

Projektarbeit

Darstellung und Auswertung des Datensatzes

"Zepp Fitness App"

Volkan Korunan

13. September 2024

Inhaltsverzeichnis

1	Einleitung	3
1.1	Ziele dieser Arbeit	3
1.2	Begründung des Themas	3
1.3	Darstellung eines persönlichen Erkenntnisinteresses	4
2	Nachvollziehbare Schritte	4
2.1	Der Stand der Forschung	5
2.2	Erster Schritt (Datenerfassung)	5
2.3	Notwendige Python Bibliotheken	5
2.4	Daten einlesen und umwandeln ins passende Format	6
2.5	Extrahieren der Daten	6
2.5.1	Erklärung der Datentypen von den Variablen point und track_point	8
2.6	Erzeugen eines Dataframe mit Pandas und Ausgabe der ersten Informationen	9
3	Darstellung der Ergebnisse als Plots	12
4	Darstellung der Laufstrecke	18
5	Berechnung der zurückgelegte Strecke	20
6	Ausblick	21

1 Einleitung

Diese Projektarbeit beschäftigt sich mit der Darstellung und Auswertung eines Datensatzes der Zepp Fitness App, der Daten zur Herzfrequenz, Geschwindigkeit, Schrittzahl usw. Zurückgelegter Strecke und den entsprechenden Zeiten erfasst. Die gesammelten Daten werden visualisiert, indem die zurückgelegte Strecke auf einer Karte dargestellt wird, dies ermöglicht eine detaillierte Analyse der Trainingsaktivitäten. Ziel dieser Arbeit ist es, durch die Auswertung dieser Daten ein besseres Verständnis der individuellen Trainingsmuster zu gewinnen und Potenziale zur Optimierung des persönlichen Trainings zu identifizieren. Die Analyse unterstützt dabei gezielte Empfehlungen zur Verbesserung der Trainingsleistung und der allgemeinen Fitness zu geben, indem Muster erkannt und Schwachstellen im Training aufgezeigt werden. So können Nutzer ihre sportlichen Aktivitäten effizienter gestalten und ihre Fitnessziele effektiver erreichen.

1.1 Ziele dieser Arbeit

Die Analyse von Daten aus Fitness-Apps wie der Zepp Fitness App ist heutzutage von großer Bedeutung, da immer mehr Menschen sportliche Aktivitäten in ihren Alltag integrieren, um ihre Gesundheit zu verbessern und ihre Fitnessziele zu erreichen. Diese Projektarbeit ist besonders relevant für verschiedene Zielgruppen wie Läufer oder Hobbysportler, die ihr Training analysieren wollen, in dem sie:

1. das Erlernen der Fähigkeit Daten zu bekommen und die wichtigsten Informationen zu extrahieren
2. Darstellung der wichtigen Datenmerkmale in unterschiedlichen Grafiken
3. Erkenntnisgewinnung über seinen Trainings- und Fitnesszustand, um die nächsten Trainingseinheiten zu optimieren.

1.2 Begründung des Themas

- **Sportler:** Für ambitionierte Sportler ist es wichtig, ihre Trainingsdaten detailliert auszuwerten, um die eigene Leistung zu steigern. Die Analyse der Herzfrequenz, Geschwindigkeit und Schrittzahl sowie die Visualisierung der zurückgelegten Strecke ermöglichen es, Trainingsmuster zu erkennen und Schwachstellen zu identifizieren. Dies unterstützt die Optimierung von Trainingsplänen und die gezielte Vorbereitung auf Wettkämpfe.
- **Menschen, die fit werden wollen:** Viele Menschen streben ein gesünderes Leben an, sei es aus persönlichen oder gesundheitlichen Gründen. Für diese Zielgruppe bieten die gewonnenen Daten wertvolle Einblicke in ihre sportlichen Aktivitäten und zeigen Fortschritte auf dem Weg zu einem aktiveren Lebensstil auf. Durch die Analyse lassen sich individuelle Schwachstellen und Potenziale erkennen, die zu gezielten Handlungsempfehlungen führen.

- **Hobbysportler:** Für Hobbysportler, die regelmäßig, aber nicht professionell trainieren, ist eine solche Datenauswertung ein hilfreiches Mittel, um ihre Aktivitäten effektiver zu gestalten. Die Visualisierung der Trainingsstrecken und die Analyse von Parametern wie Herzfrequenz und Geschwindigkeit unterstützen dabei, das Training an die eigenen Ziele anzupassen und die Motivation durch sichtbare Fortschritte zu erhöhen.
- **Ältere Menschen und Gesundheitsförderung:** Besonders für ältere Menschen kann eine gezielte Auswertung der Trainingsdaten helfen, die eigene Gesundheit zu fördern. Da im zunehmenden Alter das Risiko für Herz-Kreislauf-Erkrankungen steigt, ist die Überwachung der Herzfrequenz während sportlicher Betätigung besonders wichtig. Die Analyse der Trainingsdaten ermöglicht es, ein optimales Training zu gestalten, das die Gesundheit fördert und gleichzeitig das Risiko für Überanstrengung minimiert.

Die Arbeit zeigt, dass die Auswertung und Visualisierung der Fitnessdaten nicht nur für professionelle Athleten von Interesse ist, sondern für alle, die ihre körperliche Fitness verbessern oder erhalten möchten. Indem individuelle Trainingsmuster analysiert und Optimierungspotenziale aufgezeigt werden, kann jeder seine sportlichen Aktivitäten effizienter gestalten und seine persönlichen Fitnessziele effektiver erreichen.

1.3 Darstellung eines persönlichen Erkenntnisinteresses

Mein persönliches Erkenntnisinteresse an diesem Thema liegt in der detaillierten Analyse von Fitnessdaten, um den aktuellen Fitnesszustand zu bewerten und das Training gezielt zu optimieren. Dabei sind Metriken wie die Herzfrequenz (min, max, durchschnittlich) entscheidend, da sie Aufschluss über die Belastungsintensität und die kardiovaskuläre Fitness geben. Die Analyse der Geschwindigkeit (min, max, durchschnittlich) ermöglicht es, die Leistungsfähigkeit und den Trainingsfortschritt präzise zu beurteilen.

Weitere wichtige Parameter sind die zurückgelegte Distanz und der Kalorienverbrauch, die Rückschlüsse auf Ausdauer und Energieaufwand zulassen und besonders bei der Gewichtskontrolle hilfreich sind. Die Schrittfrequenz unterstützt die Verbesserung der Lauftechnik und -effizienz, während der Höhenverlauf der Strecke wertvolle Informationen über die Trainingsbelastung liefert. Höhenveränderungen beeinflussen unmittelbar die Herzfrequenz, die Geschwindigkeit und den Kalorienverbrauch und helfen, den Trainingsplan gezielt anzupassen.

Zusammengefasst bieten diese Daten ein umfassendes Bild des aktuellen Fitnesszustandes, dokumentieren Fortschritte und unterstützen dabei, gezielte Trainingsanpassungen vorzunehmen. So wird es möglich, die eigene Trainingsleistung objektiv zu bewerten und effektive Maßnahmen zur Steigerung der Fitness zu ergreifen.

2 Nachvollziehbare Schritte

In diesem Abschnitt wird das Projekt Schritt für Schritt erklärt. Dazu gehört die Darstellung des Quellcodes. Angefangen wird dieser Abschnitt mit dem Stand der Forschung.

2.1 Der Stand der Forschung

Die Fitness-Datenanalyse im Laufen ist ein wichtiger Bereich der Sportwissenschaft, der sich mit der Optimierung von Trainingsmethoden und der Verbesserung der sportlichen Leistung beschäftigt.

2.2 Erster Schritt (Datenerfassung)

Um einen Datensatz zu erhalten, müsst ihr im Internet nach einem passenden Datensatz suchen oder euch euren eigenen Datensatz erstellen. Dafür benötigt ihr eine Fitnessuhr, z.B. die Amazfit GTR 3 Pro, und die dazugehörige Fitness-App Zepp. Anschließend könnt ihr euch sportlich anziehen und die Trainingsaufzeichnung entweder von der Uhr aus starten oder, wie in meinem Fall, von der App aus. Nach dem Lauf müsst ihr den Datensatz exportieren, dabei gibt es drei Dateiendungen als Exportmöglichkeiten.

