

# Analyse und Auswertung von Echtzeit-Fahrplänen der Deutschen Bahn (Project D-Railing)

## STUDIENARBEIT

für die Prüfung zum

Bachelor of Engineering

des Studienganges Informationstechnik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

**Alexander Bierenstiel, André Schmitt, Dominik Schmitt**

Abgabedatum 14. Mai 2018

Bearbeitungszeitraum

900 Stunden

Matrikelnummer

2496963, 3272367, 7191584

Kurs

TINF15B3

Ausbildungsfirma

Sick AG, E.G.O. Gerätebau, netcup GmbH  
Waldkirch, Oberderdingen, Karlsruhe

Gutachter der Studienakademie

Prof. Dr. Jürgen Vollmer

## Erklärung

Wir versichern hiermit, dass wir unsere Studienarbeit mit dem Thema: „Analyse und Auswertung von Echtzeit-Fahrplänen der Deutschen Bahn“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

---

Ort      Datum

Unterschrift

## Zusammenfassung

### Nochmal lesen und eventuelle Schreibfehler beheben

Diese Studienarbeit befasst sich mit den Alltagssituationen vieler Reisender. Viele Pendler sind täglich auf pünktliche öffentliche Verkehrsmittel angewiesen. Die Idee für die Studienarbeit stammt aus dem Problem, dass ein Pendler vor seiner Abfahrt wenige Informationen über die eventuelle Verspätung erhält. Ein weiterer Punkt ist das fehlen einer Historie der Zugverbindung, auf welcher der Pendler sehen kann, ob der Zug in letzter Zeit häufiger zu spät kam. Als die Deutsche Bahn im Spätjahr 2017 die Live Fahrplan API für die Öffentlichkeit freigab, war die Idee, die Daten sinnvoll zu sammeln und auszuwerten. Ein Ziel war die statistische Analyse der Daten auf verschiedene Faktoren und Einflüsse. Als optionale Problemstellung sollte anhand der Vergangenheit die Verspätung in der Zukunft mittels neuronaler Netze prognostiziert werden. Ein erwartetes Ergebnis war eine große Menge an Datensätzen und deren anschließende Auswertung. Darüber hinaus sollen die Ergebnisse der Öffentlichkeit auf einer Website präsentiert werden.

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>8</b>
1.1	Motivation . . . . .	8
1.2	Ziele und Ablauf der Studienarbeit . . . . .	9
1.3	Stand der Technik . . . . .	11
1.4	Planung . . . . .	12
1.4.1	Zeitliche Einteilung der Studienarbeit . . . . .	12
1.4.2	Versionsverwaltung . . . . .	12
<b>2</b>	<b>Grundlagen</b>	<b>14</b>
2.1	Data Mining . . . . .	14
2.2	Datenmodell . . . . .	16
2.3	Modellierung realer Größen . . . . .	16
<b>3</b>	<b>Datenbeschaffung</b>	<b>17</b>
3.1	Die DB Timetable API . . . . .	17
3.1.1	Der Station Endpunkt . . . . .	17
3.1.2	Plan . . . . .	18
3.1.3	fchg . . . . .	18
3.1.4	rchg . . . . .	19
3.1.5	Gespeicherte Werte in der Datenbank . . . . .	20
3.2	Programmierung des Data Miners . . . . .	21
3.3	Weatherminer . . . . .	25
3.3.1	OpenWeatherMap . . . . .	27
3.4	Datenbank und Schema . . . . .	28
3.4.1	Datenbank Schema des Wetterminers . . . . .	30
3.5	Backup der Datenbank . . . . .	31
<b>4</b>	<b>Datenverarbeitung mit Data Mining</b>	<b>34</b>
4.1	Grundlagen von Data Science und KDD . . . . .	34
4.1.1	Data Mining . . . . .	34
4.1.2	Knowledge Discovery in Databases (KDD) . . . . .	34
4.2	Grundlagen . . . . .	34
4.3	Vorverarbeitung der Daten . . . . .	35
4.4	Software-Architektur der Datenauswertung . . . . .	37

4.4.1	Query Suite . . . . .	38
4.4.2	Processing Utils . . . . .	40
4.5	Statistische Auswertungen . . . . .	41
4.5.1	Definition der Verspätungen . . . . .	41
4.5.2	Analyse der Verspätungen eines Zuges . . . . .	45
4.5.3	Durchschnitt der Verspätungen einer Haltestelle . . . . .	48
4.5.4	Durchschnitt der Zeiten einer Strecke . . . . .	50
<b>5</b>	<b>Datenverarbeitung mit neuronalem Netz</b>	<b>52</b>
5.1	Programmierung der automatischen Datenverarbeitung . . . . .	52
5.2	Vorverarbeitung der Datensätze . . . . .	53
5.3	Einrichten der Tensorflow Umgebung . . . . .	54
5.4	Begriffsdefinitionen für ein neuronales Netz . . . . .	58
5.5	Eingabe der Datensätze in Tensorflow . . . . .	60
5.6	Eingabe- und Ausgabe-Parameter für das Neuronale Netz . . . . .	62
5.7	Anlernen des Netzes . . . . .	62
5.8	Verifizieren des Netzes . . . . .	63
5.9	Vorhersagen anhand des Netzes . . . . .	65
5.10	Bewertung der Ergebnisse . . . . .	65
<b>6</b>	<b>Bereitstellung der Daten im Internet</b>	<b>69</b>
6.1	Aufbau der Website . . . . .	69
6.2	Erstellung der Webrouten . . . . .	71
6.3	Erstellung der Seiten . . . . .	71
6.3.1	Idee des dynamischen Nachladen . . . . .	72
6.3.2	Die Stationsübersicht . . . . .	73
6.3.3	Die Zugübersicht . . . . .	74
6.4	Testen der Seiten mit Unit Test . . . . .	74
6.5	Visualisierung der Datensätze . . . . .	76
6.6	Darstellung der Datensätze . . . . .	80
<b>7</b>	<b>Schlussfolgerung</b>	<b>82</b>
7.1	Rückblick . . . . .	82
7.2	Fazit . . . . .	82
7.3	Ausblick . . . . .	83
	<b>Anhang</b>	<b>85</b>
	<b>Literaturverzeichnis</b>	<b>85</b>
	<b>Liste der ToDo's</b>	<b>86</b>

# Abbildungsverzeichnis

1.1	Ablauf der Implementierung . . . . .	10
3.1	Grundablauf des Miners . . . . .	22
3.2	Ablauf des Miners . . . . .	24
3.3	Übersicht der Postleitregionen von Deutschland. Erstellt von Stefan Kühn, vom Wikimedia Commons . . . . .	26
3.4	Datenbank Schema in der ersten Version . . . . .	29
3.5	Datenbank Schema in der zweiten Version . . . . .	30
4.1	Übersicht des KDD-Prozesses [USAMA FAYYAD 1996, S. 41]. . . . .	35
4.2	Grundablauf des Data Minings . . . . .	38
4.3	Verspätungsanalyse von RE4725 von Karlsruhe nach Konstanz . . . . .	46
4.4	Verspätungsanalyse von ICE74 von Basel Bad. Bf. nach Kiel mit Verspätung bei Abfahrt . . . . .	47
4.5	Verspätungsanalyse von ICE74 von Basel Bad. Bf. nach Kiel ohne Verspätung bei Abfahrt . . . . .	48
4.6	Prozentuale Anteile der Durchschnittlichen Verspätung bei ankommenden Zügen . . . . .	50
5.1	Datenfluss aus der Datenbank in das Tensorflow Modell . . . . .	55
5.2	Der Verzeichnisbaum mit der Aufteilung der Datensätze . . . . .	57
5.3	Ausschnitt aus einem Vocabfile, hier zu sehen Gleis (Soll) . . . . .	58
5.4	Verlauf der Loss Funktion auf der Website des Tensorflow Playground . . . . .	64
5.5	Verteilung der Wahrscheinlichkeiten bei 288 Klassen und allen Spalten als Eingabeparameter . . . . .	66
5.6	Verteilung der Wahrscheinlichkeiten bei 288 Klassen mit nur der Ankunftszeit als Eingabeparameter . . . . .	67
6.1	Struktur der einzelnen Views der Website . . . . .	70
6.2	Mockup der Stationsübersicht . . . . .	72
6.3	Dynamisches Nachladen der Subseiten . . . . .	73
6.4	Ausgabe der Website im Debug Modus im Fehlerfall . . . . .	75
6.5	Ausgabe der Website im Production Modus im Fehlerfall . . . . .	75
6.6	Ausgabe eines Unit Tests (oberer Teil) . . . . .	75
6.7	Ausgabe eines Unit Tests (unterer Teil) . . . . .	76

6.8	Vergleich von statischer URL zur Laravel Routen Engine . . . . .	77
6.9	Gleisbelegung am Beispiel Wiesloch-Walldorf . . . . .	78
6.10	Gleisbelegung in Berlin Ostbahnhof (ausgewählte Zugtypen S-Bahn und NightJet) . . . . .	78
6.11	Ausgegebenes JSON an den Webbrowser bei Aufruf der Route . .	79
6.12	Dreidimensionales Histogramm aus dem TensorBoard . . . . .	80

# Tabellenverzeichnis

1.1	Aufgaben über die Zwei Semester . . . . .	12
3.1	Vergleich des Funktionsumfangs der beiden Miner . . . . .	25
3.2	Tabelle mit allen Wetterverhältnisse . . . . .	28
4.1	Struktur der Durchschnitts Tabelle . . . . .	51
5.1	Vorverarbeitung der Datenbank-Daten . . . . .	56
5.2	Vergleich der Lerndauer in Bezug auf die Anzahl der Klassen . . .	63
6.1	Vergleich der Ladezeit zweier Webseiten (Angaben in mm:ss) . . .	73



# Liste der Quellcodeausschnitte

3.1	Ausschnitt der Datenabfrage . . . . .	23
3.2	Skript zum Aufteilen eines großen MySQL Dumps in mehrere kleine Dumps, Teil 1 . . . . .	31
3.3	Befehl zum Einfügen der SQL Dateien . . . . .	32
3.4	Skript zum Aufteilen eines großen MySQL Dumps in mehrere kleine Dumps, Teil 2 . . . . .	33
4.1	Some Python File . . . . .	36
4.2	Zerlegen der Zug ID in seine Komponenten . . . . .	37
4.3	Beispiel einer Query-Methode . . . . .	40
4.4	Berechnung der SARV . . . . .	44
4.5	SQL Query für neue Halte einer Haltestelle . . . . .	50
5.1	Ausschnitt aus der Datei generate_csv.py . . . . .	54
5.2	Ausschnitt von der Input Funktion aus der Datei train_test_predict.py	61
6.1	Beispiel einer Webroutendefinition . . . . .	71
6.2	Funktion zur Auswertung der Gleisbelegung eines Bahnhofs . . . .	79

# Abkürzungsverzeichnis

<b>HTTP</b>	Hypertext Transfer Protocol .....	17
<b>API</b>	Application Programming Interface .....	11
<b>PLZ</b>	Postleitzahl .....	25
<b>PLR</b>	Postleitregion .....	25
<b>SARV</b>	Streckenabschnitt-respektive Verzögerung	
<b>KDD</b>	Knowledge Discovery in Databases .....	1
<b>GPU</b>	Graphics Processing Unit .....	57

# Kapitel 1

## Einleitung

### TODO

Ratschlage von meinem Ausbilder: Roter Faden ausbauen

Einleitung überarbeiten: Den Scope einengen: Wir haben Grundsteine gelegt auf dem Weg zur Prognose Implementierungsablauf vorstellen, wie sind wir vorgegangen und weshalb? (Diagram) Datenaquirierung: Wo bekommen wir sie her? Dateneingrenzung: Welche Daten verwenden wir? Datenspeicherung: Wie speichern wir sie? etc. Motivation starker ausformulieren: Praxisbezug zum Leser herstellen

Website mit Visualisierungen besser darstellen + Screenshots! Ansätze des Neuronalen Netzes beschreiben + Probleme schildern, die sich dabei ergeben haben

Ausblick: Prognose im Ausblick beschreiben und wie die bisherige Arbeit für die Prognosenbildung hilft

### TODO

## 1.1 Motivation

### gut

Im Rahmen der Open Data Bewegung veröffentlicht auch die Deutschen Bahn<sup>1</sup> die Echtzeitdaten der Fahrpläne ihrer Personenzüge. Aus diesem Umstand heraus, hat sich der Wunsch aufgetan, die Daten der Deutschen Bahn für eine Studienarbeit zu verwenden. Um die Daten der Deutschen Bahn in einer Studienarbeit zu verwenden, benötigt es noch eine Idee, was aus den Daten gewonnen werden kann. Hierbei hat sich folgende Idee entwickelt:

Verspätungen im öffentlichen Nah- und Fernverkehr treten täglich auf. Oftmals wartet der Fahrgast mehrere Minuten, bis sein Zug endlich einfährt. Besonders früh morgens oder im Winter wünscht sich der Fahrgast, dass er die Wartezeit zu

---

<sup>1</sup>Siehe <https://developer.deutschebahn.com/store/apis/info?name=Timetables&version=v1&provider=DBOpenData>

Hause hätte verbringen können, anstatt im Bahnhof auf die Ankunft des Zuges zu warten. Die Idee ist es hierbei, mit den Echtzeitdaten der Deutschen Bahn Vorhersagen abzuleiten, die dem Fahrgast mitteilen, wie groß die Verspätung des Zuges ist, auf den er wartet. Mit diesem Wissen kann es sich der Fahrgast mit einem Kaffee daheim nochmals gemütlich machen, bevor er zum Bahnhof aufbricht. Genauso gut kann der Fahrgast die Prognosen nutzen, um abschätzen zu können, ob er Verbindungszüge erreicht oder gegebenenfalls mehr Zeit einplanen muss.

Da zur Prognosenbildung die Ursachen der Verspätung häufig nicht direkt erkennbar sind, soll mit dieser Studienarbeit Grundsteine für die Vorhersage von Verspätungen gelegt werden. Da die Prognose von Verspätungen als komplexe Aufgabe eingeschätzt wird, beschränkt sich diese Studienarbeit auf die Analyse der Daten. Die Erkenntnisse aus der Analyse der Daten sollen die Grundsteine für die Entwicklung eines Prognose-Modells bilden. Die Prognosenbildung für Verspätungen wird somit als optionales Ziel definiert.

## 1.2 Ziele und Ablauf der Studienarbeit

gut

In diesem Abschnitt werden die vorgesehenen und die optionalen Ziele der Studienarbeit formuliert und dem Leser aufgezeigt in welche Schritte sich die Studienarbeit aufteilt.

**Vorgesehene Ziele** Die Ziele der Studienarbeit sind es, Verspätungen von Zügen festzustellen und anschließend zu analysieren. Hierbei sollen unterschiedliche Aspekte bei der Analyse betrachtet werden: Es sollen die Verspätungen nach Abhängigkeit der Zeit, Ort und Strecke, sowie kritische Punkte analysiert werden. Dabei sollen die Analysen auch visualisiert werden, um Schlüsse aus ihnen ziehen zu können.

**Optionale Ziele** Da die Prognosenbildung für Verspätungen als komplexe Aufgabe eingeschätzt wird, ist die Vorhersage der Verspätung eines Zuges als optionales Ziel eingestuft. Für die Prognosenbildung sollen neben den Erkenntnissen aus der Analyse der Daten zusätzlich auch Wetterdaten einbezogen werden. Es wird vermutet, dass das Wetter besonders bei Stürmen, Kälte oder Hitze einen Einfluss auf die Pünktlichkeit von Zügen hat. Aus diesem Grund werden bei dem Wetter die Windgeschwindigkeit, die Art des Niederschlags, die Menge des Niederschlags und die Höhenlage der Bahnhöfe als zu nutzende Wetterdaten in Betracht gezogen.

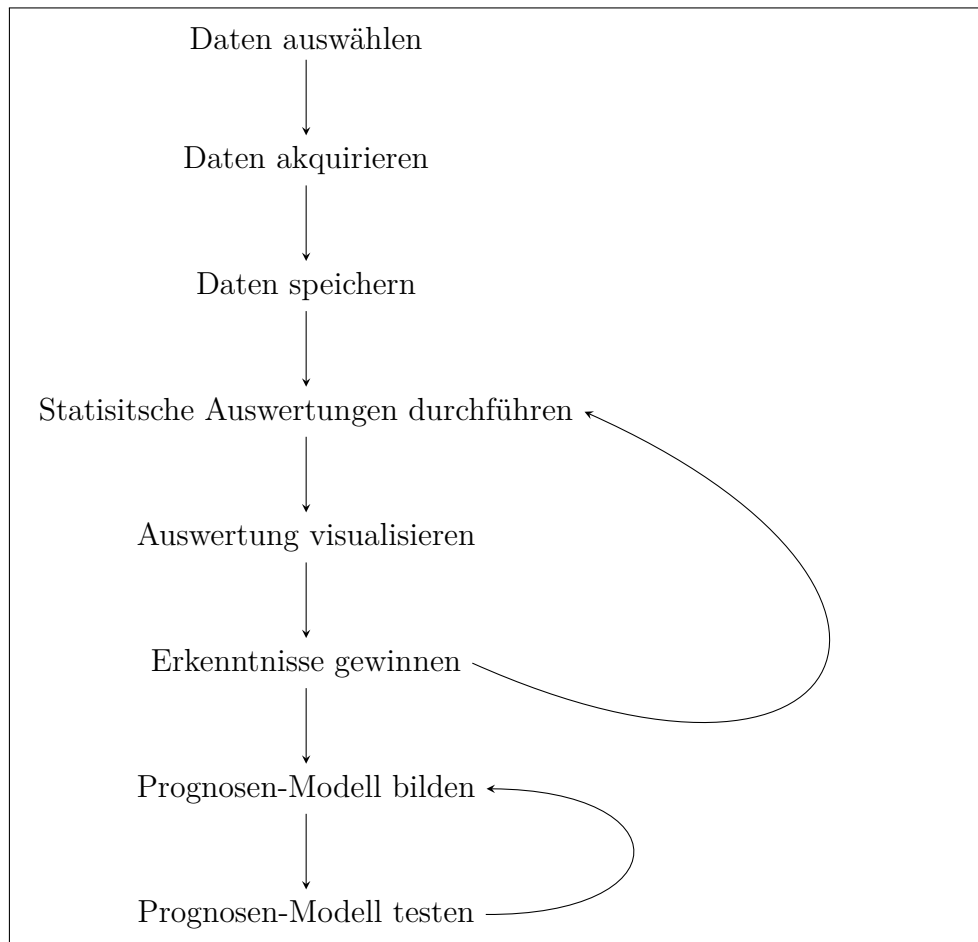


Abbildung 1.1: Ablauf der Implementierung

**Struktur der Studienarbeit** Die Studienarbeit teilt sich in einzelne Schritte auf. In Abbildung 1.1 werden die einzelnen Schritte visualisiert. Hierbei führt jeder Schritt auf das Ziel zu, Prognosen für die Verspätung von Zügen vorhersagen zu können. Der erste Schritt, der in der Studienarbeit durchzuführen ist, ist die Beschaffung der Daten über die API der Deutschen Bahn. Dieser Schritt unterteilt sich in drei Teilschritte. Der erste Teilschritt ist, diejenigen Daten auszuwählen, die nützlich für die Auswertung und Prognose sein können und diejenigen Daten zu ignorieren, die für die Analyse und Prognose unnütz erschienen. Dieser Schritt wird in der Abbildung “Daten auswählen” genannt. Danach folgt der Schritt “Daten akquirieren” bei dem die Daten von der API abgerufen und im nächsten Schritt “Daten speichern” in der Datenbank gespeichert werden. Diese genannten drei Teilschritte werden in dem Kapitel *3 Datenbeschaffung* behandelt.

Der nächste Schritt ist das statistische Auswerten der Daten, um anschließend das Ergebnis der Auswertung zu visualisieren. Mithilfe der Visualisierung soll die Interpretation der Auswertung leichter fallen, um anschließend Erkenntnisse über die analysierten Daten zu gewinnen. Diese Schritte werden in Kapitel *4 Daten-*

*verarbeitung mit Data Mining* durchgeführt. Hierbei können die drei Schritte als Schleife durchlaufen werden, um mit den bisherigen Kenntnissen weitere Hypothesen aufstellen zu können und durch weitere Analysen zu bestätigen oder zu widerlegen.

Mit den erlangten Kenntnissen aus den Auswertungen kann anschließend ein Prognosen-Modell entworfen werden. Dieses Modell muss anschließend auf seine Qualität getestet werden. Diese zwei Schritte können ebenfalls als Schleife durchlaufen werden, um das Prognosen-Modell solange anzupassen, bis es die gewünschte Qualität erreicht.

**Neuronales Netz für Prognosenbildung** In dieser Studienarbeit wird parallel zu der Analyse der Daten versucht, ein neuronales Netz für die Vorhersage der Verspätung eines Zuges zu trainieren. Das neuronale Netz wird unter der Annahme entwickelt, dass es Zusammenhänge selbst aus den Trainingsdaten erlernt. Unter dieser Annahme ist die Entwicklung des Netzes unabhängig von den zu erlangenden Erkenntnissen aus den Datenanalysen, sodass das Netz parallel zu den Datenanalysen entwickelt werden kann. Die Entwicklung des neuronalen Netzes wird in Kapitel 5 *Datenverarbeitung mit neuronalem Netz* beschrieben.

## 1.3 Stand der Technik

gut

Derzeit ist der Begriff *Maschinelles Lernen* ein wichtiger Punkt im Fortschritt von Software. In dieser Studienarbeit sollen verschiedene Disziplinen maschinellen Lernens, über Data Mining und Visualisierungstechniken, bis hin zur Bereitstellung der Ergebnisse behandelt werden. Es ist wichtig vor Beginn der Arbeit die Gebiete voneinander abzugrenzen, um die Bearbeitung in kleineren Schritten durchzuführen. Eine gewisse Reihenfolge muss dabei beachtet werden, weshalb im ersten Abschnitt der Studienarbeit auf die Grundlagen eingegangen wird. Zuerst muss der Begriff der Datenbeschaffung und des Data Minings geklärt werden.

„Data mining refers to extracting or „mining“ know ledge from large amounts of data“

[JIAWEI HAN 2000]

Erst nach der Beschaffung können die Daten in Zusammenhang gebracht werden. Die sinnvolle Visualisierung der Datensätze ist sehr wichtig, um eventuelle Zusammenhänge besser erkennen zu können. Die Visualisierung wird in späteren Kapiteln genauer betrachtet. Eine wichtige Änderung in den letzten Jahren war die Entscheidung einiger Großunternehmen, Daten über Application Programming Interface (API) Schnittstellen für Entwickler verfügbar und nutzbar zu machen. Dies erleichtert vor allem den Schritt der Datenbeschaffung für diese Studienarbeit.

## 1.4 Planung

gut/naja

Die grobe Planung der Studienarbeit sieht eine Aufteilung in die beiden Semester vor. In Abbildung 4.2 wird diese Aufteilung anhand einer Tabelle dargestellt.

### 1.4.1 Zeitliche Einteilung der Studienarbeit

Im ersten Teil der Studienarbeit steht die Erfassung der Daten der Deutschen Bahn im Vordergrund. Neben der Programmierung des Data Miners für die Bahn API wird eine Literaturrecherche, welche für die anschließende Aufbereitung und Visualisierung der Datensätze benötigt wird durchgeführt. Des weiteren wird weiterführende Literatur gelesen und auf nutzbare Inhalte für die Studienarbeit untersucht. Da die Erfassung der Daten einige Zeit in Anspruch nimmt, und die zeitlichen Freiräume im fünften Semester fehlen, wird die Hauptarbeit in das sechste Semester verlegt. Im sechsten Semester soll die Analyse der Daten, sowie deren Auswertung erfolgen. Parallel dazu soll die Prognose anhand eines neuronalen Netzes realisiert werden. Anschließend sollen die Auswertungen und Statistiken auf einer Webseite zugänglich gemacht werden.

