

Abgabe 1 für Computergestützte Methoden

Gruppe 100

Name	Matrikelnummer
Shainthavi Suthakaran	4250125
Melissa Tursucu	4020830
Gözde Ünal	4250543

Abgabedatum: 02.12.2024

Inhaltsverzeichnis

1	Der zentrale Grenzwertsatz	2
1.1	Aussage	2
1.2	Erklärung der Standardisierung	2
1.3	Anwendungen	3
2	Datenhaltung & -aufbereitung	4
2.1	Datenverarbeitung	4
2.1.1	Aufgabe 1	4
2.1.2	Aufgabe 2	5
2.1.3	Aufgabe 3	6
2.2	Datenhaltung	7
2.2.1	Aufgabe 2	7
2.2.2	Aufgabe 3	9
2.2.3	Aufgabe 4	10
2.2.4	Aufgabe 5	16

1 Der zentrale Grenzwertsatz

Der zentrale Grenzwertsatz (ZGS) ist ein fundamentales Resultat der Wahrscheinlichkeitstheorie, das die Verteilung von Summen unabhängiger, identisch verteilter (i.i.d.) Zufallsvariablen (ZV) beschreibt. Er besagt, dass unter bestimmten Voraussetzungen die Summe einer großen Anzahl solcher ZV annähernd normalverteilt ist, unabhängig von der Verteilung der einzelnen ZV. Dies ist besonders nützlich, da die Normalverteilung gut untersucht und mathematisch handhabbar ist.

1.1 Aussage

Sei X_1, X_2, \dots, X_n eine Folge von i.i.d. ZV mit dem Erwartungswert $\mu = \mathbb{E}(X_i)$ und der Varianz $\sigma^2 = \text{Var}(X_i)$, wobei $0 < \sigma^2 < \infty$ gilt. Dann konvergiert die standardisierte Summe Z_n dieser ZV für $n \rightarrow \infty$ in Verteilung gegen eine Standardnormalverteilung:¹

$$Z_n = \frac{\sum_{i=1}^n X_i - n\mu}{\sigma\sqrt{n}} \xrightarrow{d} \mathcal{N}(0, 1). \quad (1)$$

Das bedeutet, dass für große n die Summe der ZV näherungsweise normalverteilt ist mit Erwartungswert $n\mu$ und Varianz $n\sigma^2$:

$$\sum_{i=1}^n X_i \sim \mathcal{N}(n\mu, n\sigma^2). \quad (2)$$

1.2 Erklärung der Standardisierung

Um die Summe der ZV in eine Standardnormalverteilung zu transformieren, subtrahiert man den Erwartungswert $n\mu$ und teilt durch die Standardabweichung $\sigma\sqrt{n}$. Dies führt zu der obigen Formel (1). Die Darstellung (2) ist für $n \rightarrow \infty$ nicht wohldefiniert.

¹Der zentrale Grenzwertsatz hat verschiedene Verallgemeinerungen. Eine davon ist der **Lindeberg-Feller-Zentrale-Grenzwertsatz** [1, Seite 328], der schwächere Bedingungen an die Unabhängigkeit und die identische Verteilung der ZV stellt.

1.3 Anwendungen

Der ZGS wird in vielen Bereichen der Statistik und der Wahrscheinlichkeitstheorie angewendet. Typische Beispiele sind:

- **Meinungsumfragen²:** Bei Meinungsumfragen wird der Zentrale Grenzwertsatz verwendet, um die Verteilung des Stichprobenmittelwerts zu analysieren. Beispielsweise kann die Zustimmung zu einer politischen Partei in einer großen Population basierend auf einer zufälligen Stichprobe geschätzt werden. Durch den ZGWS folgt der Mittelwert der Zustimmung bei ausreichend großen Stichproben einer Normalverteilung.
- **Qualitätskontrolle in der Produktion²:** In der industriellen Qualitätskontrolle wird der ZGWS genutzt, um die Schwankungen des Mittelwerts von Produktmerkmalen, wie dem Gewicht oder der Größe, zu überwachen. Wenn beispielsweise eine Stichprobe von 30 Verpackungen gezogen wird, nähert sich der Durchschnitt des Verpackungsgewichtes der Normalverteilung an. Dies ermöglicht es, Abweichungen vom Sollgewicht einfach zu erkennen und die Prozessqualität zu bewerten.

²Die Anwendung des zentralen Grenzwertsatzes ist vielfältig. Besonders in der Wissenschaft und Mathematik wird dieser genutzt [3]

2 Datenhaltung & -aufbereitung

2.1 Datenverarbeitung

2.1.1 Aufgabe 1

Der zu untersuchende Datensatz umfasst detaillierte Informationen zum Betrieb eines Fahrradverleihs aus dem Jahr 2023. Unsere Gruppe wird sich mit der Station *William St & Pine St* beschäftigen.

Der Datensatz dokumentiert folgende Informationen:

- *group*: Zuordnung der Stationen für die Gruppen
- *station*: Stationsname des Fahrradverleihs
- *date*: das Datum, an dem die Daten erfasst wurden
- *day_of_year*: Numerische Darstellung des Tages im Jahr
- *day_of_week*: Numerische Darstellung des Tages als Wochentag
- *month_of_year*: Numerische Darstellung des Monats
- *precipitation*: Niederschlag in mm
- *windspeed*: Windgeschwindigkeit in mph
- *min_temperature*: Der niedrigste Temperaturwert des Tages in °F
- *average_temperature*: Die durchschnittliche Temperatur des Tages in °F
- *max_temperature*: Der höchste Temperaturwert des Tages in °F
- *count*: Anzahl der Ausleihen

Der zu analysierende Datensatz beinhaltet 12 Spalten und 364 Zeilen. Daraus lässt sich schließen, dass bei insgesamt 365 Tagen eines Jahres genau ein Tag im Datensatz fehlt. Dabei handelt es sich um den 16.10.23.

