Duale Hochschule Baden-Württemberg Mannheim

Projektberichtarbeit

Intel Core I7 Prozessoren

Studiengang Informatik

Studienrichtung Angewandte Informatik

Verfasser(in):	Moritz Werr, Phil Richter, Max Stege
Matrikelnummer:	5401527
Matrikelnummer:	4164342
Matrikelnummer:	7285772
Kurs:	TINF22AI2
Dozent:	Dr. Frank Schulz
Bearbeitungszeitraum:	01.10.2024 – 01.12.2024

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Titel "*Intel Core I7 Prozessoren*" selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Quelltextverzeichnis

2.1 clean.sh 4

3.1 database.R 6

5.1 load data from the database 13
5.2 Funktion extraxct_generation 14
5.3 Code für den Scatterplot in Figur 5.1 15

1 Einleitung (Max Stege)

Die rasante Entwicklung der Computertechnologie wird seit Jahrzehnten von einer zentralen Beobachtung geprägt: dem Mooreschen Gesetz. Dieses Gesetz postuliert, dass sich die Anzahl der Transistoren auf integrierten Schaltkreisen etwa alle zwei Jahre verdoppelt, was zu exponentiellen Leistungssteigerungen bei gleichzeitiger Reduzierung der Kosten führt [14]. Doch angesichts physikalischer und technischer Grenzen wird zunehmend diskutiert, ob dieses Gesetz auch heute noch seine Gültigkeit hat. Die Analyse der Entwicklung von Prozessoren, wie der Intel Core i7-Reihe, bietet eine spannende Möglichkeit, diese Frage aus einer datengetriebenen Perspektive zu untersuchen.

Neben der Überprüfung des Mooreschen Gesetzes steht die Frage im Fokus, in welchen Bereichen die größten Fortschritte erzielt wurden. Hat die Erhöhung der Kernanzahl maßgeblich zur Leistungssteigerung beigetragen, oder sind es optimierte Taktraten und eine verbesserte Energieeffizienz, die den Unterschied ausmachen? Gleichzeitig wird analysiert, welche Metriken über die Generationen hinweg stagniert sind und welche Faktoren möglicherweise als limitierende Größen der technologischen Entwicklung wirken.

Dieses Projekt, durchgeführt im Rahmen der Vorlesung *Big Data Analytics*, widmet sich der strukturierten Analyse und Visualisierung von Daten zu Intel Core i7-Prozessoren der letzten Generationen. Dabei soll nicht nur der technische Fortschritt dokumentiert werden, sondern auch ein umfassendes Verständnis für die Datengrundlage und deren Interpretation entwickelt werden. Um dies zu ermöglichen, wurde eine mehrstufige Pipeline realisiert, die sich über folgende Schritte erstreckt:

- **Datenbeschaffung:** Sammlung technischer Spezifikationen direkt von der Herstellerwebsite, da keine öffentliche API verfügbar ist.
- **Datenaufbereitung:** Transponierung und Bereinigung der Rohdaten, um diese für die weitere Analyse nutzbar zu machen.
- **Datenbankintegration:** Speicherung der Daten in einer relationalen PostgreSQL-Datenbank, um eine flexible und effiziente Abfrage der Datensätze zu ermöglichen.
- **Analyse und Visualisierung:** Nutzung moderner Datenanalyse- und Visualisierungswerkzeuge, um Muster und Trends über verschiedene Prozessorgenerationen hinweg zu identifizieren.

Durch diese strukturierte Herangehensweise können entscheidende Fragen beantwortet werden: Welche Metriken zeigen die stärksten Fortschritte? Wo sind technologische Grenzen erreicht? Und welche Entwicklungen könnten in den kommenden Jahren dominieren? Die

gewonnenen Erkenntnisse liefern nicht nur einen detaillierten Überblick über die Entwicklung der Intel Core i7-Serie, sondern geben auch Impulse für zukünftige Innovationen und die strategische Ausrichtung der Halbleiterindustrie.

Diese Dokumentation beschreibt im Detail den technischen und methodischen Ablauf des Projekts. Ziel ist es, eine nachvollziehbare und robuste Datenpipeline zu präsentieren, die als Grundlage für weiterführende Analysen und Forschungsprojekte dienen kann. Darüber hinaus wird kritisch reflektiert, welche Limitierungen und Herausforderungen sich in der Umsetzung ergeben haben, und wie diese überwunden werden können.

2 Vorbereitung der Daten (Moritz Werr)

Intel bietet auf ihrer Webseite die Möglichkeit unterschiedliche Prozessoren miteinander zu vergleichen. Es werden alle Desktop-Prozessoren der Intel Core I7-Reihe an Prozessoren ausgewählt, damit es eine große Anzahl an Datensätzen gibt und auch die Spannweite zwischen gewissen Werten groß genug ist, dass Unterschiede leichter zu erkennen sind. Z.B. ist die Kernanzahl pro Prozessor seit der 4. Generation stark gestiegen, wohingegen die Anzahl an Prozessorkernen bei Intel Core I3-Modellen nicht so stark angestiegen sind. Die Daten werden direkt von Intels Webseite bezogen. Da Intel keine öffentliche API anbietet, um deren CPU-Spezifikationen zu vergleichen und zu exportieren, muss die CSV-Datei selber heruntergeladen werden. Bevor die Daten aus der CSV ausgelesen werden können, müssen diese noch etwas aufbereitet werden. Wie in Abbildung 2.1 von der Herstellerseite zu sehen ist, sind hier leider die Prozessormodelle die Spalten anstatt der Zeilen.

