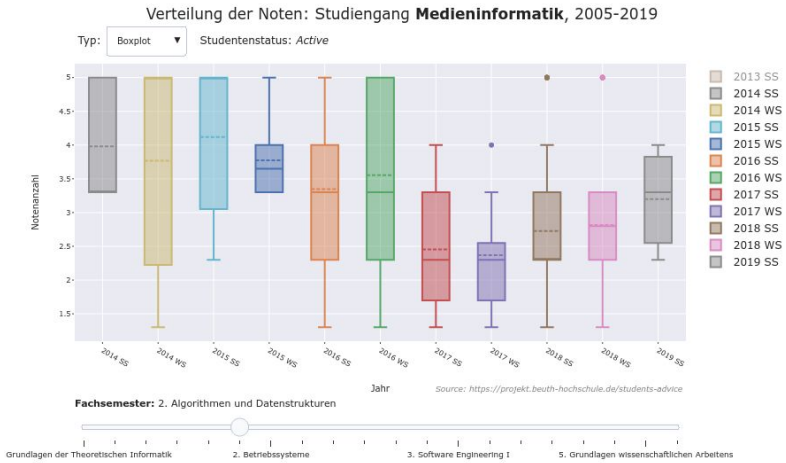


Abbildung 3.8: Beispiel eines Boxplots der Notenverteilung gegenüber der Zeit.

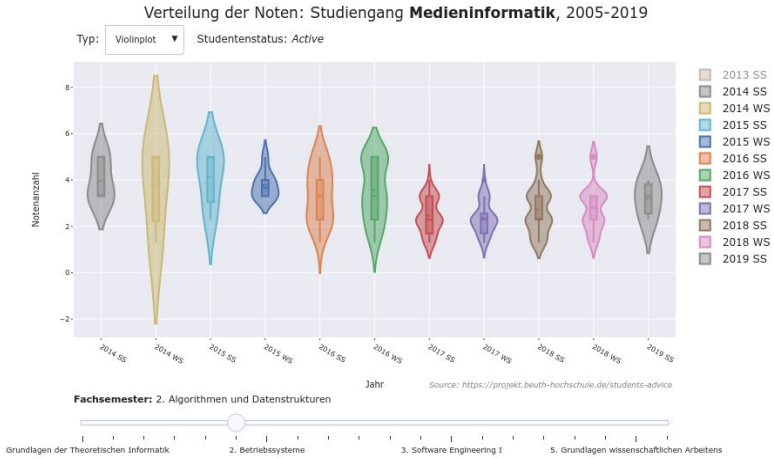


Abbildung 3.9: Beispiel eines Boxplots der Notenverteilung gegenüber der Zeit.

4. Umsetzung

In diesem Kapitel werden zunächst die Datenquellen beschrieben, die im Rahmen dieser Arbeit verwendet werden. Danach wird die Datenvorbereitung erläutert. Im Anschluss werden die einzelnen Schritte für die Implementierung der Diagramme vorgestellt. Die Implementierung der einzelnen Diagramme besitzt einen ähnlichen Aufbau. Der einzige Unterschied liegt an der Datenaufbereitung, deshalb werden zunächst exemplarisch die einzelnen Umsetzungsschritte anhand des Diagramms *GradesTimeChart* beschrieben. Ergänzend dazu wird auf die Aufbereitungsschritte zum Erstellen von *GradesOverviewStatisticsHeatmap* eingegangen. Die Vorbereitung aller Datensätze sowie Visualisierung erfolgt anhand *Jupyter Notebooks*.

4.1 Blick in die Daten

Der Datenbestand umfasst die Daten eines sechssemestrigen Bachelor-Studium mit 6025 Studenten seit dem Jahr 2005 bis 2019 mit den Studiengängen *Medieninformatik*, *Architektur* und *Druck- und Medientechnik*. Dabei gehören die Studiengänge Medieninformatik und Druck- und Medientechnik zum Fachbereich *Informatik und Medien*, wohingegen der Studiengang Architektur dem Fachbereich *Architektur und Gebäudetechnik* gehört. Die Daten stehen als CSV-Tabellen zur Verfügung und sind im UTF-8-Format codiert.
Insgesamt gibt es sieben CSV-Dateien: *equivalence_data.csv, grades_data.csv, module_data.csv, module_grades_data.csv, programs_csv, students_data.csv* und *units.csv*. Die Daten sind bereinigt und wurden durch Mitglieder des Projekts *Students Advice* auf Vollständigkeit, Einheitlichkeit und Fehler geprüft. Für die Arbeit ist folgende CSV-Tabelle *grades_data.csv* relevant.
Der Datensatz *grades_data.csv*, im Folgenden **(GR**) genannt, verfügt über 27 Spalten und 348.230 Zeilen. Zur Veranschaulichung werden mittels der *Pandas*-Methode `info()` die grundlegenden Informationen über die Spalten und Datentypen dargestellt. (vgl. Abbildung 4.1) Dabei beinhaltet der Datensatz die Informationen über die Studierenden und deren erzielten Noten für alle Studiengänge. Die Angaben über Studierenden umfassen u.a. die Kennung des Studiengangs (`Student_Program_ID`), den Abschlussstatus (`Student_Label`) für derzeit aktive Studierende, die Studiengangsabbrecher und -Absolventen, den Beginn des Studiums (`Start_Semester`) sowie das Geschlecht (`Gender`). Bei den Informationen über die Noten wird zwischen Modulnote (`Module_Grade`) und Übungsnote (`Unit_Grade`) unterschieden. Die beiden Notenangaben nutzen eine aggregierte Notenskala, die aus folgenden Notenwerten besteht: 1.3,