Bei der Analyse der Daten ist anzumerken, dass die Anzahl an Ausleihen in den wärmeren Monaten tendenziell höher ist, als in den kälteren Monaten. Die meisten Ausleihen finden im Monat Juli statt. Somit können wir die Erkenntnis gewinnen, dass die Temperatur einen signifikanten Einfluss auf die Ausleihen von Fahrrädern hat.

Bedauerlicherweise weist dieser Datensatz fehlerhafte Einträge auf. Am 24.01.23 wurde eine Ausleihanzahl von -1 angegeben, was unrealistisch ist. Des Weiteren gibt es auch Daten, die mit NAs versehen sind, wie zum Beispiel am 20.05.23 bei *average_temperature*.

2.1.2 Aufgabe 2

Um den Datensatz in Excel zu importieren, öffnen wir Excel und klicken auf den Reiter *Daten* der Ribbon-Leiste. Dort klicken wir folgendermaßen durch:

- *Daten abrufen → Aus Datei → Aus Text/CSV*

Nun können wir unsere CSV-Datei in Excel importieren. Das sieht folgendermaßen aus:

group	gs	date	day_of_year	day_of_week	month_of_year	precipitation	windspeed	min_temperature	average_temperature	max_temperature	count
1	Ave & E 18 St	01.01.2023	1	1		0	1007.42	50			56 117
1	Ave & E 18 St	02.01.2023	2	1		2	649.39	46			55 125
1	Ave & E 18 St	03.01.2023	3	1		36	582.46	48			50 96
1	Ave & E 18 St	04.01.2023	4	1		1	649.45	51			61 154
1	Ave & E 18 St	05.01.2023	5	1		5	626.45	49			51 156
1	Ave & E 18 St	06.01.2023	6	1		25	805.39	44			47 166
1	Ave & E 18 St	07.01.2023	7	1		0	1208.36	41			47 120
1	Ave & E 18 St	08.01.2023	8	1		0	917.32	37			41 123
1	Ave & E 18 St	09.01.2023	9	2	NA	2	984.37	40			46 199
1	Ave & E 18 St	10.01.2023	10	3	1	0	962.35	39			41 151
1	Ave & E 18 St	11.01.2023	11	4	1	0	783.32	38			40 164
1	Ave & E 18 St	12.01.2023	12	5	1	27	1186.1	42			53 121
1	Ave & E 18 St	13.01.2023	13	6	1	3	1901.38	49			52 134
1	Ave & E 18 St	14.01.2023	14	7	1	0	2259.30	35			38 149
1	Ave & E 18 St	15.01.2023	15	1	1	0	2304.29	33			39 129
1	Ave & E 18 St	16.01.2023	16	2	1	0	1655.29	36			48 146
1	Ave & E 18 St	17.01.2023	17	3	1	0	748.31	39			45 147
1	Ave & E 18 St	18.01.2023	18	4	1	0	1432.37	46			56 168
1	Ave & E 18 St	19.01.2023	19	5	1	96	917.37	42			45 71
1	Ave & E 18 St	20.01.2023	20	6	1	0	1476.39	44			50 172
1	Ave & E 18 St	21.01.2023	21	7	1	0	1141.NA	40			43 150
1	Ave & E 18 St	22.01.2023	22	1	1	47	828.36	38			40 133
1	Ave & E 18 St	23.01.2023	23	2	1	24	1722.35	38			39 81
1	Ave & E 18 St	24.01.2023	24	3	1	0	1644.35	39			48 153
1	Ave & E 18 St	25.01.2023	25	4	1	97	123.31	38			52 105
1	Ave & E 18 St	26.01.2023	26	5	1	23	2147.40	46			52 136
1	Ave & E 18 St	27.01.2023	27	6	1	0	1186.32	39			43 184
1	Ave & E 18 St	28.01.2023	28	7	1	0	1297.33	41			53 164
1	Ave & E 18 St	29.01.2023	29	1	1	0	1029.35	43			47 160

Importierter Datensatz

Um die zu untersuchenden Daten zu extrahieren, nutzen wir den Filterpfeil der Spalte *group*, entfernen das Häkchen neben *Alles auswählen* und setzen anschließend nur ein Häkchen bei der Zahl 100, sodass unser Datensatz nun so aussieht:

group	gs	date	day_of_year	day_of_week	month_of_year	precipitation	windspeed	min_temperature	average_temperature	max_temperature	count
100	William St & Pine St	14.11.2023	318	3	11	-1	1342.39	47			55 93
100	William St & Pine St	01.01.2023	1	1		0	1007.42	50			56 55
100	William St & Pine St	10.01.2023	10	3	1	0	962.35	39			41 64
100	William St & Pine St	10.04.2023	100	2	4	0	1163.38	46			58 97
100	William St & Pine St	11.04.2023	101	3	4	0	1432.45	56			73 111
100	William St & Pine St	12.04.2023	102	4	4	0	1297.56	68			84 124
100	William St & Pine St	13.04.2023	103	5	4	0	1118.53	69			85 118
100	William St & Pine St	14.04.2023	104	6	4	0	94.55	68			82 109
100	William St & Pine St	16.04.2023	106	1	4	0	693.52	59			66 101
100	William St & Pine St	18.04.2023	108	3	4	0	1924.45	55			58 88
100	William St & Pine St	19.04.2023	109	4	4	0	1186.43	51			64 111
100	William St & Pine St	11.01.2023	11	4	1	0	783.NA	38			40 62
100	William St & Pine St	20.04.2023	110	5	4	0	693.43	50			59 122
100	William St & Pine St	21.04.2023	111	6	4	0	1141.47	54			66 117
100	William St & Pine St	24.04.2023	114	2	4	0	1141.44	53			59 109
100	William St & Pine St	25.04.2023	115	3	4	0	85.45	52			58 114
100	William St & Pine St	26.04.2023	116	4	4	0	94.42	50			56 127
100	William St & Pine St	27.04.2023	117	5	4	0	761.48	52			57 109
100	William St & Pine St	05.05.2023	125	6	5	0	671.44	52			61 94
100	William St & Pine St	06.05.2023	126	7	5	0	805.48	58			72 91
100	William St & Pine St	07.05.2023	127	1	5	0	1007.53	61			73 112
100	William St & Pine St	08.05.2023	128	2	5	0	1007.57	66			78 105
100	William St & Pine St	09.05.2023	129	3	5	0	85.50	60			63 118
100	William St & Pine St	10.05.2023	130	4	5	0	872.44	55			66 137
100	William St & Pine St	11.05.2023	131	5	5	0	984.51	62			75 140
100	William St & Pine St	12.05.2023	132	6	5	0	917.56	68			83 108
100	William St & Pine St	13.05.2023	133	7	5	0	962.64	71			80 125
100	William St & Pine St	14.05.2023	134	1	5	0	828.56	65			69 83
100	William St & Pine St	15.05.2023	135	2	5	0	1208.51	62			72 130