Bevor die CSV bereinigen können, muss also die Matrix der CSV noch einmal transponiert werden. Sonst wären die einzelnen Prozessoren die Spalten in der Datenbank, in der die bereinigte Daten der CSV abgespeichert werden. Weiterhin sind nicht alle Spalten in der CSV interessant für die Auswertung der Daten. Viele der Einträge in der CSV listen nur Features oder Prozessorerweiterungen auf, die nicht einfach zu vergleichen sind oder sich nur schwierig visualisieren lassen. Weiterhin sind manche Einträge bei älteren Prozessoren komplett leer, weswegen die Datenmenge in diesen Spalten sehr gering ist. Daher werden nur ausgewählte Spalten gespeichert, die für die Fragestellung interessant sind und auch genügend Datenpunkte haben.

56 Intel® Produkte vergleichen

Vergleich exportieren

	Intel® Core™ i7-8700K Processor	Intel® Core™ i7-8700 Processor	Intel® Core™ i7-8700T Processor	Intel® Core™ i7-12700 Processor	Intel® Core™ i7-12700F Processor	Intel® Proce
Hauptdaten						
Produktsortiment	Intel® Core™ i7 Prozessoren der 8. Generation	Intel® Core™ i7 Prozessoren der 8. Generation	Intel® Core™ i7 Prozessoren der 8. Generation	Intel® Core™ i7 Prozessoren der 12. Generation	Intel® Core™ i7 Prozessoren der 12. Generation	Intel® der 12
Vertikales Segment	Desktop	Desktop	Desktop	Desktop	Desktop	Deskt
Prozessornummer	i7-8700K	i7-8700	i7-8700T	i7-12700	i7-12700F	i7-127
Lithographie	14 nm	14 nm	14 nm	Intel 7	Intel 7	Intel 7
Empfohlener Kundenpreis	$359.00	$303.00	$303.00	$373.00-$383.00	$345.00-$355.00	$450.0
CPU-Spezifikationen						
Anzahl der Kerne	6	6	6	12	12	12

Figure 2.1: Webseite zum Produktvergleich von Intel-Prozessoren
Quelle: [9]

Aufgrund der mächtigen Standardwerkzeugen eines POSIX-kompatiblen Betriebssystem wie Linux, wurde `awk` verwendet, um die Spalten in Zeilen zu konvertieren bzw. die Zeilen zu Spalten zu konvertieren. `awk` ist ein Werkzeug zur Erkennung und Verarbeitung von Textmustern [4]. `awk` arbeitet nicht nur mit Zeilen, wie z.B. `grep` oder `sed`, sondern kann auch in einzelnen Spalten bzw. Feldern filtern [4]. Normalerweise sind die Standardtrennzeichen von Spalten für `awk` Leerzeichen und Tabulatoren. Es ist aber möglich mit dem '-F' Flag anzugeben, dass ein anderes Trennzeichen verwendet wird [4]. In unserem Fall wird das Trennzeichen das Komma sein, weil wir mit einer CSV arbeiten. In `awk` wird häufig mit regulären Ausdrucken gefiltert, aber da unsere Datei eine statische CSV-Datei ist, kann direkt nach Spaltennamen sortiert werden. Das entsprechende Skript sieht folgendermaßen aus:

```
1  #!/bin/bash
2
3  awk -F ',' '
4  NR==5   ||
5  NR==7   ||
6  NR==8   ||
7  NR==12  ||
8  NR==13  ||
9  NR==14  ||
10 NR==15  ||
11 NR==16  ||
12 NR==17  ||
13 NR==18  ||
14 NR==19  ||
```

```
15 NR==22  ||
16 NR==23  ||
17 NR==24  ||
18 NR==25  ||
19 NR==26  ||
20 NR==27  ||
21 NR==28  ||
22 NR==29  ||
23 NR==30  ||
24 NR==31  ||
25 NR==36  ||
26 NR==42  ||
27 NR==56  ||
28 NR==57  ||
29 NR==58  ||
30 NR==59  ||
31 NR==60  ||
32 NR==61  ||
33 NR==62  ||
34 NR==82  ||
35 NR==90
36 {print $0}' "Intel_UPE_ComparisonChart_2024_11_04_i7.csv" |
37
38 awk '
39 {
40         for (i=1; i<=NF; i++)
41         a[i] = a[i] ? a[i] "," $i : $i
42 }
43 END {
44         for (i=1; i in a; i++)
45         print a[i]
46 }
47 ' FS=, OFS=, > clean.csv
```

Quelltext 2.1: `clean.sh`

Das Skript besteht aus zwei Teilen: Im ersten Teil werden die entsprechenden Spalten mit `awk` nach Spaltennummer ausgefiltert, welche vorher als interessant deklariert wurden, und in den zweiten Teil gepiped. Im zweiten Teil werden die jeweiligen Spalten in einer for-Schleife in neue Zeilen geladen. Nachdem die gesamte Datei eingelesen wurde, kann diese wieder transponiert ausgegeben werden. Nun kann mit dem eigentlichen Säubern der Daten begonnen werden.

3 Säubern der Daten (Moritz Werr)

Nun ist die CSV vorbereitet für die eigentliche Säuberung. Die Daten liegen in den entsprechenden Spalten vor und alle uninteressanten Daten wurden entfernt. Bevor die Daten jedoch in einer Datenbank in Tabellenform gespeichert werden können, sollten noch entsprechende Spalten bereinigt werden, dass nur noch Zahlenwerte ohne Einheiten dort aufzufinden sind. Weiterhin beinhalten viele Spaltennamen immer noch Sonderzeichen und Leerzeichen, welche es schwierig macht die Spalten zu lesen.