1.7, 2.3, 3.3, 4.0, 5.0. Zusätzlich werden die Ergebnisse der Noten durch die Spalte `Module_Label` mit einem entsprechenden Label versehen. Für die bestandenen Module wird das Label *bestanden*, für nicht bestandenen Module - *nicht bestanden* und bei den Modulen, bei denen eine Belegung vorliegt, *belegt* verwendet. Des weiteren liefert der Datensatz die Informationen über die Art der Lehrveranstaltung (`Elective_Module`), hierzu wird zwischen einem Pflicht- und Wahlpflichtmodul unterschieden. Die weiteren Informationen umfassen das Semester des Prüfungsergebnisses (`Unit_Semester`), den Modultitel (`Module_Title`) sowie das empfohlene Fachsemester (`Plan_Semester`), an dem das Modul belegt werden sollte.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 348230 entries, 0 to 348229
Data columns (total 27 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Student_Program_ID     348230 non-null  int64
 1   Student_Program_Short  348230 non-null  object
 2   Student_ID             348230 non-null  int64
 3   Gender                 348230 non-null  object
 4   Start_Semester         348230 non-null  int64
 5   Student_Label          348230 non-null  object
 6   Module_Program_ID      348230 non-null  int64
 7   Module_Program_Short   348230 non-null  object
 8   Module_ID              348230 non-null  object
 9   Module_Grading         348230 non-null  object
 10  Module_Title           348230 non-null  object
 11  Plan_Semester          348230 non-null  int64
 12  Elective_Module        348230 non-null  bool
 13  Module_Grade           292892 non-null  float64
 14  Module_Label           348230 non-null  object
 15  Unit_ID                348230 non-null  int64
 16  Equivalent             348230 non-null  bool
 17  Unit_Type              348230 non-null  object
 18  Unit_Semester          348230 non-null  int64
 19  Nbr_Student_Semester   348230 non-null  int64
 20  Unit_Grade             200786 non-null  float64
 21  Unit_Label             348230 non-null  object
 22  Initial_Unit_ID        348230 non-null  int64
 23  Initial_Exam_Grade     130776 non-null  float64
 24  Initial_Exam_Label     348230 non-null  object
 25  Initial_Unit_Grade     194356 non-null  float64
 26  Initial_Unit_Label     348230 non-null  object
dtypes: bool(2), float64(4), int64(9), object(12)
memory usage: 67.1+ MB
```

Abbildung 4.1: Beschreibung des Datensatzes GR

4.2 Datenvorbereitung

Wie im Kapitel 4.1 angedeutet, wurden die Daten bereits vorverarbeitet. Die weitere Vorbereitung besteht darin, die Daten auf der Ebene von *Modul* zu aggregieren. Der Grund dafür war, dass es einige Module im Studiengang *Druck- und Medientechnik* gab, bei denen sowohl Seminar als auch Übung eine Note erhalten, in den Studiengängen *Architektur* und *Medieninformatik* hingegen nur die Seminarnote zählt. Die Aggregationsschritte wurden in einer Funktion zusammengefasst `aggregateToModule(df)` (siehe Listing 4.1). Die Funktion erwartet als Parameter das *Dataframe*, das aggregiert werden soll. Weiterhin werden alle relevanten Spalten durch eine Bedingung gefiltert, die für das Umsetzen aller Visualisierungen verwendet werden. Dazu gehören die Kennung des Studiengangs, der Beginn des Studiums, die Modulnote

sowie der Modultitel, der Abschlussstatus und das empfohlene Fachsemester. Anschließend werden einige der Spalten umbenannt, Duplikate entfernt sowie der Index zurückgesetzt.

```
def prepare_data(df):
    grade_results = df[df['Student_Program_ID'] == df['Module_Program_ID']][[\
        'Student_Program_ID',
        'Student_Program_Short',
        'Student_ID',
        'Student_Label',
        'Start_Semester',
        'Module_ID',
        'Elective_Module',
        'Unit_Semester',
        'Gender',
        'Module_Title',
        'Plan_Semester',
        'Nbr_Student_Semester',
        'Module_Grade',
        'Module_Label',
        ]].rename(columns={'Student_Program_ID': 'Program_ID','Student_Program_Short':
'Program_Short'})\
          .drop_duplicates().reset_index(drop=True)
    return grade_results
```

Listing 4.1: Funktion für Modul Aggregation

Die Rückgabe ist ein neues *DataFrame* mit den aggregierten Daten für alle Studiengänge (vgl. Abbildung 4.2), das für den weiteren Gebrauch im nächsten Abschnitt beschrieben wird.

	Program_ID	Program_Short	Student_ID	Student_Label	Start_Semester	Module_ID	Elective_Module	Unit_Semester	Gender	Module_Title	Plan_Semester	Nbr_Student_Semester	Module_Grade	Module_Label	Unit_Fsemester
0	113	B-DMT	8	Graduate	4028	B01	False	4028	M	Drucktechnologie I	1	1	2.3	Passed	2014 SS
1	113	B-DMT	8	Graduate	4028	B02	False	4028	M	Grundlagen Statistik	1	1	3.3	Passed	2014 SS
2	113	B-DMT	8	Graduate	4028	B03	False	4028	M	Grundlagen Informationstechnik	1	1	2.3	Passed	2014 SS
3	113	B-DMT	8	Graduate	4028	B04	False	4028	M	Druckvorstufe	1	1	3.3	Passed	2014 SS
4	113	B-DMT	8	Graduate	4028	B05	False	4028	M	Grafik-Design	1	1	1.7	Passed	2014 SS
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
144284	153	B-ARCH	7014	Dropout	4024	B01	False	4024	M	Gebäudeentwurf I	1	1	5.0	Failed	2012 SS
144285	153	B-ARCH	7014	Dropout	4024	B02	False	4024	M	Entwerfen und Konstruieren in Massivbauweise	1	1	NaN	Enrolled	2012 SS
144286	153	B-ARCH	7014	Dropout	4024	B03	False	4024	M	Darstellende Geometrie in der Architektur	1	1	5.0	Failed	2012 SS
144287	153	B-ARCH	7014	Dropout	4024	B04	False	4024	M	Gestaltung und Präsentation I	1	1	NaN	Enrolled	2012 SS
144288	153	B-ARCH	7014	Dropout	4024	B05	False	4024	M	Baugeschichte und Architekturlehre	1	1	NaN	Enrolled	2012 SS

144209 rows × 15 columns

Abbildung 4.2: Datensatz GR nach der Aggregation auf Modulebene

4.3 Entwicklungsumgebung

Für die zu erstellenden Visualisierungen wird eine neue isolierte Entwicklungsumgebung mit Hilfe von *miniconda* eingerichtet, in der alle nötigen Pakete installiert werden. (vgl. Abschnitt 2.4.2) Dazu wird zunächst mithilfe vom Paketmanager *conda* der Befehl `conda create -n bachelor` ausgeführt, mit dem das Anlegen der Umgebung namens `bachelor` bezweckt wird. Um sie zu aktivieren, wird der Befehl `conda activate bachelor` ausgeführt. Im Anschluss werden die benötigten Bibliotheken mit dem Befehl `conda install <paketname>` installiert. Hierzu wird beispielsweise für die Installation von *Plotly* der Befehl `conda install -c Plotly Plotly` ausgeführt und für die Installation von *Jupyter Notebook* Erweiterungen der Befehl `conda install -c conda-forge jupyter_contrib_nbextensions` verwendet. Dabei bestimmt die Option `-c` (channel) einen Kanal (in diesem Fall *Plotly* und *conda-forge*), über den die Pakete installiert werden können. Um die Paketliste, deren Versionen und Kanäle zu überprüfen, wird danach der Befehl `conda list` ausgeführt. Daraufhin wird für jede zu erstellende Visualisierung ein neues *Jupyter Notebook* Dokument verwendet (vgl. Anhang).

4.4 Datenaufbereitung

GradesTimeChart

Die Datenaufbereitung beginnt mit dem Einlesen des Datensatzes (**GR)** mittels der *Pandas* Methode `read_csv()`. Folglich wurde die im Abschnitt 4.2 beschriebene Funktion `aggregateToModule(df)` aufgerufen, die die auf der Modulebene aggregierten Daten zurückliefert (vgl. Abbildung 4.2). Sie werden entsprechend in einem neuen *DataFrame* gespeichert. Anschließend werden die aggregierten Daten in jeweils drei *DataFrames* aufgeteilt und benannt: *MI* für Medieninformatik, *AR* für Architektur sowie *DM* für Druck und Medientechnik, damit danach die Titelangabe für jeweiligen Studiengang angezeigt werden kann. (siehe Listing 4.2)