Der Export des Datensatzes ist in drei verschiedenen Dateiformaten möglich:

- GPX (GPS Exchange Format): Ein weit verbreitetes Format für GPS-Daten, das häufig in Kartenanwendungen verwendet wird.
- TCX (Training Center XML): Ein Format, das speziell für Trainingsdaten entwickelt wurde und von vielen Fitness-Apps und -Geräten unterstützt wird.
- FIT (Flexible and Interoperable Data Transfer): Ein kompaktes Format, das von vielen Fitnessgeräten verwendet wird und eine hohe Datenintegrität bietet.

Meine Entscheidung fiel auf das GPX-Format, da es ein XML-Format ist, welches ich in anderen Projekten genutzt habe.

2.3 Notwendige Python Bibliotheken

Für die Datenanalyse wurden folgende Bibliotheken in Python verwendet.

```
1 # xmldict Bib fuer Umwandlung XML Datei in ein Dict Format in Python
2 import xmldict
3 # JSON Bib dazu da um JSON-formate zu erstellen
4 import json
5 # Pandas Bib fuer die Datenanalyse
6 import pandas as pd
7 # Datetime Bib um mit Datum und Zeiten zu Arbeiten
8 import datetime
9 # Seaborn Bib notwenig fuer Grafische Plots
10 # Seaborn Bib notwenig fuer Grafische Plots
11 import seaborn as sns
12 # Matplotlib Bib notwenig fuer Grafische Plots
13 import matplotlib.pyplot as plt
```

```

14 # Numpy Bib fuer Numerische Berechnung
15 import numpy as np

```

2.4 Daten einlesen und umwandeln ins passende Format

Der gegebene Code liest eine GPX-Datei (eine XML-Datei, die typischerweise GPS-Daten enthält), konvertiert sie in ein JSON-Format und speichert diese JSON-Daten in einer neuen Datei. Danach versucht er, die JSON-Daten in ein Python-Dictionary zu laden.

```

1 # Definieren des Pfads zur GPX-Datei
2 file_path = r'C:\Users\vkko\spyder-py3\Woche 4\Projekt\datein\zepp\datei2.gpx'
3
4 # XML-Datei lesen und in ein Dictionary umwandeln
5 try:
6     with open(file_path, 'r', encoding='utf-8') as xml_file:
7         doc = xmltodict.parse(xml_file.read())
8
9     # Dictionary in JSON umwandeln und speichern
10    json_data = json.dumps(doc, indent=4)
11    with open('testJsonConvert.json', 'w') as json_file:
12        json_file.write(json_data)
13
14 except FileNotFoundError:
15     print(f"Die Datei '{file_path}' wurde nicht gefunden.")
16     exit()
17 except Exception as e:
18     print(f"Ein Fehler ist aufgetreten: {e}")
19     exit()
20
21 # JSON in ein Python-Dictionary laden
22 try:
23     data = json.loads(json_data) #JSON in ein Python-Dictionary umwandeln
24 except json.JSONDecodeError as e:
25     print(f"Fehler beim Lesen der JSON-Daten: {e}")
26     exit()

```

2.5 Extrahieren der Daten

Der untere Code dient zur Verarbeitung von GPS-Daten, die im JSON-Format vorliegen. Hierbei werden relevante Informationen aus GPS-Datenpunkten extrahiert und in eine Liste namens [track_points](#) gespeichert. Die extrahierten Datenpunkte beinhalten unter anderem die geographischen Koordinaten, die Höhe, die Zeit sowie zusätzliche Messwerte wie Geschwindigkeit, Herzfrequenz und Schrittfrequenz (Kadenz).

Im Folgenden erkläre ich die verschiedenen Teile des Programmcodes: Zeile 2 erzeugt eine leere Liste aus Tracking Punkte, dort werden die extrahierte Informationen abgespeichert. In der Zeile 6 wird die JSON-Datei durchlaufen, dabei wird versucht, über die GPS-Datenpunkte in der verschachtelten Struktur des JSON-Objekts zu iterieren. Die GPS-Daten befinden sich in einer Struktur, die unter den Schlüsseln "gpx", "trk", "trkseg", und schließlich "trkpt" verschachtelt sind.

In der Zeile 13 bis 21 werden die Daten aus der JSON-Datei in den Datentyp Dictionary point abgespeichert und in der Zeile 22 in die liste track_points eingefügt.

```
1      # Liste zum Speichern extrahierter Informationen
2 track_points = []
3
4 # Iteration ueber JSON-Daten und Extraktion von Informationen
5 try:
6     for trkpt in data['gpx']['trk']['trkseg']['trkpt']:
7         # Verwenden Sie Standardwerte, wenn ein Schluessel fehlt
8         extensions = trkpt.get('extensions', {}).get('ns3:TrackPointExtension', {})
9         speed = extensions.get('ns3:speed', None)
10        heart_rate = extensions.get('ns3:hr', None)
11        cad = extensions.get('ns3:cad', None)
12
13        point = {
14            'latitude': float(trkpt['@lat']),
15            'longitude': float(trkpt['@lon']),
16            'elevation': float(trkpt['ele']),
17            'time': pd.to_datetime(trkpt['time']),
18            'speed': float(speed) if speed is not None else None,
19            'heart_rate': float(heart_rate) if heart_rate is not None else None,
20            'cadence': float(cad) if cad is not None else None
21        }
22        track_points.append(point)
23 except KeyError as e:
24     print(f"Key error: {e}. Bitte    berprfen    Sie die JSON-Struktur.")
25     exit()
26 except ValueError as e:
27     print(f"Value error: {e}.    berprfen    Sie die Datenwerte.")
28     exit()
```

2.5.1 Erklärung der Datentypen von den Variablen point und track_point

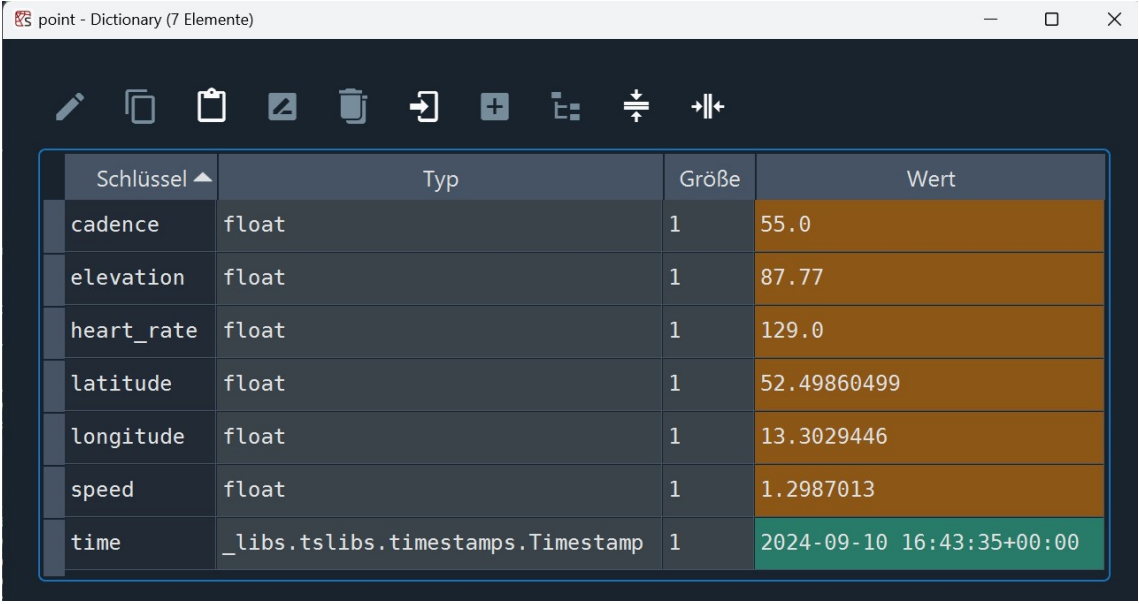
Die Variable point ist ein Dictionary in Python, das Daten im Key-Value-Prinzip speichert. Das bedeutet, dass jeder Key (Schlüssel) einem spezifischen Value (Wert) zugeordnet ist. In diesem Dictionary werden die Informationen zu einem einzelnen GPS-Datenpunkt gespeichert.

