Programmierübung 2

SoSe 2023

Julian Huber & Yannic Heyer

Termin 4 - find_peaks

Organisatorisches

Benotung der Lehrveranstaltung

- 3 Abgaben in Sakai
  - **A1: Implementierung der Funktion `find_peaks()`**
  - A2: Algorithmus zur Wiederherstellung von EKG-Daten `reconstruct()`
  - A3: TBD z.B. Datenvisualisierung
- 1 Abschlusspräsentation: Vorstellung des Dashboards
- Code-Reviews
  - Abgaben werden besprochen und Fragen zum Code gestellt

Gruppenbildung

- 2er Gruppen
- frei **innerhalb** der Termine

A1: Implementierung der Funktion `find_peaks()`

Überblick

- Peak Detection
- Dokumentation der existierenden Funktion von `scipy.signal.find_peaks()`
- Planung und Design der `FindPeaks` Klasse
- Coding der Klasse und Methoden
- Testen und Anwenden an Beispieldatensätzen
- Abgabe der Übung A1

Wofür verwenden wir Peak Detection?

- Die Quantifizierung von Peak-Eigenschaften ermöglicht Messungen und Charakterisierung von Merkmalen in den Daten
- Peaks können als Grenzen dienen, um Daten in sinnvolle Intervalle oder Ereignisse zu segmentieren
- Die Automatisierung der Peak-Detektion unterstützt effiziente Datenanalyse und Echtzeitüberwachung
- Die Peak-Detektion trägt zu einer verbesserten Diagnose, Überwachung und Behandlung im Gesundheitswesen bei, indem z.B. EKG Daten analysiert werden

Welche Möglichkeiten gibt es, Peaks zu detektieren?

1. `scipy.signal.find_peaks`
2. `peakutils.indexes`
3. `pyroomacoustics`

   ...

ODER **eigene Funktionen**

Ziel der Übung und Abgabe

- Identifikation von Anforderungen
- Programmierung der Klasse `FindPeaks` mit folgenden Methoden
  - Laden der Daten
  - Detektion der Peaks (eigens geschriebene Funktion)
  - Berechnung der Eigenschaften der Peaks
    - Abstand zum nächsten Peak (in ms) / Boxplot
    - Hochpunkt der Peaks (Absolutwert) / Histogramm
    - Höhe der Peaks (Absolutwert zur Baseline) / Boxplot
  - Darstellen der Peaks und der Eigenschaften
- Testen der Funktionalität anhand von Beispieldaten

Was muss abgegeben werden?

Im Rahmen der Übung wird gemeinsam die Struktur der Klasse und der Methoden erstellt. Die Inhalte der Methoden sollen dann in Partnerarbeit gelöst werden. Für die Bewertung wird auf die Art der Programmierung (richtiges Benennen, Struktur und Effizenz) sowie auf die Qualität und Kreativität der Darstellungen geachtet.

Achtung: Es werden mündliche Abfragen zu den Lösungen gemacht, um das Verständnis zu prüfen.

Dokumentation von `find_peaks()`

Scipy ist eine wertvolle Python Bibliothek die zum Beispiel zur Signalverarbeitung verwendet werden kann. Eine der bekanntesten Anwendungen ist die `scypi.signal.find_peaks()` Funktion.

Aufgabe:
Schauen Sie sich die Dokumentation der Funktion unter folgenden Link an und machen Sie sich mit den Input Parametern sowie dem Output vertraut. (10 min)

Klassen in Python

Keyword: `class`

Wird verwendet, um eine Klasse zu erstellen. Dafür kann folgender Code verwendet werden:

```
# Create a class named "zoo"
class Zoo:
    # Here come the __init__ method

    # method 1

    # method 2

    # ...
```

__init__ Methode

Diese Methode wird immer aufgerufen, wenn eine Objekt dieser Klasse erzeugt wird. Die Metode ermöglicht es der Klasse, die Attribute des Objekts zu initialisieren. Hier sind die Attribute *name*, *location* und eine Liste von *animals*.

```
    # Create a class named "zoo"
    class Zoo:
        def __init__(self, name, location):
            self.name = name
            self.location = location
            self.animals = []

        # method 1

        # method 2

        # ...
```

self

In der `__init__` Methode werden die Variablen durch das Keyword `self.attribut_name` aufgerufen. **Self** repräsentiert die Instanz der Klasse, hier also den Zoo, welchen wir eben erzeugt haben. Mit **self** können wir auf die Methoden und Attribute der Instanz zugreifen. Dafür werden die Attribute mit den Argumenten beim Erzeugen der Instanz "gebindet".

