

## Data Science 2

### Übung 8

#### Einführung in die Programmierung



#### Über dieses Übungsblatt

Im letzten Übungsblatt haben wir Daten eingelesen und angezeigt. Nun möchten wir etwas mehr auf den Verarbeitungsaspekt eingehen.

## Aufgabe Konzeptionelle Fragen

*Letzte Änderung: 15. June 2023, 12:38 Uhr*

**12 Punkte** – [im Detail](#)

Ansicht:  |  Im Folgenden sollen eher konzeptionelle Fragen beantwortet werden. Geben Sie kurze Erklärungen im Freitext (typischerweise 1 Satz pro Aussage). Es kann durchaus viele verschiedene richtige Antworten geben, Mehrfachantworten sind aber nicht nötig.

**1) Statische Typisierung:** In Python können Typen von Variablen mit Typannotationen versehen (und dann mit mypy geprüft werden). Nennen Sie je einen Vor- und einen Nachteil davon, Variablen im Programmcode mit festen Typen zu versehen (für ExpertInnen: wir nutzen keine Generics).

**2) Objekte & Referenzen:** Lesen Sie den folgenden Programmcode und geben Sie das Ergebniss der Vergleiche (a) und (b) (c) und (d) **mit Begründung** an.

```
# This class can be used to store a fraction. The denominator has to be != 0
class Frac:
    def __init__(self, num, denum):
        self.num = num
        if denum != 0:
            self.denum = denum
        else:
            raise ValueError("Denominator has to be != 0")

    def __eq__(self, other):
        return other.num == self.num and other.denum == self.denum

# Create two Instances of Frac
x = Frac(2,3)
y = Frac(2,3)
z = x

# (a)
x == y

# (b)
x is y

# (c)
x == z

# (d)
x is z
```

3) Objekte & Referenzen: die Funktion `replace_nan` soll alle `nan`-Werte in einer Liste durch den Wert 0 ersetzen. Dies kann z.B. bei einem Experiment mit fehlenden Messwerten notwendig sein. Welche der beiden Funktionen liefert das korrekte Ergebniss und **warum**?

```
from copy import deepcopy

x = [None, 2, 3, None, 5]
y = deepcopy(x)

# Replaces nan values with 0
def replace_nan1(arr):
    for val in arr:
        if val is None:
            val = 0

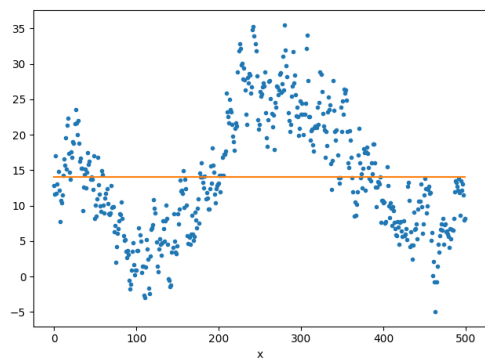
# Replaces nan values with 0
def replace_nan2(arr):
    for i in range(len(arr)):
        if arr[i] is None:
            arr[i] = 0
```

## Aufgabe (Laufende) Mittelwerte

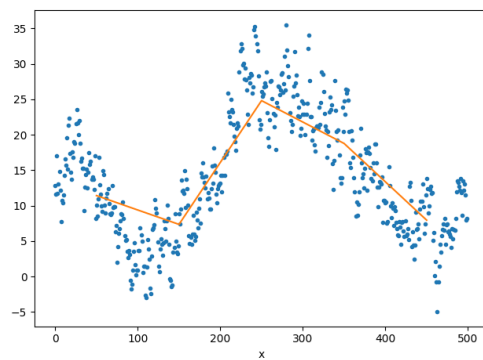
*Letzte Änderung: 06. July 2023, 13:26 Uhr*

28 Punkte — [im Detail](#)

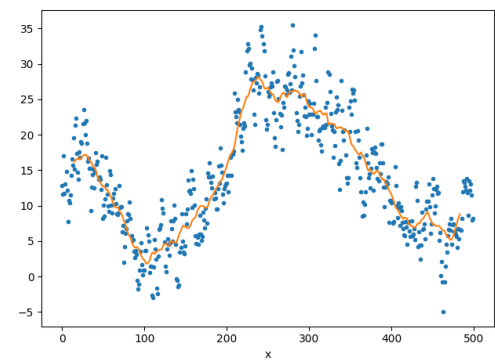
Ansicht:  | 



(a) konstante Approximation



(b) stückweiser Mittelwert



(c) laufender Mittelwert.

Verschiedene Approximationsergebnisse, basierend auf (laufenden) Mittelwerten.

In dieser Aufgabe arbeiten wir erneut mit dem Wetterdaten aus Blatt07. Wir möchten die Daten allerdings diesmal nicht nur visualisieren, sondern auch interpolieren, also Vorhersagen über ungesehene Datenpunkte machen. Dies können wir mithilfe einer leichten Abwandlung eines einfachen Konzepts erreichen, den Sie bereits aus der Schule kennen sollten, des **Mittelwertes**.