Um diese Probleme zu addressieren, wurde R zur Bereinigung der Daten ausgewählt. R ist eine Programmiersprache für statistische Berechnungen [27]. Es bietet mächtige Werkzeuge zur Auswertung und Speicherung von Daten [27]. Damit ist es eine exzellente Wahl um dieses Problem zu lösen. Es bietet Möglichkeiten die Spaltennamen automatisch mit der Funktion `clean_names()` aus der Bibliothek `janitor` umzubenennen [10], indem es problematische Zeichen, wie z.B. Leerzeichen und Sonderzeichen, erkennt und durch unproblematische Zeichen ersetzt. Weiterhin werden auch alle Großbuchstaben durch Kleinbuchstaben ersetzt. Darüber hinaus ist R eine dynamisch typisierte Programmiersprache [7]. Datentypen werden je nach Kontext automatisiert erkannt. Wenn die entsprechenden Spalten von allen nichtnumerischen Zeichen gesäubert werden, lassen sich diese problemlos in Zahlen umwandeln. Die entsprechenden Einheiten werden nach der Spaltensäuberung stattdessen noch in den Spaltennamen hinzugefügt.

Wenn alle Spalten und Spaltennamen gesäubert bzw. angepasst wurden, können die Daten in einer Datenbank gespeichert werden.

Mit den vorher genannten Überlegungen im Hinterkopf ist folgendes Skript entstanden:

```
1  #!/usr/bin/Rscript
2
3  library(RPostgres)
4
5  intel <- read.csv("clean.csv")          #Reading
6  intel <- janitor::clean_names(intel)    #Cleaning the column
       names
7
8  #cleaning and mutating columns to numbers
9  intel$max_turbo_taktfrequenz <- as.numeric(gsub(" GHz", "",
       intel$max_turbo_taktfrequenz))
10 intel$lithographie <- gsub("Intel 7", "10 nm",
       intel$lithographie)
```

```
11 intel$lithographie <- as.numeric(gsub(" nm", "",
     intel$lithographie))
12 intel$intel_turbo_boost_technik_2_0_taktfrequenz <- as.
     numeric(gsub(" GHz", "",
     intel$intel_turbo_boost_technik_2_0_taktfrequenz))
13 intel$grundtaktfrequenz_des_prozessors <- gsub(" \\| ", ".",
     intel$grundtaktfrequenz_des_prozessors)
14 intel$grundtaktfrequenz_des_prozessors <- as.numeric(gsub("
     GHz", "", intel$grundtaktfrequenz_des_prozessors))
15 intel$cache <- gsub(" MB Intel Smart Cache", "", intel$cache)
16 intel$cache <- as.numeric(gsub(" MB", "", intel$cache))
17 intel$bus_taktfrequenz <- as.numeric(gsub(" GT/s", "",
     intel$bus_taktfrequenz))
18 intel$verlustleistung_tdp <- as.numeric(gsub(" W", "",
     intel$verlustleistung_tdp))
19 intel$intel_turbo_boost_max_technology_3_0_frequency <- gsub
     (" \\| ", ".",
     intel$intel_turbo_boost_max_technology_3_0_frequency)
20 intel$intel_turbo_boost_max_technology_3_0_frequency <- as.
     numeric(gsub(" GHz", "",
     intel$intel_turbo_boost_max_technology_3_0_frequency))
21 intel$single_p_core_turbo_frequency <- as.numeric(gsub(" GHz
     ", "", intel$single_p_core_turbo_frequency))
22 intel$single_e_core_turbo_frequency <- as.numeric(gsub(" GHz
     ", "", intel$single_e_core_turbo_frequency))
23 intel$e_core_base_frequency <- gsub(" GHz", "",
     intel$e_core_base_frequency)
24 intel$e_core_base_frequency <- as.numeric(gsub("900 MHz",
     "0.9", intel$e_core_base_frequency))
25 intel$total_l2_cache <- as.numeric(gsub(" MB", "",
     intel$total_l2_cache))
26 intel$processor_base_power <- as.numeric(gsub(" W", "",
     intel$processor_base_power))
27 intel$maximum_turbo_power <- as.numeric(gsub(" W", "",
     intel$maximum_turbo_power))
28 intel$grundtaktfrequenz_der_grafik <- as.numeric(gsub(" MHz",
      "", intel$grundtaktfrequenz_der_grafik))
29 intel$max_dynamische_grafikfrequenz <- as.numeric(gsub(" GHz
     ", "", intel$max_dynamische_grafikfrequenz))
```

```
30 intel$max_videospeicher_der_grafik <- as.numeric(gsub(" GB",
     "", intel$max_videospeicher_der_grafik))
31 intel$x4k_unterstutzung <- gsub("Hz", "",
     intel$x4k_unterstutzung)
32 intel$x4k_unterstutzung <- as.numeric(gsub("Yes \\|  at ",
     "", intel$x4k_unterstutzung))
33
34
35 #Putting the units back into the column names
36 colnames(intel)[colnames(intel) == "max_turbo_taktfrequenz"]
     <- "max_turbo_taktfrequenz_GHz"
37 colnames(intel)[colnames(intel) == "lithographie"] <- "
     litographie_nm"
38 colnames(intel)[colnames(intel) == "
     intel_turbo_boost_technik_2_0_taktfrequenz"] <- "
     intel_turbo_boost_technik_2_0_taktfrequenz_GHz"
39 colnames(intel)[colnames(intel) == "
     grundtaktfrequenz_des_prozessors"] <- "
     grundtaktfrequenz_des_prozessors_GHz"
40 colnames(intel)[colnames(intel) == "cache"] <- "cache_MB"
41 colnames(intel)[colnames(intel) == "bus_taktfrequenz"] <- "
     bus_taktfrequenz_GT_per_s"
42 colnames(intel)[colnames(intel) == "verlustleistung_tdp"] <-
     "verlustleistung_tdp_W"
43 colnames(intel)[colnames(intel) == "
     intel_turbo_boost_max_technology_3_0_frequency"] <- "
     intel_turbo_boost_max_technology_3_0_frequency_GHz"
44 colnames(intel)[colnames(intel) == "
     single_p_core_turbo_frequency"] <- "
     single_p_core_turbo_frequency_GHz"
45 colnames(intel)[colnames(intel) == "
     single_e_core_turbo_frequency"] <- "
     single_e_core_turbo_frequency_GHz"
46 colnames(intel)[colnames(intel) == "e_core_base_frequency"]
     <- "e_core_base_frequency_GHz"
47 colnames(intel)[colnames(intel) == "total_l2_cache"] <- "
     total_l2_cache_MB"
48 colnames(intel)[colnames(intel) == "processor_base_power"] <-
      "processor_base_power_W"
```

```
49 colnames(intel)[colnames(intel) == "maximum_turbo_power"] <-
     "maximum_turbo_power_W"
50 colnames(intel)[colnames(intel) == "
     grundtaktfrequenz_der_grafik"] <- "
     grundtaktfrequenz_der_grafik_MHz"
51 colnames(intel)[colnames(intel) == "
     max_dynamische_grafikfrequenz"] <- "
     max_dynamische_grafikfrequenz_GHz"
52 colnames(intel)[colnames(intel) == "
     max_videospeicher_der_grafik"] <- "
     max_videospeicher_der_grafik_GB"
53 colnames(intel)[colnames(intel) == "x4k_unterstutzung"] <- "
     x4k_unterstutzung_at"
54
55
56 con <- dbConnect(
57 RPostgres::Postgres(),
58 dbname = "bda",
59 host = "localhost" ,
60 port = 5432,
61 user = "bda",
62 password = "bda",
63 )
64
65 dbWriteTable(con, "intel", intel, row.names = FALSE,
     overwrite = TRUE)
66 dbDisconnect(con)
```