5. Semester	6. Semester
Data Miner Literatur Recherche	Analyse der Daten Statistische Analyse Prognose Visualisierung

Tabelle 1.1: Aufgaben über die Zwei Semester

### 1.4.2 Versionsverwaltung

Wichtig für die Planung ist neben der Zeiteinteilung auch die Versionsverwaltung des Quellcodes und der Ausarbeitung selbst. Da ständig eine aktuelle Version beider Arbeiten zu Verfügung stehen muss und alle Autoren parallel schreiben und editieren können müssen, fiel die Wahl der Gruppe auf den Dienst Github<sup>2</sup> für den Quellcode und ShareLaTeX<sup>3</sup> für die Ausarbeitung. In Github wird eine öffentliche Organisation angelegt, welcher alle Autoren beitreten. Innerhalb der Organisation werden die Repositories zur Verwaltung von Website, Data Miner, Visualisierungstoolkit und Dokumentation angelegt. Alle Teilnehmer bekommen Zugriff auf den gesamten Quellcode. Damit ist gleichzeitig ein Backup und der aktuelle Wissensaustausch zwischen den Gruppenmitgliedern sichergestellt. Die

<sup>2</sup>Siehe: <https://github.com/>

<sup>3</sup>Siehe: <https://www.sharelatex.com/>

gemeinsame Arbeit an Quellcode wird durch die Versionsverwaltung erleichtert, da parallel an verschiedenen Stellen des selben Quellcodes gearbeitet werden kann.



# Kapitel 2

## Grundlagen

### WOHIN DAMIT

Dieses gesamte Kapitel aufsplitten und in die Grundlagen der einzelnen Kapitel einordnen

Ist halt eigentlich das Thema wo man am meisten hätte schreiben können.

## 2.1 Data Mining

### Data Mining Einführung und dessen Bedeutung für das Projekt

Das Data Mining ist ein essentieller Bestandteil des Projektes. Ohne genügend Daten als Grundlage kann dieses Projekt nicht funktionieren, da ein neuronales Netz nur mithilfe von ausreichenden Daten trainiert werden kann. Hierbei gilt als Grundsatz: Je mehr Daten zur Verfügung steht, desto genauer das neuronale Netz am Ende. Um die weitere Automatisierung des Datenflusses zu ermöglichen, werden die Datensätze in einem offenen und weiterverwendbaren Format gespeichert. In frühen Phasen der Arbeit sollte darauf Wert gelegt werden, dass keine Daten als unwichtig betrachtet werden. Sollte im späteren Verlauf der Arbeit weitere Funktionen hinzugefügt werden, können solche Daten schnell wichtig werden. Deshalb ist es sinnvoll nichts wegzuerwerfen was später noch nützlich werden könnte. Die Rohdaten können allerdings selten direkt gespeichert werden, da dies oft zu großen Datenmengen führen kann. Es ist bei der Aufarbeitung allerdings zu beachten, dass die Daten nicht angepasst oder verfälscht werden.

Datenformat und Aufbau erklären. Wieso sollte im ersten Schritt beim Mining nicht direkt alles angepasst werden? Wieso müssen die Daten aufbereitet werden? Stichwort: FehlerAPI, Fehlende Datensätze, Bucketlist, Konvertierung

Dinge die wir brauchen:

- Bahnhofsnnummer

- Linie als Folge von angefahrenen Bahnhöfen (z.B. ICE 690, EC 378, R856)
- Zugreferenz (gleicher Zug auf Linie?)
- Ankunftszeit geplant
- Ankunftszeit real
- Abfahrtszeit geplant
- Abfahrtszeit real
- Historic Delay Element?  
Angeblich kann man damit die vorherigen Verspätungen auf der Linie auslesen
- Wetter je PLZ[Postleitregionen] (Wind, Niederschlag, Temperatur)
- Die Bahnhof Tabelle mit PLZ ergänzen, um Wetterdaten zuordnen zu können (Postleitregionen)

Mögliche Auswertungen:

- Relative Verspätung pro Streckenabschnitt  
Pro Streckenabschnitt kann ein Zug Verspätung aufbauen oder abbauen. Jedem Streckenabschnitt wird die Summe aller Verspätungen, die die Züge auf diesem Streckenabschnitt aufbauen oder abbauen zugeordnet. Diese Summe aller relativen Verspätungen pro Streckenabschnitt wird anschließend visualisiert.
- Verzögerung im Bahnhof  
Pro Bahnhof kann die Verspätung eines Zuges zunehmen oder abnehmen. Pro Bahnhof werden von allen Zügen die Verspätungen, die sie in dem jeweiligen Bahnhof aufbauen oder abbauen, aufsummiert. Anschließend wird für jeden Bahnhof die gebildete Summe visualisiert.
- Welche Wetterlagen bringen Verspätungen

Mögliche Arten der Visualisierung

- Welche Strecken bringen die meiste Verspätung? Heatmap? Top10?
- Welche Bahnhöfe haben die größte Verzögerung? Heatmap? Top 10? Diagramm?

Auswahl der Wetterstationen: Die Wetterstationen werden pro Postleitregion so gewählt, sodass diese möglichst im Zentrum der jeweiligen Region liegen.

## 2.2 Datenmodell

Datenmodell erläutern, welche Rohdaten aus der DB-API

Ein Datenmodell ist sowohl erforderlich, um Datenobjekte bezüglich ihrer Bedeutung zu interpretieren, als auch, um Beziehungen zwischen Datenobjekten festzustellen oder zu beschreiben. Im Rahmen dieser Arbeit gilt es, ein Datenmodell zu definieren, das unterschiedliche Aufgaben erfüllen soll:

**Modellierung realer Größen** Zu aller erst definiert das Datenmodell die Modellierung von Größen der realen Welt, die später für die folgende Datenverarbeitung benötigt werden. Hierbei werden mathematische Definitionen entwickelt, die die Bedeutung der jeweiligen Größe, wie zum Beispiel Verspätung, beschreibt.

**Modellierung der Rohdaten** Anschließend definiert das Datenmodell, wie die beschriebenen Größen der realen Welt in den Rohdaten abstrahiert und abgebildet werden. Dies ist wichtig, um die Rohdaten, wie sie beispielsweise von der Timetable-API der Deutschen Bahn geliefert werden, interpretieren und weiterverarbeiten zu können. Insbesondere muss die Modellierung die Beziehungen unter den Datenobjekten der Rohdaten definieren, um aus diesen wieder die realen Größen ableiten zu können.

**Modellierung der Auswertung** Nachdem die Bedeutung von realen Größen und deren Abbildung in den Rohdaten definiert ist, muss die Auswertung der Daten konzipiert und modelliert werden. Hierzu zählen sowohl die Beschreibung der internen Darstellung der Daten zum Zwecke der weiteren Auswertung, als auch die Beschreibung des auswertenden Algorithmus. Zu den internen Darstellungen können Datenstrukturen in Programmen oder Datenbank-Schemata gezählt werden.

Um die Gliederung der Arbeit übersichtlich zu halten, sind die Modellierungen der oben genannten Punkte in separaten Kapiteln dargestellt.

## 2.3 Modellierung realer Größen

Schauen, ob Kapitel noch Sinn macht

n

icht wirklich wenn nix mehr drinne steht

In diesem Abschnitt werden die realen Größen, die zur Datenauswertung benötigt werden, modelliert. Eine der wichtigsten realen Größen in dieser Arbeit ist die Verspätung oder Verzögerung von Zügen. Im folgenden werden die verschiedenen Arten von Verzögerungen dargestellt.

# Kapitel 3

## Datenbeschaffung

### 3.1 Die DB Timetable API

naja

Die DB Timetable API seit dem 18.04.2017 öffentlich verfügbar. Um sie nutzen zu können ist es notwendig, sich einen Account auf der Developer Seite der Deutschen Bahn<sup>1</sup> anzulegen. Mit diesem Account ist es möglich bis zu 20 Abfragen pro Minute an die API zu stellen. Folgende Endpunkte können in der Deutschen Bahn API<sup>2</sup> abgefragt werden. Die Dokumentation der API kann in der Swagger-Specification abgerufen werden.<sup>3</sup>

#### 3.1.1 Der Station Endpunkt

Dieser Endpunkt gibt Informationen über Bahnhöfe zurück. Dafür kann sowohl der Name der Station, als auch die eindeutige EVA Nummer oder die ds100 beziehungsweise rl100 Nummer zur Identifikation angegeben werden. Der Kleene-Stern kann verwendet werden, um alle Stationen abzurufen. War der Aufruf erfolgreich, so gibt die API den Hypertext Transfer Protocol (HTTP) Status **200** zurück. Neben dem Status Code wird auch ein Ergebnis mit den angefragten Stationen zurückgegeben. Innerhalb eines Stations-Objekt, werden die verschiedenen Identifikationsmöglichkeiten angegeben. Darunter auch die von der Timetable API genutzte EVA-Nummer. Mit ihr kann jede Bahnstation in Deutschland eindeutig identifiziert werden.

Des Weiteren werden teilweise die Anordnung der Bahnsteige der Haltestellen angegeben. Der Meta-Eintrag gibt weitere EVA-Nummern an, die mit diesem Bahnhof zusammenhängen (Subbahnhof). Konnte der gesendete String nicht iden-

---

<sup>1</sup>Siehe: <https://developer.deutschebahn.com>

<sup>2</sup> Siehe API-URL: <http://api.deutschebahn.com/timetables/v1>

<sup>3</sup>Die Dokumentatation ist unter der der URL <https://developer.deutschebahn.com/store/apis/info?name=Timetables&version=v1&provider=DBOpenData> im Reiter Dokumentation zu finden.

tifiziert werden, so wird ein leeres Objekt zurückgegeben.

Beispiel:

Request:

```
https://api.deutschebahn.com/timetables/v1
/station/Heidelberg HBF
```

Response:

```
<stations>
  <station p="4|5" meta="518168|8070043"
    name='Heidelberg Hbf' eva="8000156" ds100="RH"/>
</stations>
```

### 3.1.2 Plan

Durch Angabe der EVA Nummer (String), eines Datums und einer Stunde, können planmäßige Abfahrten an dem gewählten Bahnhof innerhalb der angegebenen Stunde abgefragt werden. Dabei ist das Datum als String im „YYMMDD“ Format anzugeben. Die Stunde ist ebenfalls als String anzugeben, diese soll im „HH“ Format angegeben werden.

```
/timetable/plan/{evaNo}/{date}{hour}:
  evaNo: Angabe des Bahnhofs
  date: Angabe des gesuchten Datums (YYMMDD)
  hour: gesuchte Stunde (HH)
```

Gibt ein Timetable-Objekt zurück, in dem alle geplanten Abfahrten in der angegebenen Stunde enthält. Dabei werden keine Änderungen durch Verspätungen berücksichtigt.

Responses:

```
200 Successfull operation
```

Gibt ein Timetable-Objekt zurück. In ihm ist der Stationsname und die EVA-Nummer der Station gekapselt. Außerdem enthält es Listen von Timetable-Stop und Message-Objekten. In einer Plan-Response werden keine Messages übertragen. Es werden nur die „planned“ Attribute genutzt. Darunter ist die Geplante Ankunfts- bzw. Abfahrtszeit, das Gleis an dem der Zug ankommen soll, sowie eine Liste von den nächsten Haltestellen auf der Strecke.

### 3.1.3 fchg

Der „fchg“ Endpunkt nimmt eine EVA-Nummer (String) entgegen und gibt ein Timetable-Objekt zurück. Darin werden alle Änderungen vom Zeitpunkt der Anfrage an gespeichert.

```
/timetable/fchg/{evaNo}:  
    evaNo: Angabe des Bahnhofs
```

Innerhalb des Timetable wird der Name der Station, die EVA Nummer, eine Liste von Timetable-Stops und Messages zurückgegeben, bei denen sich Änderungen ergeben haben.

### 3.1.4 rchg

Durch Angabe einer EVA-Nummer können alle Änderungen der letzten zwei Minuten zurückgegeben werden. Alle 30 Sekunden werden diese aktualisiert.

```
/timetable/rchg/{evaNo}:  
    evaNo: Angabe des Bahnhofs
```

Der rchg Endpunkt unterscheidet sich ausschließlich durch die übertragenen Änderungen von dem fchg Endpunkt, die im Fall des rchg in der Vergangenheit liegen.

**Timetablestop** In einem Timetablestop werden eine ID aus einer Daily-Trip-ID, Abfahrtsdatum des Zuges am Beginn der Linie und der Nummer des Stops gespeichert. Außerdem die aktuelle EVA-Nummer, die Bezeichnung der Strecke, eine Referenz zum eigentlichen Zug - falls es sich um einen Ersatzzug handelt und die Events *Ankunft* und *Abfahrt*, in denen vor allem die An- bzw. Abfahrtszeiten und das Gleis untergebracht sind. Es kann jeweils die geplante, als auch die prognostizierte Information enthalten sein, eine Message, weshalb eine Änderung stattfand, sowie die Information wie viel Verspätung die Bahn hat und ob sie auf ein anderes Gleis umgeleitet wurde.

**Message** Eine Message besteht aus einer Message-Id, einem Message-Typ und einem Timestamp. Folgende Informationen können zusätzlich angehängt werden:

- Information auf welche Uhrzeit der Zug verlegt wurde, aber auch wann der Zug eigentlich geplant war
- Code um die Message zu identifizieren
- Text der Nachricht
- Kategorie der Nachricht
- Priorität der Nachricht
- Eigentümer der Nachricht
- externer Link
- Indikator ob Nachricht gelöscht

- Nachricht des Verteilers
- Name des Zuges

### 3.1.5 Gespeicherte Werte in der Datenbank

**id** ID als Primärschlüssel zur Speicherung in der Datenbank.

**zugid** Beispiel: **-7714364757423921343-1712081222-8**

Zug-ID die von der Timetable-API zur Zugidentifizierung genutzt wird.

**zugverkehrstyp** Beispiel: **F**

Verkehrstyp des Zuges. Oft verwendet werden Nahverkehr und Fernverkehr.

**zugtyp** Beispiel: **p**

Typ des Zuges: Personen / Frachtverkehr.

**zugowner** Beispiel: **80**

Besitzer des Zuges.

**zugklasse** Beispiel: **ICE**

Kurze Bezeichnung der Zug Klasse.

**zugnummer** Beispiel: **788**

Nummer des Zuges.

**zugnummerfull** Beispiel: **ICE788**

Kombination aus Zugklasse und Zugnummer. Diese wird auch im alltäglichen Leben verwendet, um Züge zu identifizieren.

**linie** Beispiel: **2**

Linie die der Zug befährt sofern sie eine Bezeichnung besitzt

**evanr** Beispiel: **8000152**

Eindeutige ID des Bahnhofes an dem der Zug gehalten hat.

**arzeitsoll** Beispiel: **16:32:00**

Uhrzeit, an dem der Zug an der Haltestelle ankommen soll.

**arzeitist** Beispiel: **16:33:00**

Uhrzeit, an dem der Zug an der Haltestelle angekommen ist.

**dpzeitsoll** Beispiel: **16:36:00**

Uhrzeit, an dem der Zug an der Haltestelle abfahren soll.

**dpzeitist** Beispiel: **16:38:00**

Uhrzeit, an dem der Zug an der Haltestelle abgefahren ist.

**gleissoll** Beispiel: **7**

Gleis auf dem der Zug ankommen soll.

**gleisist** Beispiel: **7**

Gleis auf dem der Zug angekommen ist.

**datum** Beispiel: **2017-12-08**

Datum, an dem der Zug an der Haltestelle angekommen ist.

**streckengeplanthash** Beispiel: **4d0bc383**

Strecke die der Zug Abfahren soll, sofern keine Änderungen der Strecke vorliegen.

**streckenchangedhash** Beispiel: **bd84c25a**

Strecke die der Zug in Verlauf seiner Fahrt wirklich abgefahren ist.

**zugstatus** Beispiel: **n**

Der Zugstatus kann drei Werte annehmen. Das Zeichen n steht für normal, das Zeichen c steht für cancelled und das Zeichen p steht für planned

## 3.2 Programmierung des Data Miners

gut/naja

Der Data Miner wird im Laufe der Studienarbeit immer weiter entwickelt und stetig verbessert. Die erste Version zeigte nach nur wenigen Wochen erhebliche Schwachstellen im Quellcode und der Datenbank auf. Die erste Version des Data Miners besitzt folgende Funktionen:

- Bahn API aufrufen
- Daten ungeprüft in die Datenbank schreiben

Durch die geringere Datenmenge (Reduzierung der Anzahl abgerufener Stationen von 6600 auf 1200), konnte die erste Umsetzung des Data Miners schnell realisiert werden. Da es sehr viele Optionen und Probleme gab, wurde die erste Version nach etwa acht Wochen durch die zweite Version des Miners dauerhaft ersetzt. Diese besitzt neben neuen Funktionen auch die Erweiterung zur vollständigen Abfrage der API. Außerdem konnten die Probleme der ersten Version des Data Miners behoben werden. Die zweite Version kann durch ihre bessere Performance zudem alle Daten abfragen und nutzt deutlich mehr Informationen, die in der API der Bahn bereitgestellt werden. Die wichtigste Änderung ist die Erkennung von Fehlern in der Abfrage der Datensätze. Hierdurch wird vermieden, dass ungewollt zu viele Datensätze fehlen, ohne dass dies vermerkt wird. Die zweite Version des Data Miners ist in der Lage über 600.000 Datensätze am Tag zu verarbeiten und abzuspeichern. Zu Beginn gab es jedoch noch Probleme mit denen



aus der API Dokumentation erhalten Datenstrukturen. So sollte zum Beispiel ein Gleis angeblich ein Integer sein. Dies trifft jedoch im Falle von "3 A-G", also Gleis 3 Abschnitt A bis G nicht zu. Daher musste die Datenbankspalte für das Gleis entsprechend angepasst werden. Ebenfalls von Fehlern betroffen war die Zugnummer, diese sollte eine gewisse Länge nicht überschreiten, es gab jedoch Zugnummern mit einer Ziffer zu viel, dadurch konnten zu Beginn nicht alle Züge korrekt in der Datenbank gespeichert werden.

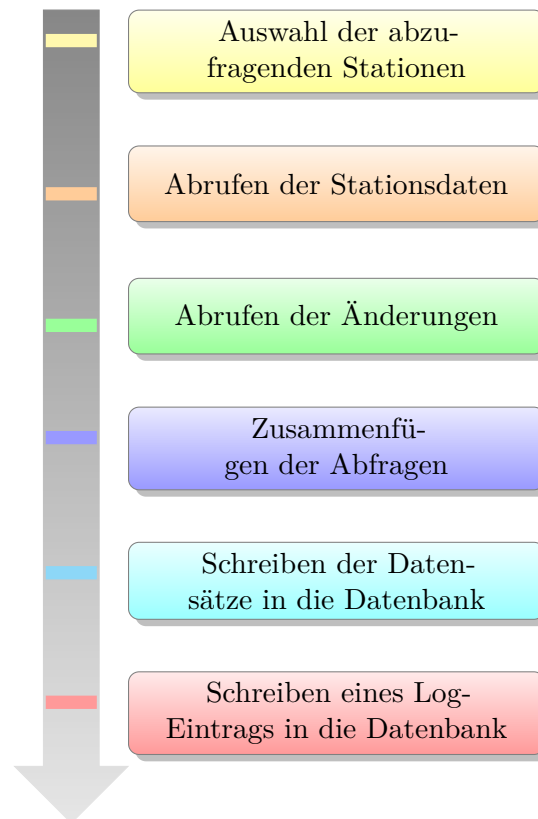


Abbildung 3.1: Grundablauf des Miners

In Listing 3.1 ist ein Ausschnitt des Quellcodes aus der Datenabfrage zu sehen. Zu Beginn werden durch eine MySQL Abfrage die abzufragenden Stationen ermittelt. Diese werden in einer Schleife nach und nach mit den ermittelten API Schlüsseln abgefragt und abgespeichert. Am Ende werden eventuell aufgetretene Fehler in einer Datenbanktabelle abgespeichert.

```

1 include_once './settings.php';
2 require_once 'classes/bahnapi.php';
3 require_once 'classes/MysqliDb.php';
4 $apikey = SETTING_APIKEYTEST;
5 $apikey2 = array_reverse($apikey);
6 $minute = date("i", time());
7 // switching key order
8 if ($minute % 2 == 0) {
9     $bahnapi = new bahnapi($apikey);
10 } else {
11     $bahnapi = new bahnapi($apikey2);
12 }
13 $params = date("Y-m-d_H:i:s", time() - 3600);
14 $mysqlslave = new mysqli(SETTING_DB_IP, SETTING_DB_USER
    , SETTING_DB_PASSWORD, SETTING_DB_NAME);
15 // resetting station fetching
16 if ($minute == 0 || $minute == "00" || $minute == "0") {
17     $stationsquery = $mysqlslave->query("UPDATE_
        haltestellen2_set_fetchtime='2017-12-01_00:00:00'
        ");
18 }
19 $stationsquery = $mysqlslave->query("SELECT_ EVA_NR_ as_
        nr, _NAME_ FROM_ haltestellen2_ WHERE_ fetchactive2=1_ AND_
        fetchtime_ <_ '$params'_ ORDER_ BY_ fetchtime_ ASC_ LIMIT_ 0,
        135");
20 $stationen = array();
21 while ($row = $stationsquery->fetch_assoc()) {
22     $stationen[] = array('nr' => $row['nr'], 'name' => $
        row["NAME"]);
23 }
24 $bahnapi->addToErrorLog("Anz._Fetch:_ " . count($stationen
    ));
25 usleep(5000000);
26 foreach ($stationen as $key => $station) {
27     // get all trains and delays for station
28     $timeold = time() - 7200;
29     $zuege = $bahnapi->getTimetable($station['nr'], $
        timeold);
30     $string = "Station:_ " . $station['nr'] . "_fetched";
31 }
32 // write error log
33 $db = new MysqliDb(SETTING_DB_IP, SETTING_DB_USER,
    SETTING_DB_PASSWORD, SETTING_DB_NAME);
34 $errors = json_encode($bahnapi->getErrorLog());
35 $errordata = array("log" => $errors);
36 $db->insert("errorlog2", $errordata);

```

Quellcode 3.1: Ausschnitt der Datenabfrage

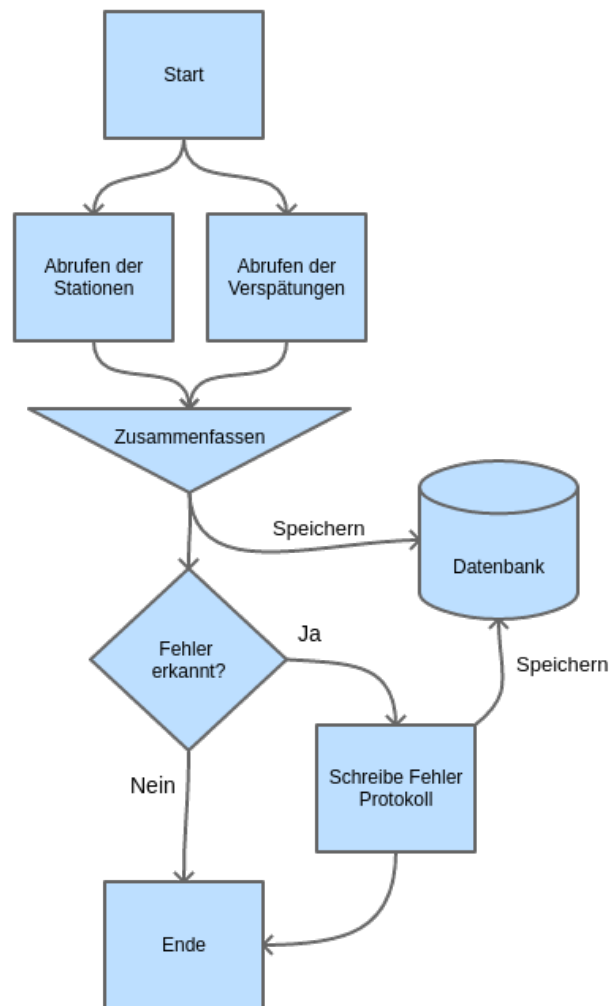


Abbildung 3.2: Ablauf des Miners

	Version 1	Version 2
Fehlererkennung	Nein	Ja
Datensätze abrufen	Ja	Ja
Datensätze speichern	Ja	Ja
HTTP Code erkennen	Nein	Ja
Alle Stationen abfragen	Nein	Ja
Fehler protokollieren	Nein	Ja
Mehr relevante Spalten speichern	Nein	Ja

Tabelle 3.1: Vergleich des Funktionsumfangs der beiden Miner

Die zweite Version des Data Miners kann im Vergleich zur ersten Version zudem mit den HTTP Status Codes automatisch erkennen, ob es auf der Seite der Deutschen Bahn API gerade ein Problem gibt. So wird auch erkannt, dass es am Abend oft zu kurzen Ausfällen der API mit dem HTTP Statuscode 503 [FIELDING 1999] kommt. Dies hilft herauszufinden, ob ein Fehler auf der Seite der Bahn API oder des Data Miners vorliegt. Ein weiterer häufiger Fehler der mangelhaften Initialisierung von Variablen wurde im Laufe der Entwicklung behoben.

