“sensormap package”

Hausübung 2

April 15, 2019

1 Datensätze

Wir werden weiter mit den Feinstaubdaten, die schon in Informatik 1 aufbereitet wurden, arbeiten:
Originaldaten: https://luftdaten.info/

Laden Sie die Angabedaten[1] auf Ihr System. Das ZIP Archiv besteht aus drei Dateien. Erstens dem Datensatz der Sensoraufzeichnungen (sensors_ass2_small.csv), zweitens der selbe Datensatz mit mehr Sensoren (sensors_ass2.csv) und drittens der Datensatz aller verzeichneten Städte und deren Daten (worldcities.csv). Achten Sie darauf, dass Sie diese Dateien bei Ihrer Abgabe nicht mit hochladen!

2 Tasks

2.1 Klasse Coordinates (3Pt)

Schreiben sie eine Klasse `Coordinates`. Diese dient dazu, Breiten- und Längengrade zu speichern. Sie soll folgende *protected*-Attribute beinhalten:

- `_lat`: Breitengrad (in °), float
- `_lon`: Längengrad (in °), float

Implementieren Sie für beide Attribute Getter und Setter über Properties (diese haben *keinen* Unterstrich im Namen). Überprüfen Sie im jeweiligen Setter, ob der zuzuweisende Wert eine passende Zahl (d.h. in einen float konvertierbar) ist. Werfen Sie andernfalls eine *passende* Exception mit *hilfreicher* Fehlermeldung. Kontrollieren Sie weiters, ob der gegebene Wert im passenden Wertebereich liegt (ansonsten wieder entsprechende Exception+Fehlermeldung).

Abgesehen von den Properties sollen folgende Methoden implementiert werden:

`__init__(self, lat, lon)`: Der Konstruktor soll Werte für Breiten- und Längengrade übernehmen und sie über die entsprechenden *properties* speichern.

`@classmethod`
`distance(cls, coord_1, coord_2)`: Die *Klassenmethode* (`staticmethod` ist auch in Ordnung) `distance`

[1] https://palme.iicm.tugraz.at/wiki/images/INF/ass_2_files.zip

soll zwei *Coordinates*-Objekte erwarten . Rückgabewert soll die Distanz zwischen den beiden Koordinaten in km sein (als float). Diese berechnet sich wie folgt:

$$distance = 6371 \cdot arccos(sin(lat_1) \cdot sin(lat_2) + cos(lat_1) \cdot cos(lat_2) \cdot cos(lon_2 - lon_1))$$

mit Breiten- und Längengradswerte in *Radianten*!

2.2 Klasse City (2Pt)

Schreiben sie eine Klasse `City`. Sie soll folgende *protected*-Attribute beinhalten:

- `_name`: Name der Stadt, string
- `_population`: die Bevölkerungszahl der Stadt, ~~int~~ float
- `_coords`: Koordinaten des Stadtzentrums, object of Coordinates
- `_country`: Land in dem die Stadt liegt, str
- `_colorcode`: Farbwahl für die Marker der Stadt, ~~tuple~~ string
- `_sensors`: Liste von Sensoren, die der Stadt zugeordnet sind, list of Sensor objects

Legen Sie über folgende Attribute read-only properties, um diese vor Zugriffen von Außen zu schützen: `_name`, `_population`, `_coords`, `_country`, `_sensors`.
Schreiben Sie für das Attribut `_colorcode` einen Getter und Setter über Properties.

Abgesehen von den Properties sollen folgende Methoden implementiert werden:

`__init__(self, name, coords, population, country, colorcode)`: ~~Der Konstruktor soll Werte für die angegebenen Attribute übernehmen und sie über die entsprechenden *properties* speichern.~~
Der Konstruktor soll die angegebenen Parameter übernehmen und in die entsprechenden Attribute speichern.

`add_sensor(self, sensor)`: Die Methode soll ein Objekt der Klasse Sensor übergeben bekommen und diesen dann zur Liste der Sensoren in dem Attribut `_sensors` der Klasse `City` hinzufügen. Achten Sie durch eine geeignete Überprüfung darauf, dass das übergebene Objekt auch wirklich eine Instanz der Klasse `Sensor` ist.