Quelltext 3.1: `database.R`

Als erstes wird im Skript die transponierte CSV eingelesen und die Spaltennamen mit Hilfe der `janitor` Library gesäubert. Danach kommt ein großer Block an Funktionen, die die einzelnen Spalten von Einheiten und komischer Formatierung säubern. Danach kommt ein großer Block zur Umbenennung der Spaltennamen. Im letzten Schritt wird der Dataframe in einer existierenden Datenbank gespeichert. Ein großer Vorteil an der Funktion ist, dass die Tabelle innerhalb der Datenbank vorher nicht existieren muss. R erstellt von selber eine neue Tabelle mit entsprechenden Datentypen basierend auf den Datentypen innerhalb des Dataframes. Daher ist es wichtig die Datentypen innerhalb des Dataframes schon zu Nummern zu konvertieren.

4 Speichern der Daten (Moritz Werr)

Aus dem vorherigen Kapitel ist bereits hervorgegangen, dass die Daten in einer Datenbank gespeichert werden. In diesem Fall ist die Wahl auf PostgreSQL gefallen. PostgreSQL ist ein Open-Source, relationales Datenbankmanagementsystem [24]. Es ist komplett ACID-kompatibel [24] und speichert Daten in geordneter Form als Tabellen ab [2]. Da die Datenquelle bereits als CSV geliefert wird, bietet es sich an diese in einer entsprechenden Datenbank zu speichern. CSV-Dateien speichern bereits Daten in Tabellenform, weswegen die Wahl direkt auf eine relationale SQL-Datenbank gefallen ist.

Es gibt viele Alternativen zu PostgreSQL, diese sind aber teils kostenpflichtig oder bieten die gleichen bzw. weniger Features wie PostgreSQL. Proprietäre Datenbankmanagementsysteme wie Oracle RDBMS oder Microsoft SQL-Server sind für große, Enterprise-Skala Deployments konzipiert und entsprechend kommerzielle Lizenzen [13][17]. Kostenlose Optionen sind daher bevorzugt. MariaDB bzw. MySQL und SQLite wären Open-Source Alternativen zu PostgreSQL, welche auch gut geeignet sind[29][16]. SQLite jedoch bietet keine Serverfunktionalität weswegen nur lokal gearbeitet werden kann[29]. MariaDB/MySQL haben keinen nennenswerten Funktionalitätsunterschied, aber wurden nicht gewählt, weil mit PostgreSQL mehr Erfahrung im Team existiert[16][23].

Nachdem das Skript aus dem vorherigen Kapitel durchgelaufen ist, ist das Datenbankschema in Abbildung 4.1 zu sehen.

Tabelle »public.intel«

Spalte	Typ	Sortierfolge	NULL erlaubt?	Vorgabewert
produktsortiment	text			
prozessornummer	text			
litographie_nm	double precision			
empfohlener_kundenpreis	text			
anzahl_der_kerne	integer			
gesamte_threads	integer			
max_turbo_taktfrequenz_GHz	double precision			
intel_turbo_boost_technik_2_0_taktfrequenz_GHz	double precision			
grundtaktfrequenz_des_prozessors_GHz	double precision			
cache_MB	double precision			
bus_taktfrequenz_GT_per_s	double precision			
verlustleistung_tdp_W	double precision			
x_of_performance_cores	integer			
x_of_efficiency_cores	integer			
intel_turbo_boost_max_technology_3_0_frequency_GHz	double precision			
single_p_core_turbo_frequency_GHz	double precision			
single_e_core_turbo_frequency_GHz	double precision			
p_core_base_frequency	text			
e_core_base_frequency_GHz	double precision			
total_l2_cache_MB	double precision			
processor_base_power_W	double precision			
maximum_turbo_power_W	double precision			
einfuhrungsdatum	text			
expected_discontinuance	text			
grundtaktfrequenz_der_grafik_MHz	double precision			
max_dynamische_grafikfrequenz_GHz	double precision			
max_videospeicher_der_grafik_GB	double precision			
x4k_unterstutzung_at	double precision			
max_auflosung_hdmi	text			
max_auflosung_dp	text			
max_auflosung_e_dp_integrierter_flachbildschirm	text			
pci_express_konfigurationen	text			
thermische_spezifikation_des_kuhlers	text			