```
grades_df = pd.read_csv('data/v1_4/grades_data.csv', sep=';')
grade_results = utils.prepare_data(grades_df)
grade_results["Unit_Fsemester"]= np.array(utils.format_semester(grade_results,"Unit_Semester"))
MI = grade_results[grade_results.Program_ID==143]
MI.name = 'Medieninformatik'
AR = grade_results[grade_results.Program_ID==153]
AR.name = 'Architektur'
DM = grade_results[grade_results.Program_ID==113]
DM.name = 'Druck und Medientechnik'
```

Listing 4.2: Vorbereitung des Datensatzes GR

Da der Schwerpunkt dieser Visualisierung auf der Darstellung der erzielten Noten im zeitlichen Verlauf liegt, muss weiterhin die für die Zeitangabe benötigte Spalte `Unit_Semester` entsprechend formatiert werden. Die Spaltenwerte sind folgend kodiert: das Sommersemester ist durch Jahreszahl × 2 kodiert und das Wintersemester durch Jahreszahl × 2 + 1. Zu diesem Zweck wird eine Hilfsfunktion `formatSemester(df, semester)` verwendet. (siehe Listing 4.3) Diese gibt die formatierten Werte als Liste mit einem entsprechenden Suffix *WS* für Wintersemester und *SS* für Sommersemester zurück.

```
def formatSemester(df,semester):
    semester_formatted=[]
    for i in df[semester]:
        if i % 2 == 0:
            year = "{}{}".format( int(i/2), ' SS')
        else:
            year = "{}{}".format( int(i/2), ' WS')
        semester_formatted.append(year)
    return semester_formatted
```

Listing 4.3: Hilfsfunktion format_semester(df, semester)

Die Überprüfung der Notenstufen erfolgt mit Hilfe von Methode `unique()`. Neben den Notenstufen existieren ebenfalls die undefinierten Werte *NaN* (Not a Number), also fehlende Werte. (vgl. Abb. 4.3) In diesem Zusammenhang bedeuten die NaN Werte in Verbindung mit der Spalte `Module_Label`, dass es sich hier um keine Notenstufe handelt, sondern um eine Belegung von einer Veranstaltung (vgl. Abb. 4.4).

```
MI.Module_Grade.unique()
array([1.3, 1.7, nan, 5. , 3.3, 2.3, 4. ])
```

Abbildung 4.3: NaN Werte in der Notenskala

```
MI[MI.Module_Grade.isnull()]['Module_Label'].unique()
array(['Enrolled', 'Passed'], dtype=object)
```

Abbildung 4.4: Bedeutung der NaN Werte

Aus diesem Grund werden die genannten Werte mit einem String *NT* (nicht teilgenommen) befüllt. Die übrigen NaN Werte werden mit der Methode `notna()` entfernt (vgl. Listing 4.4).

```
MI.loc[MI['Module_Label'] == 'Enrolled', 'Module_Grade'] = "NT"
```

```
MI = MI[MI['Module_Grade'].notna()]
```

Listing 4.4: Befüllen und Entfernen der NaN Werte

Die nächste Anpassung besteht darin, das *DataFrame* nach Notenstufen mittels der Methode `groupby()` zu gruppieren. Dabei werden mehrere Spalten in die Gruppierung einbezogen. (vgl. Listing 4.5). Anschließend wird die Gesamtzahl einzelner Notenstufen für das jeweilige Studienjahr durch eine Aggregatfunktion `count()` ermittelt. Das Ergebnis lieferte zunächst ein `DataFrameGroupBy-Objekt` zurück, das in Verbindung mit der Aggregatfunktion `count()` ein neues *DataFrame* mit einem mehrspaltigen Index zurückgab, das folglich durch die Methode `unstack()` auf herkömmliche Weise konvertiert wurde (vgl. Abb. 4.5).