Nach der Migration des Miners auf einen größeren und schnelleren Server wird die Performance der Datenbank erheblich verbessert. Die Datenbank profitiert hier vor allem von deutlich mehr Arbeitsspeicher (64 Gigabyte statt 16 Gigabyte), um Abfragen zwischenspeichern. Des weiteren ist der Data Miner nun IPv6 fähig, da der alte Hostserver noch keine eigene IPv6 Adresse hatte. Dies sichert die Funktionalität im Falle einer IPv6 Umstellung der Deutschen Bahn API Schnittstelle in der Zukunft.

### 3.3 Weatherminer

gut/naja

Um eine bessere Prognose der Verspätungen zu ermöglichen, sollte auch das Wetter miteinbezogen werden. Aus diesem Grund wurde entschieden, die Wetterdaten Deutschlands abzuspeichern, um sie später mit einbeziehen zu können. Dafür wurde eine Datenbankstruktur entwickelt, die das Auslesen der Wetterdaten früherer Zeitpunkte ermöglicht. Es ist allerdings zu beachten, dass, wenn der Wert nicht belegt ist, das Feld nicht übertragen wird. Hat es zum Beispiel in den letzten 3 Stunden nicht geregnet, wird das Feld Regen auch nicht übertragen. Um nicht zu viele Daten speichern, beziehungsweise abfragen zu müssen, wurden nicht die Postleitzahlen Postleitzahl (PLZ) genutzt, um das Wetter zu speichern, sondern das nächst größere Gebiet Postleitregion Postleitregion (PLR) genutzt. Die Postleitregionen entsprechen den ersten beiden Zahlen der Postleitzahl und sind wie in Abbildung 3.3 in Gebieten zusammengefasst.

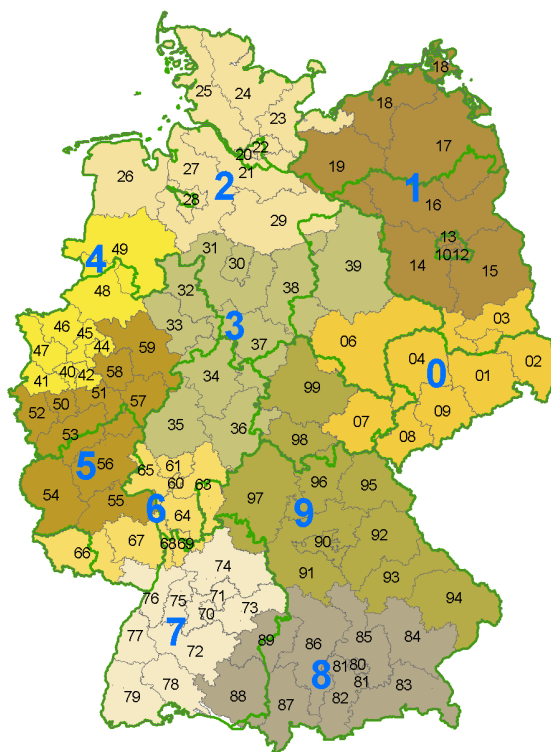


Abbildung 3.3: Übersicht der Postleitregionen von Deutschland. Erstellt von Stefan Kühn, vom Wikimedia Commons

### 3.3.1 OpenWeatherMap

Durch Angabe der Postleitzahl, Koordinaten oder des Namens der Stelle an der das Wetter abgefragt werden soll. In der Antwort können folgende Parameter ausgelesen werden:

**Koordinaten** Die geografische Lage der Stadt die angegeben wurde.

**Wetter** Ein oder mehrere Wetterlagen ID's. Diese geben Aufschluss über die Wetterlage an dem ausgewählten Ort. Eine Liste mit allen Wetterlagen ist in der Tabelle 3.2 auf Seite 28 zu finden. Diese enthält die Wetterlagen ID, die Bezeichnung der Gruppe der Wetterlage, die Beschreibung der Wetterlage und die Kennung des entsprechenden Bildes.

**Main** In diesem Teil der Antwort werden Werte wie die Temperatur in Kelvin, der atmosphärische Druck in hPa, die Luftfeuchtigkeit, die minimale und die maximale Temperatur sowie Druck auf Meereshöhe und Druck auf Normalhöhe angegeben.

**Wind** In diesem Abschnitt wird sowohl die Windgeschwindigkeit als auch die Richtung des Windes abgelegt.

**Wolken** In diesem Abschnitt wird abgelegt, wie viel Prozent des Himmels mit Wolken bedeckt sind.

**Regen** Unter diesem Key wird die Niederschlagsmenge in l/m<sup>2</sup> sowie die Niederschlagsmenge der letzten drei Stunden abgelegt.

**Schnee** Sowie im Abschnitt Regen, wird auch im Abschnitt Schnee die Menge des momentan fallenden Schnees und die der letzten drei Stunden abgelegt.

**Zeitstempel** Zeitpunkt der Datenerhebung

**System** Neben drei nicht weiter beschriebenen Parametern, wird der angegebene Ländercode nochmal zurückgegeben. Auch die Zeit des Sonnenaufgangs in der Region sowie die Zeit des Sonnenuntergangs wird in diesem Abschnitt in Unix Zeitstempelform abgelegt.

**Stadt** Hier wird die ID und der Name der abgefragten Stadt abgelegt.

Die OpenWeatherMap API hat allerdings momentan einen Bug, wodurch die

ID	WCondition
200	thunderstorm with light rain
201	thunderstorm with rain
202	thunderstorm with heavy rain
210	light thunderstorm
211	thunderstorm
212	heavy thunderstorm
221	ragged thunderstorm
230	thunderstorm with light drizzle
231	thunderstorm with drizzle
232	thunderstorm with heavy drizzle
300	light intensity drizzle
301	drizzle
302	heavy intensity drizzle
310	light intensity drizzle rain
311	drizzle rain
312	heavy intensity drizzle rain
313	shower rain and drizzle
314	heavy shower rain and drizzle
321	shower drizzle
500	light rain
501	moderate rain
502	heavy intensity rain
503	very heavy rain
504	extreme rain
511	freezing rain

Tabelle 3.2: Tabelle mit allen Wetterverhältnisse

Parameter Regen und Schnee in Deutschland nicht übermittelt werden. Um trotzdem abspeichern zu können ob es regnet, werden zusätzlich zu den Wind, Regen, Schnee Parametern die Wetterlagen abgespeichert.

### 3.4 Datenbank und Schema

gut (event Skript noch rein,mysql sharding erklären?)

Ein wichtiger Bestandteil des Projektes ist neben dem Abrufen der API das dauerhafte Abspeichern von Datensätzen. Die Struktur dieser Datensätze hat sich mit der Entwicklung des Data Miners ebenfalls verändert. Es werden mit der zweiten Version deutlich mehr Informationen aus der API abgespeichert, als mit der ersten Version. Ein Datensatz benötigt in der ersten Version 140 Bytes und in der zweiten Version 320 Bytes. Viele der neuen Informationen sind für die spätere Arbeit sehr wahrscheinlich wichtig, daher wurden diese in der zweiten Version des

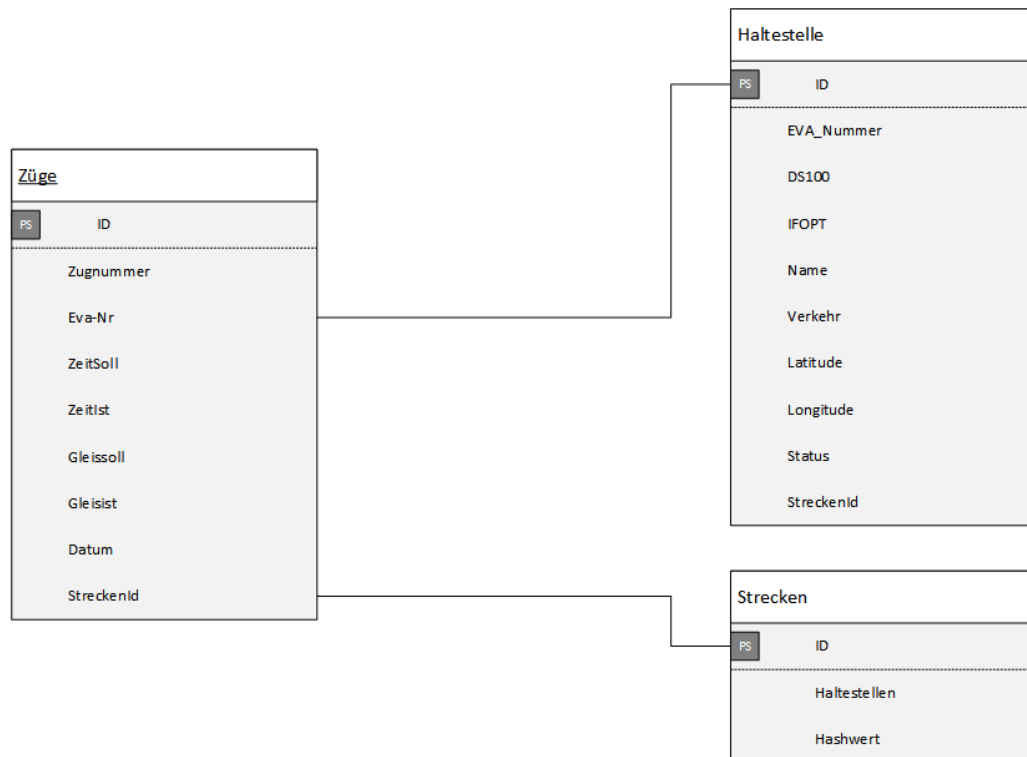


Abbildung 3.4: Datenbank Schema in der ersten Version

Data Miners ausgewählt. Zum Beispiel kann nun der Verlauf eines Zuges besser verfolgt werden und es werden Informationen zum Zugstatus und der Pünktlichkeit strikt getrennt. In Abbildung 3.4 ist das Schema von der ersten Version abgebildet. Im Vergleich zum Schema der zweiten Version fehlen hier einige Spalten und damit Informationen aus der Bahn API. In Abbildung 3.5 dagegen ist das Schema der zweiten Version zu sehen. Dieses Schema besitzt deutlich mehr Spalten pro Datensatz und benötigt daher auch etwas mehr Speicherplatz. Trotzdem beträgt die Größe der Datenbank nach mehr als 20 Millionen Datensätzen noch unter sechs Gigabyte. Ein wichtiger Punkt hierbei ist die Menge an Datensätzen. In wie weit sich die Menge an Datensätzen einen Einfluss auf das Training mit Tensorflow hat wird in Kapitel 5.7 beschrieben. Gegen Abschluss der Studienarbeit befanden sich über 70 Millionen Datensätze in der Datenbank.

Bei dem Umzug des Data Miners mitsamt der Datenbank auf einen neuen Server, musste die bereits vorhandene Datenbank mit einer Größe von 15 Gigabyte migriert werden. Diese Aufgabe war schwieriger als erwartet, da Im- und Export mehrere Stunden Zeit in Anspruch nehmen und nicht exportierte Einträge des Data Miners während des Umzuges mit dem neuen Server synchronisiert werden müssen. Dies ist bei einer Datenbanktabelle, welche dauerhaft mehrere Transaktionen des Miners erfährt, sehr mühsam umzusetzen, da sich die Tabelle ständig ändert. Um den Prozess so schonend wie möglich zu gestalten, wurde ein Skript geschrieben,



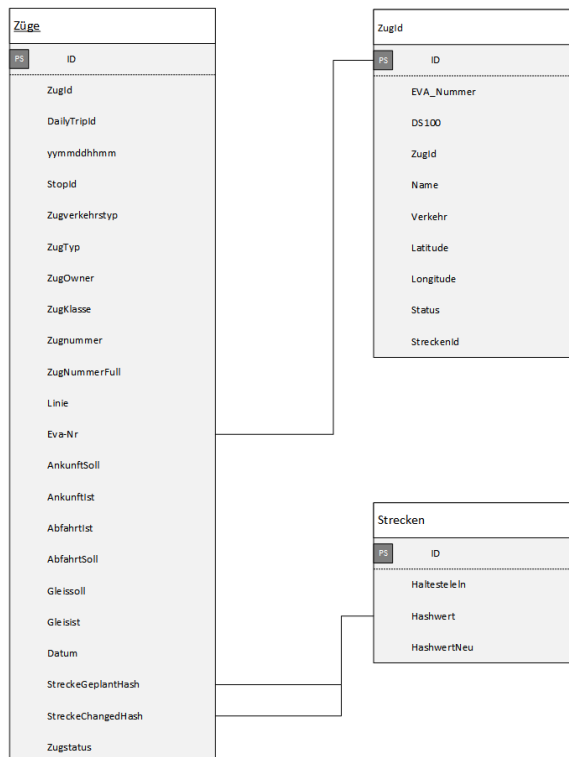


Abbildung 3.5: Datenbank Schema in der zweiten Version

das nach der fertigen Migration der Datenbank die Tabellen miteinander in eine Richtung synchronisiert. Ein MySQL Sharding mit Master- und Slave-Modus war aufgrund inkompatibler Versionen und fehlender Rechte am Server nicht möglich. Nach der Synchronisation der Tabellen durch das Skript wurde der alte Data Miner gestoppt und der Data Miner auf dem neuen Server gestartet. Die Downtime des Data Miners betrug durch den Umzug des gesamten Systems nur circa 60 Sekunden.

### 3.4.1 Datenbank Schema des Wetterminers

Für das Datenbankschema wurden weitestgehend die Felder der API Antwort als Spalten der Datenbank übernommen. In der Spalte Time wird ein Timestamp abgespeichert, an dem der Datensatz aufgenommen wurde. Die Spalte PLR wird genutzt, um die Postleitregion abzulegen, Temperatur, Luftfeuchtigkeit, Luftdruck, Windgeschwindigkeit und Windrichtung werden in je einer Spalte gespeichert. Da mehrere Wetterlagen in einem Datensatz möglich sind, musste für die Speicherung der Wetterlagen eine zusätzliche Tabelle angelegt werden, die die Verlinkung zwischen Datensatz und Wetterlage herstellt. Diese speichert die ID des Datensatzes und die ID der Wetterlage.

### 3.5 Backup der Datenbank

gut

Jeder Datensatz des Data Miners ist wichtig. Daher soll für diese kritische Stelle, der persistenten Speicherung der Daten, ein automatisiertes und verifizierbares Backup entstehen. Hierbei gibt es zwei Hauptprobleme zu lösen: Zum einen muss während des Backups eine große Transaktion im Cache oder auf der Festplatte zwischengespeichert werden. Zum anderen ist durch die Menge der Datensätze eine manuelle Verifikation, ob die Datensätze auch wieder einspielbar sind, sehr aufwändig. Daher wird ein kleines Skript geschrieben, welches mit linearem Aufwand (Größe der Datei) die Datensätze an bestimmten Stellen aufsplittet. So entstehen viele kleinere Dateien. Diese können in unter einer Minute mit einem Datenbankimport auf Funktion geprüft werden. Dies ermöglicht nach einem vollständigen Backup die einzelnen Dateien automatisiert nach und nach in einer kleineren Datenbank zu prüfen. Sollte ein Fehler beim Import auftreten, wird dieser in den MySQL eigenen Fehler Log geschrieben. Hier gilt der Grundsatz, nutzen was schon vorhanden ist (K.I.S.S). In Listing 3.2 und 3.4 wird der Quellcode des Skriptes zum Aufteilen der Datensätze gezeigt. Die Laufzeit wird grundsätzlich durch die I/O-Geschwindigkeit der Festplatte bestimmt. Die Begrenzung der erstellten Dateien erwies sich bei der Implementierung als Hilfe, um ein ungewolltes, fehlerhaftes Anlegen von tausenden kleinen Dateien zu vermeiden.

```

1 set_time_limit(600);
2 ini_set("auto_detect_line_endings", true);
3 ini_set('memory_limit', '1024M');
4 /* Number of 'insert' statements per file */
5 $max_lines_per_split = 50;
6 $dump_file = "G:\\Backup\\Studienarbeits\\DB\\backup_zuege
  2_090218.sql";
7 $split_file = "dump-split-%d.sql";
8 $dump_directory = "G:\\Backup\\Studienarbeits\\DB\\sql-
  dump\\";
9 $line_count = 0;
10 $file_count = 1;
11 $total_lines = 0;
12 $handle = fopen($dump_file, "r");
13 $buffer = "";

```

Quellcode 3.2: Skript zum Aufteilen eines großen MySQL Dumps in mehrere kleine Dumps, Teil 1

Um die erstellten Dateien wieder in eine Datenbank einzufügen, wird ein kleines Skript implementiert, dass mit dem Befehl 3.3 die Dateien nach und nach wieder einfügt. Das Skript wird auf einem Server dauerhaft genutzt, um das Backup zu

prüfen. Da diese Schleife eine sehr hohe Laufzeit haben kann werden die Befehle direkt auf dem Server ausgeführt. Denn auf einer Web Oberfläche würde der Befehl nach einiger Zeit abbrechen, da das Skript in einen Timeout läuft.

```
1      mysql -u test Zuege < dump-split-$fn.sql || echo  
      "error_in_file  
      *****"
```

Quellcode 3.3: Befehl zum Einfügen der SQL Dateien

```

1 if ($handle) {
2   while(($line = fgets($handle)) !== false) {
3     $total_lines++;
4     if($total_lines > 50000000 || $file_count > 200) {
5       break;
6     }
7     if(preg_match("/insert_into/i", $line)) {
8       /* Copy buffer to the split file */
9       if($line_count >= $max_lines_per_split) {
10        $file_name = $dump_directory . sprintf($split_file ,
11          $file_count);
12        $out_write = @fopen($file_name, "w+");
13        fputs($out_write, $buffer);
14        fclose($out_write);
15        $buffer = '';
16        $line_count = 0;
17        $file_count++;
18      }
19      $line_count++;
20      $buffer .= $line;
21    }
22    if($buffer && strlen($buffer) < 200) {
23      /* Write out the remaining buffer */
24      $file_name = $dump_directory . sprintf($split_file , $
25        file_count);
26      $out_write = @fopen($file_name, "w+");
27      fputs($out_write, $buffer);
28      fclose($out_write);
29    } elseif ($buffer) {
30      echo 'Warning_please_check_last_file_if_complete';
31    }
32    fclose($handle);
33    echo "Split_done.";
34  } else {
35    echo 'No_Handle_obtained.';
36  }

```

Quellcode 3.4: Skript zum Aufteilen eines großen MySQL Dumps in mehrere kleine Dumps, Teil 2

# Kapitel 4

## Datenverarbeitung mit Data Mining

### 4.1 Grundlagen von Data Science und KDD

#### 4.1.1 Data Mining

leere subSection

Knowledge Discovery

#### 4.1.2 Knowledge Discovery in Databases (KDD)

Das Ziel von KDD ist es, das menschliche Vermögen, Daten zu analysieren und zu untersuchen, zu steigern, indem die Datenanalyse automatisiert wird. Die Automatisierung ist nötig, um mit den immer größer werdenden Datenmengen umgehen zu können. [USAMA FAYYAD 1996, S. 39]

Der KDD-Prozess ist ein mehrstufiger Prozess, der sich nach [] in folgende Schritte untergliedern lässt:

zitat leer

KDD verwendet zur Analyse von Daten auch das Data Mining. Jedoch wird ein Unterschied zwischen KDD und Data Mining gemacht: Data Mining ist das Anwenden von spezifischen Algorithmen auf Daten, um Muster in den Daten zu finden. KDD hingegen ist ein allumfassender Prozess, um Wissen aus den Daten zu gewinnen. [USAMA FAYYAD 1996, S. 39]

### 4.2 Grundlagen

Hier noch text

Begriffsdefinition

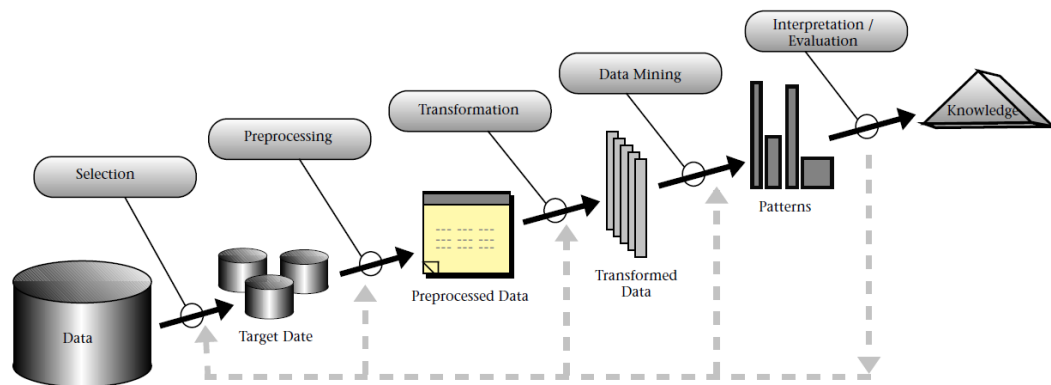


Abbildung 4.1: Übersicht des KDD-Prozesses [USAMA FAYYAD 1996, S. 41].

## Statistische Methoden

## Machine Learning

## Visualisierungsmethoden

**Denkbare Auswertungen** - Verspätungsarten pro Linie - davon der Durchschnitt über mehrere Züge die die gleiche Linie zu verschiedenen Zeiten befahren - damit können Heatmaps erzeugt werden

## 4.3 Vorverarbeitung der Daten

Bevor die gesammelten Daten analysiert werden können, müssen Teile der Datensätze vorverarbeitet werden, um sie in ein brauchbares Datenformat zu bringen.

Strecken eines Zuges werden in langen Zeichenketten statt EVA-Nummern abgelegt

Die Datenbank enthält mehrere tausend Datensätze von Zügen die an verschiedenen Haltestellen und Bahnhöfen halten. Die Strecke, die ein Zug fährt ist eine der wichtigsten Informationen, die aus den Datensätzen ausgelesen werden muss. Jedoch ist das in dem ursprünglichen Format der Datensätze sehr ineffizient umzusetzen.