Die Struktur sieht wie folgt aus:

```
1 point = {  
2     'latitude': 52.5200,  
3     'longitude': 13.4050,  
4     'elevation': 34.5,  
5     'time': pd.to_datetime("2024-09-12 14:00:00"),  
6     'speed': 3.5,  
7     'heart_rate': 85.0,  
8     'cadence': 90.0  
9 }
```

In der IDE von Spyder kann man dies noch genauer betrachten ([Abbildung 1](#)).

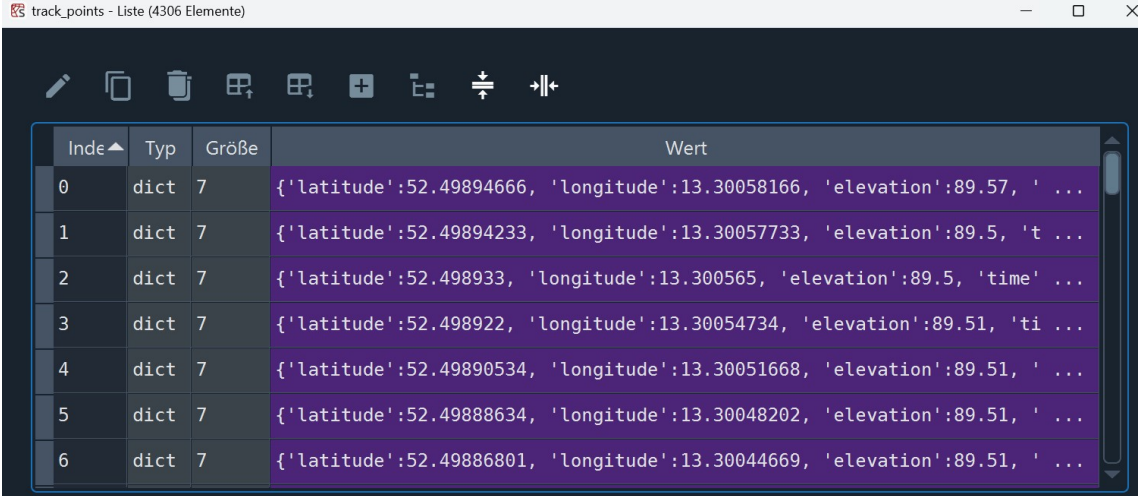
Dort werden Schlüssel, Typ, Größe und der Wert jeweils dargestellt.



Schlüssel ▲	Typ	Größe	Wert
cadence	float	1	55.0
elevation	float	1	87.77
heart_rate	float	1	129.0
latitude	float	1	52.49860499
longitude	float	1	13.3029446
speed	float	1	1.2987013
time	_libs.tslibs.timestamps.Timestamp	1	2024-09-10 16:43:35+00:00

Abbildung 1: point

Die Variable `track_point` ist eine Liste, welche Messpunkte aus den Variable Point, wie in der (Abbildung 1) dargestellt, abspeichert. Diese Liste enthält 4306 Einträge, (Abbildung 2).



Index	Typ	Größe	Wert
0	dict	7	<code>{'latitude':52.49894666, 'longitude':13.30058166, 'elevation':89.57, ' ...</code>
1	dict	7	<code>{'latitude':52.49894233, 'longitude':13.30057733, 'elevation':89.5, 't ...</code>
2	dict	7	<code>{'latitude':52.498933, 'longitude':13.300565, 'elevation':89.5, 'time' ...</code>
3	dict	7	<code>{'latitude':52.498922, 'longitude':13.30054734, 'elevation':89.51, 'ti ...</code>
4	dict	7	<code>{'latitude':52.49890534, 'longitude':13.30051668, 'elevation':89.51, ' ...</code>
5	dict	7	<code>{'latitude':52.49888634, 'longitude':13.30048202, 'elevation':89.51, ' ...</code>
6	dict	7	<code>{'latitude':52.49886801, 'longitude':13.30044669, 'elevation':89.51, ' ...</code>

Abbildung 2: TrackingPoint

2.6 Erzeugen eines Dataframe mit Pandas und Ausgabe der ersten Informationen

In dem unterem Python Code wird ein Dataframe aus `track_points` erzeugt. Anschließend wird das Dataframe ausgegeben auf der Konsole. Anschließend Folgen zwei weitere Ausgaben einmal mit `df.info()` wird in Pandas eine prägnante Zusammenfassung eines DataFrames auf die Konsole dargestellt. Sie liefert wichtige Informationen über den DataFrame. Mit `df.describe()` wird in Pandas eine Zusammenfassung der deskriptiven Statistiken für die numerischen Spalten eines DataFrames erstellt. Diese Methode liefert wichtige statistische Kennzahlen, die helfen, die Verteilung und Eigenschaften der Daten zu verstehen. Die Ausagbe in der Konsole kann man aus den folgenden Abbildungen begutachten: (Abbildung 3) und (Abbildung 4)

```

1 # Erstellen des DataFrames
2 df = pd.DataFrame(track_points)
3
4 # DataFrame ausgeben
5 print(f"df: \n {df}")
6
7 print("DataFrame info Ausgabe: \n ")
8 df.info()
9
10 print("DataFrame Describe Ausgabe: \n")
11 df.describe()

```

```
df:
  latitude longitude elevation ... speed heart_rate cadence
0  52.498947  13.300582    89.57 ... 0.555556    140.0    0.0
1  52.498942  13.300577    89.59 ... 0.555556    140.0    0.0
2  52.498933  13.300565    89.58 ... 0.555556    140.0    0.0
3  52.498922  13.300547    89.51 ... 0.555556    140.0    0.0
4  52.498905  13.300517    89.51 ... 0.555556    140.0    79.0
...
4301 52.498730  13.302843    84.92 ... 0.537634    157.0    44.0
4302 52.498725  13.302848    84.92 ... 0.537634    156.0    44.0
4303 52.498721  13.302854    84.91 ... 0.550059    155.0    44.0
4304 52.498715  13.302866    84.86 ... 1.282051    154.0    44.0
4305 52.498605  13.302945    87.77 ... 1.298701    129.0    55.0
[4306 rows x 7 columns]
```

Abbildung 3: Ausgabe des Dataframe

```
DataFrame info Ausgabe:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4306 entries, 0 to 4305
Data columns (total 7 columns):
#   Column             Non-Null Count  Dtype
---  -
0   latitude           4306 non-null   float64
1   longitude           4306 non-null   float64
2   elevation           4306 non-null   float64
3   time               4306 non-null   datetime64[ns, UTC]
4   speed              4306 non-null   float64
5   heart_rate          4306 non-null   float64
6   cadence             4306 non-null   float64
dtypes: datetime64[ns, UTC](1), float64(6)
memory usage: 235.6 KB

DataFrame Describe Ausgabe:
Out[69]:
   latitude  longitude  ... heart_rate  cadence
count  4306.000000  4306.000000  ...  4306.000000  4306.000000
mean    52.499771    13.302088  ...    156.318857    72.688806
std      0.005264     0.004502  ...     14.667030    12.187549
min     52.483135    13.291047  ...    122.000000     0.000000
25%     52.484437    13.300576  ...    146.000000    59.000000
50%     52.488828    13.302437  ...    160.000000    79.000000
75%     52.494686    13.303684  ...    167.000000    81.000000
max     52.499131    13.312168  ...    182.000000   105.000000
[8 rows x 6 columns]
```

Abbildung 4: Ausgabe der Informationen und wichtige statistische Kennzahlen des Dataframes

Der folgende Python-Code unten bereinigt und verarbeitet Daten eines Dataframes. Dabei werden Messwerte wie Herzfrequenz, Kadenz und Zeitstempel bearbeitet. Verschiedene Techniken werden angewendet, um fehlende Daten zu behandeln und neue Informationen zu extrahieren.