2.3 Klasse Sensor (4Pt)

Schreiben sie eine Klasse 'Sensor'. Sie soll folgende *protected*-Attribute beinhalten:

- `_id`: Sensor-ID, int
- `_city`: Die Stadt, in welcher der Sensor steht, City
- `_coords`: Koordinaten des Sensors, Coordinates
- `_distance_to_centre`: Distanz zum Stadtzentrum, float
- `_data`: Sensordaten, pandas-Dataframe
- `_address`: Adresse des Sensors, None/string

Schreiben sie für die Attribute `_id, _city, _coords, _distance_to_centre` und `_data` read-only properties.

Die Objekte sollen wie folgt initialisiert werden:
`__init__(self, sensor_id, city, coords, data)`: Die übergebenen Werte werden in die entsprechenden Attribute gespeichert, `_address` soll mit `None` initialisiert werden.

Schreiben Sie eine property `address`, welche beim erstmaligen Aufruf die Adresse über die ArcGIS-API abruft und in das zugehörige Attribut speichert. Bei Folgeaufrufen soll dann der API-Aufruf übersprungen und die Adresse direkt aus dem Attribut gelesen werden. Die Addresse kann wie folgt abgerufen werden:

```python
from geopy.geocoders import ArcGIS
geolocator = ArcGIS()
address = geolocator.reverse((lat, lon)).address
```

Schreiben Sie weiters eine Methode `create_bokeh_plot(self, smooth=False)`. Diese soll - analog zu Assignment 3 in Informatik 1[2] - aus den Daten des Sensors eine bokeh-Figure erstellen und zurückgeben. Wird der Parameter `smooth` gesetzt, sollen die Daten über die angegebene Fensterbreite geglättet werden. Dies kann über die Dataframe-Methode `rolling` wie folgt erledigt werden:

```python
data[['P1', 'P2']] = data.rolling("30d", on="timestamp").mean()[['P1', 'P2']]
```

`"30d"` ist hier ein Beispielwert für `smooth`, und gibt die Breite des gleitenden Mittelwertfilters an (in diesem Fall 30 Tage). Die Daten sollten allerdings nicht inplace geglättet werden, da ansonsten spätere Plots mit `smooth=False` zu falschem Ergebnis führen.
Der Plot soll so aussehen wie der Beispielplot im Wiki. Die Farben sind grundsätzlich frei wählbar, sollen aber etablierten Standards für Farbenblindheit entsprechen. Bei dem bereitgestellten Codestück vom letzten Assignment wurde fill_between() von Matplotlib verwendet. Die gefüllten Bereiche unterhalb der Kurven lassen sich mit `Band` erstellen, plotten sie aber dennoch die Linien selbst, da sonst die Achsen nicht richtig angepasst werden.
Verwenden Sie auch das Bokeh HoverTool um einen Mouseover Effekt bei den Popups zu erreichen. Dies soll (wie in der Referenz) jeweils das Datum, den P1 und den P2 Wert an der aktuellen Mausposition anzeigen. Achten Sie darauf, dass das Datum wirklich als Datum angezeigt wird (und nicht als große Zahl). Setzen Sie beim Erstellen des HoverTools den Parameter mode auf ”vline” und den Parameter point_policy auf ”follow_mouse”. Dies führt zu einem Verhalten wie bei der Referenz. Damit das HoverTool richtig funktioniert binden Sie es nur an die Linie von P1 (die Sie zusätzlich zum Band geplottet haben). Dies können Sie mit dem Parameter renderers() beim HoverTool erreichen. Setzen Sie die Höhe des Bokeh Plots auf 300px, die Breite auf 400px.

2.4 Weitere Funktionen (7Pt)

2.4.1 read_cities (2Pt)

Schreiben Sie eine Funktion `read_cities(path, countries=[])`. Diese soll aus der im Parameter `path` (string) spezifizierten Datei diejenigen Städte einlesen, die in vom Parameter `countries` bestimmten Ländern liegen. Groß- und Kleinschreibung der Ländernamen soll hier ignoriert werden! Ist `countries` eine leere Liste (Defaultwert), sollen alle in der Datei vorkommenden Städte eingelesen werden. Für jede eingelesene Stadt soll ein `City`-Objekt erstellt und an eine Liste angehängt werden. Diese Liste soll am Ende der Funktion zurückgegeben werden. Bei der initialen Zuweisung der Farbcodes haben Sie freie Hand. Wichtig ist lediglich, dass die Codes beim Plotten richtig dargestellt werden. Die mögliche Farben können Sie in der Dokumentation unter ”class Icon” finden.[3] ~~Seaborn color_palette bietet Ihnen zum Beispiel hilfreiche Listen[4].~~