Gefilterter Datensatz nach Gruppennummer 100

2.1.3 Aufgabe 3

Damit wir die höchste mittlere Temperatur in $^{\circ}C$ bestimmen können, fügen wir eine zusätzliche Spalte mit dem Namen **average c** ein.

Um die Temperatur von $^{\circ}F$ in $^{\circ}C$ umzuwandeln, wird die folgende Formel angewendet³:

$$^{\circ}C = (^{\circ}F - 32) * \frac{5}{9} \quad (3)$$

Um die mittlere Temperatur von Fahrenheit in Celsius umzurechnen, wird die oben genannte Formelverwendet. Diese wird direkt in die erste Zelle unter dem Spaltennamen der neu erstellten Spalte eingetragen.

average_c ▾	
=([@average_temperature]-32)*5/9	

Formeingeabe zur Umrechnung

Durch `[@average_temperature]` wird die Formel auf jede Zeile unter *average temperature* angewendet.

Anschließend klicken wir auf den Filterpfeil und sortieren die Daten nach absteigender Größe.

date	day_of_year	day_of_week	month_of_year	precipitation	windspeed	min_temperature	average_temperature	max_temperature	count	average_c
20.05.2023	140	7	5		165	94.57	NA		63	31 #WERT!
28.07.2023	209	6	7		4	738.76	83		91	113 28,33333333
05.09.2023	248	3	9		0	693.74	82		93	118 27,7777778
06.09.2023	249	4	9		0	537.74	82		93	123 27,7777778
12.07.2023	193	4	7		0	984.72	81		90	125 27,2222222
07.08.2023	250	5	9		0	85.73	81		92	120 27,2222222
29.07.2023	210	7	7		64	1432.70	81		90	107 27,2222222
27.07.2023	208	5	7		2	123.73	80		92	129 26,6666667

Sortierte Daten für die mittlere Temperatur in $^{\circ}C$

Die höchste mittlere Temperatur wurde am 28.07.23 erfasst und beträgt somit 28,33 $^{\circ}C$.

³Die Formel für die Umrechnung von Fahrenheit in Celsius wurde von [2] entnommen

2.2 Datenhaltung

2.2.1 Aufgabe 2

Im Folgenden entwerfen wir mithilfe der 1. und 2. Normalform ein Datenbankschema. Die Normalformen sind folgendermaßen definiert⁴:

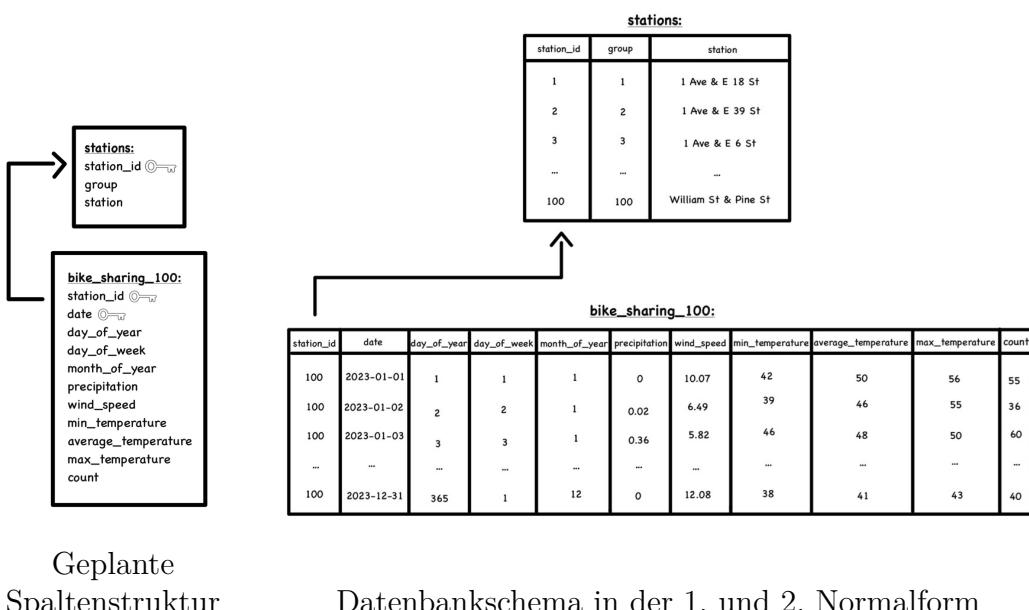
Eine Relation befindet sich in der **ersten Normalform**, wenn:

- alle Werte atomar sind
- alle Zeilen eindeutig identifizierbar sind
- alle Spalten eindeutige Namen besitzen

Eine Relation befindet sich in der **zweiten Normalform**, wenn:

- wenn sie sich bereits in der ersten Normalform befindet
- jedes Nicht-Schlüsselattribut vollständig vom Primärschlüssel abhängt

Um diese nun in unsere Datenbank zu implementieren, teilen wir die Daten in zwei Tabellen auf. Unsere Idee dabei ist es, die zeitabhängigen Variablen - wie das Datum und die Wetterbedingungen - von den stationären Variablen wie *group* und *station* zu trennen.



Geplante
Spaltenstruktur

Datenbankschema in der 1. und 2. Normalform

⁴Die Normalisierung von Datenbanken sind grundlegend um Daten zu strukturieren. Die Definitionen wurden von [4] und [5] entnommen.