```
grouped_MI = MI.groupby(['Unit_Fsemester','Plan_Semester','Elective_Module','Module_Title',
'Module_Grade'])['Module_Grade'].count().unstack().fillna(0).stack()
grouped_MI = grouped_MI.reset_index()
grouped_MI.rename(columns = {0:'Count'}, inplace=True)

grouped_MI['Module_TitleF'] = grouped_MI.apply(lambda x: \
            '{}. {}'.format(x['Plan_Semester'], x['Module_Title']),axis=1)
grouped_MI.sort_values(['Unit_Fsemester','Module_TitleF'], inplace=True)
```

Listing 4.5: Gruppierung der Datensätze

Die dadurch entstandenen *NaN* Werte wurden mit 0 Werten mittels der Methode `fillna()` befüllt. Das *DataFrame* wurde anschließend in umgekehrter Richtung mittels der Methode `stack()` konvertiert, um die Ursprungsform des *DataFrames* herzustellen.

			Module_Grade	1.3	1.7	2.3	3.3	4.0	5.0	NT
Unit_Fsemester	Plan_Semester	Elective_Module	Module_Title							
2005 WS	1	False	Grundlagen der Theoretischen Informatik	NaN	NaN	3.0	19.0	8.0	4.0	28.0
			Mathematik I	NaN	NaN	1.0	1.0	1.0	NaN	NaN
			Mediendesign Grundlagen	9.0	13.0	24.0	5.0	NaN	NaN	19.0
			Programmierung I	17.0	8.0	8.0	7.0	3.0	9.0	13.0
			Technische Grundlagen der Informatik	12.0	9.0	17.0	12.0	1.0	2.0	16.0
...	...	...	...	...	...	...	...	...	...	...
2019 SS	45	True	Interaktions- und Interface-Design	NaN	NaN	1.0	NaN	NaN	NaN	NaN
			Mobile Anwendungsentwicklung	1.0	NaN	1.0	NaN	NaN	NaN	NaN
			Software Engineering: Architekturen und Werkzeuge	NaN	NaN	NaN	NaN	1.0	NaN	NaN
			Softwarequalität und -test	1.0	NaN	1.0	NaN	NaN	NaN	NaN
			Spieleentwicklung und Creative Coding	NaN	NaN	1.0	NaN	NaN	NaN	NaN

Abbildung 4.5: DataFrame nach Anwenden der Methode unstack()

Daraufhin wurde mit Hilfe der Methode `reset_index()` ein neues *DF* mit den ursprünglichen Spalten, die sich vorher im Index befanden, zurückgegeben (vgl. Abb. 4.6). Die neu erstellte Spalte wurde entsprechend mit der Methode `rename()` umbenannt, wobei der zweite Parameter `inplace` dazu dient, die Operation direkt auf das *DataFrame* anzuwenden.

	Unit_Fsemester	Plan_Semester	Elective_Module	Module_Title	Module_Grade	0	Module_TitleF
0	2005 WS	1	False	Grundlagen der Theoretischen Informatik	1.3	0.0	1. Grundlagen der Theoretischen Informatik
1	2005 WS	1	False	Grundlagen der Theoretischen Informatik	1.7	0.0	1. Grundlagen der Theoretischen Informatik
2	2005 WS	1	False	Grundlagen der Theoretischen Informatik	2.3	3.0	1. Grundlagen der Theoretischen Informatik
3	2005 WS	1	False	Grundlagen der Theoretischen Informatik	3.3	19.0	1. Grundlagen der Theoretischen Informatik
4	2005 WS	1	False	Grundlagen der Theoretischen Informatik	4	8.0	1. Grundlagen der Theoretischen Informatik
...	...	...	...	...	...	...	...
5245	2019 SS	45	True	Spieleentwicklung und Creative Coding	2.3	1.0	45. Spieleentwicklung und Creative Coding
5246	2019 SS	45	True	Spieleentwicklung und Creative Coding	3.3	0.0	45. Spieleentwicklung und Creative Coding
5247	2019 SS	45	True	Spieleentwicklung und Creative Coding	4	0.0	45. Spieleentwicklung und Creative Coding
5248	2019 SS	45	True	Spieleentwicklung und Creative Coding	5	0.0	45. Spieleentwicklung und Creative Coding
5249	2019 SS	45	True	Spieleentwicklung und Creative Coding	NT	0.0	45. Spieleentwicklung und Creative Coding

5250 rows × 7 columns

Abbildung 4.6: DataFrame nach Anwenden der Methode reset_index()

Zusätzlich zum *Dataframe* wurde eine neue Spalte `Module_TitleF` hinzugefügt, die aus Werten der Spalte `Plan_Semester` und `Module_Titel` zusammengesetzt ist und später für die Anzeige der *Slider* Werte benötigt wird. Dazu wurde die Methode `apply()` benutzt, die es ermöglicht, eine beliebige Funktion auf alle Werte eines *Series* Objektes anzuwenden. Dazu wird eine anonyme *Lambda*-Funktion verwendet, die entlang der Zeilen die Formatierung der beiden Spalten vornimmt. Im Anschluss wurde das *DataFrame* nach Modultitel und Jahr sortiert.

GradesOverviewStatisticsHeatmap

In diesem Fall erfolgte die Datenaufbereitung mit Hilfe einer Pivot-Tabelle. Eine Pivot-Tabelle nimmt als Eingabe die Spaltendaten entgegen und arrangiert die Einträge in einer zweidimensionalen Tabelle. Für *GradesOverviewStatisticsHeatmap* sind zunächst neben den im Listing 4.2 beschriebenen Vorbereitungsschritten alle Modulnamen formatiert. Folglich wurde eine Pivot-Tabelle mit der Methode `pivot_table()` erstellt, dabei wurden die leeren Zeilen mit Nullen befüllt und eine Aggregatfunktion *mean* aufgerufen (vgl. Listing 4.6).

```
df.loc[df['Elective_Module'] == True, 'Plan_Semester'] = 'WP'
df['Unit_Title_Formatted'] = df.apply(lambda x: \
```

```
            '<b>{}</b>. {}'.format(x['Plan_Semester'], x['Module_Title']),axis=1)
grades_heatmap = df.pivot_table(values='Module_Grade',
                                index='Unit_Title_Formatted',
                                columns='Unit_Fsemester',
                                aggfunc='mean', fill_value=0)
grades_heatmap = grades_heatmap.sort_values('Unit_Title_Formatted',ascending=False)
```

Listing 4.6: Datenaufbereitung mit Hilfe von Pivottabelle

4.5 Visualisierung

Nachdem das *DataFrame* in ein entsprechendes Format umgewandelt wurde (vgl. Abb. 4.6), entstand die erste Visualisierung für ein Modul. Um eine neue Graphik im Falle von *GradesTimeChart* zu erstellen, werden dem `Figure` Objekt die Daten als Liste von *traces* in diesem Fall vom Typ *Bar* übergeben (vgl. Abschnitt 2.4.6).