Die einzelnen

fehlt hier was

Dieses Kapitel bezieht sich auf den Tabellen cache vorgang, also generierung einer neuen besser select baren tabelle, das Quellcode listing ist noch nicht korrekt

```

1 import sys
2 import argparse
3 from glob import glob
4 import os
5 import datetime
6 import json
7 import re
8 FLAGS, unparsed = parser.parse_known_args()
9
10 def timetotimeint(input):
11     input = str(input)
12     if input == 'None':
13         input = "24:00:00"
14     hhmmss = input
15     (h, m, s) = hhmmss.split(':')
16     result = int(h) * 60 + int(m)
17     result = math.floor(result/5)
18     return result
19
20 def openvocalfile(name):
21     lines = []
22     filename = "./vocabfiles/" + str(name) + ".txt"
23     with open(filename, mode="w+", encoding="utf-8") as
         file:
24         for line in file:
25             line = line.rstrip('\n')
26             lines.append(line)
27     return lines
28

```

Quellcode 4.1: Some Python File

Ebenfalls für Abfragen ineffizient, ist der "Primary key" der Deutschen Bahn. Dieser besteht, wie in 3.1.5 beschrieben aus drei Teilen. Um eine Abfrage der Datenbank auf einen bestimmten Zug zu machen muss die Daily Trip ID sowie das Datum im "yymmddhhmm"Format angegeben werden. Diese Werte müssten dann allerdings von der Datenbank mit dem String der Zug Id verglichen werden. Das ist natürlich nicht sehr performant. Deshalb wurde entschieden, dass die Datenbankstruktur um drei Spalten erweitert wird. Da sie im Nachhinein hinzugefügt worden sind, müssen alle schon vorhandenen Einträge bearbeitet werden. Dafür wurde ein Algorithmus geschrieben, der die Zugid aus der Datenbank ausliest und wie in Listing 4.2 beschrieben aufteilt. Diese Komponenten werden dann in die

jeweiligen Spalten eingebunden.

```

1     if temp.empty:
2         print("actual_id_{ }_is_empty".format(
3             actual_id))
4         missing_IDs.write('{}\n'.format(actual_id))
5         missing_IDs.flush()
6     else:
7         # check if id is already filled
8         if temp["stopid"][0] is None:
9             # split zugid in komponenten
10            zugid = temp["ttsid"][0]
11            print("actual_id_{ }\t\tzugid:_{ }\n\r".format(
12                actual_id, zugid))
13            zugid = zugid.split("-")
14            # if first komponent is empty dailytripid
15            was negative
16            try:
17                if zugid[0] == "":
18                    zugid[1] = int(zugid[1]) * (-1)
19                    qs.insert_3tuple_with_id(actual_id,
20                        zugid[1], zugid[2], zugid[3])
21            else:
22                qs.insert_3tuple_with_id(actual_id,
23                    (zugid[0]), zugid[1], zugid[2])
24            except ConnectionResetError:

```

Quellcode 4.2: Zerlegen der Zug ID in seine Komponenten

## 4.4 Software-Architektur der Datenauswertung

gut

In diesem Abschnitt wird kurz die Software-Architektur dargestellt, die bei dem Data Mining angewandt wird. In Abbildung 4.2 zeigt sich, wie der allgemeine Ablauf des Datenabrufes mit der Architektur zusammenhängt: Grundsätzlich müssen die Daten zuerst beschafft werden, bevor sie ausgewertet werden können. Aus diesem Grund werden zuerst die nötigen Daten aus der Datenbank abgerufen. Für die Datenbankabfrage wird die Python-Klasse “QuerySuite” eingeführt. Die “QuerySuite” gibt die Daten der Abfrage in Dataframes zurück, die geeignet für den Transport der Daten sind. Nach der Abfrage werden die Daten weiterverarbeitet. Zu diesem Zweck wird das Python-Package “ProcessingUtils” konzipiert, das Funktionen enthält, um Berechnungen mit den Dataframes ausführen zu können. Im Folgenden wird genauer auf die genannten Komponenten eingegangen.



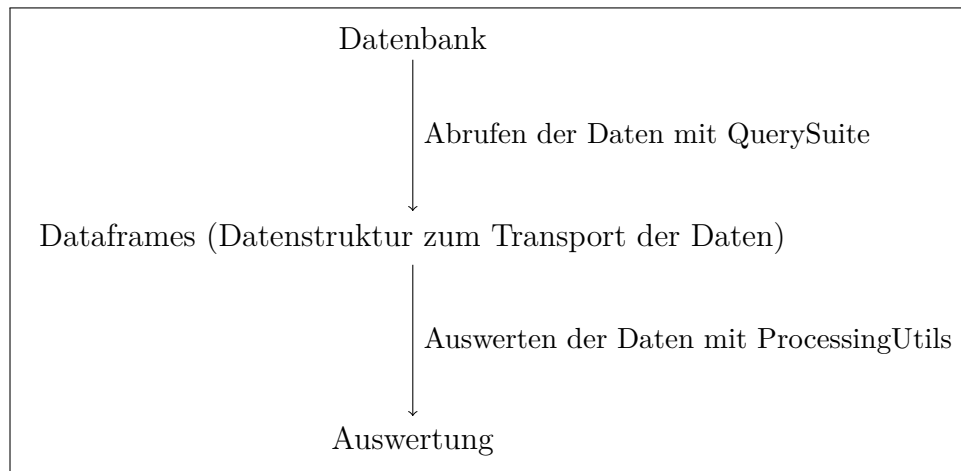


Abbildung 4.2: Grundablauf des Data Minings

#### 4.4.1 Query Suite

**Kapselung der SQL-Queries** Da, je nach Auswertung, verschiedene Datensätze aus der Datenbank gebraucht werden, sind auch mehrere SQL-Queries notwendig, um diese Daten von der Datenbank abzufragen. Zu diesem Zweck werden die Abfragen von der Klasse “QuerySuite” durchgeführt. Die Klasse enthält für jede SQL-Abfrage eine spezielle Methode, die diese Abfrage durchführt. Die Abfragen werden in Methoden gekapselt, um das Programmieren übersichtlicher zu gestalten. Jede Methode kann zusätzliche Parameter bei Aufruf entgegennehmen, die dann in der SQL-Abfrage verwendet werden können. Auf diese Weise müssen die SQL-Abfragen nicht ständig kopiert und abgeändert werden, wenn sich nur die Parameter, nicht jedoch die Struktur der Abfrage ändern. Diese Maßnahme verhindert Code-Duplication, die in der Software-Entwicklung vermieden werden soll. Sollte sich während der Entwicklung beispielsweise das Datenbankmodell ändern, so müssen nur die einzelnen SQL-Abfragen, die in den jeweiligen Methoden der QuerySuite gekapselt sind, angepasst werden, anstatt nach jeder SQL-Abfrage in jeder Auswertung zu suchen und anzupassen.

**Verwaltung der Verbindung** Die Klasse ist neben den Abfragen auch verantwortlich für die dafür benötigte Verbindung zur Datenbank. Der Klasse wird einmalig die zu verwendende Datenbankverbindung übergeben, die für alle nachfolgenden Aufrufe der Abfrage-Methoden verwendet werden. Bevor die SQL-Queries an die Datenbank gestellt werden, fügt die Klasse noch eine “LIMIT”-Klausel an die aufzuführende SQL-Query an, sodass die Anfragen implizit begrenzt werden. Während der Entwicklung ist das ein wichtiges Feature, um bei fehlerhaften Abfragen die Datenbank nicht unnötig zu überlasten. Werden von einer Query mehr Datensätze erwartet, als es die Begrenzung zulässt, so wird für diese spezielle Query die Begrenzung davor explizit als aufgehoben.

**Umformatierung der Datensätze in ein Dataframe** Nach der Query liefert die Datenbank die Ergebnisse. Die Python-Bibliothek PyMySQL, die für die Datenbankinteraktionen verwendet wird, speichert hierbei die Ergebnisse in einem zweidimensionalen Python-Tupel. Das Tupel enthält zeilenweise weitere Tupel, die die einzelnen Datensätze enthalten. Die inneren Tupel speichern jeweils die Elemente eines Datensatzes in der Reihenfolge der Tabellenspalten aus der Datenbank. Für die Weiterverarbeitung der Datensätze ist das Datenformat des zweidimensionalen Tupels jedoch nachteilig. Dies liegt daran, dass Tupel dafür gedacht sind, Folgen von Elementen mit einem Reihenfolgencharakter zu speichern. Um zu wissen, welches Datum in welcher Spalte und Zeile des Tupels welcher Tabellenspalte aus der Datenbank entspricht, muss die vorangegangene Query bekannt sein. Die nachfolgende Datenverarbeitung soll also nicht eine Reihenfolge von Elementen als Eingabedaten erhalten, sondern ein Datenformat erhalten, das dem Datenmodell der Datenbank nachempfunden wird.

Zu diesem Zweck wird die Python-Bibliothek Pandas verwendet. Pandas ermöglicht es, sogenannte Dataframes zu erzeugen. Ein Dataframe ist eine tabellenähnliche Datenstruktur, sodass die enthaltenen Daten nach Spalten und Zeilen organisiert werden. Die Spalten können hierbei mit Labels und die Zeilen mit Indizes versehen werden. Auf diese Weise können die Ergebnisse der Datenbankabfrage in Dataframes eingefügt werden, sodass die Spalten der Dataframes auch die entsprechenden Spaltennamen der Datenbanktabellen tragen.

Das Umformatieren der Query-Daten in ein Dataframe hat den Vorteil, dass die nachfolgende Datenverarbeitung die gewünschten Daten nach den Tabellennamen und den Indizeswerten auflösen kann und nicht auf eine feste Reihenfolge von Elementen in einem Tupel angewiesen ist. Aus diesem Grund werden Dataframes als Datenstruktur verwendet, um die Daten während der Verarbeitung zu transportieren.

**Beispiel** In Quellcode 4.3 ist als Beispiel “get\_tts\_by\_ttsid” als eine der Query-Methoden in der QuerySuite-Klasse aufgelistet. An diesem Beispiel soll gezeigt werden, wie die oben genannten Punkte sich in dem Quellcode äußern. In Zeile 2 werden zunächst die Labels beschrieben, die für die Benennung der Dataframe-Spalten verwendet werden. In Zeile 9 steht die Methode “\_get\_tts\_by\_ttsid\_query”. In Zeile 10 fügt die Methode, den übergebenen Parameter “ttsid” in die SQL-Query ein. In Zeile 12 findet über die Methode “\_do\_query” die Ausführung der Query statt.

In Zeile 15 wird die Methode “get\_tts\_by\_ttsid” definiert. In Zeile 19 erhält die Methode das Query-Ergebnis als Tupel zurück. Anschließend wird in Zeile 20 das Tupel in ein Dataframe umformatiert. Wie zu sehen ist, wird vor der Erzeugung des Tupels das zweidimensionale Tupel mittels der Funktion “list” in eine

Liste von eindimensionalen Tupeln konvertiert, damit Pandas die Struktur der Daten richtig deuten kann. Bei der Erzeugung werden der “DataFrame”-Funktion die zuvor definierten Labels für die Spalten mitgegeben.

```

1 # labels for table "time table stops" (named "zuege" in
  database)
2 TABLE_LABELS_TTS = ["id", "ttsid", "dailytripid",
3   "ymmddhhmm", "stopindex", "zugverkehrstyp", "zugtyp",
4   "zugowner", "zugklasse", "zugnummer", "zugnummerfull",
5   "linie", "evanr", "arzeitsoll", "arzeitist", "dpzeitsoll",
6   "dpzeitist", "gleissoll", "gleisist", "datum",
7   "streckengeplanthash", "streckenchangedhash", "zugstatus"]
8
9 def _get_tts_by_ttsid_query(self, ttsid):
10     query = "SELECT * FROM zuege WHERE zuege.zugid = "
11           "\{}\{}\".format(ttsid)
12     result = self._do_query(query)
13     return result
14
15 def get_tts_by_ttsid(self, ttsid):
16     """
17     Retrieves full row of database table by given 'ttsid'
18     (named 'zugid' in database).
19     """
20     result = self._get_tts_by_ttsid_query(ttsid)
21     result_df = pd.DataFrame(data=list(result), columns=TABLE_LABELS_TTS)
22     return result_df

```

Quellcode 4.3: Beispiel einer Query-Methode

#### 4.4.2 Processing Utils

Das Package “ProcessingUtils” beinhaltet verschiedene Python-Funktionen, die bei den unterschiedlichen Datenauswertungen benutzt werden. Die Funktionen operieren auf den von der QuerySuite gelieferten Dataframes. Das Package dient nur dazu, die verschiedenen Funktionen zu gruppieren, die für die Datenauswertung geschrieben werden. Wie die Funktionen schließlich kombiniert werden, hängt von der jeweiligen Auswertung ab, die Gebrauch von diesen Funktionen machen kann.

## 4.5 Statistische Auswertungen

naja (nicht ganz gelesen da muede)

### 4.5.1 Definition der Verspätungen

Für die Auswertung der Daten ist die Verspätung eine interessante Größe. Hierbei können verschiedene Verspätungen definiert und in dem Datenbestand untersucht werden. In diesem Abschnitt werden verschiedene Verspätungsarten definiert und anschließend dargestellt, wie diese in dem Datenbestand analysiert werden.

**Verspätung bei Ankunft** Die Verspätung der Ankunft  $\Delta an$  eines Zuges  $zug_m$  im Bahnhof  $bhf_n$  ist definiert als

$$\Delta an(bhf_n, zug_m) := an_{real}(bhf_n, zug_m) - an_{plan}(bhf_n, zug_m) \quad (4.1)$$

**Verspätung bei Abfahrt** Die Verspätung der Abfahrt  $\Delta ab$  eines Zuges  $zug_m$  im Bahnhof  $bhf_n$  ist definiert als

$$\Delta ab(bhf_n, zug_m) := ab_{real}(bhf_n, zug_m) - ab_{plan}(bhf_n, zug_m) \quad (4.2)$$

**Geplante Haltedauer** Die geplante Haltedauer lässt sich mit folgender Formel ermitteln:

$$halten_{plan}(bhf_n, zug_m) := ab_{plan}(bhf_n, zug_m) - an_{plan}(bhf_n, zug_m) \quad (4.3)$$

**Reale Haltedauer** Die reale Haltedauer lässt sich mit folgender Formel ermitteln:

$$halten_{real}(bhf_n, zug_m) := ab_{real}(bhf_n, zug_m) - an_{real}(bhf_n, zug_m) \quad (4.4)$$

**Verspätung durch Haltedauer** Hier werden die zwei zuvor aufgestellten Formeln für die geplante und reale Haltedauer wieder aufgegriffen und mit ihnen die Verspätung definiert, die durch eine zu lange außerplanmäßige Haltedauer entsteht.

$$\Delta halten(bhf_n, zug_m) := halten_{real}(bhf_n, zug_m) - halten_{plan}(bhf_n, zug_m) \quad (4.5)$$

**Geplante Fahrtdauer** Zum Ermitteln der geplanten Fahrtdauer werden nun zwei Bahnhöfe benötigt. Der eine ist der Start-Bahnhof ( $bhf_{n-1}$ ) und der andere ist der Zielbahnhof ( $bhf_n$ ), mit denen nun über die Ankunfts- und Abfahrtszeit die Fahrtdauer ermittelt werden kann:

$$fahren_{plan}(bhf_{n-1}, bhf_n, zug_m) := an_{plan}(bhf_n, zug_m) - ab_{plan}(bhf_{n-1}, zug_m) \quad (4.6)$$

**Reale Fahrtdauer** Die reale Fahrtdauer wird nach dem gleichen Schema wie die geplante Fahrtdauer berechnet:

$$fahren_{real}(bhf_{n-1}, bhf_n, zug_m) := an_{real}(bhf_n, zug_m) - ab_{real}(bhf_{n-1}, zug_m) \quad (4.7)$$

**Verspätung durch Fahrtdauer** Aus der geplanten und der realen Fahrtdauer, lässt sich nun die Verspätung, die sich durch eine außerplanmäßige Fahrtdauer ergibt, ermitteln:

$$\Delta fahren(bhf_{n-1}, bhf_n, zug_m) := fahren_{real}(bhf_{n-1}, bhf_n, zug_m) - fahren_{plan}(bhf_n, bhf_{n-1}, zug_m) \quad (4.8)$$

**Beispiel** Die Berechnungen der verschiedenen Verspätungen sind mit der Programmiersprache Python implementiert. Folgendes Beispiel 4.4 zeigt die Funktion, die die Verspätung bedingt durch die Fahrtzeit berechnet wird.

Zu Beginn erhält die Funktion Parameter, die bestimmen, welcher Abschnitt einer Route untersucht werden soll. Hierfür bekommt die Funktion zwei Dataframes übergeben. Im ersten Dataframe wird der Startpunkt des Abschnitts angegeben (“train\_stop\_from\_df”) und im zweiten Dataframe wird der Zielpunkt des Abschnitts angegeben (“train\_stop\_to\_df”).

Bevor mit der Berechnung der Verspätung begonnen wird, validiert die Funktion zunächst, ob die nötigen Parameter übergeben wurden. Es kann während der Analyse einer ganzen Route vorkommen, dass der Startpunkt oder Zielpunkt eines Abschnittes nicht angegeben werden und somit den Wert “None” haben.

Im nächsten Schritt wird das “ttsid”-Attribut, das einen Haltepunkt eindeutig identifiziert, von Start- und Zielpunkt ausgelesen. Die “ttsid”-Attribute von Start- und Zielpunkt wird zum Schluss im Ergebnis-Dataframe zusammen mit der berechneten Verspätung gespeichert, um die Verspätung dieser Strecke in späteren Auswertungen zuordnen zu können.

Im nächsten Schritt wird nun die Verspätung, die bedingt durch die Abweichung von der geplanten Fahrtzeit auftritt, berechnet. Auch hier wird überprüft, ob beide Punkte definiert wurden. Ist ein Punkt nicht angegeben worden (und besitzt somit den Wert “None”), so wird die Verspätung mit einem Wert von “NaT” (Not a Time) angegeben, um zu signalisieren, dass für den gegebenen Start- und Zielpunkt keine Verspätungsberechnung durchgeführt werden konnte. Die Repräsentation von nicht vorhandenen Verspätungen mittels “NaT” ist hierbei sinnvoll, da diese Werte bei der späteren Weiterverarbeitung oder Visualisierung ignoriert werden können.

Im letzten Schritt wird der Ergebnis-Dataframe konstruiert erstellt. In der Spalte “traveltime\_real” speichert dieser, die ermittelte Verspätung. In der Spalte “ttsid\_from” und “ttsid\_to” werden die “ttsid”-Attribute des Start- und Zielpunktes gespeichert. Der befüllte Dataframe wird schließlich an den Aufrufenden der Funktion zurückgegeben.

```

1 def calc_delay_by_traveltime_df(train_stop_from_df,
2   train_stop_to_df):
3     """
4     Calculates the delay that has been caused by the
5     travel of the train.
6     Positive value means, that the travel time caused
7     additional delay.
8     Negative value means, that the travel time decreased
9     the delay.
10    :param train_stop_from_df: Pandas dataframe. Input
11    for the train stop the train comes from.
12    :param train_stop_to_df: Pandas dataframe. Input for
13    the train stop the train arrives at.
14    :return: Returns a pandas dataframe with columns '
15    delay_by_traveltime', 'ttsid_from', 'ttsid_to'.
16    """
17    if train_stop_from_df is None:
18        ttsid_from = None
19    else:
20        ttsid_from = train_stop_from_df["ttsid"].iloc[0]
21
22    if train_stop_to_df is None:
23        ttsid_to = None
24    else:
25        ttsid_to = train_stop_to_df["ttsid"].iloc[0]
26
27    if train_stop_from_df is None or train_stop_to_df is
28    None:
29        delay = pd.NaT
30    else:
31        traveltime_real = calc_traveltime_real_df(
32            train_stop_from_df, train_stop_to_df)["
33            traveltime_real"].iloc[0]
34        traveltime_scheduled =
35            calc_traveltime_scheduled_df(
36                train_stop_from_df, train_stop_to_df)["
37                traveltime_scheduled"].iloc[0]
38        delay = traveltime_real - traveltime_scheduled
39
40    result = pd.DataFrame(
41        data=[[delay, ttsid_from, ttsid_to]],
42        columns=["delay_by_traveltime", "ttsid_from", "
43            ttsid_to"])
44    return result

```

Quellcode 4.4: Berechnung der SARV

### 4.5.2 Analyse der Verspätungen eines Zuges

Mit den in Abschnitt 4.5.1 definierten Verzögerungen ist es bereits möglich, erste statistische Auswertungen auszuführen. In diesem Abschnitt wird dargestellt, wie die Verspätungsarten eines Zuges entlang seiner Route analysiert werden. Hierbei werden bei der Analyse diese Verspätungsarten berücksichtigt.

- Verspätung bei Ankunft
- Verspätung bei Abfahrt
- Verspätung durch Haltezeit
- Verspätung durch Fahrtzeit

**Beispiel** Als Beispiel ist nun Abbildung 4.3 zu betrachten, die den RE4725 von Karlsruhe Hbf. nach Konstanz zeigt. Anhand des Beispiels wird die Bedeutung von Visualisierungen im Data Mining recht deutlich, da hier wichtige Fakten schnell erkannt werden können.

Die blaue Kurve ist sehr auffällig und zeigt, dass die Verspätung bei Ankunft ab Offenburg kurz aber stark zunimmt und erst nach Donaueschingen wieder abnimmt. Das Maximum der Verspätung bei Ankunft liegt bei +5 Minuten, das bei den Stationen Haslach, Hausach, Hornberg und Donaueschingen erreicht wird.

Wird die orangene Kurve betrachtet, wird auch deutlich, weshalb die Verspätung kurz nach Offenburg zunimmt: Der Zug wartet unplanmäßig lange in Offenburg (+4 Min.) und verursacht so die Verspätung. Ansonsten ist gut zu sehen, dass die orangene Kurve sehr wenig schwankt und die Verspätung durch Haltezeit für die weitere Fahrt des Zuges keine wesentliche Ursache für weitere Verspätungen darstellt.

Wird nun die rote Kurve betrachtet, so ist auch hier zu erkennen, dass die Verspätung durch Fahrtzeit im Mittel keine Verspätungen verursacht. Zwischen Hornsberg und Allensbach ist die Fahrtzeit dafür verantwortlich, dass die in Offenburg verursachte Verspätung wieder abgebaut wird. Jedoch zeigt sich noch eine Besonderheit: Zwischen Konstanz-Petershausen und Konstanz nimmt die Verspätung des Zuges (bei Ankunft) wieder deutlich zu, da der Zug für die Strecke außerplanmäßig +5 Minuten länger benötigt.

Zuletzt ist die violette Kurve zu betrachten. Hier ist zu sehen, dass die Verspätung bei Abfahrt kaum Mehrwert liefert, da sie größtenteils Deckungsgleich mit der Verspätung bei Ankunft ist.

Nun sollen Interpretationen dargestellt werden, die aus den visualisierten Daten



gewonnen werden können. Die erste Interpretation lautet, dass Offenburg womöglich ein Bahnhof ist, der oft Verspätungen durch zu lange Haltezeit aufbaut. In Offenburg werden viele Verbindungen zu anderen Zügen hergestellt, sodass Abweichungen von der geplanten Wartedauer plausibel erscheinen, wenn ein Zug auf einen anderen Verbindungszug warten muss. Um dieser Vermutung nachzugehen, wäre es eine Idee, den Durchschnitt der Verspätung durch Verspätung von mehreren Zügen zu bestimmen, die in Offenburg halten. Wenn der Durchschnitt eine erhebliche Verspätung zeigt, könnte damit die Hypothese gestützt werden, dass das Warten auf Verbindungszüge ein möglicher Grund für Verspätungen ist.