Zunächst beginnt der Code mit der Behandlung fehlender Werte in der Spalte `heart_rate` (Herzfrequenz). Hierbei wird eine lineare Interpolation durchgeführt, um die fehlenden Herzfrequenzwerte aufzufüllen. Die Methode `interpolate` mit der Option `'linear'` wird verwendet, um die Lücken basierend auf den benachbarten Werten zu schließen. Anschließend werden die interpolierten Werte mit der Methode `round()` gerundet, um sicherzustellen, dass sie als ganze Zahlen dargestellt werden.

Falls es am Anfang der Datenreihe fehlende Werte gibt, die nicht durch die Interpolation aufgefüllt werden konnten, werden diese mithilfe der Methode `backfill()` auf den ersten gültigen Wert gesetzt. Diese Methode füllt alle fehlenden Werte von unten nach oben auf, um sicherzustellen, dass auch die Anfangslücken geschlossen werden.

Anschließend werden in der Spalte `cadence` (Kadenz) alle fehlenden Werte (NaN) durch die Zahl 0 ersetzt. Dies geschieht mit der Methode `where()`, die überprüft, ob die Werte in der Spalte vorhanden sind (`notna()`). Wenn nicht, wird 0 eingesetzt. Danach wird eine lineare Interpolation durchgeführt, um die Werte aufzufüllen und eine glattere Verteilung der Kadenzwerte zu erzielen.

Nachdem die Interpolation durchgeführt wurde, wird der Median der Cadence-Spalte berechnet, wobei nur Werte berücksichtigt werden, die nicht 0 sind. Dies dient dazu, eine typische Schrittfrequenz zu ermitteln. Anschließend werden alle 0-Werte in der Kadenz-Spalte durch diesen Median ersetzt, um eine realistischere Darstellung der Kadenzdaten zu erhalten.

Im nächsten Schritt erfolgt die Extraktion von Minuten und Sekunden aus den Zeitstempeln. Hierbei wird die Zeitdifferenz zwischen den Zeitstempeln in der Spalte `time` und einem festen Referenzzeitpunkt berechnet. Dieser Referenzzeitpunkt (timeaware) wird als UTC-Zeit definiert. Die Differenz zwischen dem Zeitstempel und dem Referenzzeitpunkt wird in der Spalte `time_delta` gespeichert.

Anschließend werden aus der Zeitdifferenz die Komponenten Sekunden, Minuten und Stunden extrahiert und jeweils in separate Spalten (seconds, minutes und stunde) gespeichert. Schließlich wird die Gesamtzahl der Minuten (Fullminutes) berechnet, indem die Stunden in Minuten umgerechnet und mit den Minuten addiert werden.

```
1 # Behandlung von fehlenden Herzfrequenzdaten durch Interpolation
2 df['Heart_rate'] = df['heart_rate'].interpolate(method='linear')
3 df['Heart_rate'] = df['Heart_rate'].round()
4
5 # um die ersten nan Werte die nicht interpoliert werden konnten,
6 # auf den ersten gemessenen wert zu setzten
7 df['Heart_rate'] = df['Heart_rate'].backfill()
8
9 # NaN-Werte in der Spalte 'Cadence' durch 0 ersetzen
10 df['Cadence'] = df['cadence'].where(df['cadence'].notna(), 0)
11 df['Cadence'] = df['Cadence'].interpolate(method='linear')
12
13 # Median der Cadence-Spalte berechnen, unter Ausschluss von Nullen
14 median_cadence = df['Cadence'][df['Cadence'] != 0].median()
15
16 # Nullen durch den Median ersetzen
17 df['Cadence'] = df['Cadence'].replace(0, median_cadence)
18
19 # Extrahieren der Minuten und Sekunden in separate Spalten
20 time1 = datetime.datetime.strptime('10:09:2024:15:26:42', "%d:%m:%Y:%H:%M:%S")
21 timeaware = time1.replace(tzinfo=datetime.timezone.utc)
22
23 df['time_delta'] = df['time'] - timeaware
24 df['seconds'] = df['time_delta'].dt.components['seconds']
25 df['minutes'] = df['time_delta'].dt.components['minutes']
26 df['stunde'] = df['time_delta'].dt.components['hours']
27 df['Fullminutes'] = df['stunde'] * 60 + df['minutes']
```

Nun folgt der Code Abschnitt für die Erzeugung der grafischen Ausgaben siehe Kapitel 3. Es werden dort die Plots sowie der Code, der die Plots erzeugt, erläutert.

3 Darstellung der Ergebnisse als Plots

Der erste Plot ist ein Scatterplot, welcher mit der Bibliothek Seaborn erzeugt wird. Es wird die Herzfrequenz auf der Y-Achse und die Geschwindigkeit auf der X-Achse dargestellt.

```
1 # Scatterplot von der Herzfrequenz vs Geschwindigkeit
2 sns.scatterplot(data=df, x='Speed', y='Heart_rate')
3
4 # Achsenbeschriftungen und Titel hinzufügen
5 plt.xlabel('Geschwindigkeit (km/h)')
6 plt.ylabel('Herzfrequenz (bpm)')
7 plt.title('Herzfrequenz vs. Geschwindigkeit')
8
9 # Herzfrequenzlinien hinzufügen
10 hr_max = df['Heart_rate'].max()
11 hr_mean = df['Heart_rate'].mean()
12 hr_min = df['Heart_rate'].min()
13 plt.axhline(y=hr_max, color='red', linestyle='--', label='Max Herzfrequenz')
14 plt.axhline(y=hr_mean, color='green', linestyle='--', label='Durchschnitt
    Herzfrequenz')
15 plt.axhline(y=hr_min, color='violet', linestyle='--', label='Min Herzfrequenz')
16
17 # Legende hinzufügen und anpassen
18 plt.legend(loc='lower right', fontsize='8') # Position und Größe anpassen
19
20 # Plot anzeigen
21 plt.show()
```

Die Grafik ([Abbildung 6](#)) zeigt die Beziehung zwischen der Herzfrequenz (in Schlägen pro Minute) und der Geschwindigkeit (in km/h) während einer sportlichen Aktivität des Laufens. Die Punktwolke erstreckt sich über einen Geschwindigkeitsbereich von etwa 0 bis 20 km/h und eine Herzfrequenzspanne von ungefähr 120 bis 180 Schlägen pro Minute. Auffällig sind drei horizontale gestrichelte Linien, die wichtige Herzfrequenzwerte markieren: Die oberste rote Linie zeigt die maximale Herzfrequenz bei etwa 182 bpm, die mittlere grüne Linie die durchschnittliche Herzfrequenz bei circa 157 bpm und die untere violette Linie die minimale Herzfrequenz bei ungefähr 122 bpm. Die Datenpunkte zeigen eine deutliche Konzentration im Geschwindigkeitsbereich zwischen 5 und 10 km/h, wobei die Mehrheit der Herzfrequenzwerte zwischen 140 und 170 bpm liegt. Dies deutet auf eine moderate bis intensive Belastung während des Großteils der Aktivität hin. Bei niedrigen Geschwindigkeiten (unter 5 km/h) ist eine breitere Streuung der Herzfrequenzwerte zu beobachten, was möglicherweise auf Ruhephasen oder langsame Bewegungen wie Bergaufstrecken hindeutet. Im höheren Geschwindigkeitsbereich (über 15 km/h) gibt es weniger Datenpunkte, aber tendenziell höhere Herzfrequenzen, was intensive Anstrengungen oder Sprintphasen vermuten lässt.