```
grades = list(MI.Module_Grade.unique())
data = []
for grade in grades:
        mask = (grouped_MI.Module_Title.values == "Mathematik I")&\
                                    (grouped_MI.Module_Grade.values == grade)
        trace = go.Bar(
            name = grade,
            x = grouped_MI.loc[mask, 'Unit_Fsemester'],
            y = grouped_MI.loc[mask, 'Count'],
        )
        data.append(trace)
fig = go.Figure(data=data)
fig.show()
```

Listing 4.7: Beispiel einer Erstellung einer Abbildung mithilfe des *Figure* Objektes

Dabei referenzieren die Schlüsselwortargumente *x*, *y* auf Spalten aus dem DataFrame, die mit Hilfe eines Indizierungsoperators `loc` und einer sog. booleschen Maskierung die gefilterten Werte als *Series* Objekte jedem *trace* hinzufügen.[120] (vgl. Listing 4.7). Zur Verdeutlichung wird mit einem einfachen *print* die zugrunde liegende Datenstruktur des `Figure` Objektes in der Abbildung 4.7 dargestellt.

[120] vgl. [VL18] S.99f

```
Figure({
    'data': [{'name': '1.3',
              'type': 'bar',
              'x': array(['2005 WS', '2006 SS', '2006 WS', '2007 SS', '2007 WS', '2008 SS',
                          '2008 WS', '2009 SS', '2009 WS', '2010 SS', '2010 WS', '2011 SS',
                          '2011 WS', '2012 SS', '2012 WS', '2013 SS', '2013 WS', '2014 SS',
                          '2014 WS', '2015 SS', '2015 WS', '2016 SS', '2016 WS', '2017 SS',
                          '2017 WS', '2018 SS', '2018 WS', '2019 SS'], dtype=object),
              'y': array([ 0.,  6.,  0.,  4.,  7.,  5., 10.,  2.,  8., 16., 10.,  9., 20., 16.,
                          18., 12., 14., 13., 18., 13., 17., 19., 24., 12., 13.,  5., 17.,  0.])},
             {'name': '1.7',
              'type': 'bar',
              'x': array(['2005 WS', '2006 SS', '2006 WS', '2007 SS', '2007 WS', '2008 SS',
                          '2008 WS', '2009 SS', '2009 WS', '2010 SS', '2010 WS', '2011 SS',
                          '2011 WS', '2012 SS', '2012 WS', '2013 SS', '2013 WS', '2014 SS',
                          '2014 WS', '2015 SS', '2015 WS', '2016 SS', '2016 WS', '2017 SS',
                          '2017 WS', '2018 SS', '2018 WS', '2019 SS'], dtype=object),
              'y': array([ 0.,  2.,  2.,  0.,  4.,  1.,  2.,  1.,  3., 13., 10.,  7.,  2.,  8.,
                           8.,  9.,  5.,  3.,  5.,  6.,  6.,  2.,  6.,  5.,  4.,  9.,  7.,  0.])},
```

Abbildung 4.7: Datenstruktur des *Figure* Objektes

4.6 Interaktionen

Alle Diagramme, außer dem *GradesOverviewStatisticsHeatmap*, bieten die Möglichkeit, die Module auszuwählen und die Darstellungsform zu ändern. Zu diesem Zweck werden die durch *Plotly* bereitgestellten interaktiven Steuerelemente: Drop-Down Listen (sog. `Updatemenus`) und Schieberegler (`Sliders`) verwendet. Es ist anzumerken, dass die interaktiven Steuerelemente zum *Layout*-Attribut gehören (vgl. Kapitel 2.4.6). Dabei kann das Verhalten der interaktiven Steuerelemente durch die Angabe einer `updatemenu()` Methode bestimmt werden. Darunter gibt es vier möglichen Methoden, die sich mit einem gewählten Steuerelement verbinden lassen. Hierzu zählt die Methode *restyle*, mit der die Daten oder Datenattribute geändert werden können, die Methode *relayout* für die Modifikation der Layout-Attribute und die Methode *update*, die beides Daten und Layout modifizieren kann, sowie eine *animate* Methode, die das Aus- und Einblenden der Animationen unterstützt.

Im Falle von *GradesTimeChart* werden die *Sliders* dazu verwendet, ein gewünschtes Modul auszuwählen. Die Umsetzung von Sliders wurde neben der Anlehnung an die *Plotly* Dokumentation anhand des Tutorials durchgeführt.[121] Die Idee dahinter ist, für jedes Modul eine korrekte Diagrammansicht über die Jahre anzuzeigen. Dabei werden zuerst *steps* innerhalb eines *Sliders* definiert, die zuerst einer Liste (*sliders*) und anschließend zum *Layout*-Attribut hinzugefügt werden. Ein *step* ist dabei ein *Dictionary*, das die Anzeige von einem bestimmten *trace* festlegt (vgl. Listing 4.8). Folglich werden der `updatemenu` Methode die Argumentwerte übergeben. Als Argumentwert für die Anzeige von mehreren *traces* ist der Schlüssel *visible* ausschlaggebend. Dabei geben die in ihm enthaltenen booleschen Werte Auskunft darüber, welches Modul anzuzeigen ist.

[121] vgl. [PLS]

```
grades = list(MI.Module_Grade.unique())
steps = [dict(method='restyle', args=['visible', [True, False]],label="Mathematik II"),
         dict(method='restyle', args=['visible', [False, True]],label="Mathematik II" )]
sliders = [dict(
    active=0,
    steps=steps
)]
module_titles=["Mathematik I", "Mathematik II"]
data = []
for grade in grades:
    for title in module_titles:
        mask = (grouped_MI.Module_Title.values == title)&\
                                    (grouped_MI.Module_Grade.values == grade)
        trace = go.Bar(
            name = grade,
            x = grouped_MI.loc[mask, 'Unit_Fsemester'],
            y = grouped_MI.loc[mask, 'Count'],
            visible=True,
            text=title
            )
        data.append(trace)
layout = go.Layout(sliders=sliders)
fig = go.Figure(data=data, layout=layout)
fig.show()
```

Listing 4.8: Implementierung von Sliders

Zur Veranschaulichung wird in der Abbildung 4.8 eine vereinfachte Version der Umsetzung von *Sliders* dargestellt. Zusätzlich ist die Datenstruktur des erstellten Figure Objektes in der Abbildung 4.8 zu sehen.