Eine andere Interpretation ist, dass die Strecke zwischen Konstanz-Petershausen und Konstanz möglicherweise auch ein Grund für Verspätungen ist. Hier ist eine Durchschnittsberechnungen von mehreren Zügen auch eine gute Idee, um dieser Vermutung nachzugehen. Mögliche Gründe für Verspätung auf einer Strecke könnten sein, dass der Zug beispielsweise auf Signalfreigabe zur Fortsetzung der Fahrt warten muss.

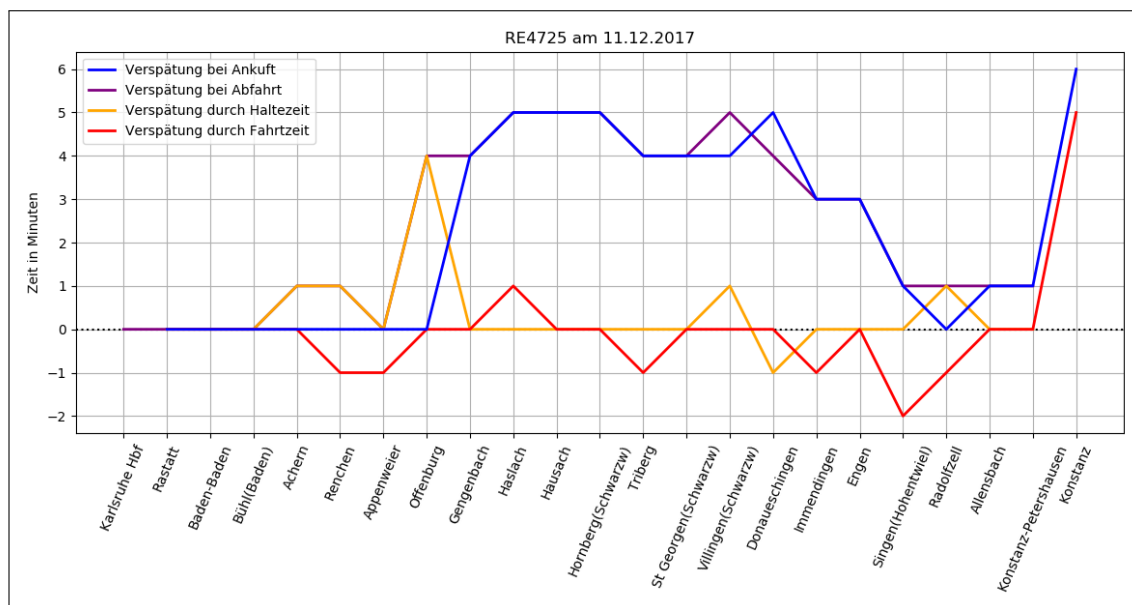


Abbildung 4.3: Verspätungsanalyse von RE4725 von Karlsruhe nach Konstanz

**Verspätung bei Abfahrt liefert kaum Mehrwert** Aus der zuvor gezeigten Abbildung 4.3 wurde die Beobachtung gemacht, dass die Verspätung bei Abfahrt größtenteils deckungsgleich mit der Verspätung bei Ankunft ist und deshalb kaum neue Informationen liefert. Auch in Abbildung 4.4 kann sehr gut beobachtet werden, dass sich der Verlauf der Verspätung bei Abfahrt nur geringfügig von dem Verlauf der Ankunft unterscheidet.

Die Verspätung bei Abfahrt trägt auch an sich kaum Bedeutung: Es ist bei

der Analyse der Verspätungen vorwiegend die Verspätung bei Ankunft interessant. Schließlich ist es für einen Fahrgast wichtig, dass ein Zug pünktlich am Start- und Zielbahnhof ankommt. Ob der Zug verspätet aus einem Bahnhof abfährt, ist dem Fahrgast gleichgültig, solange sich die Ankunft nicht verspätet.

Aus diesen Gründen wird die Verspätung bei Abfahrt nicht mehr in der Darstellung der Verspätungen berücksichtigt. Dies hat auch den Vorteil, dass die Visualisierungen der Verspätungen übersichtlicher wird. Dies kann in Abbildung 4.5 nachvollzogen werden, die den gleichen Zug wie in Abbildung 4.4 zeigt, aber die Verspätung bei Abfahrt ausgelassen wird.

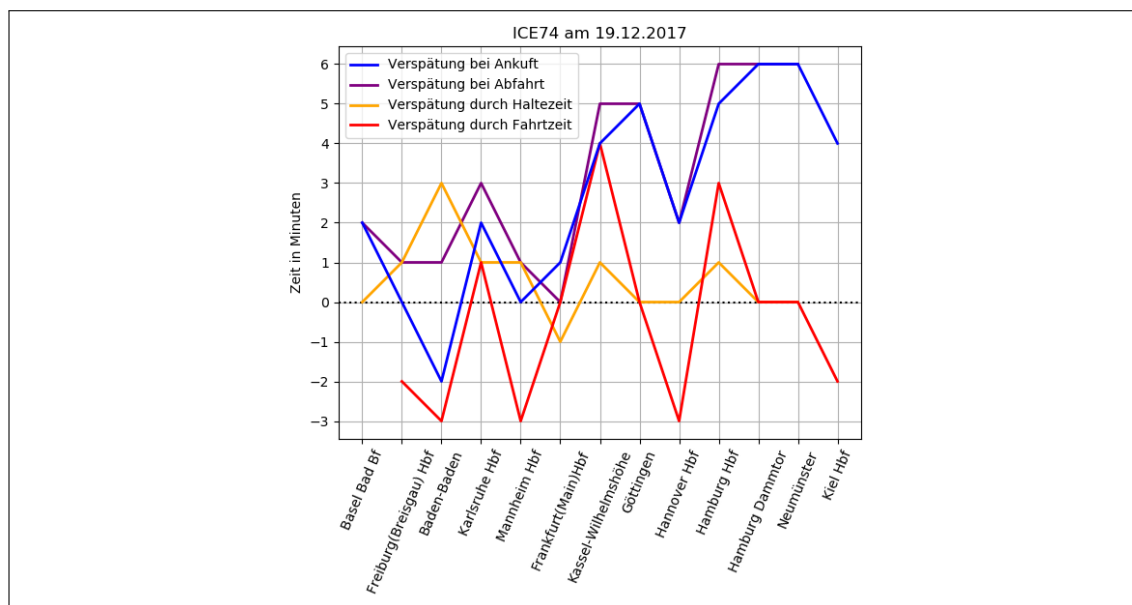


Abbildung 4.4: Verspätungsanalyse von ICE74 von Basel Bad. Bf. nach Kiel mit Verspätung bei Abfahrt

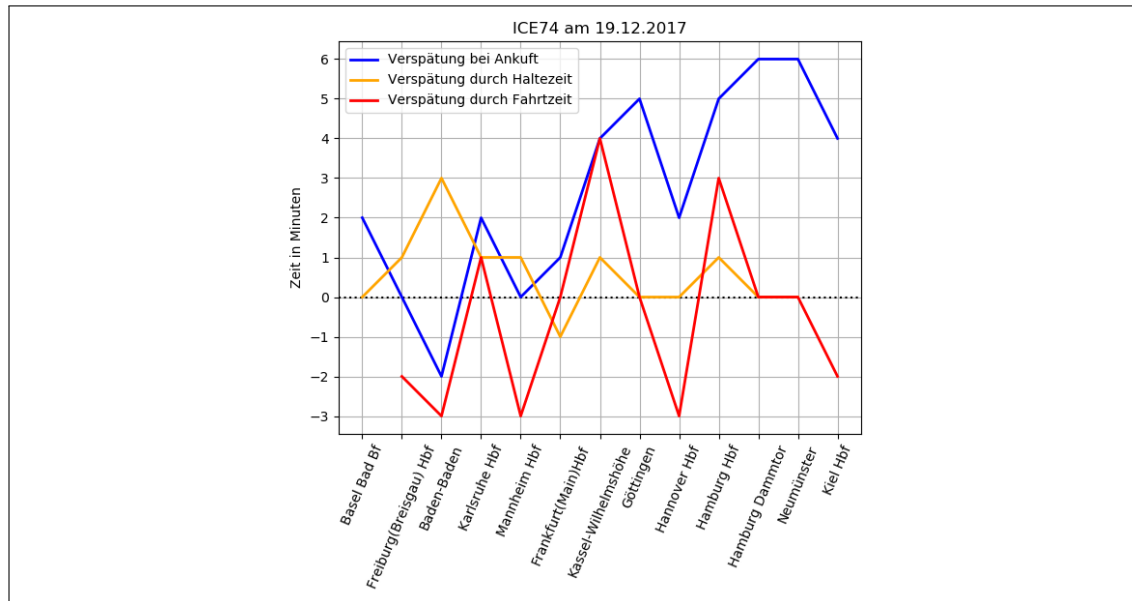


Abbildung 4.5: Verspätungsanalyse von ICE74 von Basel Bad. Bf. nach Kiel ohne Verspätung bei Abfahrt

**Zusammenhang der Verspätungen** Somit verbleiben drei Kategorien von Verspätungen: Die Verspätung durch Haltezeit, die Verspätung durch Fahrzeit und Verspätung bei Ankunft. Interessant ist hierbei immer die Verspätung bei Ankunft, da sie die Verspätung beschreibt, die wichtig für den Fahrgast ist. Hierbei kann die Verspätung bei Ankunft des nächsten Bahnhof durch die Verspätung durch Haltezeit und Fahrzeit sowie der vorherigen Verspätung bei Ankunft des vorherigen Bahnhofs beschrieben werden. Die folgende Formel stellt diese Beziehung dar:

$$\Delta an(b_n, z_m) = \Delta an(b_{n-1}, z_m) + \Delta halten(b_{n-1}, z_m) + \Delta fahren(b_{n-1}, b_n, z_m) \quad (4.9)$$

Aus diesem Zusammenhang zwischen diesen drei Verspätungsarten wird deutlich, dass die Verspätung bei Abfahrt nicht benötigt wird, um die Verspätungen, die sich bei der Deutschen Bahn ergeben, nachzuvollziehen.

### 4.5.3 Durchschnitt der Verspätungen einer Haltestelle

An einer Haltestelle können Verspätungen an einer Strecke festgemacht werden. In diesem Abschnitt werden die durchschnittliche Verspätung bei Ankunft, die durchschnittliche Verspätung bei Abfahrt und die durchschnittliche zusätzliche Standzeit der Züge in einer Haltestelle berechnet. In der ersten Version wurde für jede Berechnung der Werte alle Werte in der Datenbank verwendet. Die Verspätungen werden addiert, und später durch die Anzahl der Summanden geteilt. Diese Berechnungsweise führt allerdings zu dem Problem, dass die Berechnung sehr lange dauert. Alleine für die Berechnung mit einem Drittel der Daten dauert 6-7h. Da das nicht praktikabel ist, muss diese Funktion performater gemacht

werden.

Um dies zu erreichen, wird ein Speicher angelegt, in dem die Daten der Durchschnittsberechnung abgespeichert werden können. Dadurch ist es nicht mehr nötig, immer alle Halte einer Haltestelle abzufragen, damit die Durchschnitte berechnet werden können. Stattdessen werden die in Tabelle 4.1 gezeigten Daten in der Datenbank abgespeichert. Die Spalte ID wird lediglich als Primary Key benutzt. Die EVA Nummer wird verwendet, um die Durchschnitte den Haltestellen zuzuweisen. Die Description ID wird verwendet, um die verschiedenen Durchschnitte der Stationen zu den Strings „Ankunfts Verspätung“, „Abfahrts Verspätung“ und „zusätzliche StandZeit“ zuzuweisen. Diese sind in einer weiteren Tabelle in der Datenbank Abgelegt.

Die Ziffer eins entspricht dementsprechend dem String „Ankunfts Verspätung“, die zwei „Abfahrts Verspätung“ und die drei „zusätzliche StandZeit“. Dadurch kann die Homepage mit wenig Aufwand alle verfügbaren Durchschnitte anfordern und Verarbeiten. Sollten weitere Durchschnittsberechnungen vorhanden sein, können diese sogar ohne Anpassung der Homepage angezeigt werden. In der Spalte Average der Tabelle 4.1 wird der aktuelle Durchschnitt abgelegt, der zuletzt berechnet wurde. Die aktuelle Anzahl der verwendeten Datensätze wird durch die Spalte Count abgebildet. In Last Value wird die Letzte ID abgelegt. Diese wird verwendet, um alte Einträge der Datenbank von den neuen zu trennen.

Um die Durchschnitte zu berechnen, werden die Zustände der jeweiligen Durchschnitte aus der Datenbank abgefragt. Die Werte Average und Count werden zu der letzten Summe multipliziert.

$$Summe = Average * Count \quad (4.10)$$

Count und Last Value werden zwischengespeichert. Nun werden die Datensätze aus der Datenbank abgefragt. Um nicht alle Datensätze der Datenbank abrufen zu müssen, wird die letzte ID (Last Value) verwendet. Sie wird als Bedingung in der Datenabfrage 4.5 eingesetzt. Dadurch werden nur die Datensätze aus der Datenbank geladen, die eine höhere ID besitzen als die der letzten ID. Die im Pandas Dataframe formatierten Daten werden nun zeilenweise durchlaufen, und die jeweiligen Summen berechnet sowie Counts erhöht. Sollte die Summe und Count '0' sein, wird kein Durchschnitt berechnet. Sind dagegen Werte vorhanden, werden nacheinander die verschiedenen Durchschnitte berechnet. Die neuen Werte werden dann direkt wieder in die Datenbank geschrieben. Ein weiterer Vorteil dieser Implementierung, ist dass die Werte nicht aus einer Json oder CSV Datei eingelesen werden müssen, sondern direkt von der Homepage aus der Datenbank geladen werden können. Die Description ID gibt dabei wie oben beschrieben Aufschluss über die Art des Durchschnittes.

Mit diesen Durchschnitten haben wir herausgefunden, dass Züge an 67,46% der Haltestellen "pünktlich" kommen. Zu Spät kommen dagegen 28,43% und 4,11% kommen zu früh. Bei den abfahrenden Zügen fahren 33,68% nicht pünktlich von den Haltestellen ab. 3,60% fahren zu früh und 62,72.% pünktlich von ihrer Haltestelle ab. Wobei pünktlich im Sinne der Deutschen Bahn berechnet wird.

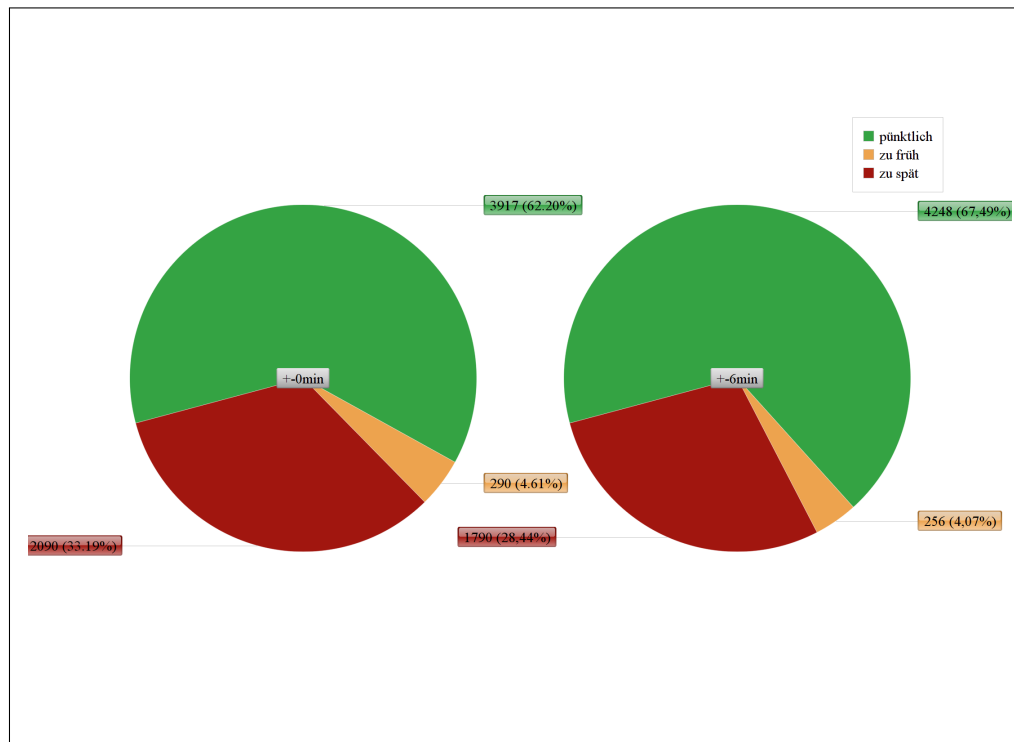


Abbildung 4.6: Prozentuale Anteile der Durchschnittlichen Verspätung bei ankommenden Zügen

Also mit 6 Minuten Toleranz. Würde man pünktlich mit  $\pm 0$  Minuten rechnen, wären 62,2% tatsächlich pünktlich. Zu früh kommen Züge im Durchschnitt nur bei 4,61%. Zu spät kommen 33,19% der Züge. Bei den Abfahrenden Zügen dagegen kommen 4,08% zu früh. 39,00% der Züge kommen zu spät und 56,91% der Züge kommen pünktlich. zu sehen ist das nochmal in Abbildung 4.6 Bei der Analyse der Standzeiten wurde allerdings herausgefunden, dass 84,33% der Züge Genau so lange in den Haltestellen stehen, wie es geplant war. Allerdings ist nicht gesagt, dass die Züge pünktlich sind. Gesamt wurden 6274 Haltestellen im Zeitraum vom 03.12.2017 bis zum 01.05.18 beobachtet und Analysiert.

```
1 SELECT zuege.arzeitist , zuege.arzeitsoll , zuege.
   dpzeitist , zuege.dpzeitsoll , `zuege`.`ID` FROM `zuege`
   `WHERE` `zuege`.`evanr` = {} AND `zuege`.`ID` > {}
   ORDER BY `zuege`.`ID` ASC
```

Quellcode 4.5: SQL Query für neue Halte einer Haltestelle

#### 4.5.4 Durchschnitt der Zeiten einer Strecke

Um die Durchschnitte der Fahrten zu berechnen, wird zu Beginn jede Strecke einmal analysiert (siehe 4.5.2). Die daraus resultierenden Daten werden dann

Name der Zelle	Datentyp
ID	INT
EVA_nummer	INT
DescriptionID	INT
Average	double
Count	INT
lastValue	INT

Tabelle 4.1: Struktur der Durchschnitts Tabelle

Bahnhof für Bahnhof summiert und danach, wie für einen Durchschnitt nötig, mit der Anzahl der Summanden geteilt. Problematisch sind vor allem Züge die frühzeitig ihre Fahrt beenden, da diese dann natürlich einige Bahnhöfe nicht anfahren. Das kann dazu führen, dass bei der Berechnung der Durchschnitte Fehler passieren, die das Ergebnis verfälschen. Um das zu umgehen, wird in jedem Datensatz geprüft, ob die Eva Nummer des Haltes auch zu der aktuell benötigten Nummer passt. Ist das nicht der Fall, wird dieser nicht berücksichtigt.

# Kapitel 5

## Datenverarbeitung mit neuronalem Netz

### 5.1 Programmierung der automatischen Datenverarbeitung

gut (naja)

In diesem gesamten Chapter/Kapitel fehlen noch Quellenangaben und weitere Literaturhinweise

Neuronale Netze benötigen zum Trainieren der Neuronen des Netzes und zum Erreichen einer hohen Genauigkeit viele Trainingsdatensätze. Diese Datensätze werden als lokale Dateien auf dem Dateisystem der Computer, welche das Netz trainieren, zwischengespeichert. Dies ist notwendig, um die Performance beim Trainieren zu erhöhen, da beim Trainieren sehr viele Aufrufe auf die Datensätze über eine Schnittstelle durchgeführt werden. Diese Aufrufe dauern bei einzelnen Datenbankabfragen deutlich länger. Dadurch wird ein schnelles Anlernen des Netzes verhindert. Alleine die Latenzzeiten eines Aufrufs übersteigt die Dauer einer lokalen Ladezeit um den Faktor 100 (20 Millisekunden Latenzzeit im Vergleich 0.2 Millisekunden SSD Lesezeit). Daher wird für die Datenverarbeitung ein Skript programmiert, welches die Daten vorverarbeitet und auf dem lokalen Dateisystem ablegt. Dabei werden auch bereits die drei verschiedenen Modi (Training, Test, Vorhersage) beachtet. Dies ist wichtig, da in den Test Datensätzen keine Trainingsdatensätze vorkommen sollten. Denn dann kann es passieren, dass die Neuronen gezielt nur diese Datensätze beachten und das Training nur auf die gegebenen Testdatensätze ausgerichtet ist. Diese Faktoren müssen bei der automatischen Datenverarbeitung beachtet werden, um das genaue Anlernen eines neuronalen Netzes erreichen zu können.

## 5.2 Vorverarbeitung der Datensätze

gut (Tabelle nicht im Text verwendet vllt noch einbauen)

Bei der automatischen Vorverarbeitung werden die Datensätze aus der Datenbank in einzelne .csv-Dateien geschrieben. Hierbei werden je nach Modus Trainings-, Test-, Vorhersagedatensätze in einer anderen Struktur generiert. Bei einem Vorhersagedatensatz werden die unbekannten Spalten mit dem Wert None aufgefüllt. Bei Trainings- und Testdatensätzen werden die abgefragten Datenbankdatensätze in zwei Gruppen aufgeteilt, da diese sich nicht überschneiden sollen. In Abbildung 5.2 ist die Verzeichnisstruktur zu sehen, diese wird bei der Vorverarbeitung automatisch angelegt. Des Weiteren ist es notwendig, die Datentypen in Zahlen zu konvertieren, da Tensorflow nur mit numerischen Typen sinnvoll umgehen kann. Die allgemeine Lösung führt zu einem Encoding der Strings als Integer oder Float Datenwert. Diese Konvertierung wird anhand von sogenannten Vocabfiles vorgenommen. Diese beinhalten alle möglichen Strings der Datensätze mit je einem String pro Zeile. Die Zeilennummer wird dabei von Tensorflow automatisch als Integerwert für den String verwendet. In Abbildung 5.3 ist ein Ausschnitt des Vocabfile für die Gleisbelegung zu sehen.



```

1 def timetotimeint(input):
2     input = str(input)
3     if input == 'None':
4         input = "24:00:00"
5     hhmss = input
6     (h, m, s) = hhmss.split(':')
7     result = int(h) * 60 + int(m)
8     result = math.floor(result/5)
9     return result
10
11 def openvocalfile(name):
12     lines = []
13     filename = "./vocabfiles/" + str(name) + ".txt"
14     with open(filename, mode="w+", encoding="utf-8") as
        file:
15         for line in file:
16             line = line.rstrip('\n')
17             lines.append(line)
18     return lines
19
20 def writevocalfile(name, vocab):
21     lines = vocab
22     filename = "./vocabfiles/" + str(name) + ".txt"
23     with open(filename, mode="w+", encoding="utf-8") as
        file:
24         for item in lines:
25             if item == "":
26                 print("No item found, dont save")
27             else:
28                 file.write("%s\n" % item)
29

```

Quellcode 5.1: Ausschnitt aus der Datei generate\_csv.py

In der Abbildung 5.1 ist ein Ausschnitt aus der Datei generate\_csv.py zu sehen. In dem Ausschnitt befinden sich die wichtigsten Funktionen zur Vorverarbeitung der Datensätze und zur Erstellung der einzelnen Vocabfiles. Aus der Datenbank werden hierbei die Daten abgerufen und für die Verwendung in Tensorflow vorbereitet. In Abbildung 5.1 wird dieser Datenfluss aufgezeigt.