Methoden

Methoden fügen der Klasse Funktionalität hinzu. Die Methoden können über das Objekt aufgerufen werden und könnnen auf die Attribute der Instanz zugreifen.
Eine Methode der Klasse `zoo` könnte z.B. folgendermaßen aussehen:

```
    # Define a method, that adds an andimal to the list of animals

    def add_animal(self, animal):
        self.animals.append(animal)
        print(f"{animal} just joined the zoo.")
```

Aufgabe: Methoden 1 (10 min)

Schreiben Sie eine Methode für die Klasse `zoo`, welche ein Tier aus der Liste von Tieren entfernt, wenn das Tier in der Liste enthalten ist.
Wenn die Aktion erfolgreich durchgeführt wurde, gebe folgenden Text aus:

```
Animal XY just left the zoo!
```

Wenn kein Tier in der Liste ist oder nicht der korrekte Namen genannt wurde, soll folgender Text ausgegeben werden:

```
Animal XY is not is not part of zoo!
```

Aufgabe 2: Methoden 2 (20 min)

Schreiben Sie zwei Methoden:

1. Eine Methode (**get_animals()**), welche die Liste der Tiere als `return` Wert zurückgibt
2. Eine Methode (**describe()**), welche die Attribute des Objekts (name, location, animals) als eine Art Beschreibung ausgibt. Ein möglicher Output könnte sein:

```
# Methode 2
Welcome to *name* in *location*!
Our zoo has the following animals:
- tiger
- cat 1
- cat 2
- goat
```

Verwenden Sie eine *for-loop* um die Namen der Tiere im Zoo untereinander auszugeben.

Klassen verwenden, Instanzen erzeugen und Methoden aufrufen

Um die Klasse `zoo` sinnvoll nutzen zu können, müssen wir ein Objekt der Klasse `zoo` erzeugen. Dies können wir entweder unter der Definition der Klasse platzieren oder in einer weiteren Python Datei, welche die Klasse bzw. das Modul importiert. Wir werden hier die erste Variante wählen, um weniger Dateien zu erzeugen.
Eine Instanz der Klasse `zoo` wird mit folgendem Python-Code erzeugt:

```
test_zoo = Zoo("Kölner Zoo", "Köln")
```

Lassen Sie sich die Attribute ausgeben, welche die Instanz `test_zoo` besitzt und verwenden sie dafür die *describe()* Methode:

```
test_zoo.describe()
```

Erzeugen Sie einen weiteren Zoo, geben Sie jedoch nur einen Input-Parameter in der Klammer an.

```
second_zoo = Zoo("Wiener Zoo")
```

Warum wird Ihnen ein Fehler angezeigt?

Verwenden Sie mehrmals Ihre Methode, um dem Zoo Tiere hinzuzufügen.
Prüfen Sie dann mit der Metode `get_animals()`, welche Tiere im Zoo sind.
Entfernen Sie einige Tiere und verwenden Sie die Methode erneut.

Funktioniert Ihr Code?? - Beginnen Sie jetzt mit der Implementierung der `FindPeaks` Klasse.