2.4.2 find_nearest_city (1Pt)

Schreiben Sie eine Funktion `find_nearest_city(cities, coords)`. Die Funktion bekommt eine Liste von Städten, sowie ein Objekt der Klasse `Coordinates` übergeben. Ziel dieser Funktion ist es, die näheste Stadt zu finden und zurück zu geben. Um die Distanz zwischen einer Stadt und den Koordinaten zu berechnen, soll die bereits geschriebene Funktion `distance()` verwendet werden. Falls keine der Städte näher als 10km entfernt liegt, so soll `None` returned werden.

[2] https://palme.iicm.tugraz.at/wiki/Info1BM
[3] https://github.com/python-visualization/folium/blob/master/folium/map.py
[4] https://seaborn.pydata.org/generated/seaborn.color_palette.html

2.4.3 parse_sensors (2Pt)

Schreiben Sie eine Funktion `parse_sensors(data_path, cities)`. Der Parameter `data_path` (string) gibt an, aus welcher Datei die Sensordaten gelesen werden, `cities` ist eine Liste aus `City`-Objekten. Die Funktion soll nun diejenigen Sensoren einlesen, die einer Stadt zugeordnet werden können (siehe `find_nearest_city`). Für jeden Sensor soll ein `Sensor`-Objekt angelegt werden, das im Parameter `city` eine Referenz auf die nächstgelegene Stadt (als Objekt, nicht als string!) bekommt. Das fertige `Sensor`-Objekt soll außerdem an die `sensors`-Liste der entsprechenden Stadt angehängt werden.

2.4.4 create_map (2Pt)

Schreiben Sie eine Funktion `create_map(plot_list, filename, plot_sensor_values=False, smooth=False, zoom=?)`. Die Funktion soll die html Map mit den verschiedenen Markern der Sensoren und deren Popups erstellen.
Übergabeparameter:

- `plot_list`: Entweder eine Liste von Städten oder Sensoren (auch gemischte Listen möglich), die auf der Karte darzustellen sind, Objects of Sensor and/or City
- `filename`: Name unter dem die Map gespeichert werden soll, string
- `plot_sensor_values`: Boolean, welcher angibt ob nur die Informationen der Stadt (bei False) oder der Plot der Sensordaten (bei True) als Popup auf der Map angezeigt werden sollen, Boolean mit Default auf False
- `smooth`: gibt an ob (bzw. mit welchem Fenster) die Daten geglättet im Popup der map dargestellt werden sollen, default False
- `zoom`: Wert für den Anfangszoom auf die Map, int mit Default (Bitte setzen Sie den Default auf einen geeigneten Wert).

Überprüfen Sie ob die Elemente der Liste `plot_list` jeweils Objekte der Klassen Sensor oder City sind. Werfen Sie andernfalls eine *passende* Exception mit *hilfreicher* Fehlermeldung.
Erstellen Sie nun eine Map mit Hilfe des Packages *folium*: Setzen sie das Zentrum der Karte auf den Median der Koordinaten der zu plottenden Sensoren. Als width and height sollen jeweils 1000 gewählt werden. Der übergebene Zoom soll hier eingesetzt werden.
Erstellen Sie Marker-Cluster mit der Funktion *MarkerCluster(overlay=True, control=True,)*.
Sollte der `plot_sensor_values` auf True sein, soll für jeden Sensor ein Popup mit dem Plot der Feinstaubdaten des jweiligen Sensors erscheinen, dafür können sie folgenden Aufruf verwenden:

```
popup = folium.Popup(
        folium.IFrame(
            folium.Html(
                file_html(bokeh_plot, CDN),
                script=True),
            height=325,
            width=425), max_width=425)
```

bokeh_plot ist hier eine fertige bokeh-Figure. Die heights und widths sollen alle so hardgecoded übernommen werden.
Ist `plot_sensor_values` auf False, soll ein Popup mit folgenden Beschreibungen (in fetter Schrift) und den dazugehörigen Daten (in normaler Schrift) des jeweiligen Sensors erscheinen. (Nach dem ":" soll ein Leerzeichen stehen)

Sensor ID: "sensor ID" **City:** "nearest City to the sensor"
Population: "Population of the nearest City"
Address: "Address of the Sensor"
Distance to centre: "Distance from Sensor to Citycentre"

(The "" are only for demonstration sake and should not appear in your Popups)

Fügen Sie die Marker zur Map hinzu (siehe `folium`). Nun muss nur noch der Marker zum Cluster von Markern und das Cluster dann zur Map hinzugefügt werden und die Map ist fertig!
Speichern Sie die Map unter dem Namen, der durch den Parameter `filename` spezifiziert wird.