## 5.3 Einrichten der Tensorflow Umgebung

gut (naja)

Bevor ein neuronales Netz mit Tensorflow realisiert werden kann, muss die Umgebung auf den jeweiligen Computern eingerichtet werden. Hier unterscheiden

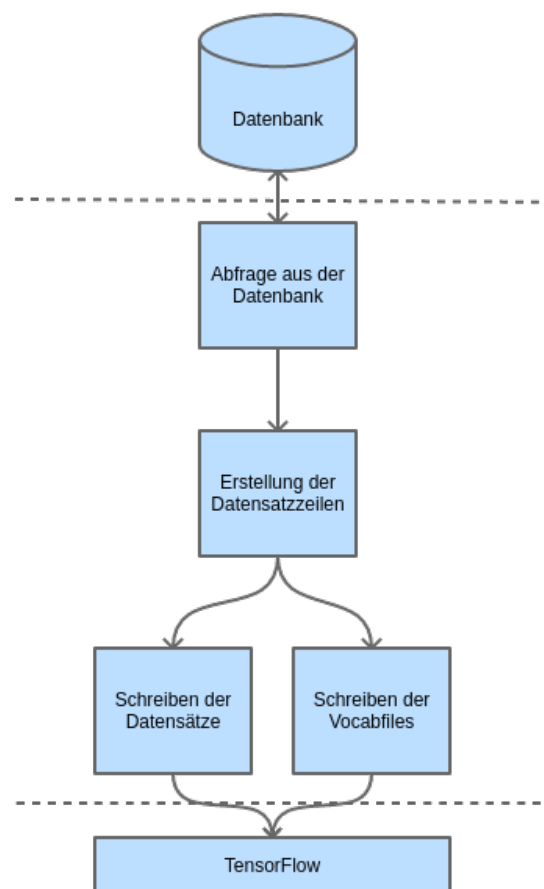


Abbildung 5.1: Datenfluss aus der Datenbank in das Tensorflow Modell

Spalte	Beispiel	Datenbank Datentyp	Konvertierter Datentyp (Python)
id	4092195	VARCHAR	nicht genutzt
zugid	-7714364757423921343-1712081222-8	VARCHAR	Aufgeteilt in drei Integer
zugverkehrstyp	F	VARCHAR	String
zugtyp	p	VARCHAR	String
zugowner	80	VARCHAR	VocabFile
zugklasse	ICE	VARCHAR	VocabFile
zugnummer	788	VARCHAR	Integer
zugnummerfull	ICE788	VARCHAR	nicht genutzt
linie	53	VARCHAR	Integer
evanr	8000152	INT	Integer
arzeitsoll	16:32:00	TIME	Integer
arzeitist	16:33:00	TIME	Integer
dpzeitsoll	16:36:00	TIME	Integer
dpzeitist	16:38:00	TIME	Integer
gleissoll	7	VARCHAR	VocabFile
gleisist	7	VARCHAR	VocabFile
datum	2017-12-08	DATE	Integer
streckengeplanthash	4d0bc383	VARCHAR	nicht genutzt
streckenchangedhash	bd84c25a	VARCHAR	nicht genutzt
zugstatus	n	VARCHAR	nicht genutzt

Tabelle 5.1: Vorverarbeitung der Datenbank-Daten

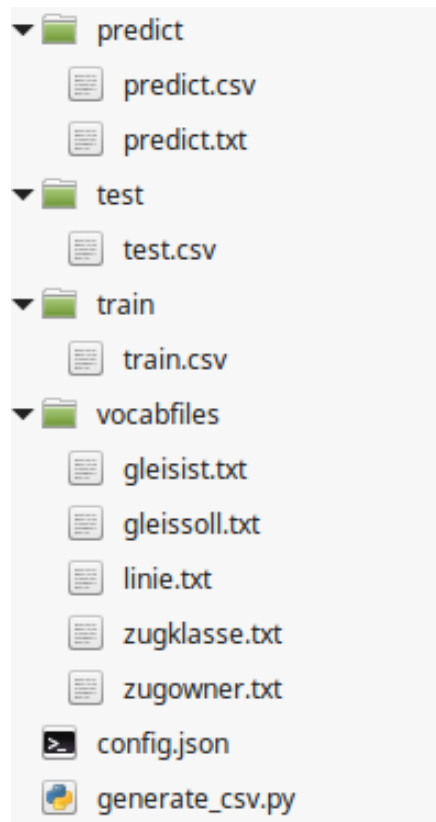
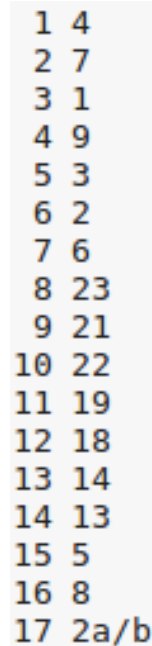


Abbildung 5.2: Der Verzeichnisbaum mit der Aufteilung der Datensätze

sich die Schritte der Einrichtung je nach Betriebssystem. Unter Windows wird die Einrichtung von Python 3.5.2+ via Installer fertiggestellt. Daraufhin wird mit PIP Installs Packages<sup>1</sup> das Paket von Tensorflow heruntergeladen und installiert. Daraufhin steht die Grundversion von Tensorflow dem Nutzer bereit. Da Tensorflow vor allem durch eine Graphics Processing Unit (GPU) beschleunigt wird, sollte bei der Verwendung als langfristige Umgebung, die GPU Unterstützung installiert werden. Dies spart vor allem Zeit und somit auch unnötigen Leerlauf beim ausprobieren eines neuen Modells. Unter Linux müssen die Schritte ebenfalls vorgenommen werden, da jedoch die Unterstützung für Linux Server bereits vorhanden ist, wird die Installation vereinfacht und benötigt mehrere Stunden weniger, im Falle einer Fehlersuche. Unter Windows gab es beim Einrichten des GPU Support unerwartete Probleme mit den Systemumgebungsvariablen, wodurch die Treiber für die Grafikkarte nicht geladen werden konnten. Da jedoch keine hilfreiche Fehlermeldung erschien, musste die Installation manuell Schritt für Schritt überprüft werden. Nach diesen Schritten kann Tensorflow mit und ohne GPU Unterstützung auf den Rechnern ausgeführt werden. Ein kurzer Vergleich zeigte, dass die Geschwindigkeit bei Berechnungen mit der Grafikkarte, je nach Modell, in etwa verzehnfacht.

<sup>1</sup>Siehe: <https://pip.pypa.io/en/stable/quickstart/>



```
1 4
2 7
3 1
4 9
5 3
6 2
7 6
8 23
9 21
10 22
11 19
12 18
13 14
14 13
15 5
16 8
17 2a/b
```

Abbildung 5.3: Ausschnitt aus einem Vocabfile, hier zu sehen Gleis (Soll)

## 5.4 Begriffsdefinitionen für ein neuronales Netz

gut (naja)

Beim Einstieg in das Themengebiet neuronale Netze fallen viele fremde Begriffe. Diese sollten vorab geklärt sein, um Missverständnisse zu vermeiden. In folgender Auflistung werden die allerwichtigsten Begriffe erklärt, weitere Begriffe und genauere Definitionen können zum Beispiel auf der Glossar Website von Google<sup>2</sup> nachgelesen werden. Dort sind sehr ausführliche und detaillierte Beschreibungen der einzelnen Begriffen zu finden.

**Feature** wird ein Attribut einer Zeile genannt, in diesem Fall zählt zum Beispiel die EVA Nummer als Feature in dem Datensatz.

**Label** wird als die Spalte des Datensatzes definiert, welche am Ende vom neuronalen Netz vorhergesagt werden soll. In unserem Fall wäre die Ankunftszeit (IST) eine solche Spalte.

**Layer** beschreibt eine Schicht von Neuronen, die Anzahl der Neuronen eines Layers wird anhand der sogenannten Hidden Units festgelegt. Diese gibt gleichzeitig die Anzahl der Layer vor. Ein Beispiel: [20,5,10] bedeutet 20 Neuronen im ersten Layer, fünf Neuronen im zweiten Layer und zehn Neuronen im dritten Layer.

<sup>2</sup>Siehe: <https://developers.google.com/machine-learning/glossary/>

**Loss** ist ein Float-Wert, welcher durch eine Funktion bestimmt wird, welche die falschen Vorhersagen gewichtet, je geringer der Loss-Wert, desto geringer die Lerndauer.

**Accuracy** ist die Genauigkeit der Vorhersagen, wenn zum Beispiel von 100 Vorhersagen 40 korrekt sind, beträgt die Genauigkeit 0,4 oder 40 Prozent.

**Optimizer** ist die Funktion, welche für die Einstellungen der Neuronen beim Training verwendet wird. Mit dieser Funktion kann auch die Lernrate angepasst werden. Wenn diese zu hoch eingestellt ist, beginnt die Loss Funktion stark zu schwingen.

**Estimator** ist die Grundlage für das Modell und beinhaltet alle Einstellungen der Parametern. Hier wird auch die Anzahl an Layern, Neuronen und der Optimizer angegeben.

**Input Function** nennt man die Funktion, welche für die Eingabe von Datensätzen im Training, Testen und Vorhersagen verwendet wird. Die Funktion liest die Datensätze auf der Festplatte ein (zum Beispiel eine .csv-Datei) und gibt zwei Tensoren zurück. Der erste Tensor beinhaltet alle Feature Spalten der Datensätze und der zweite Tensor die Label der Datensätze.

**Activation Function** ist eine Funktion, welche am Ende jedes Layer angewendet wird, um die Neuronen in dem Layer zu aktivieren. Dies kann mit einem Join in der parallelen Programmierung verglichen werden. Je nachdem welche Aktivierungsfunktion gewählt wird kann es zu verschiedenen Gewichtungen der Neuronen kommen.

**Dropout** ist ein Float Wert zwischen 0.0 und 1.0, wobei 0.0 für keine fehlenden Verbindungen zwischen den Layern der Neuronen steht und 1.0 bedeuten würde, dass es keine Verbindungen gäbe. Ein guter Wert liegt zwischen 0.0 (ein sogenanntes "fully connected neuronal network" oder 0.3). Der Dropout verhindert, dass alle Datenwerte direkt von Relevanz sind und vermeidet somit ein sogenanntes Overfitting des Modells auf die Trainingsdatensätze.

**Tensor** sind die mathematische Abbildung eines Skalars, Vektors oder einer Matrix. Als Vereinfachung können in diesem Modell die Tensoren als Matrix oder Vektor angesehen werden, welche je nach Parameter sich in ihrer Dimension unterscheiden.

**Epochs** ist die Anzahl an Epochen, welche das Modell durchlaufen soll.

**Steps** ist die Anzahl der Schritte, die pro Epoche von dem Modell trainiert werden soll. Bei einer Vorhersage wird die Schrittzahl auf die Anzahl der eingegebenen Datensätze gesetzt, beziehungsweise automatisch von Tensorflow erkannt.

## 5.5 Eingabe der Datensätze in Tensorflow

naja

Die Eingabe von Datensätzen und die Vorverarbeitung sind bei der Erstellung eines neuronalen Netzes von hoher Bedeutung. Die Zeit, eine gut funktionierende und schnelle Eingabefunktion zu schreiben, macht sich beim Trainieren des neuronalen Netzes bemerkbar. Da beim Training viele Datensätze in kurzer Zeit benötigt werden, muss ein Engpass an dieser Stelle wenn möglich vermieden werden. Bevor die Eingabefunktion geschrieben wird, müssen die Spalten der Datensätze im Modell angelegt werden. Es wird also ein Modell mit den Spalten als Variablen angelegt, in welches zu einem späteren Zeitpunkt von der Eingabefunktion echte Werte eingesetzt werden. Deshalb befinden sich in einem Modell des neuronalen Netzes auch niemals echte Datensätze sondern nur die Parameter, die durch das Trainieren erstellt wurden. Der Speicherplatzbedarf eines Modells richtet sich also indirekt über die Anzahl an Layern und deren Neuronen.

```
1 def input_fn_mode(mode):
2     filenames = ""
3     if mode == "train":
4         filenames = glob(os.path.join('./train', '*.csv'
5         ))
6     elif mode == "test":
7         filenames = glob(os.path.join('./test', '*.csv'
8         ))
9     else:
10        filenames = glob(os.path.join('./predict', '*.
11        csv'))
12    # get all filenames for datasets in this mode,
13    shuffel them
14    random.shuffle(filenames)
15    # select one file
16    filename = filenames[0]
17    # Extract lines from input files using the Dataset
18    API.
19    dataset = tf.data.TextLineDataset(filename)
20    if mode == "predict":
21        dataset = dataset.map(parse_csv ,
22        num_parallel_calls=5)
23        # do only one prediction
24        dataset = dataset.batch(1)
25    else:
26        shuffle = True
27        num_epochs = 4000
28        batch_size = 1
29        # shuffle if wanted
30        if shuffle:
31            dataset = dataset.shuffle(buffer_size=500000
32            )
33        dataset = dataset.map(parse_csv ,
34        num_parallel_calls=5)
35        # repeat for epoch count
36        dataset = dataset.repeat(num_epochs)
37        # generate batches of datasets
38        dataset = dataset.batch(batch_size)
39    return dataset
```

Quellcode 5.2: Ausschnitt von der Input Funktion aus der Datei train\_test\_predict.py

In Listing 5.2 ist die Input Funktion des neuronalen Netzes zu sehen. Durch



den Modus wird zwischen den einzelnen Eingabedateien unterschieden. Die Eingabedatei wird danach mithilfe der Tensorflow Dataset API in ein reproduzierbares Format gebracht. Desweiteren wird bei Trainingsdaten die Reihenfolge automatisch variiert, um ein fehlerhaftes Lernen des neuronalen Netzes zu verhindern. Am Ende der Funktion werden die Datensätze an das Modell zurückgegeben.

## 5.6 Eingabe- und Ausgabe-Parameter für das Neuronale Netz

Erläuterung welche Informationen in das Neuronale Netz eingegeben werden und welche Daten von dem Netz ausgegeben werden.

Endnutzereingaben: Startbahnhof Zielbahnhof Einsteige-Zeit Zugeingabe (welcher Zug genau?)

Eingabe: Zug-ID Ziel-Bahnhof Um Voraussagen treffen zu können, braucht das neuronale Netz noch zusätzliche Informationen: Strecke des Zuges? Vergangene Fahrten des Zuges und dessen Verspätung?

Zug-ID

Soll-Ankunftszeit des Zuges

Ausgabe: Voraussichtliche Verspätung in Minuten

## 5.7 Anlernen des Netzes

gut

Beim Anlernen eines neuronalen Netzes ist die Anzahl der Datensätze entscheidend, denn je mehr Daten zum Anlernen vorhanden sind, desto genauer das daraus entstehenden Modell und die erzielten Resultate. Angenommen wurde, dass die Anzahl an Datensätzen für das gewünschte Modell ausreichen wird. Da jedoch bei der Einarbeitung in Tensorflow und dessen Verwendung sehr viel Zeit geflossen ist, kann diese These nicht endgültig bestätigt oder widerlegt werden. Aus Zeitmangel beim Trainieren des neuronalen Netzes auf eigener Hardware muss auf ein Großteil der Datensätze vorerst verzichtet werden. Vorab gilt es die Genauigkeit für eine kleinere Testregion zu testen und daran das Modell verifizieren. Das diese Region nicht die Situationen in ganz Deutschland widerspiegeln kann ist im Vorhinein klar. Nichtsdestotrotz soll eine manuelle Vorhersage in kleinem Rahmen ermöglicht werden. Die Weiterentwicklung des neuronalen Netzes muss in die Zeit nach dem Abschluss verschoben werden oder von einer Gruppe Studenten aus dem nachfolgenden Jahr übernommen werden, da hierfür schlicht und ergreifend die Ressourcen zu knapp sind. Vorhersagen sind trotzdem möglich, auch wenn diese manuell stattfinden und mit dem aktuellen Modell nicht genau sind.

Beim Anlernen des neuronalen Netzes ergeben sich viele Probleme. Vor allem die hohe Anpassbarkeit des Netzes sorgt für viele Änderungen des Modells. Das

Anzahl Klassen	Intervall pro Klasse	erreichte Genauigkeit	Lerndauer (pro Modell)
24	60 Minuten	90 - 99%	3 Stunden
288	5 Minuten	35 - 75%	12 Stunden
1440	1 Minute	0 - 15%	96 Stunden

Tabelle 5.2: Vergleich der Lerndauer in Bezug auf die Anzahl der Klassen

Problem liegt hierbei bei der benötigten Lerndauer des gewählten Modells, da bei jeder vorgenommenen Änderung des Netzes die komplette Lerndauer erneut benötigt wird, um einen Vergleich zwischen den Modellen aufstellen zu können. Da es je nach Parametern zu unterschiedlichen Ergebnissen kommt muss eine Spanne an Eingabeparametern für jedes Modell überprüft werden. Da das Anlernen die Ressourcen eines Computers nahezu ausschöpfen, kann in dieser Zeit keine weitere Arbeit am Computer daran vorgenommen werden. Das Hauptproblem besteht daher in der Unmöglichkeit vorab zu wissen, welche der Parameter am Ende für ein genaues Vorhersagemodell relevant sind. Des weiteren kann es passieren, dass nach einer längeren Lerndauer das neuronale Netz plötzlich genauer wird. Als Beispiel zeigt die Abbildung 5.4 die Simulation einer Klassifizierung im Internetbrowser<sup>3</sup>, wie der Verlauf der Loss Funktion und daher der inversen Genauigkeit aussehen kann.

## 5.8 Verifizieren des Netzes

gut

Ein grundlegendes Problem bei neuronalen Netzen ist die Ungewissheit, ob die angenommenen Parameter des Netzes überhaupt zu einem passenden Ergebnis führen können. Als Beispiel dient die Klassifizierung in 1441 Klassen. Hierbei wurde bei den Tests eine sehr niedrige Genauigkeit festgestellt. Dies hat jedoch nicht direkt etwas zu bedeuten, denn es könnte sein, dass durch die hohe Anzahl an Klassen die Lerndauer der Neuronen exponentiell ansteigt. Dies ist im Vergleich zu den vereinfachten 24 Klassen deutlich zu sehen, denn dort kann das neuronale Netz innerhalb einer Stunde bereits über 90% Genauigkeit erreichen, sofern die passenden Parameter gewählt wurden. Dieses Problem ist vor allem kritisch, wenn noch keine Erfahrungen gemacht wurden, welche Parameter entscheidend für eine korrekte Vorhersage sind, da die Lerndauer bei 288 Klassen auf weit über sechs Stunden ansteigt und das zuvor erprobte Netz nur noch eine Genauigkeit von 35-45% erreicht hat. Alles in allem lässt sich aus diesen Tests die Erkenntnis gewinnen, dass ein neuronales Netz bei mehr Klassen nicht nur länger braucht um genauere Ergebnisse zu liefern, sondern auch deutlich mehr Trainings- und Testdatensätze benötigt.

<sup>3</sup>Siehe: <https://playground.tensorflow.org/>

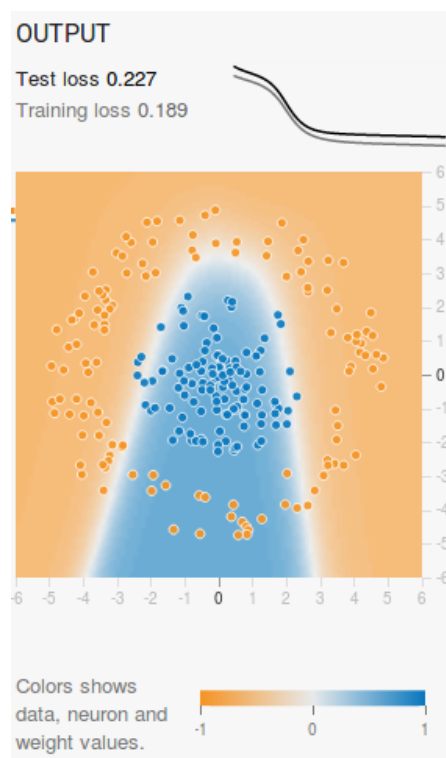


Abbildung 5.4: Verlauf der Loss Funktion auf der Website des Tensorflow Playground

## 5.9 Vorhersagen anhand des Netzes

gut

Die Vorhersage mit neuronalen Netzen unterliegen einer grundlegenden Struktur. Durch die Input Funktion werden die bekannten Größen des Modells an Tensorflow übergeben. Dort wird die Vorhersage durchgeführt und liefert einen Tensor als Antwort zurück. In diesem Falle besitzt der Tensor jeweils 24, 288 oder 1441 Klassen, welche jeweils eine Uhrzeit oder einen Zeitintervall von einer Stunde oder fünf Minuten darstellt. Jeder Uhrzeit wird über eine Softmax Funktion eine relative Wahrscheinlichkeit zugeordnet. Dies bedeutet, dass die Summe aller Klassen gleich 100% entsprechen. Ein erwartetes Ergebnis einer Vorhersage wäre also eine Normalverteilung über einen bestimmten Wert. Das würde zum Beispiel bedeuten, dass ein Zug mit der Wahrscheinlichkeit 75% genau zu dieser Zeit kommt, oder mit 95% Wahrscheinlichkeit in einer Zeitspanne von fünf Minuten um diesen Wert. Je nachdem, wie genau das Modell die Realität vorhersagen kann, kann diese Kurve schmaler werden, wodurch die Wahrscheinlichkeit einer genaueren Vorhersage größer ist. Da durch die fehlende Lerndauer und das nicht optimale neuronale Netz keine hohe Genauigkeit beim Trainieren erreicht werden konnte, ist das Ergebnis bei der Vorhersage mit Tensorflow sehr ungenau. Ein Problem hierbei ist die nicht optimierte Loss Funktion, welche das Anlernen von Datumswechseln nicht berücksichtigt. Des Weiteren sollte eine vorbereitende Gewichtung der einzelnen Spalten in den Datensätzen vorgenommen werden vor der Eingabe in das neuronale Netz vorgenommen werden. In Abbildung 5.5 ist eine Vorhersage mit 288 Klassen zu sehen, jede falsche Abweichung in der Vorhersage bedeutet also direkt fünf Minuten Fehler für den Nutzer. Das direkt aus der Abbildung erkennbare Problem ist das Rauschen der Vorhersage um mehrere zeitlich weit auseinander liegenden Klassen. Das Rauschen kommt durch die mangelnde Lerndauer und die daraus resultierende Unsicherheit der Vorhersage. Als Vergleich kann Abbildung 5.6 herangezogen werden, dieses Modell nutzt jedoch nur die Ankunftszeit ohne die Information über die Zugklassen oder den betreffenden Bahnhof. Dort kann nach etwa drei Stunden Lerndauer die erwartete Vorhersage mit einer Genauigkeit von 50-75% erreicht werden. Da jedoch durch diese Vereinfachung der Sinn hinter der Vorhersage verloren geht, ist dieses Ergebnis im Endeffekt nicht nützlich.