A1: Implementierung der Klasse `FindPeaks`

Alle Aufgaben sollen in einer Python Datei gelöst und umgesetzt werden. Verwenden Sie die Kommentarfunktion, um den die jeweilige Nummer der Aufgabe anzugeben. Beispiel

```
# A1.1
# This should be the Answer to Task A1.1

# A1.2 - define the class Zoo
# class Zoo:
```

A1.1 (2 Punkte)

- Strukturieren Sie Ihren Code zuerst, indem Sie mit Kommentaren Platzhalter für Klassen, Methoden und die Aufrufe erstellen
- Verwenden Sie die Datei `A1_Kommentare.py` als Idee dafür, wie solche Kommentare aussehen könnten

A1.2

- Programmierung der Klasse `FindPeaks` mit folgenden Methoden. Erstellen Sie dafür zuerste eine Python Datei mit dem Namen **FindPeaks.py**:

1. **Laden der Daten** (2 Punkte)
   Es sollen csv-Files, wie in Übung 3 (EKG Daten), eingelesen werden und als Variable vom Typ `pandas.DataFrame` in einem Attribut der Klasse gespeichert werden.
2. **Detektion der Peaks** (4 Punkte)
   Die Methode soll in den EKG Daten des Dataframes die Hochpunkte finden und in einem neuen Dataframe (**peaks**) speichern, wobei die Indizes, also die Stellen an denen die Hochpunkte auftreten, in der Spalte mit dem Namen `Indizes` gespeichert werden sollen.

3. **Berechnung der Eigenschaften der Peaks** (8 Punkte + 2 Extrapunkte)
   Peaks haben Eigenschafte, welche Auskunft über die Daten geben. Schreiben Sie drei Methoden, welche in der Methode `PeakInformations()` aufgerufen werden mit den folgenden Funktionalitäten:
   - *Abstand zum nächsten Peak (in ms) / Boxplot (3 Punkte)*
     Berechnen Sie den Abstand zum nächsten Peak (davor und danach) und schreiben Sie die Werte in Millisekunden in den Dataframe (**peaks**). Verwenden Sie eine Spalte für die Zeiten der Peaks davor und eine Spalte für die Peaks, die danach auftreten.
   - *Hochpunkt der Peaks (Absolutwert) / Histogramm (2 Punkte)*
     Schreiben Sie die Absoluten Messwerte in eine Spalte des Dataframes **peaks**.

- *Höhe der Peaks (Absolutwert zur Baseline) / Boxplot (fortgeschritten) (3 Punkte + 2 Extrapunkte)*
  Berechnen Sie auch die relative höhe der Peaks zur "Baseline". Dafür müssen Sie jeweils die zwei nächsten Tiefpunkte um den Höhepunkt herum identifizieren und die Absolutwerte dieser beiden mitteln. Dadurch berechnen Sie die "Baseline" und können diese dann vom Absolutwert abziehen

1. **Darstellen der Peaks und der Eigenschaften** (5 Punkte)
   Nutzen Sie Matplotlib oder eine Python Bibliothek Ihrer Wahl, um die Peaks aus *Aufgabe 2* darzustellen. Schreiben Sie auch hierfür Methoden, welche Boxplots, Histogramme und Line-Graphs erzeugen.

A1.3 Feinheiten (2 Punkte)

Schauen Sie sich nochmals Ihren Code an und versuchen Sie die Benennung der Variablen und Methoden sinnvoll zu wählen, damit dritte Personen Ihren Code einfach "lesen" können.

A1.4 Kommentare (1 Punkt)

Wenn nötig, kommentieren Sie Ihren Code! Falls Ihr Code nicht funktioniert können Sie die Kommentare nutzen, um ihre Problem zu beschreiben!

A1.5 Doctrings (2 Punkte)

Verfassen Sie für jede Methode einen **Docstring**.

A1.6 Final Check (1 Punkt)

Ihr Code funktioniert und erzeugt die verlangten Daten und Darstellungen.

Punkteverteilung

Aufgabe	Punkte
A1	2
A2	19 + 2 Extrapunkte
A3	2
A4	1
A5	2
A6	1
Gesamt	27 (29)

Abgabe

Die Aufgabe soll als *.Zip* File in SAKAI abgegeben werden. Dafür erhalten Sie separat auf SAKAI eine Benachrichtigung, welche das Abgabedatum enthält.