Figure 4.1: Datenbankschema erstellt durch das R-Skript

5 Visualisieren der Daten (Phil Richter)

Nach der Bereinigung und dem Speichern der Daten in einer Datenbank, wie in den vorherigen Kapiteln beschrieben, können die Daten visualisiert werden. Dafür wird als Grundlage die Programmiersprache Python mit der Version 3.12.7 verwendet [25]. Diese ist vorallem im Bereich der Datenanalyse und -visualisierung sehr weit verbreitet [15, 22]. Eine alternative Möglichkeit wäre die Verwendung von R, welches ebenfalls sehr weit verbreitet ist und speziell für die Datenanalyse entwickelt wurde [27]. Zudem wurde R bereits für die Säuberung der Daten, wie im Kapitel 3 beschrieben, verwendet. Da im Team allerdings mehr Erfahrung mit Python vorhanden war, wurde sich bei der Visualsierung für die Verwendung von Python entschieden.

Für das Laden und Visualisiern der Daten werden folgende Packages verwendet:

Package	**Version**	**Beschreibung**
dotenv	1.0.1	Lädt Umgebungsvariablen aus einer *.env* Datei
sqlalchemy	2.0.36	Ermöglicht Zugriff auf die Datenbank, sowie Datenbankabfragen
pandas	2.2.3	Ermöglicht die Manipulation, Analyse und Verarbeitung von Daten
matplotlib	3.9.2	Erstellt aus gegebenen Daten anpassbare Diagramme
seaborn	0.13.2	Basiert auf *matplotlib* und wird ebenfalls zur Datenvisualisierung verwendet

Table 5.1: Auflistung aller verwendeten Packages, sowie ihrer Versionen

5.1 Laden der Daten (Phil Richter)

Im ersten Schritt der Visualisieren werden die Daten aus der Datenbank geladen. Dafür werden die Datenbankverbindungsinformationen mit Hilfe von *dotenv* [26] aus einer lokal definierten *.env* Datei geladen. Es werden in diesem Fall Umgebungsvariablen verwendet, damit die Verbindungsinformationen nicht im Code hart codiert sind und somit nicht versehentlich veröffentlicht werden. Nach dem Laden der Variablen wird mit *sqlalchemy* [28] eine Verbindung zur Datenbank aufgebaut und die Daten werden mit einer SQL-Querry abgefragt [18]. Die abgefrageten Daten werden in einem *pandas* [19] DataFrame gespeichert. Ein DataFrame ist dabei eine zweidimensionale Datenstruktur, wie eine Tabelle [20]. Das DataFrame wird verwendet, um die Daten vor der Visualisierung manipulieren zu können, ohne die Originaldaten zu verändern. Im folgendem Python-Code werden die Daten wie beschrieben geladen:

```
1     import os
2     import re
3     import numpy as np
4     import pandas as pd
5     import seaborn as sn
6     import matplotlib.pyplot as plt
7     from sqlalchemy import create_engine
8     from dotenv import load_dotenv
9
10    load_dotenv() # load environment variables
11
12    # get the environment variables
13    b_user = os.getenv("DB_USER")
14    db_password = os.getenv("DB_PASSWORD")
15    db_name = os.getenv("DB_NAME")
16    db_host = os.getenv("DB_HOST")
17    db_port = os.getenv("DB_PORT")
18
19    db_url = f"postgresql://{db_user}:{db_password}@{db_host
         }:{db_port}/{db_name}" # create the db url
20    engine = create_engine(db_url)
21
22    query = "SELECT * FROM intel;" # query to get the data
         from the database
23    df = pd.read_sql(query, engine) # store the data in a
         dataframe
```

Quelltext 5.1: load data from the database

5.2 Bereinigen der geladenen Daten (Phil Richter)

Auch wenn die Daten, wie im Kapitel 3 beschrieben, bereits vor dem Speichern in die Datenbank bereinigt werden, bleinen kleine Unreinheiten in den Daten bestehen. Daher werden bestimmte Spalten im DataFrame nochmals angepasst.

Ein Beispiel dafür ist die Spalte *produktsortiment*. Diese enthält die jeweilige Generation des Prozessors, welche im Datensatz in drei verschiedenen Schreibweisen vorkommt:

- Die Generation als Zahl, z.B. *4. Generation*
- Die Generation als Zahl ausgeschrieben, z.B. *vierte Generation*

- Die Generation als englische Zahl, z.B. *4th Generation*

Um die Prozessoren für die Visualisierung nach ihrer Generation Gruppieren/Sortieren zu können, wird die Spalte *produktsortiment* in eine neue Spalte *generation* umgewandelt, wobei die Schreibweise vereinheitlicht wird. Dies wird mit Hilfe der Funktion *extract_generation* gemacht. Der Code dazu sieht wie folgt aus:

```
1     number_words = {
2         "erste": 1, "zweite": 2, "dritte": 3, "vierte": 4, "
              fuenfte": 5,
3         "sechste": 6, "siebte": 7, "achte": 8, "neunte": 9, "
              zehnte": 10
4     }
5
6     def extract_generation(gen_text):
7         # match the numeric generation
8         match_numeric = re.search(r'(\d+)(?:\.|th)?\s?Gen',
              gen_text, re.IGNORECASE)
9         if match_numeric:
10            # return the numeric generation
11            return int(match_numeric.group(1))
12    for word, num in number_words.items():
13        # check if the word is in the text
14        if word in gen_text.lower():
15            return num # return the number
16    return None
17
18    # extract the generation from the produktsortiment column
19    df['generation'] = df['produktsortiment'].apply(
          extract_generation)
```