## 5.10 Bewertung der Ergebnisse

gut

Die Bewertung der erzielten Vorhersagen muss, je nach Modell und Aufbau betrachtet werden. Als Hauptmerkmal diene die Anzahl der Klassen und Neuronen, so benötigt ein Modell mit vielen Klassen deutlich mehr Neuronen und damit Lerndauer, um die gleichen Ergebnisse zu erzielen. Außerdem sind die verwendeten

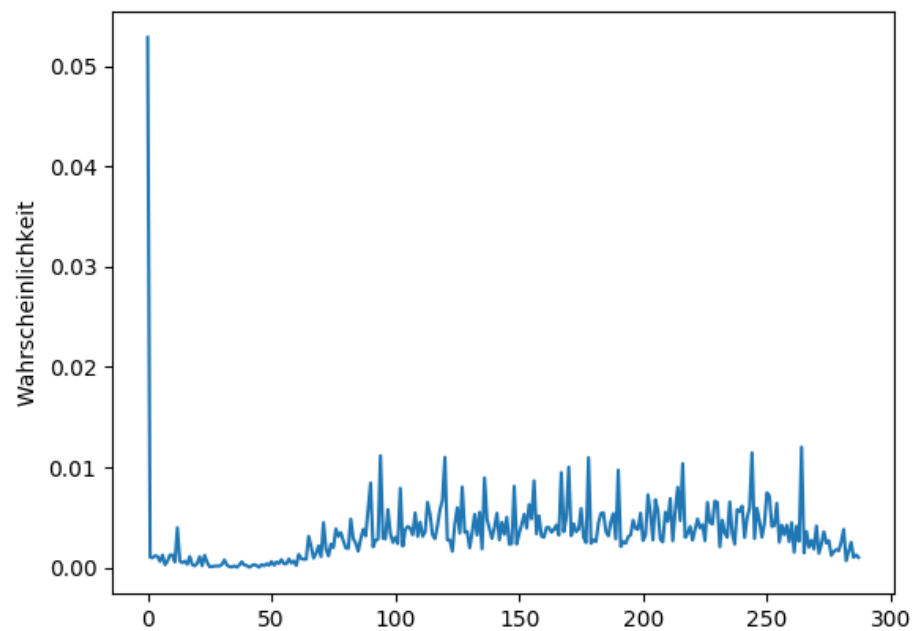


Abbildung 5.5: Verteilung der Wahrscheinlichkeiten bei 288 Klassen und allen Spalten als Eingabeparameter

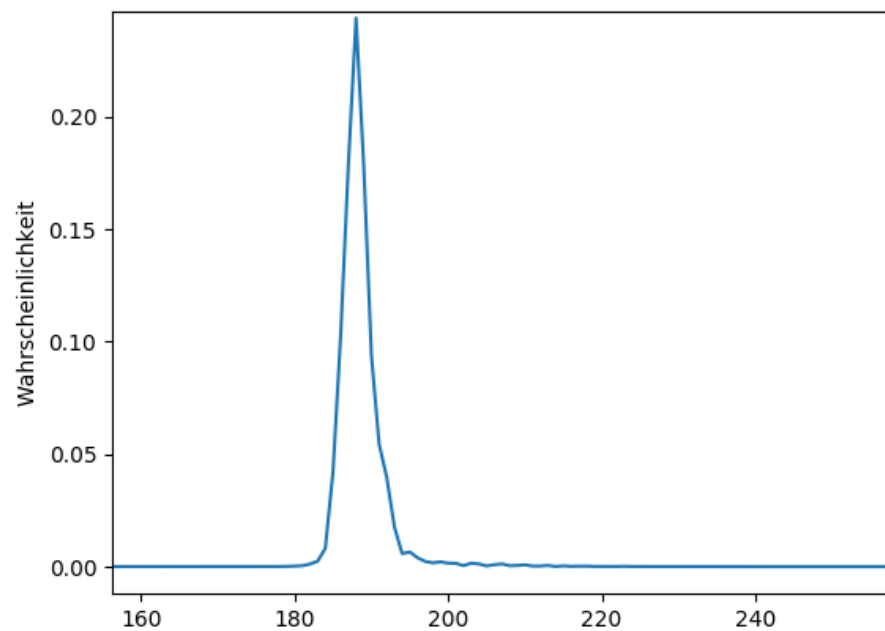


Abbildung 5.6: Verteilung der Wahrscheinlichkeiten bei 288 Klassen mit nur der Ankunftszeit als Eingabeparameter

Eingabeparameter von hoher Bedeutung für die Lerndauer und die Genauigkeit. Je mehr Datenspalten in das Modell übergeben werden, desto länger dauert ein Einschwingen auf die gewünschte Vorhersage. Als kurzes Fazit zum Thema Tensorflow kann gesagt werden, dass das Vorhaben die Vorhersage genau zu machen leider kein Erfolg hatte. Jedoch konnte die Vorverarbeitung und eine Grundoptimierung vorgenommen werden. Dies ist in Anbetracht der Zeit und der Komplexität des Modells bereits als ein Teilerfolg zu bewerten. Die ohnehin optionale Prognose wird zum Ende der Studienarbeit aufgrund von Zeitmangel nicht vollständig fertiggestellt.

# Kapitel 6

## Bereitstellung der Daten im Internet

### 6.1 Aufbau der Website

In diesem gesamten Kapitel fehlen noch Quellenangaben und weitere Literaturhinweise

Heutzutage ist die Bereitstellung einer Website eine einfache Methode Daten mit anderen Menschen zu teilen. Da die Datenbank und Website einen gewissen Sicherheitsstandard erfüllen soll, wird sich für ein Framework entschieden, welches bereits integrierte Sicherheitsfunktionen bietet. Der Name des Frameworks lautet Laravel<sup>1</sup>. Dies spart vor allem Zeit bei der Entwicklung der neuen Funktionen für die Bereitstellung der einzelnen Webviews. Ein View ist eine Seite oder der Teil einer Website, welcher in eine weitere Seite eingebettet sein kann. In Abbildung 6.1 kann die Struktur der einzelnen Ansichten erkannt werden.

Auf der Website gibt es folgende Hauptpunkte, welche jedem Nutzer zur Verfügung stehen. Diese Punkte stellen wie geschrieben nur die Hauptebene dar. Jeder dieser Punkte besitzt entsprechende Unterpunkte mit genaueren Analysen und Statistiken.

in anderen Kapiteln paragraph anstatt item, formatierung schöner

**Übersicht** ist die Startseite der Nutzer. Hier sollen Grundinformationen an die Nutzer gegeben werden, wie zum Beispiel die Anzahl an Datensätzen (gesamt). Diese Seite soll optimiert sein schnell zu laden, weshalb sie relativ wenig Daten an den Nutzer senden soll. Dies ist vor allem in Anbetracht der mobilen Nutzung der Website wichtig.

**Karte** ist eine Karte, welche als Basisoberfläche Kartenmaterial von OpenStreet-Maps<sup>2</sup> verwendet. Darauf werden mithilfe von Leaflet<sup>3</sup> einzelne Schichten

---

<sup>1</sup>Siehe: <https://laravel.com/>

<sup>2</sup>Siehe: <https://www.openstreetmap.org/>

<sup>3</sup>Siehe: <https://leafletjs.com/>





gezeichnet, wie zum Beispiel die Bahnhöfe der Deutschen Bahn.

**Bahnhöfe** ist die Hauptansicht für Statisten der einzelnen Stationen. Auf der Hauptseite befindet sich eine Suchfunktion mit grundlegenden Einstellungen. Nach erfolgreicher Suche nach einem Bahnhof kann sich der Nutzer eine der vielen erzeugten Ansichten anschauen.

**Züge** ist die Übersicht und Suche über die vorhandenen Züge in der Datenbank. Nach erfolgreicher Suche kann der Nutzer sich die einzelnen Statistiken zu dem ausgewählten Zug anschauen.

**Impressum** ist eine Verlinkung auf das nach deutschem Recht benötigte Impressum einer Website gemäß § 5 Telemediengesetz (TMG)

## 6.2 Erstellung der Webrouen

Mittlerweile gibt es in beinahe allen Webframeworks eine native Unterstützung für Restful basierte Routen. In Laravel werden hier die Routen nochmals in vier Kategorien je nach Anwendungsfeld aufgeteilt. Diese Routen sind nach deren Zuständigkeit benannt und heißen `api`, `channels`, `console` und `web`. Im Normalfall reichen die Webrouen für das Vorhaben aus, falls es eine komplette API für alle Datensätze geben soll, kann diese über die API Routen definiert werden. Der Aufbau einer Webroute ist relativ simpel, wie in Listing 6.1 zu sehen ist. Zuerst wird die Route ausgehend vom Startpunkt der Webseite angegeben. In der Route können Parameter mit geschweiften Klammern als Platzhalter dargestellt werden. So ist es möglich Routen für alle Stationen anzulegen, ohne diese einzeln programmieren zu müssen. In der Route wird dann der Parameter aus der URL genommen und anhand dessen der Inhalt der entsprechenden Seite angezeigt.

Beim Erstellen der Routen gibt es jedoch auch Fallstricke, die zu Beginn nicht direkt erkennbar sind. So muss zum Beispiel die längste Route zuerst angegeben werden, da die Routen nach dem First Match Prinzip abgearbeitet werden. Sollte unter der Hauptroute noch eine Subroute mit Parameter stehen, wird trotz Parameter nur die Hauptroute angezeigt.

```
1 Route::get('/station/{id}/timetable/{date}', '
    StationController@timetable')->name('station.
    detaildate');
```

Quellcode 6.1: Beispiel einer Webroutendefinition

## 6.3 Erstellung der Seiten

Bei der Erstellung der einzelnen Ansichten der Website wird zuvor eine grundlegende Strukturierung anhand von Mockups erstellt. Diese dienen dazu schnell

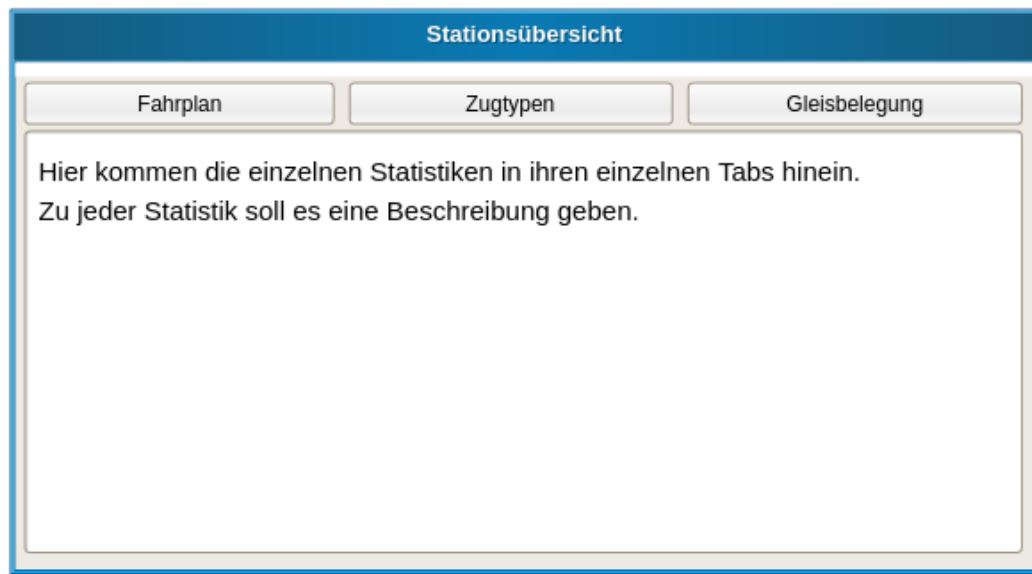


Abbildung 6.2: Mockup der Stationsübersicht

Änderungen vorzunehmen und diese anschließend nach verschiedenen Faktoren wie Ordnung und verständliche Anordnungen zu bewerten. Ein Mockup der Stationsseite ist in Abbildung 6.2 zu sehen. Dort wird bereits zu Beginn auf die verschiedenen Subseiten geachtet. Diese sollen die Datenmenge in für die Nutzer besser verständliche kleinere Teile aufspalten und ordnen. Des Weiteren gilt es zu beachten, dass durch die Struktur von Templates in Laravel eine einheitliche Ansicht für alle Stationen gegeben ist. Nur der Inhalt der Seiten unterscheidet sich von Station zu Station. Ein weiterer Vorteil ist die Nutzung von Bootstrap. Dieses Webframework nutzt CSS und Javascript, um je nach Webbrowser und Auflösung die Website trotzdem anschaulich darzustellen. So soll die Website auf dem Smartphone ohne spezielle App genauso gut benutzbar sein, wie auf dem heimischen Computer der Nutzer. Dabei ist die Ladedauer und die Größe der ausgelieferten Webseiten bereits beachtet. Die Größe ist immernoch von Relevanz, da die Nutzer noch mit geringen Bandbreiten auf Edge oder GPRS Geschwindigkeiten unterwegs sein können. In Tabelle 6.1 ist ein Vergleich der Ladezeit zwischen zwei Webseiten aufgezeigt. Da es in Deutschland immer noch viele Regionen gibt, in welchen kein schnelles mobiles Internet verfügbar ist, kann man in der ersten Zeile der Tabelle den gewaltigen Unterschied in der Ladezeit gut erkennen. Des Weiteren gibt es entlang der Bahnstrecken einige Funklöcher, welche einen Verbindungsabbruch zur Folge haben, sollte die Website nicht bereits geladen sein.

### 6.3.1 Idee des dynamischen Nachladen

Bei der Erstellung der Übersichten und Statistiken für die Züge und die Stationen, sollen immer nur die dem Nutzer sichtbaren Elemente erstellt und geladen werden.

Netzwerkgeschwindigkeit	Website (600 Kilobyte)	Webseite (2,4 Megabyte)
0,3 MBit/s [EDGE]	00:49	03:16
7,2 MBit/s [HSPA]	00:01	00:02
21 MBit/s [HSPA+]	00:01	00:01
100 MBit/s [LTE]	00:01	00:01

Tabelle 6.1: Vergleich der Ladezeit zweier Webseiten (Angaben in mm:ss)

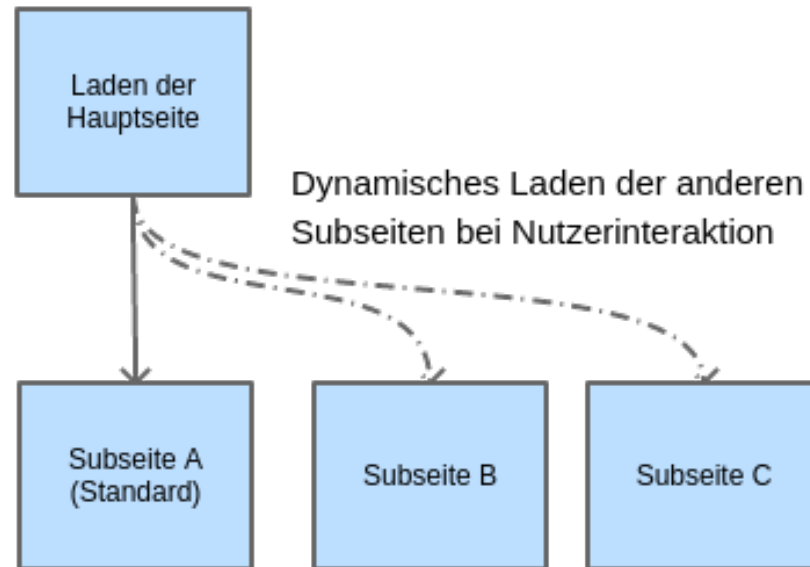


Abbildung 6.3: Dynamisches Nachladen der Subseiten

Dies spart Ressourcen auf dem Server und gleichzeitig Bandbreite beim Nutzer. Des Weiteren ist die Webseite dadurch deutlich schneller, da die Menge an Quellcode im Hintergrund besser aufgeteilt wird. Diese Unterteilung der Statistiken sorgt also nicht nur für eine bessere Übersichtlichkeit, sondern auch für einen schnelleren Seitenaufbau beim Nutzer. In Abbildung 6.3 wird das Laden der Subressourcen aufgezeigt. Dieses erfolgt via Javascript Code, welche die nicht sichtbaren Elemente gleichzeitig im Hintergrund aus dem DOM entfernt.

### 6.3.2 Die Stationsübersicht

Auf der Seite der Stationsübersicht wird dem Nutzer eine Übersicht über die vorher ausgewählte Station angezeigt. Da es viele verschiedene Statistiken gibt, wird die Navigation durch Tabs realisiert. Die folgenden Elemente sind auf der Stationsübersicht wählbar:

**Fahrplan** Hier wird dem Nutzer der aktuelle Fahrplan des Tages präsentiert. Der Nutzer kann ebenfalls ein Datum wählen, welches dann den Fahrplan des

gewählten Tages zeigt. Zu jedem Zug im Fahrplan gibt es einen Querverweis auf die Zugübersicht des gewählten Zuges.

**Zugtypen** Hier kann der Nutzer die Anzahl der verschiedenen Zugtypen sehen, welche an der gewählten Haltestelle abgefahren sind.

**Gleisbelegung** Hier sieht der Nutzer die Anzahl der Zugtypen pro Gleis, so kann man erkennen, das gewisse Gleise nur von S-Bahnen oder für den Fernverkehr genutzt werden.

### 6.3.3 Die Zugübersicht

Auf der Seite der Zugübersicht werden dem Nutzer allerhand Informationen zu dem ausgewählten Zug angezeigt. Um die Informationen besser zu ordnen wird eine Navigation mit Tabs erstellt, welche die folgenden Elemente enthält:

**Haltestellen** Hier wird dem Nutzer die Route des Zuges angezeigt. Gleichzeitig gibt es einen Querverweis auf die angefahrenen Haltestellen, um deren Statistiken anzuschauen.

**Verspätung** Hier wird dem Nutzer eine Statistik zur Verspätung über den Verlauf der Strecke angezeigt. So sollen etwaige Engpässe aufgedeckt und erkennbar werden. Des weiteren wird der Verlauf über die letzten zwei Wochen angezeigt, um eventuelle Tendenzen zu erkennen.

**Ausfallstatistik** Hier soll dem Nutzer eine Statistik zur Ausfallwahrscheinlichkeit angezeigt werden.

**Gleiswechsel** Hier kann der Nutzer sehen, ob der Zug in einem Bahnhof häufiger von einem anderen Gleis als dem Sollgleis abfährt.

**Streckenwechsel** Hier kann der Nutzer die verschiedenen Strecken sehen, im Falle eine Umleitung in der Vergangenheit

## 6.4 Testen der Seiten mit Unit Test

Mit einer steigende Komplexität der Website wird das manuelle Testen immer aufwändiger. Um bestehende Seiten auf Fehler durch eine Änderung schnell zu überprüfen werden Unit Tests eingesetzt. Neben den Unit Tests werden Integration Tests durchgeführt, um ein fehlerfreies Zusammenarbeiten der Komponenten als Gesamtes sicherzustellen. Als einfachste Testart lässt sich eine Überprüfung von HTTP Status Codes realisieren. So kann geprüft werden, ob eine Route den Code 200 (OK) oder eine Fehlercode zurück gibt. Im Falle eines internen Fehlers in Laravel, wird dem Nutzer eine benutzerdefinierte Fehlerseite angezeigt. Diese wird mit dem HTTP Code 500 (Internal Server Error) an den Nutzer gesendet. In der

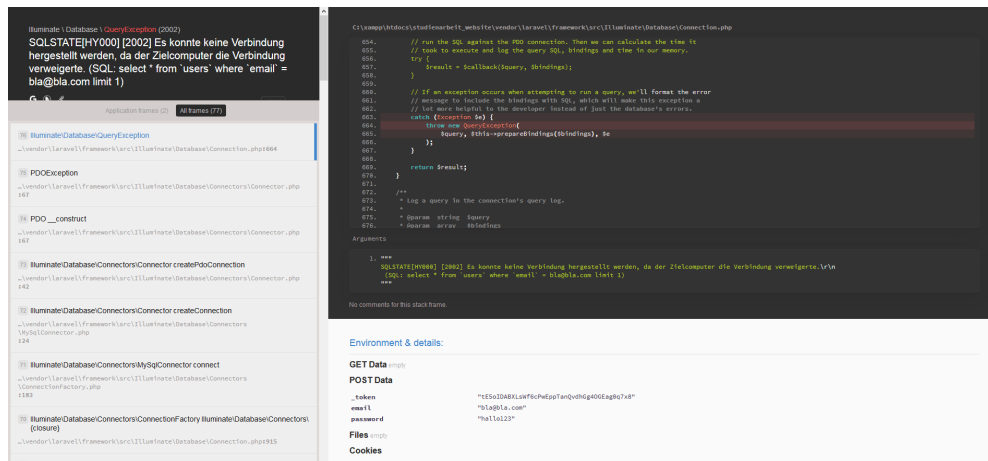


Abbildung 6.4: Ausgabe der Website im Debug Modus im Fehlerfall

Whoops, looks like something went wrong.

Abbildung 6.5: Ausgabe der Website im Production Modus im Fehlerfall

Entwicklungsumgebung werden Debug Informationen dem Entwickler ausgegeben. Diese werden im Produktiveinsatz aus Gründen der Sicherheit nicht an die Nutzer ausgegeben. In Abbildung 6.4 ist eine Fehlermeldung aus Entwicklersicht zu sehen. Der Fehler wird zuvor bereits von einem Integrationstest erkannt und in einer Logdatei mit weiteren Details vermerkt. Diese Logdatei kann durch das Ausführen des Testframeworks angezeigt werden. Die Ausgabe auf der Website ist in Abbildung 6.5 zu sehen. Die Vorteile von automatischen Tests ist die schnelle Erkennung, ob eine Änderung im Quellcode ungewollte Effekte verursacht. Die Abbildungen 6.6 und 6.7 zeigen einen Durchlauf des Unittests mit phpunit. Dort können die fehlgeschlagenen Tests direkt ausgewertet und gefunden werden.

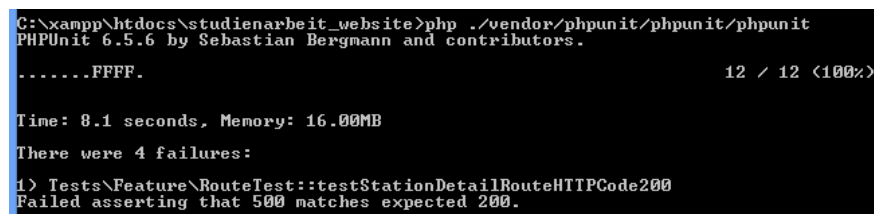


Abbildung 6.6: Ausgabe eines Unit Tests (oberer Teil)



```
FAILURES!  
Tests: 12, Assertions: 12, Failures: 4.
```

Abbildung 6.7: Ausgabe eines Unit Tests (unterer Teil)

## 6.5 Visualisierung der Datensätze

Dimensionen der Daten, ORT; ZEIT; NAME; STRECKE; etc.

Absätze setzen für bessere Lesbarkeit hier und im ff

Zur Visualisierung der Datensätze werden vorgerechnete und Echtzeit Daten verwendet. Je nachdem, ob dem Nutzer eine Interaktionsmöglichkeit gegeben werden soll, wird entschieden, welche Software zum Einsatz kommen soll. Die dafür verwendeten Tools stammen aus dem Python Modul `matplotlib`<sup>4</sup> oder aus der Javascript Library `d3js`<sup>5</sup> beziehungsweise `c3js`<sup>6</sup>. Die Anzahl der Dimensionen der Datensätzen macht es schwierig zu entscheiden, ob eine Ansicht oder Grafik für diese überhaupt Sinn ergibt. Daher werden zu Beginn der Visualisierungsprozesses verschiedene Techniken ausprobiert. Alle Skripts werden vor dem Hintergrund der Wiederverwendbarkeit geschrieben. Ein wichtiger Schritt von der Datenbank zur fertigen Grafik ist die SQL-Abfrage. Diese soll optimiert sein, um den Datenbankserver nicht unnötig zu belasten. Zur Optimierung kann das MySQL Schlüsselwort `EXPLAIN`<sup>7</sup> verwendet werden. Nachdem die Abfrage ausgeführt ist, wird die Antwort als Liste von Objekten abgespeichert. In der Art des Objektes gibt es grundlegende Unterschiede, je nachdem, ob das Objekt weitere Methoden enthält, oder ob das Objekt als simple Datensammlung darstellt.