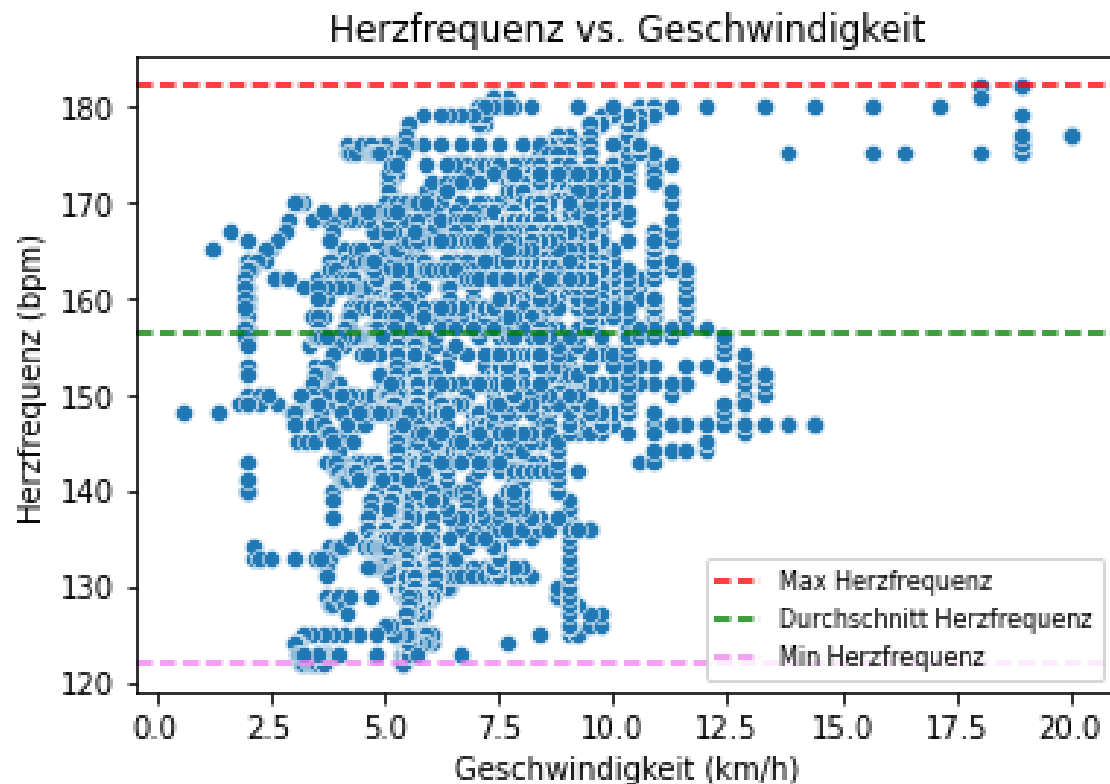


Abbildung 5: Herzfrequenz vs Geschwindigkeit

Nun folgt ein ähnlicher Plot nur diesmal ist die X-Achse nicht die Geschwindigkeit sondern die Zeit in Minuten.

Das Diagramm zeigt die Herzfrequenz (in bpm) in Abhängigkeit von der Zeit (in Minuten). Zu Beginn steigt die Herzfrequenz relativ schnell an und bewegt sich anschließend in einem Bereich zwischen 130 und 180 bpm. Es gibt mehrere Zyklen von Anstiegen und Abnahmen, die möglicherweise durch unterschiedliche körperliche Aktivitätsstufen oder Ruhephasen verursacht werden. Insgesamt zeigt das Diagramm, dass die Herzfrequenz über die Zeit hinweg eine steigende Tendenz aufweist mit mehreren Spitzenwerten, die über 180 bpm liegen.

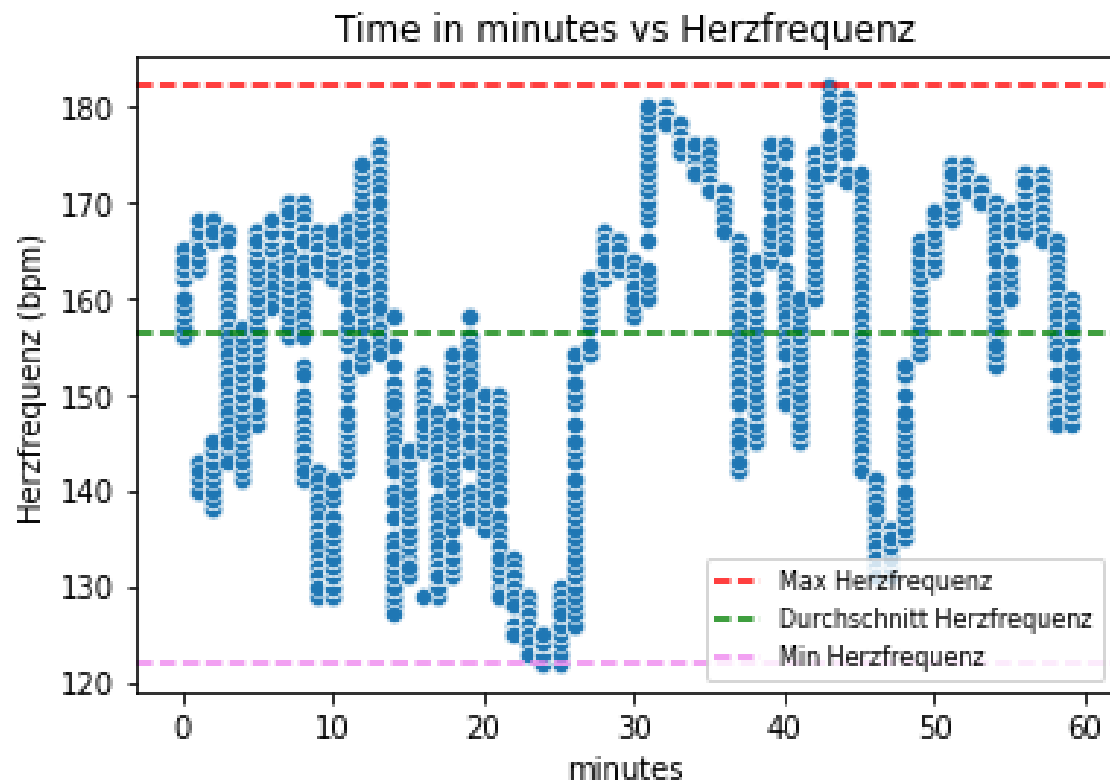


Abbildung 6: Herzfrequenz vs Zeit

Der unten stehende Python-Code stellt zwei verschiedene Merkmale der Herzfrequenz und die Geschwindigkeit als Linienplot über die Zeit in Minuten dar. Zusammengefasst kann man sagen, das Diagramm kombiniert die Herzfrequenz und die Geschwindigkeit in einem Zeitverlauf. Die Herzfrequenz (blaue Linie) und die Geschwindigkeit (orange Linie) werden über die Zeit dargestellt, um die Korrelation zwischen beiden Variablen zu verdeutlichen. Es gibt sichtbare Spitzen in der Herzfrequenz, die häufig mit einer Erhöhung der Geschwindigkeit korrelieren, was darauf hindeutet, dass die Herzfrequenz bei höheren Geschwindigkeiten ansteigt. Die maximale Herzfrequenz (rote gestrichelte Linie) liegt bei etwa 180 bpm, während die maximale Geschwindigkeit etwa 12 km/h erreicht. Die durchschnittliche Herzfrequenz (grüne gestrichelte Linie) und die minimale Herzfrequenz (violette gestrichelte Linie) bleiben ebenfalls erkennbar (Abbildung 7).

```

1 # Erstellen des ersten Scatterplots für die erste y-Achse (Herzfrequenz)
2 fig, ax1 = plt.subplots()
3
4 sns.lineplot(data=df, x='Fullminutes', y='Heart_rate', ax=ax1, color='b', label='
    Herzfrequenz')
5 ax1.set_xlabel('Minutes')
6 ax1.set_ylabel('Herzfrequenz (bpm)', color='b')
7 ax1.tick_params(axis='y', labelcolor='b')
8
9 # Herzfrequenzlinien hinzufügen
10 hr_max = df['Heart_rate'].max()
11 hr_mean = df['Heart_rate'].mean()
12 hr_min = df['Heart_rate'].min()
13 ax1.axhline(y=hr_max, color='red', linestyle='--', label='Max Herzfrequenz')