Quelltext 5.2: Funktion extraxct_generation

Im Code wird zuerst ein Dictionary [1] *number_words* definiert, welches die Zahlenwörter auf die entsprechende Zahl abbildet. Anschließend wird die Funktion *extract_generation* definiert, welche einen String als Parameter erwartet und die Prozessor-Generation aus dem Text extrahiert. Dabei wird zuerst mit Hilfe eines regulären Ausdrucks, faslls vorhanden, die numerische Generation extrahiert. Ein regulärer Ausdruck ist dabei eine Zeichenkette, die ein bestimmtes Suchmuster für ein gegebenen String definiert [3]. Der in der Funktion verwendete reguläre Ausdruck "*(\d+)(?:|th)?\s?Gen*" sucht nach einer Zahl gefolgt von einem Punkt oder *th*, sowie einem Leerzeichen und den Buchstaben *Gen*. Falls dabei ein übereinstimmender Text gefunden wird, wird dieser zurückgegeben. Anderenfalls wird überprüft, ob eines der

im Dictionary definierten Wörter im Text vorkommt. Falls ja, wird die entsprechende Zahl zurückgegeben. Abschließend wird die Funktion auf die Spalte *produktsortiment* angewendet und das Ergebnis in der neuen Spalte *generation* gespeichert, welche nur noch die Generation als Zahl enthält. Die neue Spalte kann wie jede andere Spalte im DataFrame zur Visualisierung verwendet werden.

5.3 Erstellen von Graphen (Phil Richter)

Nachdem die Daten geladen und bereinigt wurden, können diese nun visualisiert werden. Dafür wird das Package *matplotlib* [12], sowie *seaborn* [30] verwendet. Diese sind sehr weit verbreitet und bieten eine Vielzahl an Möglichkeiten, um Daten zu visualisieren [6]. Mit ihnen können verschiedene Diagramme wie Histrogramme, Scatterplots, Balkendiagramme und viele mehr erstellt werden [21, 8]. Im folgenden ist ein Beispiel für ein Scatterplot und dessen Code dargestellt:

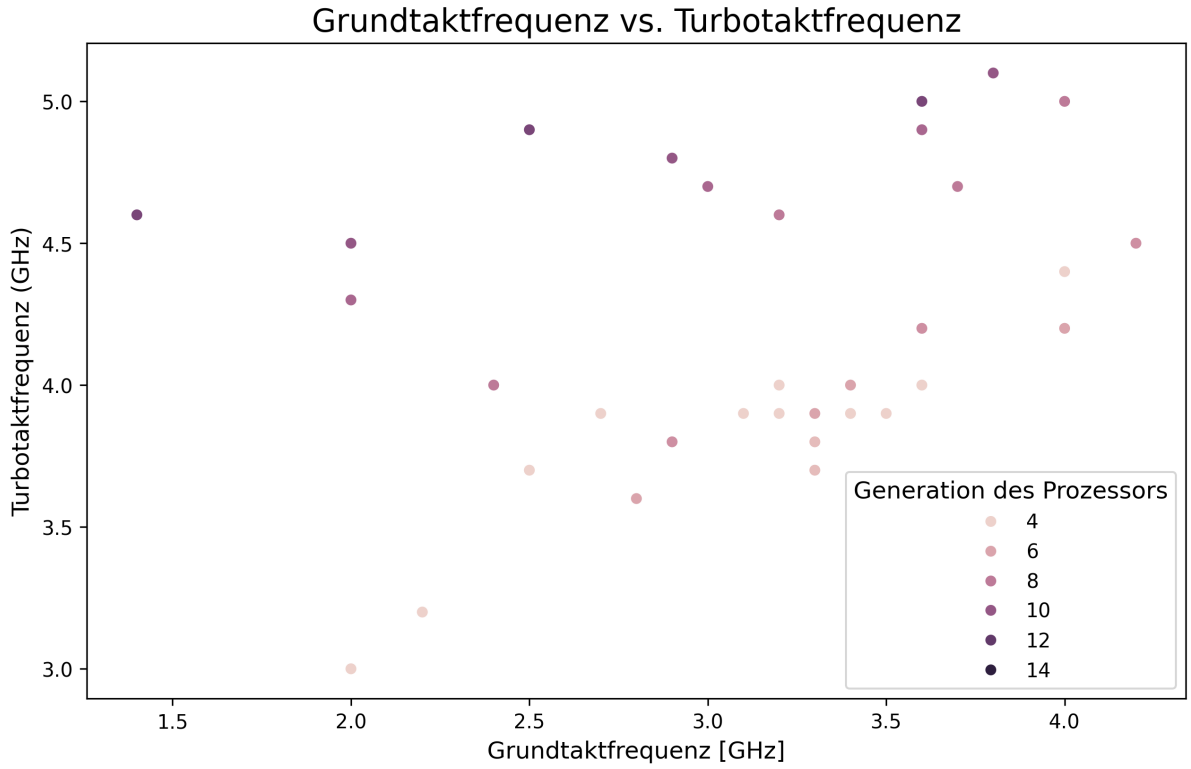

Figure 5.1: Scatterplot erstellt mit *matplotlib* und *seaborn*

```
1    plt.figure(figsize=(10, 6))
2    sn.scatterplot(data=df, x='
        grundtaktfrequenz_des_prozessors_GHz', y='
        max_turbo_taktfrequenz_GHz', hue='generation')
```

```
3     plt.title("Grundtaktfrequenz vs. Turbotaktfrequenz",
          fontsize=16)
4     plt.legend(title='Generation des Prozessors', fontsize
          =10, title_fontsize=12)
5     plt.xlabel("Grundtaktfrequenz [GHz]", fontsize=12)
6     plt.ylabel("Turbotaktfrequenz (GHz)", fontsize=12)
7     plt.show()
```