Um die Datenmenge für den Nutzer zu verringern, wird die Datenaufbreitung serverseitig durchgeführt. Der Nutzer bekommt daraufhin vorverarbeitete Datensätze, welche als simples Datenformat oder JSON in eine Grafik eingebettet werden. Die Formate unterscheiden sich abhängig von den anzuzeigenden Statistiken. Für die einfachere Wiederverwendbarkeit wird ein Controller für die Datenaufbereitung entwickelt. Dieser wird `GraphController.php` genannt und soll intern die Datenvorverarbeitung im Webserver übernehmen. Je nachdem wie die Python Skripts aufgebaut sind, können sie entweder vom Nutzer per WebCGI, oder durch die Backendschnittstelle des Laravel Frameworks ausgeführt werden. Ein Vorteil bei der Ausführung durch Laravel ist der Cache<sup>8</sup>, welche in diesem Fall problemlos mit anderen Nutzern geteilt werden kann, da keine persönlichen Nutzerdaten darin enthalten sind.

Da die Website komplett dynamisch generiert wird, müssen alle statischen Querverweise durch die in Laravel vorgesehenen Routen ersetzt werden. Diese

<sup>4</sup>Siehe <https://matplotlib.org/>

<sup>5</sup>Siehe <https://d3js.org/>

<sup>6</sup>Siehe <https://c3js.org/>

<sup>7</sup>Siehe: <https://dev.mysql.com/doc/refman/8.0/en/explain.html>

<sup>8</sup>Siehe <https://laravel.com/docs/5.6/cache>

```
HTML: Klassisch statische URL
<a href="http://www.example.org">Visit example.org</a>
Laravel: Route dynamisch eingesetzt
<a href="{ route('station.index') }">Haltestellen</a>
```

Abbildung 6.8: Vergleich von statischer URL zur Laravel Routen Engine

im ersten Moment etwas ungewohnte Programmierung hat den entscheidenden Vorteil, dass die Verlinkung durch das Framework vorgenommen wird und nur noch die reale Adresse und nicht die Datei abgeändert werden muss.

Für jede Haltestelle sollen den Nutzern verschiedene Statistiken zu Verfügung gestellt werden. Hierzu ist es notwendig sich Gedanken über die möglichen relevanten Themen zu machen. Eine interessante Betrachtung ist zum Beispiel die Verteilung von verschiedenen Zugklassen (ICE, RB, ...) auf die im Bahnhof vorhandenen Gleise. Am Beispiel Wiesloch-Walldorf in Abbildung 6.9 kann man erkennen, dass das Gleis 3 nicht für den Fernverkehr verwendet wird. Gleichzeitig kann die Verteilung der Zugklassen pro Gleis relativ zueinander erkannt werden. So gibt es Gleise welche hauptsächlich vom Fernverkehr bedient werden und Gleise, die häufig für S-Bahnen benutzt werden. Das sich daraus weitere Informationen gewinnen lassen, ist deutlich beim Berliner Ostbahnhof in Abbildung 6.10 zu erkennen. Dort fahren die S-Bahnen auf den ausgebauten Gleisen, die über eine Stromschiene verfügen. Züge in der Nacht wie der NightJet verkehren dabei hauptsächlich auf den Gleisen 1 bis 3. Als besondere Herausforderung beim Programmieren der Anzeige der Statistik kann das noch relativ unbekannte Framework c3js gesehen werden. Die Datensätze aus der Datenbank müssen bevor sie an den Nutzer gesendet werden als JSON formatiert werden. Diese Aufgabe übernimmt der GraphController des Backends. Dieser liefert für die verschiedenen Statistiken die jeweiligen Ausgaben als JSON. Eine Problematik kann die Begrenzung des PHP Memory Limits sein, da bei großen Datenabfragen dieses leicht überschritten werden kann. Weitere Probleme treten in Verbindung mit Offset Bugs auf, diese sind durch die von extern kommende Programmteile vorprogrammiert. Oftmals ist ein Index eines Gleises nicht sichtbar, da die Ausführung der Javascript Funktion einen Index zu früh aufhört und somit den letzten Datensatz verschluckt. Dieses Problem kann mithilfe von weiteren Datensatzes am Ende behoben werden, diese werden, da die Zugklasse auf NONE gesetzt ist nicht im Frontend angezeigt. Diese Funktion des Backends für die Gleisbelegungsstatistik wird in Quellcode Listing 6.2 dargestellt. Die darin verwendete MySQL Abfrage ist belastet den Server kaum, da dieser alle Einträge der Station durch eine vorherige Query bereits zwischengespeichert hat und nur eine neue Aggregatfunktion über diesen Zwischenspeicher laufen lassen. Der auf die Abfrage folgende Quellcode sorgt für eine Formatierung der Datensätze in einem für c3js günstigen Ausgangsformat. Die generierte JSON Datei ist exemplarisch in Abbildung 6.11 dargestellt.

Um die Daten in der JSON nicht dauerhaft erneut generieren zu müssen, wird der in Laravel bereits integrierte Cache benutzt. Dieser ist sehr mächtig und



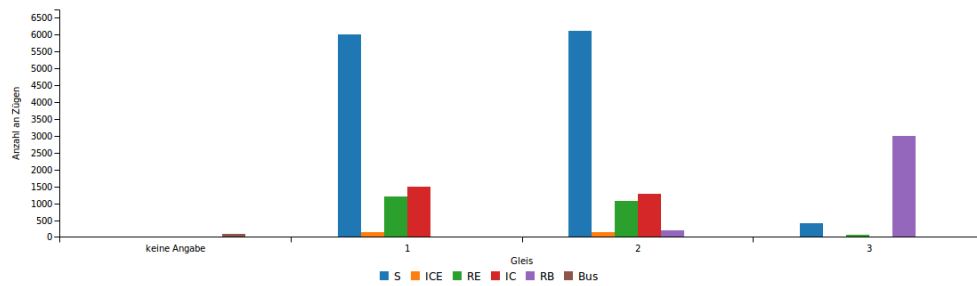


Abbildung 6.9: Gleisbelegung am Beispiel Wiesloch-Walldorf

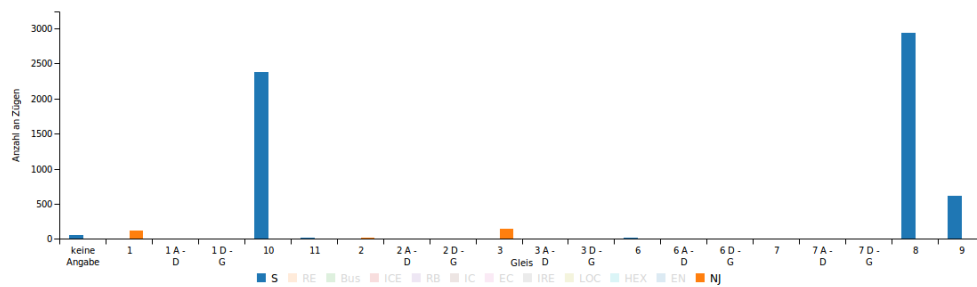


Abbildung 6.10: Gleisbelegung in Berlin Ostbahnhof (ausgewählte Zugtypen S-Bahn und NightJet)

verfügt über verschiedene Routinen, welche diverse Speichermethoden des Caches unterstützten. Entschieden wurde sich für einen Dateibasierten Cache ohne weitere Software. Dieser ist in der Regel ausreichend schnell und wird vom Webserver im Arbeitsspeicher gehalten. Die Ladezeiten einer Station nachdem diese im Cache vorhanden ist, fällt von über 250 Millisekunden auf unter fünf Millisekunden. Dies entspricht dem Faktor 50. Gerade bei den Daten der Stationen ist ein Cache ohne größere Probleme umsetzbar. Da der Miner jede Station maximal einmal die Stunde aufruft, werden dem Nutzer auch keine Daten längerfristig vorenthalten. Zudem verändern sich die bereits gespeicherten Datensätze nicht mehr. Ein Nachteil des Caches ist bei der Entwicklung ebenfalls nicht zu merken, da hier die Konfigurationsdatei auf geringer oder gar keine Cachenutzung global eingestellt werden kann.

```
[{"name": "keine Angabe", "Bus": 82}, {"name": "keine Angabe", "IC": 13}, {"name": "keine Angabe", "RB": 5}, {"name": "keine Angabe", "S": 1}, {"name": "1", "IC": 1484}, {"name": "1", "ICE": 153}, {"name": "1", "RE": 1210}, {"name": "1", "S": 6007}, {"name": "2", "IC": 1269}, {"name": "2", "ICE": 152}, {"name": "2", "RB": 210}, {"name": "2", "RE": 1078}, {"name": "2", "S": 6124}, {"name": "3", "IC": 19}, {"name": "3", "ICE": 3}, {"name": "3", "RB": 2986}, {"name": "3", "RE": 57}, {"name": "3", "S": 403}]
```

Abbildung 6.11: Ausgegebenes JSON an den Webbrowser bei Aufruf der Route

```

1  public function getTrainclassPerPlatformStatistic($
    evanr)
2  {
3      $stats = Cache::remember('
        getTrainclassPerPlatformStatistic'.$evanr, 240,
        function() use ($evanr){
4          $statsraw = DB::connection('mysql2')->select("
            SELECT Count(id) as anzahl, gleisist,
            zugklasse FROM k42174_bahnapi.zuege where
            evanr=:evanr group by gleisist, zugklasse
            limit 10000", ['evanr' => $evanr]);
5          $stats = array();
6          $savelastgleis = "";
7          $savelastzugklasse = "";
8          foreach ($statsraw as $zuginfo)
9          {
10             if ($zuginfo->gleisist == NULL) {
11                 $zuginfo->gleisist = 'keine_Angabe';
12             }
13             $stats[] = array("name"=>$zuginfo->gleisist, $
                zuginfo->zugklasse=>$zuginfo->anzahl);
14             $savelastgleis = $zuginfo->gleisist;
15             $savelastzugklasse = $zuginfo->zugklasse;
16         }
17         return Response::json($stats);
18     });
19     return $stats;
20 }
```

Quellcode 6.2: Funktion zur Auswertung der Gleisbelegung eines Bahnhofs

würde für diese abbildung eine andere Farbe für den Text nehmen das Lila finde ich irgendwie komisch zu lesen

Eine weitere interessante Statistik könnte die Verspätung eines Zuges an verschiedenen Tagen sein. So kommt ein Zug zum Beispiel chronisch zu spät oder es gibt häufig ungewollte Gleiswechsel eines Zuges. Vor allem Muster sollten am Ende von den Nutzern erkannt werden können. Die Beziehungen und Muster von Zügen untereinander ist auch bei einer Vorhersage mithilfe eines neuronalen Netzes wichtig. Diese Muster erstmals zu erkennen ist die Grundlage für eine

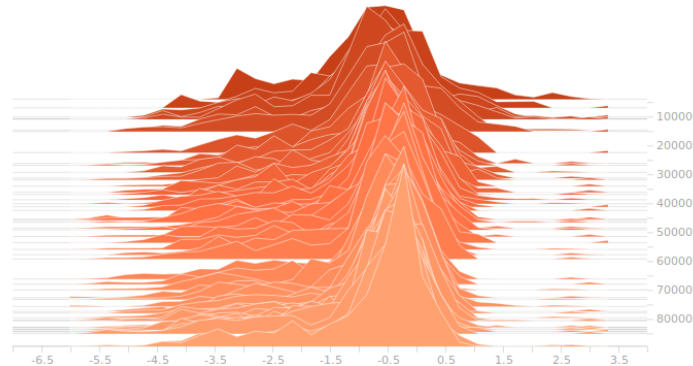


Abbildung 6.12: Dreidimensionales Histogramm aus dem TensorBoard

spätere Aufbereitung der Daten für das neuronale Netz. In der Theorie müsste das neuronale Netz diese Muster selbständig erkennen und erlernen können, dies dauert aber einige Zeit und benötigt viele Datensätze, daher soll vorab eine Sortierung der Datensätze vorgenommen werden.

Da der Nutzer eine hübsche Statistik sehen will, werden verschiedenen Grafiken und Diagramme je nach Anwendungsbereich benutzt. Eine Idee für ein Diagramm wäre ein dreidimensionales Histogramm über die Zeit mit dem Streckenverlauf und der Verspätung. Ein solches Histogramm wird derzeit im Tensorflow eigenen Tensorboard verwendet, um die Verteilungen der einzelnen Schichten im Netzwerk über die Lerndauer zu visualisieren. In Abbildung 6.12 ist ein solches Histogramm zu sehen. Die dritte Dimension stellt den zeitlichen Verlauf deutlich dar.

## 6.6 Darstellung der Datensätze

Hier was zum View Stations schreiben, um deren details

In der Bahnübersicht der Website werden den Nutzern alle Informationen und Statistiken zu diesem Bahnhof angezeigt. In der Fahrplanübersicht zu jedem Tag soll der Nutzer die Möglichkeit erhalten, zu jedem Zug Statistiken zu dessen Verspätungen über einen Zeitraum anzeigen zu lassen. Die Auswahl des Zuges findet entweder über den Fahrplan auf der Seite oder über die eigenständige Seite für die Zugsuche statt. Die Statistiken für jeden Zug werden, sofern nicht vorhanden, automatisch generiert und dann für 60 Minuten gecached. Der Einsatz eines Caches ist sinnvoll, da die Generierung der Statistiken einige Ressourcen benötigt und sich die Datensätze eines Zuges nur selten ändern. Somit wird das doppelte Senden von Anfragen an den MySQL Server verhindert. Die Routen der Website sind darauf ausgelegt möglichst kleine Teile der Website auszutauschen oder dynamisch nachzuladen. Dies soll vor allem bei dem mobilen Nutzen der

Website die Ladezeiten gering halten und den Server entlasten.

Grafik Fahrplan bzw. Suche Zug mit zugnummerfull

Erstes Laden der Seite, Weiteres laden der seite (wie tcp syn,act,etc. Diagramm)

# Kapitel 7

## Schlussfolgerung

### 7.1 Rückblick

Was ist geschehen, was würden wir anders machen, was waren wichtige Schritte

Rückblickend haben wir einiges geschafft was wir schaffen wollten. Leider aber nicht alles. Das ist vor allem dem Unterschätzen der Arbeit geschuldet, die notwendig war, um dieses Thema zu bearbeiten. Natürlich gab es auch Probleme innerhalb der Gruppe. So war die Kommunikation zwischen den Mitgliedern meist spärlich oder wenig hilfreich. Ein wichtiger Schritt waren vor allem die Datenerfassung aus der API der Deutschen Bahn. Mithilfe der Daten aus dem Miner konnten die Daten um einiges einfacher analysiert werden als die Daten die wir als Response der DBAPI erhalten haben. Müssten wir eine weitere Studienarbeit schreiben, würden wir bereits vor der Suche nach einem Dozenten für das Thema eine Gliederung für das Thema erstellen. Dadurch ist es für die Studenten und den Dozenten einfacher einen ordentlichen Umfang der Studienarbeit festzulegen. Auch sollte festgelegt werden welche Themen im Vordergrund stehen sollen.

### 7.2 Fazit

Ergebnis der Studienarbeit, was war gut, was war schlecht, hat alles so geklappt, wo gab es Probleme, wie wurden diese gelöst (kurz und knapp zusammengefasst).

WHAT? Raff ich nicht: verspätung abhängig nach zeit nicht analysiert

Insgesamt können wir sagen, dass „Analyse und Auswertung von Echtzeit-Fahrplänen der Deutschen Bahn“ ein faszinierendes Thema für eine Studienarbeit war. Problematisch war, dass wir keine klare Definition des Problems festgelegt hatten. Zu Beginn war uns nicht klar was wir innerhalb unserer Studienarbeit bearbeiten wollten. Dies wurde erst nach und nach im 6. Semester deutlich. Dadurch wurde es schwierig, die Parameter in Bezug zu setzen und zeitnah auch

praktisch umzusetzen. Eine weitere Schwierigkeit war die schlechte Sichtbarkeit unseres Erfolgs. Trotz der ganzen Arbeit die wir betrieben haben, sind unsere Ergebnisse nur schwer sichtbar. Die virtualisierbaren Anteile sind in Form von Diagrammen und Abbildungen in die Studienarbeit mit eingeflossen, schlecht darstellbar bleibt der Aufwand für die Abfrage und Vorverarbeitung der Daten. Die Berechnungen eines Durchschnitts zum Beispiel sind simpel, problematisch war jedoch die Beschaffung der dazu benötigten Daten von unseren Servern. So ist es zum Beispiel notwendig, für Berechnung der Durchschnittlichen Verspätung nicht nur die Informationen zur aktuellen Haltestelle abzufragen, sondern auch die der vorherigen Haltestelle. Werden diese in zu vielen Abfragen abgerufen, kann das zu Performanceproblemen führen. Fertig gestellt wurde die Feststellung der Verspätung der Züge auf allen Strecken die an Bahnhöfen der Deutschen Bahn Halten. Die Analyse der Daten ist ausreichend für eine Statistische Auswertung, allerdings nicht ausreichend, für die Bildung eines Prognose Modells.

Was hat funktioniert?  
hf

### 7.3 Ausblick

Wie geht es weiter, könnte es weiter gehen, was sollte verbessert werden, wo befinden sich Schwachstellen, event. ungelöste Probleme

Als ungeklärt kann man die Prognose der Ankunftszeit betrachten. Diese benötigt noch sehr viel Arbeit um eine Prognose mit einer besseren Genauigkeit zu erreichen. Dafür ist aber noch sehr viel Arbeit und Rechenleistung nötig.

**Bereitstellung der Daten** Eine weitere Erweiterungsmöglichkeit sehen wir in der Publizierung unserer Rohdaten und teilweise schon durchgeführte Analysen wie zum Beispiel die Durchschnittsberechnungen der Verspätung pro Station und Streckenabschnitt. Dies soll es anderen Studen ermöglichen unsere, bereits gesammelten Daten nutzen zu können ohne die Daten erneut sammeln zu müssen.

**Verbesserung der Website** Die Webseite stellt jetzt schon ein nützlichen Werkzeug dar, um die Mitschriebe und Analysen der Echtzeitdaten zeitnah einsehen zu können. Hierbei kann die Website weiter verbessert werden, indem die Nutzerinteraktion verbessert wird.

Auf welche Wiese soll die Nutzerinteraktion verbessert werden?

**Weitere Durchführung von Analysen** Es wurden bereits statistische Auswertungen durchgeführt.

Haben wir schon irgendwo geschildert, ob und welche Erkenntnisse wir aus den Statistischen Daten ziehen konnten?

Allerdings sind die daraus gewonnenen Erkenntnisse noch nicht ausreichend, um ein Prognose-Modell zu entwerfen. Hierbei ist es denkbar, elaboriertere Algorithmen aus der Machine Learning Domäne einzusetzen. Es wird vermutet, dass durch die Anwendung von Machine Learning-Algorithmen aussagekräftigere Erkenntnisse für die Bildung eines Prognosen-Modells erlangt werden können.

**Verbesserung des Neuronalen Netzes**

Weitere Arbeit an dem Model+ vorhersage, weit  
nutzerinteraktion.

# Literatur

- FIELDING, R. et al. [1999]. *Hypertext Transfer Protocol – HTTP/1.1*. Internet Requests for Comments. RFC. URL: <http://www.rfc-editor.org/rfc/rfc2616.txt> [siehe S. 25].
- JIAWEI HAN, Micheline Kamber [2000]. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, S. 7 [siehe S. 11].
- USAMA FAYYAD Gregory Piatetsky-Shapiro, Padhraic Smyth [1996]. „From Data Mining to Knowledge Discovery in Databases“. In: *AI Magazine Volume 17 Number 3*. Association for the Advancement of Artificial Intelligence [siehe S. 34, 35].



# Liste der ToDo's

■	Nochmal lesen und eventuelle Schreibfehler beheben . . . . .	1
■	TODO . . . . .	8
■	TODO . . . . .	8
■	gut . . . . .	8
■	gut . . . . .	9
■	gut . . . . .	11
■	gut/naja . . . . .	12
■	WOHIN DAMIT . . . . .	14
■	Dieses gesamte Kapitel aufsplitten und in die Grundlagen der einzelnen chapter einordnen . . . . .	14
■	Ist halt eigentlich das Thema wo man am meisten hätte schreiben können.	14
■	Data Mining Einführung und dessen Bedeutung für das Projekt . . . .	14
■	Datenformat und Aufbau erklären. Wieso sollte im ersten Schritt beim Mining nicht direkt alles angepasst werden? Wieso müssen die Da- ten aufbereitet werden? Stichwort: FehlerAPI, Fehlende Datensätze, Bucketlist, Konvertierung . . . . .	14
■	Datenmodell erläutern, welche Rohdaten aus der DB-API . . . . .	16
■	Schauen, ob Kapitel noch Sinn macht . . . . .	16
■	n . . . . .	16
■	naja . . . . .	17
■	gut/naja . . . . .	21
■	gut/naja . . . . .	25
■	gut (event Skript noch rein,mysql sharding erklären?) . . . . .	28
■	gut . . . . .	31
■	leere subSection . . . . .	34
■	zitat leer . . . . .	34
■	Hier noch text . . . . .	34
■	Strecken eines Zuges werden in langen Zeichenketten statt EVA-Nummern abgelegt . . . . .	35
■	fehlt hier was . . . . .	35
■	Dieses Kapitel bezieht sich auf den Tabellen cache vorgang, also generie- rung einer neuen besser select baren tabelle, das Quellcode listing ist noch nicht korrekt . . . . .	35
■	gut . . . . .	37

■ naja (nicht ganz gelesen da muede) . . . . .	41
■ gut (naja) . . . . .	52
■ In diesem gesamten Chapter/Kapitel fehlen noch Quellenangaben und weitere Literaturhinweise . . . . .	52
■ gut (Tabelle nicht im Text verwendet vllt noch einbauen . . . . .	53
■ gut (naja) . . . . .	54
■ gut (naja) . . . . .	58
■ naja . . . . .	60
■ Erläuterung welche Informationen in das Neuronale Netz eingegeben werden und welche Daten von dem Netz ausgegeben werden. . . . .	62
■ gut . . . . .	62
■ gut . . . . .	63
■ gut . . . . .	65
■ gut . . . . .	65
■ In diesem gesamten Kapitel fehlen noch Quellenangaben und weitere Literaturhinweise . . . . .	69
■ in anderen Kapiteln paragraph anstatt item, formatierung schöner . . .	69
■ Dimensionen der Daten, ORT; ZEIT; NAME; STRECKE; etc. . . . .	76
■ Absätze setzen für bessere Lesbarkeit hier und im ff . . . . .	76
■ würde für diese abbildung eine andere Farbe für den Text nehmen das Lila finde ich irgendwie komisch zu lesen . . . . .	79
■ Hier was zum View Stations schreiben, un deren details . . . . .	80
■ Grafik Fahrplan bzw. Suche Zug mit zugnummerfull . . . . .	81
■ Erstes Laden der Seite, Weiteres laden der seite (wie tcp syn,act,etc. Diagramm) . . . . .	81
■ Was ist geschehen, was würden wir anders machen, was waren wichtige Schritte . . . . .	82
■ Ergebnis der Studienarbeit, was war gut, was war schlecht, hat alles so geklappt, wo gab es Probleme, wie wurden diese gelöst (kurz und knapp zusammengefasst. . . . .	82
■ WHAT? Raff ich nicht: verspätung abhängig nach zeit nicht analysiert .	82
■ Was hat funktioniert? hf . . . . .	83
■ Wie geht es weiter, könnte es weiter gehen, was sollte verbessert werden, wo befinden sich Schwachstellen, event. ungelöste Probleme . . . . .	83
■ Auf welche Wiese soll die Nutzerinteraktion verbessert werden? . . . . .	83
■ Haben wir schon irgedwo geschildert, ob und welche Erkenntnisse wir aus den Statistischen Daten ziehen konnten? . . . . .	83
■ Weitere Arbeit an dem Model+ vorhersage, weitere visualisierungen, bessere nutzerinteraktion. . . . .	84