```

```

14 ax1.axhline(y=hr_mean, color='green', linestyle='--', label='Durchschnitt
    Herzfrequenz')
15 ax1.axhline(y=hr_min, color='violet', linestyle='--', label='Min Herzfrequenz')
16
17 # Erstellen der sekund ren y-Achse
18 ax2 = ax1.twinx()
19 sns.lineplot(data=df, x='Fullminutes', y='Speed', ax=ax2, color='orange', label='
    Geschwindigkeit')
20 ax2.set_ylabel('Speed (km/h)', color='orange')
21 ax2.tick_params(axis='y', labelcolor='orange')
22
23 # Legenden zusammenf hren und anzeigen
24 lines_1, labels_1 = ax1.get_legend_handles_labels()
25 lines_2, labels_2 = ax2.get_legend_handles_labels()
26 ax1.legend(lines_1 + lines_2, labels_1 + labels_2, loc='lower right', fontsize='6')
27 plt.title('Time in Minutes vs Herzfrequenz & Speed')
28 plt.show()

```

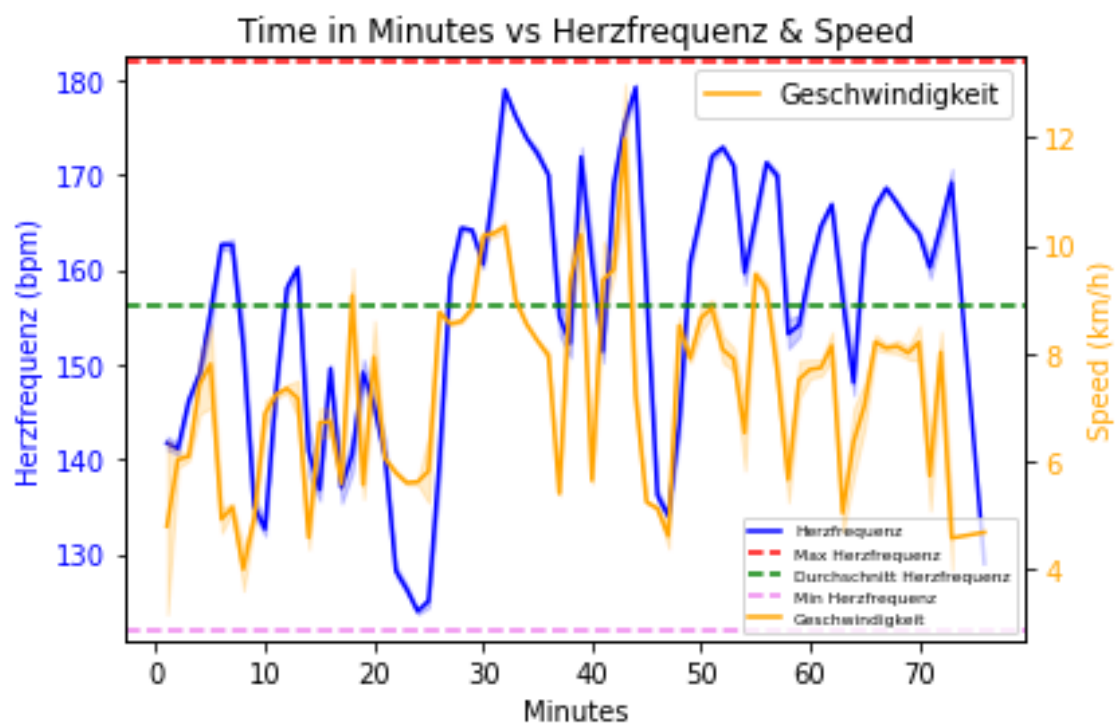


Abbildung 7: Time in Minutes vs Herzfrequenz & Speed

Der nächste Code erzeugt eine Histogramm Grafik und kann unter diesem Text betrachtet werden:

```

1 # Histogramm mit Seaborn erstellen
2 sns.histplot(df['Speed'], bins=20, color='skyblue', kde=False)
3 # Achsenbeschriftungen und Titel hinzufuegen
4 plt.xlabel('Geschwindigkeit km/h')
5 plt.ylabel('Anzahl der Vorkommen')
6 plt.title('Verteilung der Geschwindigkeit')
7 # Plot anzeigen
8 plt.show()

```

Das oben erzeugte Diagramm ([Abbildung 8](#)) zeigt die Verteilung der Geschwindigkeit (in km/h) während der Aktivität. Es handelt sich um ein Histogramm, das die Häufigkeit der unterschiedlichen Geschwindigkeiten darstellt. Die meisten Geschwindigkeiten liegen im Bereich von 5 bis 10 km/h, wobei die höchste Anzahl von Vorkommen bei etwa 7,5 bis 10 km/h liegt. Dies deutet darauf hin, dass die Aktivität überwiegend mit moderater Geschwindigkeit durchgeführt wurde, mit wenigen Ereignissen unter 2,5 km/h oder über 12,5 km/h.

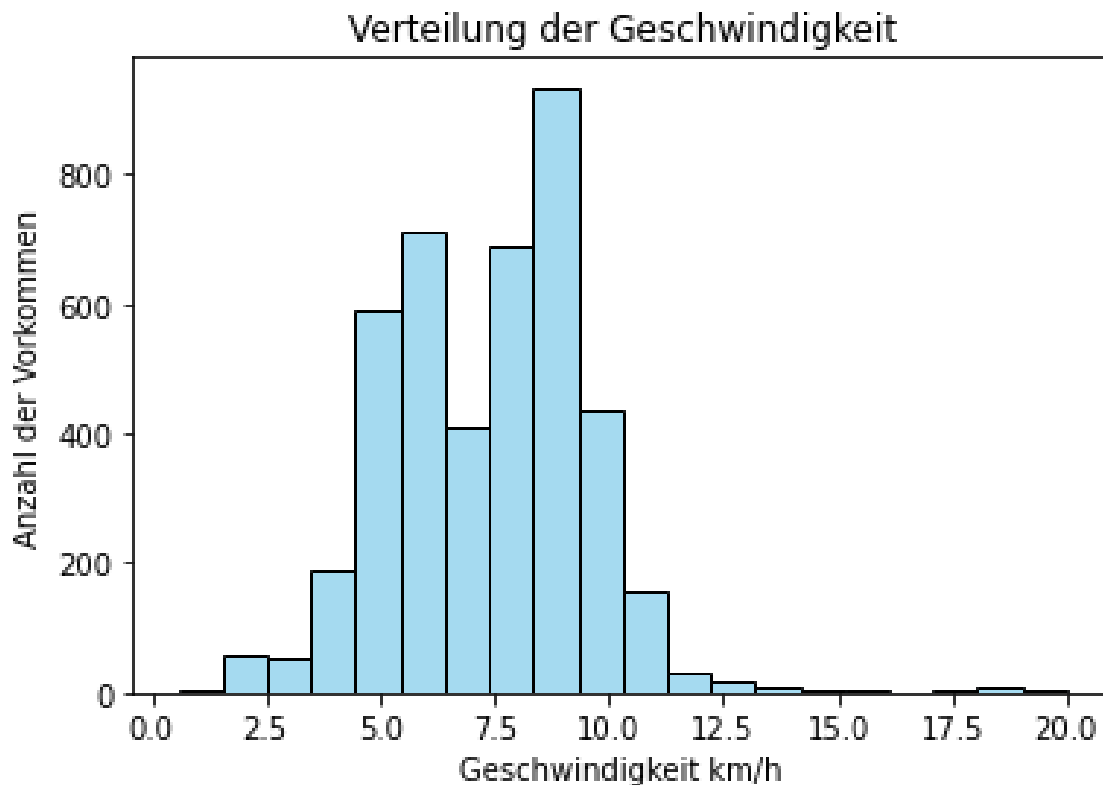


Abbildung 8: Verteilung der Geschwindigkeit

Eine Weitere Verteilung zeigt die nächste ([Abbildung 9](#)). Die Grafik zeigt die Verteilung der Herzfrequenz in Schlägen pro Minute (bpm) in einem Histogramm. Auf der x-Achse ist die Herzfrequenz in bpm dargestellt, die von 120 bpm bis 180 bpm reicht, während die y-Achse die Anzahl der Vorkommen in den entsprechenden Herzfrequenzbereichen zeigt.