Quelltext 5.3: Code für den Scatterplot in Figur 5.1

Im Code wird zuerst ein neues Diagramm mit der Funktion *plt.figure()* erstellt. Dabei wird die Größe des Diagramms auf 10x6 Zoll festgelegt. In Zeile zwei wird mit der Funktion *sn.scatterplot()* bereits der Scatterplot erstellt. Dabei wird als Datenquelle das im Kapitel 5.1 erstellte DataFrame *df* verwendet. Die x-Achse wird mit der Spalte *grundtaktfrequenz_des_prozessors_GHz* und die y-Achse mit der Spalte *max_turbo_taktfrequenz_GHz* belegt. Die Farbe des Punktes repräsentiert die Generation des Prozessors, welche in der im Kapitel 5.2 erstellten Spalte *generation* gespeichert ist. In den Spalten drei bis sechs werden der Titel, die Legende, sowie die Beschriftungen der x- und y-Achse festgelegt. Mit der Funktion *plt.show()* in Spalte sieben wird das Diagramm angezeigt.

Die restlichen Diagramme, die in dieser Arbeit erstellt wurden, sind nach einem ähnlichen Schema wie eben beschrieben erstellt worden. Dabei wurden unterschiedliche Diagrammtypen sowie unterschiedliche Spalten des DataFrames verwendet. Alle Diagramme sind im Ordner *Datenanalyse* in der Datei *visualization.ipynb* zu finden.

6 Zusammenfassung (Max Stege)

6.1 Fazit (Max Stege)

Das vorliegende Projekt hat die Entwicklung der Intel Core i7-Prozessorreihe analysiert, um zentrale Fragestellungen der modernen Prozessorentwicklung zu beantworten. Dabei wurden Daten von der Herstellerwebsite erhoben, bereinigt, in einer relationalen Datenbank gespeichert und schließlich visualisiert. Im Fokus standen die Überprüfung des Mooreschen Gesetzes, die Identifikation von Metriken mit signifikanten Leistungssteigerungen und die Analyse von Bereichen, die eine Stagnation aufweisen.

Die Ergebnisse zeigen, dass das Mooresche Gesetz zwar in seinem ursprünglichen Sinne an Relevanz verloren hat, jedoch weiterhin als Leitfaden für technologische Entwicklungen dient. Die Leistungssteigerung moderner Prozessoren wird weniger durch höhere Taktraten erreicht – die sich zunehmend den physikalischen Grenzen nähern – als vielmehr durch eine höhere Kernanzahl, fortschrittliche Fertigungstechnologien und spezialisierte Architekturansätze. Stagnation zeigte sich vor allem in Metriken wie den Taktraten und der klassischen Single-Thread-Leistung, was auf thermische und energetische Beschränkungen zurückzuführen ist.

Dieses Projekt war jedoch nicht frei von Herausforderungen. Der Aufwand für die Umstrukturierung der CSV-Dateien und die Filterung irrelevanter Informationen war beträchtlich. Zudem war die Datenqualität nicht immer zufriedenstellend, da einige Werte unvollständig oder uneinheitlich waren, was zusätzliche Bereinigungsmaßnahmen erforderlich machte.

Ein weiteres Problem ergab sich bei der Auswahl geeigneter Analysemethoden und Visualisierungen. Trotz umfangreicher Bereinigungen blieben in den Daten leichte Inkonsistenzen, die die Interpretierbarkeit bestimmter Trends erschwerten. Diese Schwierigkeiten verdeutlichen, wie entscheidend eine qualitativ hochwertige und standardisierte Datenbasis für Datenanalysen ist.

Insgesamt konnte das Projekt jedoch eine robuste Pipeline zur Datenaufbereitung und -analyse etablieren, die als Grundlage für ähnliche Untersuchungen dienen kann. Die gewonnenen Erkenntnisse geben Einblicke in die technologische Entwicklung der Prozessoren und legen die Grundlage für weiterführende Fragestellungen.

6.2 Ausblick (Max Stege)

Die bei diesem Projekt aufgetretenen Schwierigkeiten bieten zugleich Ansatzpunkte für zukünftige Arbeiten. Insbesondere die Automatisierung der Datenbeschaffung, etwa durch Web-Scraping-Techniken oder das Schaffen standardisierter APIs, könnte den Arbeitsaufwand reduzieren und die Datenqualität verbessern. Ein weiterer wichtiger Schritt wäre die Integration von Validierungsmechanismen, um die Konsistenz der Daten bereits während der Bereinigungsphase sicherzustellen.

Zukunftsgerichtete Studien könnten sich mit der Analyse neuerer Technologien befassen, wie etwa der 3D-Chip-Architektur, Fortschritten in der Materialforschung (z. B. Graphen) oder dem wachsenden Einfluss spezialisierter Rechenmodule für Künstliche Intelligenz und maschinelles Lernen. Dabei könnten auch Predictive-Analytics-Methoden eingesetzt werden, um die Trends der kommenden Jahre zu prognostizieren.

Ein weiterer interessanter Ansatz wäre der Vergleich der Entwicklungen bei anderen Prozessorherstellern wie AMD oder Apple, um unterschiedliche Innovationsstrategien und deren Auswirkungen auf den Markt zu bewerten. Zudem könnten breiter angelegte Analysen mit zusätzlichen Metriken durchgeführt werden, um ein umfassenderes Bild der technischen Fortschritte in der Halbleiterbranche zu erhalten.

Schließlich wäre eine engere Verknüpfung der Datenanalyse mit ökologischen Aspekten von Interesse. Der Energieverbrauch und die Nachhaltigkeit der Prozessorfertigung könnten entscheidende Faktoren für die zukünftige Entwicklung der Branche werden.