```
[{'method': 'restyle', 'args': ['visible', [True, False]], 'label': 'Mathematik II'}, {'method': 'restyle', 'args':
['visible', [False, True]], 'label': 'Mathematik II'}]
Figure({
    'data': [{'name': '1.3',
              'text': 'Mathematik I',
              'type': 'bar',
              'visible': True,
              'x': array(['2005 WS', '2006 SS', '2006 WS', '2007 SS', '2007 WS', '2008 SS',
                          '2008 WS', '2009 SS', '2009 WS', '2010 SS', '2010 WS', '2011 SS',
                          '2011 WS', '2012 SS', '2012 WS', '2013 SS', '2013 WS', '2014 SS',
                          '2014 WS', '2015 SS', '2015 WS', '2016 SS', '2016 WS', '2017 SS',
                          '2017 WS', '2018 SS', '2018 WS', '2019 SS'], dtype=object),
              'y': array([ 0.,  6.,  0.,  4.,  7.,  5., 10.,  2.,  8., 16., 10.,  9., 20., 16.,
                          18., 12., 14., 13., 18., 13., 17., 19., 24., 12., 13.,  5., 17.,  0.])},
             {'name': '1.3',
              'text': 'Mathematik II',
              'type': 'bar',
              'visible': True,
              'x': array(['2005 WS', '2006 SS', '2006 WS', '2007 SS', '2007 WS', '2008 SS',
                          '2008 WS', '2009 SS', '2009 WS', '2010 SS', '2010 WS', '2011 SS',
                          '2011 WS', '2012 SS', '2012 WS', '2013 SS', '2013 WS', '2014 SS',
                          '2014 WS', '2015 SS', '2015 WS', '2016 SS', '2016 WS', '2017 SS',
                          '2017 WS', '2018 SS', '2018 WS', '2019 SS'], dtype=object),
              'y': array([ 3.,  2.,  2.,  2.,  3.,  3.,  2.,  6., 17.,  1.,  8.,  8.,  9., 11.,
                          15., 13., 11., 18.,  9., 12.,  8., 18.,  9., 16.,  3.,  4.,  6.,  0.])},
```

Abbildung 4.8: Anzeige der zugrunde liegenden Datenstruktur eines Figure Objektes aus Listing 4.8

Anschließend wurden alle zur Anzeige benötigten Module einer Liste hinzugefügt und einzelne *steps* innerhalb einer *for*-Schleife mit Modulnamen befüllt (vgl. Abbildung 4.9).

```
module_titles = sorted(grouped_MI.Module_TitleF[grouped_MI.Elective_Module == False].unique())
steps = []
for (i, title) in enumerate(module_titles):
    step = dict(method='restyle', args=['visible', [False]* len(module_titles)], label=title)
    step['args'][1][i]
    steps.append(step)

sliders = [dict(
    active=0,
    steps=steps
)]
layout = go.Layout(sliders=sliders)
fig = go.Figure(layout=layout)
fig.show()
```

Listing 4.9: Befüllen der Slider-Werte in einer for-Schleife

Die Darstellung verschiedener Darstellungsformen wurde mit Hilfe von Drop-Down Listen (`Updatemenus`) umgesetzt. Sie funktionieren ähnlich wie *Sliders* und werden im Listing 4.10 präsentiert. Sie bestehen aus einer Liste von sog. `buttons`, die das Verhalten des Diagramms beeinflussen. Als Argumentwert wird hier u.a. der Schlüssel *type* festgelegt, der nach dem Aufruf der Methode *restyle*, das Diagramm mit einem neuen Tracetyp aktualisiert. Die weiteren Schlüssel wie *stackgruop* und *mode* beschreiben in diesem Fall die Gestaltung der graphischen Elemente.

```
updatemenus=[
  dict(
      direction='down',
      pad={'b': 10, 't':20},
      font={'size': 10},
      x=0.05,
      y=1.19,
      xanchor='left',
      yanchor='top',
      buttons=list([
      dict(args=[{'type': 'bar', 'barmode': ''}], method='restyle', label='Bar'),
      dict(args=[{'type': 'scatter', 'stackgroup': 'one','mode': 'lines','line': {'shape':
''}}], method='restyle', label='Area')
        ]))]
```

Listing 4.10: Implementieren von Updatemenus

4.7 Layout

Abschließend wird das Layout vom Diagramm angepasst. Dafür wurde zunächst mit dem Eintrag im Layout `template="seaborn"` ein Basisaussehen für alle Visualisierungen gesetzt. Die verfügbaren Themen können durch den Import von *Plotly.io*[122], einer Low-Level-Schnittstelle zum Anzeigen, Lesen und Schreiben von *Plotly* Figuren, hinzugefügt werden. Neben dem Titel und den Achsenbeschriftungen wurden zusätzlich Annotationen hinzugefügt und für die Einblendung von Zusatzinformationen wird das Hover-Tool von *Plotly* verwendet (vgl. Listing 4.11).