Der Mittelwert  $\frac{1}{n}(y_1 + y_2 + \dots + y_n)$  von einer Reihe von Messwerten  $y_i$  haben wir bereits in Blatt03 berechnet. Das liefert natürlich zunächst nur eine Zahl, aber mit ein bisschen Schieben, bekommt man ein leistungsfähiges Verfahren, um Rauschen aus Daten zu entfernen (und damit brauchbare Vorhersagen zu machen).

Laden Sie erneut die Wetterdaten aus Blatt07 (siehe Aufgabe *Matplotlib*) ein. Wie die Daten einzulesen sind, können Sie ggf. in Blatt07 nachlesen. Wir arbeiten erneut mit der Temperatur (Spalte *TX*). **Verwenden Sie Matplotlib für alle Diagramme.**

**1)** Berechnen Sie den Mittelwert ihres Datensatzes und fügen Sie eine Linie auf Höhe des Mittelwertes Ihres Datensatzes Ihrem Plot aus Aufgabe *Matplotlib 1c*) aus dem letzten Übungsblatt in einer anderen Farbe hinzu (siehe Abbildung (a)). Verwenden Sie das optionale Argument `linestyle=" "` im `plot`-Befehl der Rohdaten, damit die einzelnen Punkte nicht verbunden werden und ein Argument für `marker`, wie z.B. `.` oder `x`. Die Syntax des plot Befehls lautet dann beispielsweise `plt.plot([x], [y], color="blue", linestyle=" ", marker=".")`.

**Hinweis1:** Numpy bietet bereits eine Mittelwert-Funktion für Vektoren an: [np.mean](#), die Sie verwenden können.

**Hinweis2:** Um den Mittelwert einzutragen, können Sie die [plt.hlines](#) Funktion von Matplotlib verwenden, oder einfach einen konstanten Vektor der `plt.plot` Funktion übergeben. Die Syntax zur `hlines` Funktion lautet `plt.hlines(y-Wert, xmin, xmax, color="orange")` wobei `x_min` und `x_max` den Wertebereich angeben.

Der Mittelwert selbst ist natürlich wenig aussagekräftig was den Verlauf des Datensatzes betrifft. Daher überlegen wir uns ein verbessertes Schema: Wir teilen den Datensatz in gleich große Teile ein und bestimmen den Mittelwert jedes einzelnen Teils.

**2a)** Teilen Sie die  $n$  Datenpunkte der Temperaturspalte in 3 gleich große Intervalle (Abschnitte) auf (Den letzten unvollständigen Teil können Sie dabei weglassen). Berechnen Sie zu jedem der 3 Intervalle den zugehörigen Mittelwert der Daten in diesem Abschnitt. Tragen Sie nun die 3 Punkte (in einer anderen Farbe als die Datenpunkte) in das Diagramm ein (siehe Abbildung (b)). Verwenden Sie als  $x$ -Werte die jeweiligen Mittelpunkte der Intervalle. Verwenden Sie dabei die `plt.plot` Funktion mit dem Standardargument `linestyle="-"` für eine durchgängige Linie.

**Hinweis:** Um Teile der Daten zu betrachten kann die Slicing-Notation helfen: `data[10:20]` gibt die Datenpunkte 10-19 aus.

**2b)** Wir möchten nun das Vorgehen aus Aufgabenteil 2a) für  $m$  Intervalle verallgemeinern (falls Sie das nicht schon getan haben). Berechnen Sie zunächst die Länge eines Intervalles, wenn Sie  $n$  Datenpunkte in  $m$  gleich große Teile unterteilen. Berechnen Sie mit einer `for` oder `while` Schleife erneut wie in 2a) die Mittelwerte und zugehörigen  $x$ -Werte der  $n$  Intervalle und tragen Sie diese (in einer anderen Farbe als die Datenpunkte) in das Diagramm ein.

Was wir eben mit festen Teilen des Datensatzes gemacht haben können wir nun auch mit „mitlaufenden“ Teilen realisieren.

**3a)** Erstellen Sie eine leere Liste `moving_avgs`. Iterieren Sie nun über alle Datenpunkte mit einer `for`-Schleife; für jeden Datenpunkt, berechnen Sie den Mittelwert des Datenpunktes und der zwei angrenzenden Datenpunkten (davor und danach) und fügen Sie das Ergebniss der Liste hinzu: `moving_avgs.append([resultat])`. Sie können dabei den ersten und letzten Datenpunkt in der `for`-Schleife auslassen, da ja einer der Nachbarn fehlt oder einfach die fehlenden Werte im Mittelwert ignorieren.

**3b)** Visualisieren Sie die Werte aus der in 3a) erstellten Liste `moving_avgs` und die Rohdaten in einem Diagramm, ähnlich wie in Abbildung

(c). Achten Sie dabei darauf die richtigen  $x$ -Werte zu verwenden.