Zusammenfassend lässt sich sagen, dass dieses Projekt nicht nur wertvolle Einblicke in die Entwicklung moderner Prozessoren geliefert hat, sondern auch weitere Impulse für zukünftige Untersuchungen gibt.

Bibliography

[1] *5. Data Structures*. Python documentation. URL: https://docs.python.org/3/tutorial/datastructures.html (visited on 12/01/2024).

[2] *5.1. Table Basics*. PostgreSQL Documentation. Nov. 21, 2024. URL: https://www.postgresql.org/docs/17/ddl-basics.html (visited on 11/30/2024).

[3] adegeo. *Regular Expression Language - Quick Reference - .NET*. June 18, 2022. URL: https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference (visited on 12/01/2024).

[4] *awk(1): pattern scanning/processing - Linux man page*. URL: https://linux.die.net/man/1/awk (visited on 11/28/2024).

[5] Min Chen et al. *Big Data: Related Technologies, Challenges and Future Prospects*. SpringerBriefs in Computer Science. Cham: Springer International Publishing, 2014. ISBN: 978-3-319-06244-0 978-3-319-06245-7. DOI: 10.1007/978-3-319-06245-7. URL: https://link.springer.com/10.1007/978-3-319-06245-7 (visited on 11/13/2024).

[6] *Data Visualization in Python: Overview, Libraries & Graphs | Simplilearn*. Simplilearn.com. URL: https://www.simplilearn.com/tutorials/python-tutorial/data-visualization-in-python (visited on 12/01/2024).

[7] ewendorf. *The Algorithmic Expert's Tool: An introduction to the Python programming language*. ML Conference. Apr. 24, 2019. URL: https://mlconference.ai/blog/introduction-to-the-r-programming-language/ (visited on 11/30/2024).

[8] *Example gallery — seaborn 0.13.2 documentation*. URL: https://seaborn.pydata.org/examples/index.html (visited on 12/01/2024).

[9] I7. *Intel Core i7 Generation 4 bis 14*. i7. Nov. 4, 2024. URL: https://www.intel.de/content/www/de/de/products/compare.html?productIds=212279,212280,212047,212048,212251,199325,199335,199314,199316,199318,191048,191792,193738,190885,186604,148263,140642,126684,97129,93339,88200,88195,87718,88040,80807,80808,80809,80814,80806,77656,76642,75121,75122,75123,75124,75125,236781,236794,236854,236783,236789,230492,230490,230491,230500,230489,134596,134591,134592,134594,134595,129948,126686,97122,97128,88196 (visited on 11/04/2024).

[10] *janitor package - RDocumentation*. URL: https://www.rdocumentation.org/packages/janitor/versions/2.2.0 (visited on 11/28/2024).

[11] Mohsen Marjani et al. "Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges". In: *IEEE Access* 5 (2017). Conference Name: IEEE Access, pp. 5247–5261. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2689040. URL: https://ieeexplore.ieee.org/abstract/document/7888916 (visited on 11/13/2024).

[12] *Matplotlib — Visualization with Python*. URL: https://matplotlib.org/ (visited on 12/01/2024).

[13] *Microsoft SQL Server System Properties*. URL: https://db-engines.com/en/system/Microsoft+SQL+Server (visited on 11/30/2024).

[14] *Moore's Law*. URL: https://www.intel.com/content/www/us/en/newsroom/resources/moores-law.html (visited on 12/01/2024).

[15] *Most wanted data science skills U.S. 2019*. Statista. URL: https://www.statista.com/statistics/1016247/united-states-wanted-data-science-skills/ (visited on 12/01/2024).

[16] *MySQL System Properties*. URL: https://db-engines.com/en/system/MySQL (visited on 11/30/2024).

[17] *Oracle System Properties*. URL: https://db-engines.com/en/system/Oracle (visited on 11/30/2024).

[18] *ORM Quick Start — SQLAlchemy 2.0 Documentation*. URL: https://docs.sqlalchemy.org/en/20/orm/quickstart.html (visited on 12/01/2024).

[19] *pandas - Python Data Analysis Library*. URL: https://pandas.pydata.org/ (visited on 12/01/2024).

[20] *pandas.DataFrame — pandas 2.2.3 documentation*. URL: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html#pandas.DataFrame (visited on 12/01/2024).

[21] *Plot types — Matplotlib 3.9.3 documentation*. URL: https://matplotlib.org/stable/plot_types/index.html (visited on 12/01/2024).

[22] *Popular technologies in the data science tech stack 2024*. Statista. URL: https://www.statista.com/statistics/1292394/popular-technologies-in-the-data-science-tech-stack/ (visited on 12/01/2024).

[23] *PostgreSQL System Properties*. URL: https://db-engines.com/en/system/PostgreSQL (visited on 11/30/2024).

[24] *PostgreSQL: About*. URL: https://www.postgresql.org/about/ (visited on 11/30/2024).

[25] *Python Release Python 3.12.7*. Python.org. URL: https://www.python.org/downloads/release/python-3127/ (visited on 12/01/2024).

[26] *python-dotenv: Read key-value pairs from a .env file and set them as environment variables*. Version 1.0.1. URL: https://github.com/theskumar/python-dotenv (visited on 12/01/2024).

[27] *R: What is R?* URL: https://www.r-project.org/about.html (visited on 11/28/2024).

[28] *SQLAlchemy*. URL: https://www.sqlalchemy.org (visited on 12/01/2024).

[29] *SQLite System Properties*. URL: https://db-engines.com/en/system/SQLite (visited on 11/30/2024).

[30] Michael Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (Apr. 6, 2021), p. 3021. ISSN: 2475-9066. DOI: 10.21105/joss.03021. URL: https://joss.theoj.org/papers/10.21105/joss.03021 (visited on 12/01/2024).