2.5 Package (2Pt)

Implementieren Sie Ihre Abgabe als eigenständiges python Package mit dem Namen `sensormap`. Klassen sollen in entsprechenden Dateien stehen (zB. die Klasse `City` in der Datei `city.py`), die restlichen Funktionen in `sensormap_functions.py`. Erstellen Sie dafür auch eine `__init__.py` Datei, in der Sie ihre Klassen und Funktionen importieren bzw. bereitstellen. Mit folgendem Code soll man Klassen und Funktionen aus Ihrem Package importieren können:

```python
from sensormap import City, Sensor, read_cities, parse_sensors, create_map
```

WICHTIG: Achten Sie unbedingt auf die korrekte Benennung von Dateien, Klassen, Funktionen, Attributen, Methoden und Parametern, da wir Ihr Programm bzw. Package sonst nicht testen können!

2.6 Testen (2Pt)

In der Datei `assignment_2.py` sollen Sie nun Ihr Package verwenden. Lesen Sie alle Städte ein, die in Österreich und Deutschland liegen. Lesen Sie dann diejenigen Sensoren ein, die ebendiesen Städten zugeordnet werden können. Geben Sie nun für jeder Stadt mit mindestens einem Sensor aus, wie viele Sensoren dort eingelesen wurden, in folgendem Stil:

```
Graz: 2
München: 3
```

Erstellen Sie eine Karte, die für alle eingelesenen Sensoren einen Marker mit Informationen über deren Standort (als Popup) enthält und speichern sie das Ergebnis unter dem Namen 'map_info.html'. Erstellen Sie weiters eine Karte mit den Plots der Sensordaten im Popup (geglättet über 15 Tage), allerdings nur für die Sensoren in Stuttgart und Graz. Speichern sie diese unter 'map_plots.html'

3 Beschränkungen

- Fügen Sie keine nutzlosen Komponenten Ihrer Abgabe hinzu
- Sofern möglich, verwenden Sie eingebaute Funktionen
- Importieren Sie nur erlaubte Packages und solche, die Sie dann auch verwenden
- Verwenden Sie keine Kommandozeilenparameter

3.1 Erlaubte Packages

- `os`, `sys`, `math`
- `numpy`, `pandas`, `tqdm`
- `matplotlib`, `seaborn`, `bokeh`, `folium`, `geopy`

4 Datei Header

All Ihre Quelldateien (oder Notebooks) in Ihrer Abgabe müssen gleich zu Beginn einen Kommentar mit folgenden Informationen enthalten:

- Author: – Ihr Name
- MatNr: – Ihre Matrikelnummer
- Description: – Generelle Beschreibung der Datei
- Comments: – Kommentare, Erklärungen, usw

Bitte einfach den folgenden Code in Ihre Dateien kopieren und den Inhalt anpassen. Je nach PDF-Reader müssen Sie eventuell die Leerzeichen/Einrückungen per Hand anpassen. Bei jupyter Notebooks fügen Sie den Kommentarheader bitte in die erste Zelle ein.
Beispielheader:

```
#########################################################################
# Author:      [FIRST NAME] [LAST NAME]
# MatNr:       [MATR NR]
# Description: [SHORTDESCRIPTION]
# Comments:    [ANY RELEVANT COMMENTS.
#              CAN BE MULTILINE]
#########################################################################
```

5 Coding Standard (Abzug bis 50%)

Für diese Lehrveranstaltung orientieren Sie sich am offiziellen PEP 8 Standard[5]. Dieser Beschreibt grundsätzliche Formalitäten im Bezug auf Ihren Code. Folgendes ist besonders zu Beachten:

Sprache. Code schreibt man in Englisch. Im internationalen Zeitalter ist es notwendig, dass auch jemand am anderen Ende der Welt verstehen kann, was Sie programmiert haben. Ihr gesamter Quellcode muss daher auf Englisch geschrieben sein. Dies betrifft sowohl die Kommentare also auch Variablennamen und Ähnliches.