Die Verteilung der Herzfrequenz ist nicht gleichmäßig, sondern weist eine deutliche Häufung im Bereich zwischen 150 bpm und 170 bpm auf. Besonders auffällig ist der Bereich um 160 bpm, in dem die Anzahl der Vorkommen ein Maximum von über 500 erreicht. Dies deutet darauf hin, dass die meisten Messungen der Herzfrequenz in diesem Bereich liegen.

Im niedrigeren Bereich der Herzfrequenz, zwischen 120 bpm und 140 bpm sind deutlich weniger Vorkommen zu verzeichnen wobei die Anzahl der Vorkommen ab 120 bpm zunächst langsam ansteigt. Der Bereich zwischen 170 bpm und 180 bpm zeigt wiederum eine abnehmende Häufigkeit, was darauf hindeutet, dass sehr hohe Herzfrequenzen weniger häufig auftreten.

Insgesamt zeigt die Verteilung der Herzfrequenz eine asymmetrische Form mit einer leichten Rechtsverschiebung, was auf eine tendenzielle Häufung der Messwerte im oberen Bereich der Herzfrequenzen hinweist. Diese Verteilung könnte in einem medizinischen oder sportwissen-

schaftlichen Kontext von Interesse sein, da sie Aufschluss darüber gibt, in welchem Bereich die Herzfrequenz der beobachteten Population am häufigsten liegt.

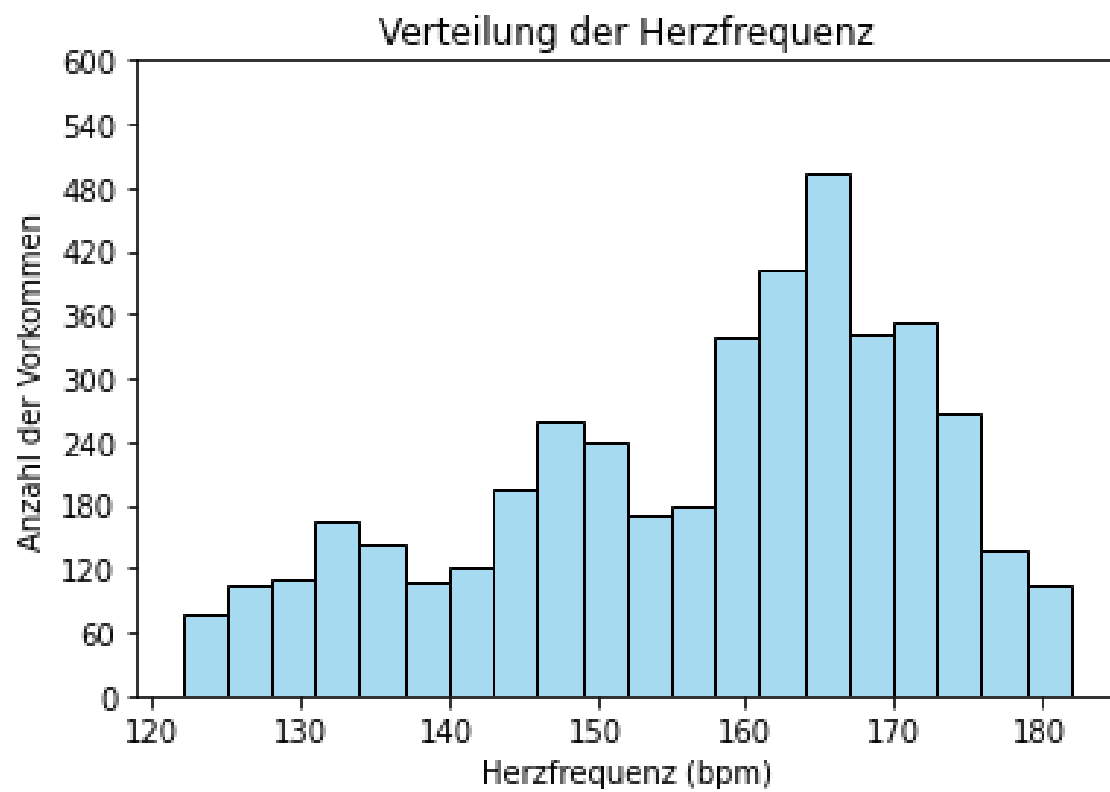


Abbildung 9: Verteilung der Herzfrequenz

4 Darstellung der Laufstrecke

Zusammenfassend zeigen die Grafiken eine detaillierte Analyse der Herzfrequenz und der Geschwindigkeit über einen bestimmten Zeitraum, wobei deutliche Muster und Korrelationen zwischen den beiden Variablen sichtbar sind. Der untere Python-Code erstellt eine interaktive Karte, die eine Laufstrecke basierend auf GPS-Koordinaten im Webbrowser anzeigt und anschließend als Bild abspeichert. Zunächst werden zwei Listen mit Koordinaten, nämlich Breitengrade und Längengrade, aus einem DataFrame (df) extrahiert. Diese Listen repräsentieren die verschiedenen geografischen Punkte der Laufstrecke.

Anschließend wird mit der Bibliothek folium eine Karte erstellt, die auf den ersten Punkt der Koordinaten zentriert ist. Die Karte wird mit einem Startpunkt und einem festgelegten Zoom-Level initialisiert, um eine geeignete Ansicht zu bieten. Danach wird eine Liste der Koordinatenpaare erstellt, die die verschiedenen Punkte der Laufstrecke darstellt. Diese Koordinaten werden als Linie (PolyLine) auf der Karte eingezeichnet, wobei die Linie blau gefärbt ist, eine Dicke von zwei Pixeln hat und vollständig deckend ist.

Die Karte wird als HTML-Datei unter dem Namen karte4.html gespeichert, sodass sie in einem Webbrowser angezeigt werden kann. Um die Karte als Bild zu speichern, verwendet der Code Selenium, eine Bibliothek zur Steuerung von Webbrowsern. Zuerst wird der Pfad zum geckodriver definiert, welcher benötigt wird, um den Firefox-Browser zu steuern. Anschließend werden die Optionen für den Browser festgelegt. Dabei kann der Browser entweder sichtbar oder unsichtbar (im sogenannten *HeadlessModus*) ausgeführt werden.

Daraufhin wird eine Instanz des Firefox-Browsers gestartet und die HTML-Datei der Karte geöffnet. Ein Screenshot der angezeigten Karte wird aufgenommen und im Arbeitsspeicher als PNG-Datei gespeichert. Mithilfe der Pillow-Bibliothek wird das Bild aus dem Arbeitsspeicher geladen und unter dem Namen *karte_screenshot4.png* auf der Festplatte gespeichert. Schließlich wird der WebDriver geschlossen und der Browser beendet.

```
1 import folium
2 from selenium import webdriver
3 from selenium.webdriver.firefox.service import Service
4 from selenium.webdriver.firefox.options import Options
5 from PIL import Image
6 import io
7 # Beispiel: Listen von Längen- und Breitengraden
8 breitengrade = df.Latitude # Breitengrade
9 laengengrade = df.Longitude # Längengrade
10
11 # Startpunkt für die Karte setzen (Mittelpunkt der ersten Koordinate)
12 karte = folium.Map(location=[breitengrade[0], laengengrade[0]], zoom_start=14.44)
13
14 # Eine Liste der Koordinatenpaare erstellen
15 koordinaten = list(zip(breitengrade, laengengrade))
16
17 # Strecke (Linie) auf der Karte einzeichnen
18 folium.PolyLine(locations=koordinaten, color='blue', weight=2, opacity=1).add_to(
    karte)
```

```

19
20 # Speichern der Karte als HTML
21 karte.save("karte4.html")
22
23 # Pfad zum geckodriver angeben
24 gecko_path = r"C:\WebDriver\geckodriver.exe"
25
26 # Firefox Options und Service konfigurieren
27 options = Options() # Setze auf True, wenn du den Browser im Hintergrund
28 options.headless = False# ausführen möchtest
29
30 # Service mit dem geckodriver-Pfad erstellen
31 service = Service(executable_path=gecko_path)
32
33 # Selenium WebDriver für Firefox initialisieren
34 driver = webdriver.Firefox(service=service, options=options)
35
36 # Screenshot als PNG im Arbeitsspeicher speichern
37 screenshot = driver.get_screenshot_as_png()
38
39 # Bild mit Pillow öffnen und speichern
40 image = Image.open(io.BytesIO(screenshot))
41 image.save('karte_screenshot4.png') # Speichert das Bild als PNG
42
43 driver.quit() # WebDriver beenden
44
45 img_data = karte._to_png(5)
46 img = Image.open(io.BytesIO(img_data))
47 img.save('Karte4.png')

```

Zusätzlich bietet der Code eine alternative Methode, die Karte direkt als PNG-Bild zu speichern, ohne den Browser zu verwenden. Die interne Funktion `_to_png` von folium wird verwendet, um die Karte in ein Bildformat zu konvertieren, welches dann mit der Pillow-Bibliothek gespeichert wird. Diese Methode ist schneller und vermeidet die Abhängigkeit von Selenium und einem Webbrowser. Das Ergebnis kann man in der unteren [Abbildung 10](#) begutachten.