Der Pfeil stellt die Verbindung der beiden Tabellen dar. Durch die Spalte *station_id* werden beide Tabellen miteinander verknüpft. Das Schema sieht wie folgt aus:

- stations(*station_id*#, *group*, *station*)
- bike_sharing_100 (*station_id*#, *date*#, *day_of_year*, *day_of_week*, *month_of_year*, *precipitation*, *wind_speed*, *min_temperature*, *average_temperature*, *max_temperature*, *count*)

Es ist hinzuzufügen, dass *station_id* in der Tabelle *bike_sharing_100* allein als Fremdschlüssel dient, jedoch in Kombination mit *date* ein Primärschlüssel ist.

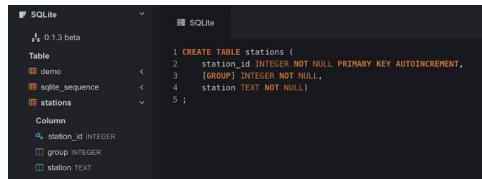
Die erste Normalform wird erfüllt, da alle Zellen in beiden Tabellen nur genau einen Wert beinhalten. In der Tabelle *stations* werden die Zeilen unter der Spalte *station_id* eindeutig identifizierbar gemacht, da wir *station_id* als Primärschlüssel nutzen. Dadurch, dass *station_id* ebenfalls in der Tabelle *bike_sharing_100* auftaucht, haben wir durch diese Spalte und der Spalte *date* auch die Eindeutigkeit der Zeilen geschaffen.

Da die erste Normalform erfüllt ist und alle Nicht-Schlüsselattribute ausschließlich vom gesamten Primärschlüssel abhängig sind, ist auch die zweite Normalform gegeben.

2.2.2 Aufgabe 3

Um nun mithilfe der *DDL (Data Definition Language)* die Datenstruktur in SQL zu definieren, öffnen wir SQLite und gehen folgende Schritte durch:

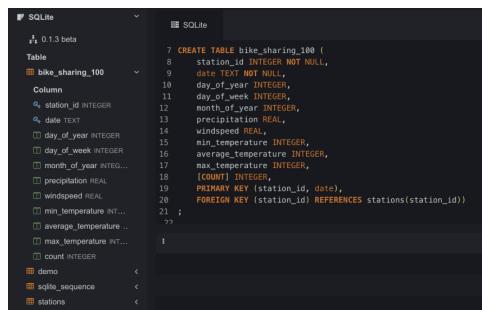
```
CREATE TABLE stations (
    station_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    [GROUP] INTEGER NOT NULL,
    station TEXT NOT NULL
);
```

A screenshot of the SQLite Manager interface. On the left, the sidebar shows the database version as 0.1.3 beta and lists tables: demo, sqlite_sequence, and stations. The stations table has columns: station_id (INTEGER), group (INTEGER), and station (TEXT). On the right, the main pane displays the SQL code for creating the stations table:

```
CREATE TABLE stations (
    station_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    [GROUP] INTEGER NOT NULL,
    station TEXT NOT NULL
);
```

Definition der Tabelle *stations*

```
CREATE TABLE bike_sharing_100 (
    station_id INTEGER NOT NULL,
    date TEXT NOT NULL,
    day_of_year INTEGER,
    day_of_week INTEGER,
    month_of_year INTEGER,
    precipitation REAL,
    windspeed REAL,
    min_temperature INTEGER,
    average_temperature INTEGER,
    max_temperature INTEGER,
    [COUNT] INTEGER,
    PRIMARY KEY (station_id, date),
    FOREIGN KEY (station_id) REFERENCES stations(station_id)
);
```

A screenshot of the SQLite Manager interface. On the left, the sidebar shows the database version as 0.1.3 beta and lists tables: bike_sharing_100, demo, sqlite_sequence, and stations. The bike_sharing_100 table has columns: station_id (INTEGER), date (TEXT), day_of_year (INTEGER), day_of_week (INTEGER), month_of_year (INTEGER), precipitation (REAL), windspeed (REAL), min_temperature (INTEGER), average_temperature (INTEGER), max_temperature (INTEGER), and count (INTEGER). The primary key is defined as (station_id, date), and a foreign key constraint links station_id to the stations table. On the right, the main pane displays the SQL code for creating the bike_sharing_100 table:

```
CREATE TABLE bike_sharing_100 (
    station_id INTEGER NOT NULL,
    date TEXT NOT NULL,
    day_of_year INTEGER,
    day_of_week INTEGER,
    month_of_year INTEGER,
    precipitation REAL,
    windspeed REAL,
    min_temperature INTEGER,
    average_temperature INTEGER,
    max_temperature INTEGER,
    [COUNT] INTEGER,
    PRIMARY KEY (station_id, date),
    FOREIGN KEY (station_id) REFERENCES stations(station_id)
);
```

Definition der Tabelle *bike_sharing_100*

2.2.3 Aufgabe 4

Um den Datensatz passend in unsere Tabellen zu importieren, nehmen wir einige Veränderungen an der ursprünglichen Tabelle *bike_sharing_with_NAs* vor. Zunächst müssen wir dafür die CSV-Datei importieren. Da wir *DB Browser for SQLite* benutzen, müssen wir uns folgendermaßen durchklicken:

- *File → Import → Table from CSV-file*

```

CREATE TABLE stations (
    station_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    [group] INTEGER NOT NULL,
    station TEXT NOT NULL
);

CREATE TABLE bike_sharing_100 (
    station_id INTEGER NOT NULL,
    date TEXT NOT NULL,
    day_of_year INTEGER,
    day_of_week INTEGER,
    month_of_year INTEGER,
    precipitation REAL,
    windspeed REAL,
    min_temperature INTEGER,
    average_temperature INTEGER,
    max_temperature INTEGER,
    [count] INTEGER,
    PRIMARY KEY (station_id, date),
    FOREIGN KEY (station_id) REFERENCES stations(station_id)
);