Leerzeichen statt Tabulatoren. Python basiert auf Einrückungen, anstatt auf geschwungenen Klammern . Theoretisch gibt es die Möglichkeit, Leerzeichen (spaces) oder Tabulatoren (tabs) zu verwenden. PEP8 schreibt aber klar 4 Leerzeichen als Einrückungen vor. Die meisten Python Programmierumgebungen werden automatisch 4 Leerzeichen einfügen, wenn Sie auf die Tabulator-Taste drücken.

Sprechende Namen. Verwenden Sie kurze, aber sprechende Namen für Ihre Variablen, Funktionen, (und Ähnliches). Es muss eindeutig aus dem Namen hervorgehen, was die Aufgabe des Elements ist. Für simple Iterationen kann ein einfaches i ausreichend sein, aber dies kann schnell zu Chaos führen. Sollten Sie Variablen haben, die keine Aufgabe haben und nicht verwendet werden, schreibt PEP 8 vor, einen einfachen Unterstrich (_) zu verwenden. Achten Sie bei der Groß-Kleinschreibung auf die Richtlinien aus PEP8.

120 **Zeichen Zeilenlänge.** PEP8 sieht Zeilenlängen von maximal 79 Zeichen vor. Ein anderes, etwas großzügigeres Limit, welches sich in der Szene etabliert hat, sieht eine Länge von 120 Zeichen vor. In dieser LV verwenden wir daher das erweiterte Limit. Bitte achten Sie darauf, dass keine Zeile in Ihrem Code diese Länge von 120 Zeichen überschreitet (gilt auch für Kommentare). *Hinweis:* Zeilenlänge inkludiert die Einrückungen mittels Leerzeichen!

[5] `https://www.python.org/dev/peps/pep-0008/`

6 Automatisierte Tests

Ihr Programm wird automatisiert getestet. Achten Sie daher, dass Sie die Angabe genau einhalten. Dies gilt im Besonderen für vorgeschriebene Variablen- und Funktionsnamen!
Vergewissern Sie sich, dass Ihre Abgabe allen Beschränkungen, die in dieser Angabe erwähnt sind, konform ist. Zusätzlich wird jede Abgabe auch von einem Mitglied des Tutorenteams begutachtet. Ihre Tutoren führen auch die finale Bewertung (Puntke) durch.

7 Abgabe

7.1 Deadline

05. Mai 2019 um 23:59:59.

Eine Spätabgabe ist nicht vorgesehen. Ausnahme bilden hier Notfälle. Sollte das Abgabesystem nicht online sein, verlängert sich die Deadline automatisch um 24 Stunden. Sollten Sie, aus diversen Gründen, nicht in der Lage sein, Ihre Abgabe hochzuladen, kontaktieren Sie Ihren Tutor *VOR* der Deadline. (*Hinweis:* Urlaub oder Ähnliches wird nicht als Grund akzeptiert!)

7.2 Hochladen der Abgaben

Assignments werden stets als Archive abgegeben. Erlaubt sind hier die Formate *.zip*, und *.tar.gz*.
Zusätzlich zu ihren Quelldateien, soll Ihre Abgabe auf eine Datei namens *readme.txt* beinhalten. Das Vorhandensein der Datei ist Pflicht, ihr Inhalt aber optional. Sie soll folgenden Inhalt haben: (i) Die Zeit, die Sie benötigt haben, um die Aufgabenstellung zu absolvieren. (ii) Feedback, wo Sie Probleme hatten. Ihr Feedback ermöglicht es, verbreitete Probleme zu erkennen und die Vorlesung und Tutoriumseinheiten entsprechend anzupassen.
Abgaben erfolgen auf der Palme Website. Bitte prüfen Sie vor der Abgabe diese Kriteren:

- Datei- und Ordnerstruktur (siehe unten)
- Kommentarheader in jeder Quelldatei
- Coding Standard

7.3 Struktur der Abgabe

```
assignment_2.zip (or assignment_2.tar.gz)
    assignment_2.py
    readme.txt
    sensormap
        __init__.py
        city.py
        sensor.py
        coordinates.py
        sensormap_functions.py
```