```
layout = go.Layout(
  template="seaborn",
  hovermode='x',
  annotations=[dict(
                    x=1.1,
                    y=-0.25,
                    xref='paper',
                    yref='paper',
                    text='<i>Source:https://projekt.beuth-hochschule.de/students-advice</i>',
                    showarrow=False,
                    arrowhead=7,
                    font=dict(color='grey', size=9)),
                dict(text="Studentenstatus: <i>%s</i>" ,
                    x=0.25,
                    y=1.11,
                    xref="paper",
                    yref="paper",
                    align="left",
                    showarrow=False),
                 dict(text="Typ:",
                    x=0,
                    y=1.11,
                    xref="paper",
                    yref="paper",
                    align="left",
                    showarrow=False)])
```

Listing 4.11: Hinzufügen von Annotationen und Zusatzinformationen zum Layout

[122] vgl. [PL] Seite: *Static Image Export*

5. Zusammenfassung und Ausblick

Im Rahmen dieser Bachelorarbeit entstanden die interaktiven Visualisierungen zum Thema Studienperformanz der Studierenden, die für drei Studiengänge einsetzbar sein können. Die Visualisierungen folgen dem Ansatz der explorativen Analyse und können als Mittel zum Erkenntnisgewinn verwendet werden. Die Grundlage der erstellten Visualisierungen bilden zwei Anwendungsfälle. Zum einen wird die Verteilung der Noten hervorgehoben, zum anderen wird der Schwerpunkt auf dem Vergleich von Lehrveranstaltungen gelegt. Zudem orientierte sich die Umsetzung von Visualisierungen nach Erkenntnissen über die Anforderungen an eine gute Visualisierung sowie unter Berücksichtigung der Bewertungskriterien: der Expressivität, der Effektivität und der Angemessenheit.
Dafür wurden zunächst wichtige Grundlagen aus dem Bereich Visualisierung erarbeitet und verschiedene Visualisierungsarten aus dem Bereich der explorativen Datenanalyse recherchiert. Zusätzlich wurde ein Überblick über verschiedene *Python* Grafikbibliotheken gegeben, in dem deren Möglichkeiten und hervorstechende Merkmale aufgezeigt wurden. Ein großer und gleichzeitig aufwendigster Teil dieser Arbeit lag an der Datenaufbereitung. Zusätzlich hat sich auch die Wahl der zuvor nicht verwendeten Technologien wie *Plotly* und *Pandas* als arbeitsintensiv erwiesen. Allerdings bilden die entstandenen Visualisierungen eine mögliche Basis für die weiteren Analysen und erleichtern viele explorative Betrachtungen der Daten.

Die Umsetzung der Visualisierungen mittels *Jupyter Notebook* und *plotly.py* bietet zwar die Möglichkeit die einzelnen Diagrammen in einem Notebook aufzurufen bzw. im *HTML*-Ausgabeformat zu speichern. Eine Zusammenführung von Diagrammen in einem Dashboard würde die Exploration wesentlicher verbessern. Dazu kann ein *Python*- Framework *Dash* verwendet werden. *Dash*[123] ist zur Erstellung von Webapplikationen mit fertigen HTML Steuerkomponenten gedacht. Das Framework ermöglicht die in *Plotly* erstellten Visualisierungen direkt anzubinden. Da das Framework u.a. auf einem Micro-Webframework *Flask*[124] basiert, könnte alternativ ein Dashboard mit *Flask* implementiert werden. Dadurch kann man mehr Flexibilität beim Einsatz von *JavaScript* und *HTML* Elementen verschaffen. Weiterhin können die erstellten Visualisierungen durch Animationen angereichert werden. Diese Verbesserung könnte etwa bei den Zeitreihen-Diagrammen ihre Anwendung finden.

[123] [DA]
[124] [FL]

Quellenverzeichnis

Literatur

[BK18] Bubenhofer N., Kupietz M., *Visualisierung Sprachlicher Daten: Visual Linguistics – Praxis – Tools*. Heidelberg University Publishing, 2018.

[HQ07] Josef Hoffmann, Franz Quint, *Signalverarbeitung mit MATLAB und Simulink: Anwendungsorientierte Simulationen,* München, 2007.

[KMS+08] Keim, D. A.,Mansmann, F., Schneidewind, J., Thomas, J., Ziegler, H.: *Visual Analytics: Scope and Challenges* / Universität Konstanz und Pacifc Northwest National Laboratory, National Visualization and Analytics Center(NVAC) URL:http://kops.uni-konstanz.de/bitstream/handle/123456789/5631/Visual_Analytics_Scope_and_Challenges.pdf?sequence=1&isAllowed=y [10.05.2020]

[LW17] Lilienfeld, S., and Waldman, I. D. , *Psychological Science Under Scrutiny: Recent Challenges and Proposed Solutions*. Wiley-Blackwell, 2017.

[P94] Polasek W., EDA *Explorative Datenanalyse Einführung in die deskriptive Statistik*, Springer, Berlin, 1994.

[PD10] Bernhard Preim, Raimund Dachselt. *Interaktive Systeme Band 1: Grundlagen, Graphical User Interfaces, Informationsvisualisierung*. 2.Auflage, Berlin Heidelberg.Springer 2010.

[RS94] Rönz, B.,G. Strohe. *Lexikon Statistik*. Wiesbaden. Roth-Kleyer, 1994

[SM00] Heidrun Schumann, Wolfgang Müller. *Visualisierung: Grundlagen und allgemeine methoden*. Berlin, Heidelberg, Springer. 2000.

[TH06] Toutenburg, H., Heumann, C., *Deskriptive Statistik: Eine Einführung in Methoden und Anwendungen Mit SPSS*. Springer, 2006.

[VL18] VanderPlas, Jake, and Knut Lorenzen. *Data Science Mit Python: Das Handbuch für den Einsatz Von IPython, Jupyter, NumPy, Pandas, Matplotlib Und Scikit-Learn*. Mitp, 2018.

[WGK15] Matthew Ward, Georges Grinstein, Daniel Keim. *Interactive Data Visualization: Foundations, Techniques, and Applications*. Boca Raton USA: CRC Press. 2015.

[W20] Claus. O Wilke. *Fundamentals of Data Visualization,* O'Reilly, 2020.

Online-Quellen