```

Aufbau der importierten Tabelle *bike_sharing_data_with_NAs*

Nachdem wir die Tabelle erfolgreich importiert haben, ist zu erkennen, dass die Spalten mit generischen Namen wie c1, c2, c3 usw. versehen sind. Um diese Bezeichner durch die Spaltennamen der Tabelle *bike_sharing_with_NAs* zu ersetzen, verwenden wir den folgenden Befehl:

```
ALTER TABLE bike_sharing_data_with_NAs
RENAME COLUMN old_column TO 'new_column';
```

Mit diesem Befehl passen wir die Spaltenbezeichner an, um die Tabelle besser an die Struktur des tatsächlichen Datensatzes anzupassen.

```

ALTER TABLE bike_sharing_100
RENAME COLUMN c1 TO "group";
ALTER TABLE bike_sharing_data_with_NAs
RENAME COLUMN c2 TO "station";
ALTER TABLE bike_sharing_data_with_NAs
RENAME COLUMN c3 TO "date";
ALTER TABLE bike_sharing_data_with_NAs
RENAME COLUMN c4 TO "day_of_year";
ALTER TABLE bike_sharing_data_with_NAs
RENAME COLUMN c5 TO "day_of_week";
ALTER TABLE bike_sharing_data_with_NAs
RENAME COLUMN c6 TO "month_of_year";
ALTER TABLE bike_sharing_data_with_NAs
RENAME COLUMN c7 TO "precipitation";
ALTER TABLE bike_sharing_data_with_NAs
RENAME COLUMN c8 TO "windspeed";

```

Umbenennung von c1 - c8

```

ALTER TABLE bike_sharing_data_with_NAs
RENAME COLUMN c9 TO "min_temperature";
ALTER TABLE bike_sharing_data_with_NAs
RENAME COLUMN c10 TO "average_temperature";
ALTER TABLE bike_sharing_data_with_NAs
RENAME COLUMN c11 TO "max_temperature";
ALTER TABLE bike_sharing_data_with_NAs
RENAME COLUMN c12 TO "count";

```

Umbenennung von c9 - c12

Dadurch haben wir den Datensatz korrekt beschriftet und die Spaltennamen entsprechend angepasst.

	group	station	date	day_of...	day_of...	month...	precipit...	windspe...	min_te...	averag...	max_te...	count
1	1 Ave ...	2023-0...	1	1	1	0	10.07	42	50	56	117	
1	1 Ave ...	2023-0...	2	2	1	0.02	6.49	39	46	55	125	
1	1 Ave ...	2023-0...	3	3	1	0.01	6.49	45	51	61	154	
1	1 Ave ...	2023-0...	4	4	1	0.05	6.26	45	49	51	156	
1	1 Ave ...	2023-0...	5	5	1	0.25	8.85	39	44	47	166	
1	1 Ave ...	2023-0...	6	6	1	0	12.08	36	41	47	120	
1	1 Ave ...	2023-0...	7	7	1	0	9.17	32	37	41	123	
1	1 Ave ...	2023-0...	8	1	1	0.02	9.84	37	40	46	199	
1	1 Ave ...	2023-0...	9	2	NA	0.02	9.62	35	39	41	151	
1	1 Ave ...	2023-0...	10	3	1	0	7.83	32	38	40	164	
1	1 Ave ...	2023-0...	11	4	1	0	11.86	-1	42	53	121	
1	1 Ave ...	2023-0...	12	5	1	0.27	19.01	38	49	52	134	
1	1 Ave ...	2023-0...	13	6	1	0.03	19.01	38	49	52	134	

Visualisierung der Tabelle *bike_sharing_with_NAs*

Bei der Visualisierung der CSV-Datei fällt auf, dass die Spaltennamen fälschlicherweise erneut in der ersten Zeile stehen. Um dies zu korrigieren, verwenden wir den folgenden Befehl:

```
DELETE FROM bike_sharing_data_with_NAs
WHERE column_name = 'column_name';
```

Auf diese Weise werden alle Zellen in der ersten Zeile entfernt.

61	DELETE FROM bike_sharing_data_with_NAs
62	WHERE [GROUP] = 'group'
63	AND station = 'station'
64	AND date = 'date'
65	AND day_of_year = 'day_of_year'
66	AND day_of_week = 'day_of_week'
67	AND month_of_year = 'month_of_year'
68	AND precipitation = 'precipitation'
69	AND windspeed = 'windspeed'
70	AND min_temperature = 'min_temperature'
71	AND average_temperature = 'average_temperature'
72	AND max_temperature = 'max_temperature'
73	AND [COUNT] = 'count';

Eliminierung der ersten Zeile von *bike_sharing_with_NAs*

Damit haben wir die Tabelle *bike_sharing_with_NAs* erfolgreich geändert.

	group	station	date	day_of_week	day_of_month	month	precip	windspeed	min_temp	average_temp	max_temp	count
1	1 Ave ...	1	2023-0...	1	1	0	18.07	42	50	56	117	
1	1 Ave ...	2	2023-0...	2	1	0.02	6.49	39	46	55	125	
1	1 Ave ...	3	2023-0...	3	1	0.36	5.82	46	48	58	96	
1	1 Ave ...	4	2023-0...	4	1	0.01	6.49	45	51	61	154	
1	1 Ave ...	5	2023-0...	5	1	0.05	6.26	45	49	51	156	
1	1 Ave ...	6	2023-0...	6	1	0.25	8.05	39	44	47	166	
1	1 Ave ...	7	2023-0...	7	1	0	12.08	36	41	47	120	
1	1 Ave ...	8	2023-0...	8	1	0	9.17	32	37	41	123	
1	1 Ave ...	9	2023-0...	9	2	NA	0.02	9.84	37	40	46	199
1	1 Ave ...	10	2023-0...	10	3	1	0	9.62	35	39	41	151
1	1 Ave ...	11	2023-0...	11	4	1	0	7.83	32	38	40	164
1	1 Ave ...	12	2023-0...	12	5	1	0.27	11.06	-1	42	53	121
1	1 Ave ...	13	2023-0...	13	6	1	0.03	19.01	38	49	52	134
1	1 Ave ...	14	2023-0...	14	7	1	0	22.59	30	35	38	149

Aktualisierter Output der Tabelle *bike_sharing_with_NAs*

Nachdem wir diese Tabelle erfolgreich angepasst haben, können wir nun den Datensatz auf unsere Tabellen *stations* und *bike_sharing_100* übertragen. Dabei fangen wir mit der Tabelle *stations* an.