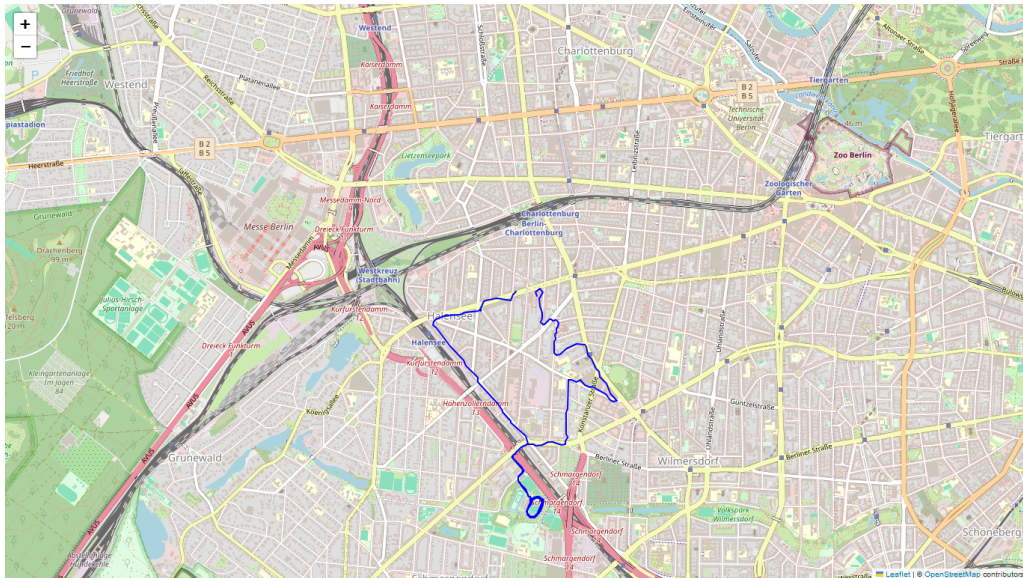


Abbildung 10: Laufstrecke auf der Karte

5 Berechnung der zurückgelegte Strecke

Der gegebene Python-Code berechnet die Gesamtdistanz einer Strecke basierend auf einer Liste von GPS-Koordinaten (Breiten- und Längengraden). Diese Daten stammen vermutlich aus einer Datenquelle wie einem DataFrame (df), der die geografischen Koordinatenpunkte der Strecke enthält.

Zuerst werden die Listen der Breiten- und Längengrade (breitengrade und laengengrade) aus dem DataFrame extrahiert. Anschließend wird eine Konstante für den Erdradius `EARTH_RADIUS_KM` in Kilometern definiert, die später in der Berechnung verwendet wird.

Der Code definiert eine Funktion `haversine_distance(lat1, lon1, lat2, lon2)`, die die Entfernung zwischen zwei geografischen Punkten auf der Erdoberfläche berechnet. Die Funktion nutzt die Haversine-Formel, eine mathematische Formel, die verwendet wird, um die kürzeste Entfernung zwischen zwei Punkten auf einer Kugel zu berechnen. Zunächst werden die geografischen Koordinaten (Breiten- und Längengrade) von Grad in Bogenmaß (Radiant) umgerechnet. Dann werden die Unterschiede zwischen den Breiten- und Längengraden berechnet.

Anschließend wird die Haversine-Formel angewendet, die den Abstand zwischen zwei Punkten auf einer Kugel unter Berücksichtigung der Erdkrümmung berechnet. Das Ergebnis ist die Entfernung zwischen den beiden Punkten in Kilometern.

Nach der Definition der Funktion wird eine Variable `total_distance` initialisiert, die die gesamte zurückgelegte Strecke speichert. Mithilfe einer Schleife wird die Entfernung zwischen jedem aufeinanderfolgenden Koordinatenpaar berechnet, indem die `haversine_distance`-Funktion für jeweils zwei aufeinanderfolgende Punkte aufgerufen wird. Die berechnete Entfernung wird schrittweise zur `total_distance` addiert, um die Gesamtdistanz zu ermitteln.

Abschließend gibt der Code die Gesamtdistanz auf zwei Dezimalstellen genau in Kilometern aus. Der Ausdruck `print(f"Die zurückgelegte Strecke beträgt etwa total_distance:.2f km.")` formatiert die berechnete Gesamtstrecke und gibt sie als Nachricht auf der Konsole aus, um den

Benutzer über die zurückgelegte Strecke zu informieren.

```
1     breitengrade = df.latitude # Breitengrade
2     laengengrade = df.longitude # L ngengrade
3     # Konstante: Radius der Erde in Kilometern
4     EARTH_RADIUS_KM = 6371.0
5     def haversine_distance(lat1, lon1, lat2, lon2):
6         # Umwandlung der Koordinaten von Grad in Radian
7         lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])
8
9         # Berechnung der Unterschiede der Koordinaten
10        dlat = lat2 - lat1
11        dlon = lon2 - lon1
12
13        # Haversine-Formel
14        a = np.sin(dlat / 2.0) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2.0)
15        ** 2
16        c = 2 * np.arcsin(np.sqrt(a))
17
18        # Berechnung der Entfernung
19        distance = EARTH_RADIUS_KM * c
20        return distance
21
22    # Berechnung der gesamten zur ckgelegten Strecke
23    total_distance = 0.0
24
25    for i in range(1, len(breitengrade)):
26        # Berechnung der Entfernung zwischen aufeinanderfolgenden Punkten
27        distance = haversine_distance(breitengrade[i-1], laengengrade[i-1],
28        breitengrade[i], laengengrade[i])
29        total_distance += distance
30
31    print(f"Die zurueckgelegte Strecke betraegt etwa {total_distance:.2f} km.")
```

So ermittelt man die zurückgelegte Strecke von etwa 9.00 km für den Trainingslauf, dies ist deckungsgleich mit der Messung der Uhr.

6 Ausblick

Die verwendete Datenanalyse könnte auf andere Trainingsdatensätze der Uhr oder der Zepp App mit einer deutlich größeren Anzahl von Messreihen oder zusätzlichen Variablen (mit kleiner Anpassung des Codes) angewendet werden. Des Weiteren ist es möglich eine andere Datenanalyse, wie zum Beispiel die Schlafaufzeichnung, als Datensatz zu untersuchen und daraus Erkenntnisse zur Optimierung des Schlafes zu gewinnen. Zudem könnte man mithilfe anderer Trainingsdaten sein Training vergleichen und feststellen, ob eine Verbesserung des Fitnesszustands nach 3–4 Wochen eingetreten oder ob gegebenenfalls eine Anpassung des Trainings erforderlich ist. Zudem könnte man noch weitere Informationen gewinnen, wenn man andere grafische Plots mit anderen Merkmalen durchführen würde.

Abbildungsverzeichnis

1	point	8
2	TrackingPoint	9
3	Ausgabe des Dataframe	10
4	Ausgabe der Informationen und wichtige statistische Kennzahlen des Dataframes	10
5	Herzfrequenz vs Geschwindigkeit	13
6	Herzfrequenz vs Zeit	14
7	Time in Minutes vs Herzfrequenz & Speed	15
8	Verteilung der Geschwindigkeit	16
9	Verteilung der Herzfrequenz	17
10	Laufstrecke auf der Karte	20