[AN] *Installieren Der Anaconda Python-Distribution Unter Ubuntu 20.04.* DigitalOcean, URL:www.digitalocean.com/community/tutorials/how-to-install-the-anaconda-python-distribution-on-ubuntu-20-04-de [18.05.2020]

[AP] (Tutorial) Altair in Python: Data Visualizations. *DataCamp Community*
URL: https://www.datacamp.com/community/tutorials/altair-in-python [25.05.2019]

[CD19] *PYTHON DATA VISUALIZATION 2019 Tools and Trends*
URL: https://www.cdslab.org/python/notes/visualization/overview/pyviz.pdf [03.05.2020]

[IL] Scheuermann, G., Informationsvisualisierung 8. Spezifische Verfahren -Parallele Koordinaten, Vorlesungsskript, Universität Leipzig
URL:https://www.informatik.uni-leipzig.de/bsv/homepage/sites/default/files/Infovis_8-4-spez-pc_0.pdf [3.06.2019]

[IN] Datenvisualisierung: R vs. Python
URL:https://www.inwt-statistics.de/blog-artikel-lesen/datenvisualisierung-r-versus-python.html [3.06.2019]

[IP] Jupyter and the Future of IPython *IPython*, ipython.org/.
URL: https://ipython.org/ [3.06.2019]

[LM] Pandey, P., Jupyter Notebook Und Lab Für (Daten)-Wissenschaftler. *Linux*, 29.03.2019
URL: www.linux-magazin.de/ausgaben/2019/05/jupyter/ [3.06.2019]

[PAT] Moffitt, C. Overview of Pandas Data Types. *Practical Business Python Atom*, 26 Mar 2018, URL: https://pbpython.com/pandas_dtypes.html [15.05.2020]

[PD] *Plotting with Plotnine* - Practical Data Science
URL: https://www.practicaldatascience.org/html/plotting_part1.html [15.05.2020]

[PLS] Sylwia Mielnicka - Visualisation, AI and Data Analysis.
URL: https://sylwiamielnicka.com/blog/advanced-plotly-sliders-and-dropdown-menus/ [18.05.2020]

[PLT] *Plotly - Introduction*.Tutorialspoint
URL: https://www.tutorialspoint.com/plotly/plotly_introduction.htm [01.04.2020]

Bildquellen

[AM] *Mosaic Plot in R. DataScience Made Simple*, Bearbeitungsstand: 15.11.2019
URL: www.datasciencemadesimple.com/mosaic-plot-in-r/ [15.05.2020]

[H19] Deng, Haozhang, et al. "PerformanceVis: Visual Analytics of Student Performance Data from an Introductory Chemistry Course." *Visual Informatics*, Elsevier, 2 Nov. 2019,
URL: www.sciencedirect.com/science/article/pii/S2468502X1930049X?via=ihub [3.06.2019]

[PV18] *Python Data Visualization 2018: Why So Many Libraries?* Anaconda
URL: https://www.anaconda.com/wp-content/uploads/2019/01/PythonVisLandscape.jpg [18.07.2020]

[S13] *Streudiagrammmatrix*, Stapelkamp, T., *Informationsvisualisierung: Web - Print -*

Signaletik. Springer, 2013.

[TS20] *Visuelle Variablen*, Tominski, C., Schumann H., *Interactive Visual Data Analysis*. CRC Press, 2020.

Dokumentationen

[ALT] Declarative Visualization in Python. *Altair*
URL: https://altair-viz.github.io/ [18.04.2020]

[BG] Bqplot. Bqplot/Bqplot. *GitHub*
URL: https://github.com/bqplot/bqplot [15.05.2020]

[BOK] Contributors, Bokeh. *Bokeh 2.1.1 Documentation*
URL: https://docs.bokeh.org/en/latest/index.html [25.05.2019]

[D3] Bostock, Mike. *Data-Driven Documents. D3.Js*
URL: https://d3js.org/ [15.05.2020]

[DA] Dash User Guide. *Plotly*, *dash.plotly.com*
URL: https://dash.plotly.com/ [05.04.2020]

[FL] Welcome to Flask. *Welcome to Flask - Flask Documentation (1.1.x)*
URL: https://flask.palletsprojects.com/en/1.1.x/ [20.06.2020]

[GL] Glumpy Documentation - *Glumpy v1.x Documentation*
URL: https://glumpy.github.io/ [15.05.2020]

[GT] The GTK Team. *The GTK Project - A Free and Open-Source Cross-Platform Widget Toolkit.*, URL: https://www.gtk.org/ [15.05.2020]

[JU] The Jupyter Notebook - *Jupyter Notebook 6.0.3 Documentation*
URL: https://jupyter-notebook.readthedocs.io/en/stable/notebook.html [01.04.2020]

[JP] Project Jupyter, jupyter.org/
URL: https://jupyter.org/ [01.04.2020]

[MA] Visualization with Python. *Matplotlib*
https://matplotlib.org/ [3.06.2019]

[MI] Miniconda *Miniconda - Conda Documentation*
URL: https://docs.conda.io/en/latest/miniconda.html [18.05.2020]

[NUM] ECOSYSTEM. *NumPy*, numpy.org/., URL: http://www.numpy.org/ [15.05.2020]

[PAN] Pandas.Pydata.org. Python Data Analysis Library
URL: https://pandas.pydata.org/ [06.04.2020]

[PL] Plotly Python Graphing Library. *Plotly*
URL: https://plotly.com/python/ [01.04.2020]

[PLA] Python API Reference for Plotly. *Python API Reference for Plotly - 4.9.0*

Documentation, plotly.com/python-api-reference/.
URL: https://plotly.com/python-api-reference/ [01.04.2020]

[PY] Welcome to Python.org. *Python.org*, URL: www.python.org/about/ [18.05.2020]

[PYL] The PyLab Vision. *PyLab - SciPy Wiki Dump*
URL: https://scipy.github.io/old-wiki/pages/PyLab [20.05.2020]

[SE] *Statistical Data Visualization.* Seaborn
URL: https://seaborn.pydata.org/ [10.05.2020]

[TP] *Welcome! - Toyplot 0.19.0 Documentation*
URL: https://toyplot.readthedocs.io/en/stable/ [15.05.2020]

[VG] A Visualization Grammar. *Vega*, *vega.github.io/vega/*
URL: https://vega.github.io/vega/ [10.06.2020]

[VI] Documentation — *Vispy*. Vispy.Org, 2020
URL: http://vispy.org/documentation.html. [15.05.2020]

Anhang

Jupyter Notebook Dateien	
GradesOverviewStatisticsHeatmap.ipynb	beinhaltet die Implementierung von GradesOverviewStatisticsHeatmap, Seite 36
GradesLabelChart.ipynb	beinhaltet die Implementierung von *GradesLabelChart,* Seite 38-39
GradesOverallChart.ipynb	beinhaltet die Implementierung von GradesOverallChart, Seite 37-38
GradesStatisticsTimeBoxplot.ipynb	beinhaltet die Implementierung von GradesStatisticsTimeBoxplot, Seite 41
GradesTimeChart.ipynb	beinhaltet die Implementierung von GradesTimeChart, Seite 40
PlotComparison.ipynb	beinhaltet die Implementierungen von Beispieldiagrammen aus dem Kapitel 2.2.1 und Kapitel 2.3
utils.ipynb	beinhaltet die Hilfsfunktionen, Seiten 45-46