Mit dem folgenden Code fügen wir die Daten in die Tabelle *stations* ein, wobei die Spalten *group* und *station* verwendet werden:

```
INSERT INTO stations ([GROUP], station)
SELECT
    CAST([GROUP] AS INTEGER) AS [GROUP],
    station AS station
FROM bike_sharing_data_with_NAs
GROUP BY station, [GROUP];
```

Da bei *bike_sharing_with_NAs* alle Spalten standardmäßig als TEXT importiert wurden, nutzen wir SELECT, um die Daten zu wählen und wandeln die *group*-Spalte des importierten Datensatzes mit CAST in einen Integer (also in eine ganze Zahl) um. Mit GROUP BY gruppieren wir die Daten nach den Spalten *station* und *group*, um möglicherweise redundante oder doppelte Werte zu aggregieren.

```
77 INSERT INTO stations ([GROUP], station)
78 SELECT
79     CAST([GROUP] AS INTEGER) AS [GROUP],
80     station AS station
81 FROM bike_sharing_data_with_NAs
82 GROUP BY station, [GROUP];
```

Befehle zur Importierung der Tabelle *stations*

Unsere Tabelle *stations* hat nun die korrekten Spaltenbezeichnungen und Datentypen.

station_id	group	station
1	1	1 Ave & E 18 St
2	2	1 Ave & E 39 St
3	3	1 Ave & E 6 St
4	4	1 Ave & E 68 St
5	5	10 Ave & W 14 St
6	6	11 Ave & W 27 St
7	7	11 Ave & W 59 St
8	8	3 Ave & E 62 St
9	9	31 Ave & E 34 St
10	10	4 Ave & E 12 St
11	11	5 Ave & E 103 St
12	12	5 Ave & E 87 St
13	13	6 Ave & W 16 St
14	14	8 Ave & W 24 St

Fertige Tabelle *stations*

Nun müssen wir ebenfalls die von uns erstellte Tabelle *bike_sharing_100* befüllen und die Datentypen auf unsere Spalten anpassen.

Dazu nutzen wir folgenden Befehl:

```
INSERT INTO bike_sharing_100 (
    CAST([GROUP] AS INTEGER) AS [GROUP],
    station_id, date, day_of_year, day_of_week,
    month_of_year, precipitation, windspeed,
    min_temperature, average_temperature, max_temperature, COUNT
)
SELECT
    s.station_id,
    CAST(b.date AS TEXT) AS date,
    CAST(b.day_of_year AS INTEGER) AS day_of_year,
    CAST(b.day_of_week AS INTEGER) AS day_of_week,
    CAST(b.month_of_year AS INTEGER) AS month_of_year,
    CAST(b.precipitation AS REAL) AS precipitation,
    CAST(b.windspeed AS REAL) AS windspeed,
    CAST(b.min_temperature AS INTEGER) AS min_temperature,
    CAST(b.average_temperature AS INTEGER) AS average_temperature,
    CAST(b.max_temperature AS INTEGER) AS max_temperature,
    CAST(b.count AS INTEGER) AS COUNT
FROM bike_sharing_data_with_NAs b
JOIN stations s
ON b.station = s.station
;
```

Durch das b von `FROM bike_sharing_data_with_NAs b`, das s von `JOIN stations s und dem Befehl darunter ON b.station = s.station wird die Spalte station beider Tabellen miteinander verknüpft.`

```
0.1.3 beta
Table
  bike_sharing_100      < 86 INSERT INTO bike_sharing_100 (
  bike_sharing_data_with... < 87   station_id, date, day_of_year, day_of_week,
  demo                   < 88   month_of_year, precipitation, windspeed,
  sqlite_sequence         < 89   min_temperature, average_temperature, max_temperature, COUNT
  stations                < 90 )
  MariaDB                < 91 SELECT
  PostgreSQL              < 92   s.station_id,
  MS SQL                 < 93   CAST(b.date AS TEXT) AS date,
  PostgreSQL              < 94   CAST(b.day_of_year AS INTEGER) AS day_of_year,
  PostgreSQL              < 95   CAST(b.day_of_week AS INTEGER) AS day_of_week,
  PostgreSQL              < 96   CAST(b.month_of_year AS INTEGER) AS month_of_year,
  PostgreSQL              < 97   CAST(b.precipitation AS REAL) AS precipitation,
  PostgreSQL              < 98   CAST(b.windspeed AS REAL) AS windspeed,
  PostgreSQL              < 99   CAST(b.min_temperature AS REAL) AS min_temperature,
  PostgreSQL              < 100  CAST(b.average_temperature AS INTEGER) AS average_temperature,
  PostgreSQL              < 101  CAST(b.max_temperature AS INTEGER) AS max_temperature,
  PostgreSQL              < 102  CAST(b.count AS INTEGER) AS COUNT
  103 FROM bike_sharing_data_with_NAs b
  104 JOIN stations s
  105 ON b.station = s.station
  106 ;
```

Befehle zur Importierung der Tabelle *bike_sharing_100*

Unsere Tabelle *bike_sharing_100* hat nun ebenfalls die korrekten Spaltenbezeichnungen und Datentypen.

Table	108 SELECT * FROM bike_sharing_100										
bike_sharing_100	station_id	date	day_of_year	day_of_week	month_of_year	precipitation	windspeed	min_temperature	average_temperature	max_temperature	count
bike_sharing_data_with_NAs	1	2023-01-01	1	1	1	0	10.07	42	50	56	117
demo	1	2023-01-02	2	2	1	0.02	6.49	39	46	55	125
sqlite_sequence											
stations	1	2023-01-03	3	3	1	0.36	5.82	46	48	50	96
MariaDB	1	2023-01-04	4	4	1	0.01	6.49	45	51	61	154
PostgreSQL	1	2023-01-05	5	5	1	0.05	6.26	45	49	51	156
MS SQL	1	2023-01-06	6	6	1	0.25	8.05	39	44	47	166
	1	2023-01-07	7	7	1	0	12.08	36	41	47	120
	1	2023-01-08	8	1	1	0	9.17	32	37	41	123
	1	2023-01-09	9	2	0	0.02	9.84	37	40	46	199
	1	2023-01-10	10	3	1	0	9.62	35	39	41	151
	1	2023-01-11	11	4	1	0	7.83	32	38	40	164
	1	2023-01-12	12	5	1	0.27	11.86	-1	42	53	121
	1	2023-01-13	13	6	1	0.03	19.01	38	49	52	134
	1	2023-01-14	14	7	1	0	22.59	30	35	38	140
	1	2023-01-15	15	1	0	22.84	30	33	30	120	

Tabelle *bike_sharing_100* nach Importierung

Anschließend filtern wir die Daten für die Tabelle `bike_sharing_100`, die wir zuvor erstellt haben. Mit dem folgenden Befehl:

```
DELETE FROM bike_sharing_100
WHERE station_id != 100;
```

entfernen wir alle Einträge in der Spalte `station_id`, deren Werte von unserer Id (100) abweicht. Nach der Ausführung dieses Befehls hat unsere Tabelle die folgende Struktur:

The screenshot shows a database interface with a sidebar on the left listing databases and tables. The main area displays the results of a SQL query:

```
0.1.3 beta
Table
bikesharing_100 <
bikesharing_data_with... <
demo <
sqlite_sequence <
stations <
MariaDB <
PostgreSQL <
MS SQL <

113 SELECT * FROM bike_sharing_100
```

station_id	date	day_of_week	day_of_month	month	precipitation	windsp...	min_temp	average...	max_temp	count
100	2023-01-01	1	1	1	0	10.07	42	50	56	55
100	2023-01-02	2	2	1	0.02	6.49	39	46	55	36
100	2023-01-03	3	3	1	0.36	5.82	46	48	50	60
100	2023-01-04	4	4	1	0.01	6.49	45	51	61	61
100	2023-01-05	5	5	1	0.05	6.26	45	49	51	71
100	2023-01-06	6	6	1	0.25	8.05	39	44	47	57
100	2023-01-07	7	7	1	0	12.08	36	41	47	40
100	2023-01-08	8	1	1	0	9.17	32	37	41	50
100	2023-01-09	9	2	1	0.02	9.84	37	40	46	62
100	2023-01-10	10	3	1	0	9.62	35	39	41	64
100	2023-01-11	11	4	1	0	7.83	0	38	40	62
100	2023-01-12	12	5	1	0.27	11.86	36	42	53	40

Tabelle `bike_sharing_100` nach `station_id = 100` gefiltert

2.2.4 Aufgabe 5

Um die Abfrage formulieren zu können, benötigen wir die Spalte *average_temperature_celsius*, die wir mithilfe dieses Befehles hinzufügen:

```
ALTER TABLE bike_sharing_100
ADD COLUMN average_temperature_celsius REAL;
```

Wir haben der Spalte ebenfalls den Datentyp REAL zugeordnet, damit wir diese mit Gleitkommazahlen befüllen können.

Im nächsten Schritt verwenden wir den folgenden Befehl:

```
UPDATE bike_sharing_100
SET average_temperature_celsius=(average_temperature * 1.0 - 32) * 5/9;
```

Dieser Befehl wandelt die Werte der Spalte *average_temperature* von Fahrenheit in Celsius um und speichert die Ergebnisse in der neuen Spalte *average_temperature_celsius*. Das Multiplizieren mit 1.0 stellt sicher, dass die Berechnung mit den richtigen Dezimalstellen erfolgt.

Nun sieht unsere Tabelle folgendermaßen aus:

The screenshot shows a SQLite database client interface. On the left, there's a sidebar with a tree view of databases and tables. Under the 'SQLite' section, the '0.1.3 beta' database is selected, showing tables like 'bike_sharing_100', 'bike_sharing_data_with...', 'demo', 'sqlite_sequence', and 'stations'. The main area displays a table with 14 columns: stat., date, day_o..., day_o..., mont..., p..., winds..., min_t..., avera..., max_t..., count, and average_temperatu... (truncated). The data consists of 14 rows, each representing a different day from January 1st to January 14th, 2023. The 'average_temperature_celsius' column is visible in the last column of each row. The table has a light gray background with white text and thin black borders for the cells.

stat.	date	day_o...	day_o...	mont...	p...	winds...	min_t...	aver...	max_t...	count	average_temperatu...
100	2023-01-01	1	1	1	0	10.07	42	50	56	55	10
100	2023-01-02	2	2	1	0	6.49	39	46	55	36	7.77777777777777
100	2023-01-03	3	3	1	0	5.82	46	48	50	60	8.88888888888889
100	2023-01-04	4	4	1	0	6.49	45	51	61	61	10.55555555555555
100	2023-01-05	5	5	1	0	6.26	45	49	51	71	9.44444444444444
100	2023-01-06	6	6	1	0	8.05	39	44	47	57	6.66666666666666
100	2023-01-07	7	1	1	0	12.08	36	41	47	40	5
100	2023-01-08	8	1	1	0	9.17	32	37	41	50	2.77777777777777
100	2023-01-09	9	2	1	0	9.84	37	40	46	62	4.44444444444444
100	2023-01-10	10	3	1	0	9.62	35	39	41	64	3.88888888888888
100	2023-01-11	11	4	1	0	7.83	0	38	40	62	3.33333333333333
100	2023-01-12	12	5	1	0	11.86	36	42	53	40	5.55555555555555
100	2023-01-13	13	6	1	0	19.01	38	49	52	57	9.44444444444444
100	2023-01-14	7	1	0	22.59	30	35	38	30	1.66600555666666	

Tabelle *bike_sharing_100* mit der hinzugefügten Spalte *average_temperature_celsius*

Unser letzter Schritt ist es, die Werte der Spalte *average_temperature_celsius* in absteigender Reihenfolge zu sortieren, um dann die höchste mittlere Temperatur zu ermitteln.

Hierfür nutzen wir diesen Befehl:

```
SELECT * FROM bike_sharing_100
ORDER BY average_temperature_celsius DESC;
```

Mit dem *DESC*-Befehl legen wir eine absteigende Reihenfolge fest (engl. 'descending'). Nach Anwendung dieses Befehls sollte unsere Tabelle wie folgt aussehen:

	stat...	date	day_o...	day_o...	mont...	p...	winds...	min_t...	avera...	max_t...	count	average_temperatu...
	100	2023-0...	209	6	7	0..	7.38	76	83	91	113	28.33333333333333
	100	2023-0...	248	3	9	0	6.93	74	82	93	118	27.77777777777777
	100	2023-0...	249	4	9	0	5.37	74	82	93	123	27.77777777777777
	100	2023-0...	193	4	7	0	9.84	72	81	90	125	27.22222222222222
	100	2023-0...	210	7	7	0..	14.32	70	81	90	107	27.22222222222222
	100	2023-0...	250	5	9	0	8.5	73	81	92	120	27.22222222222222
	100	2023-0...	208	5	7	0..	12.3	73	80	92	129	26.66666666666666
	100	2023-0...	192	3	7	0	10.51	70	79	86	133	26.111111111111
	100	2023-0...	194	5	7	0	14.54	72	79	85	132	26.111111111111
	100	2023-0...	247	2	9	0	4.7	70	79	89	88	26.111111111111
	100	2023-0...	184	2	7	0..	9.4	73	78	87	88	25.55555555555555
	100	2023-0...	186	4	7	0	7.38	70	78	88	114	25.55555555555555
	100	2023-0...	188	6	7	0	6.49	74	78	84	102	25.55555555555555
	100	2023-0...	189	7	7	0	6.71	73	78	84	124	25.55555555555555

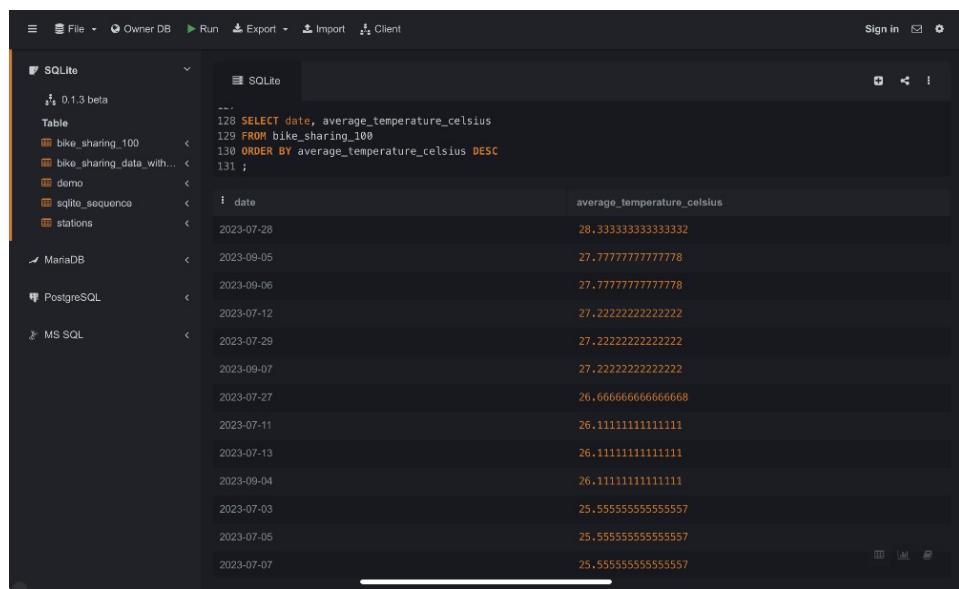
Sortierung der Spalte *average_temperature_celsius* in absteigender Reihenfolge

Abschließend haben wir zur Veranschaulichung den Befehl:

```
SELECT date, average_temperature_celsius
FROM bike_sharing_100
ORDER BY average_temperature_celsius DESC;
```

genutzt, damit wir ausschließlich die Spalten *date* und *average_temperature_celsius* angezeigt bekommen.

Final sieht es folgendermaßen aus:



The screenshot shows a SQLite database interface with the following structure:

- SQLite**:
 - 0.1.3 beta
 - Table
 - bike_sharing_100
 - bike_sharing_data_with...
 - demo
 - sqlite_sequence
 - stations
 - MariaDB
 - PostgreSQL
 - MS SQL

In the main window, a table named "date" is displayed with two columns: "date" and "average_temperature_celsius". The data is as follows:

date	average_temperature_celsius
2023-07-28	28.33333333333332
2023-09-05	27.77777777777778
2023-09-06	27.77777777777778
2023-07-12	27.22222222222222
2023-07-29	27.22222222222222
2023-09-07	27.22222222222222
2023-07-27	26.660666666666668
2023-07-11	26.11111111111111
2023-07-13	26.11111111111111
2023-09-04	26.11111111111111
2023-07-03	25.555555555555557
2023-07-05	25.555555555555557
2023-07-07	25.555555555555557

Spalte *date* und *average_temperature_celsius*

Somit können wir dem Datensatz entnehmen, dass am 28.07.23 die höchste mittlere Temperatur gemessen wurde, die **28,33°C** betrug.

Literatur

- [1] Achim Klenke. *Wahrscheinlichkeitstheorie.*, Springer, 3. edition, 2013.
- [2] <https://www.fahrenheit-umrechnen.de/>
- [3] <https://www.studysmarter.de/studium/mathematik-studium/statistik-studium/der-zentrale-grenzwertsatz/>
- [4] <https://www.sql-und-xml.de/sql-tutorial/erste-normalform-datentypen.html>
- [5] <https://www.sql-und-xml.de/sql-tutorial/zweite-normalform-funktionale-abhaengigkeit.html>