

## **Dokumentation zum Projektseminar „Angewandte Informationswissenschaft“**

### **Idee:**

Analysetool zur Prüfung von Wetterdaten auf Korrektheit von Wetterweisheiten und Bauernregeln für den Raum Düsseldorf.

### **Einleitung:**

Es gibt eine große Zahl an Wetterweisheiten und Bauernregeln. Wetterweisheiten beziehen sich auf verschiedene Faktoren an bestimmten Kalendertagen, durch die eine Aussage getroffen werden kann, wie bestimmte Wetterparameter für eine bestimmte Zeit voraussichtlich sein werden. Bauernregeln unterscheiden sich von Wetterweisheiten darin, dass es zwei Sorten dieser gibt. Die eine beschreibt allgemeingültige Sprüche, die keinerlei Aussagekraft besitzen wie: *„Kräht der Hahn auf dem Mist, ändert sich das Wetter oder es bleibt wie es ist.“*, diese Sorte an Bauernregeln wird in dieser Dokumentation und im Analysetool nicht weiter betrachtet. Die anderen sind vergleichbar zu Wetterweisheiten, da auch diese kommende Wetterwerte vorhersagen. Die zentrale Frage ist: *„Wie genau sind diese Wetterweisheiten bzw. Bauernregeln wirklich und treffen sie für den Bereich Düsseldorf zu?“* Um diese Frage klären zu können, werden zuerst valide Wetterdatenaufzeichnungen benötigt. Hierbei sollte ein größerer Bereich (1990-2015) untersucht werden, um eventuelle temporäre Abweichungen ausschließen zu können. Anschließend sollten diese Daten mittels einer Programmiersprache (Python) analysiert und bewertet, sowie das Ergebnis graphisch dargestellt werden.

### **Projektidee:**

Bei meiner Projektidee möchte ich die Daten z.B. von der Datenbank des Deutschen Wetterdienstes verarbeiten. Dabei sollen diese daraufhin untersucht werden, ob Weisheiten wie z.B. Siebenschläfer, Bauernregeln oder Ähnliches für einen bestimmten Zeitraum in einem geographischen Bereich z.B. Düsseldorf zutreffend waren oder nicht, bzw. wie hoch die Fehlerquote ist oder der Grad der Abweichung. Das Programm zur Analyse würde ich gerne mit Python umsetzen. Zur Steuerung des Programms, könnte eine GUI verwendet werden. Durch diese könnte man den Zeitraum und den Ort eingeben und die zu prüfenden Wetterweisheiten auswählen.

Anschließend sollen die Ergebnisse graphisch aufgearbeitet und dargestellt werden. Diese Darstellung sollte mit Python oder wenn möglich mit D3 umgesetzt werden.

### **Definitionen, Hintergrundinformationen, Begründungen:**

Der erste Schritt war die Datenbeschaffung der Wetterdaten über die Datenbank WESTE-XL des DWD (Deutscher Wetterdienst). Bei dieser Datenbank können Wetterdaten in verschiedenen Formaten beschafft werden. Da für das Projekt die Wetterdaten für den Bereich Düsseldorf in einem repräsentativen Zeitabschnitt (1990-2015) mit der Programmiersprache Python analysiert werden sollten, wurden die Daten im CSV-Format beschafft. Dabei wurden für jeden Wert (Luftfeuchtigkeit, maximale Lufttemperatur, minimale Lufttemperatur, mittlere Lufttemperatur, Neuschnee, Niederschlag, Schneehöhe, Sonnenscheindauer und Windspitzen) einzelne CSV-Dateien mit den entsprechenden Tageswerten erstellt.

Im zweiten Schritt wurden diese Daten mit Python eingelesen und für die weitere Analyse aufbereitet.

Als nächstes wurden Wetterweisheiten und Bauernregeln von [wetter.de](http://wetter.de) und [tz.de](http://tz.de) bezogen und in Python-Funktionen übersetzt. Dabei war zu beachten, dass die Weisheiten/Regeln mit den vorhandenen Wetterdaten kompatibel sind.

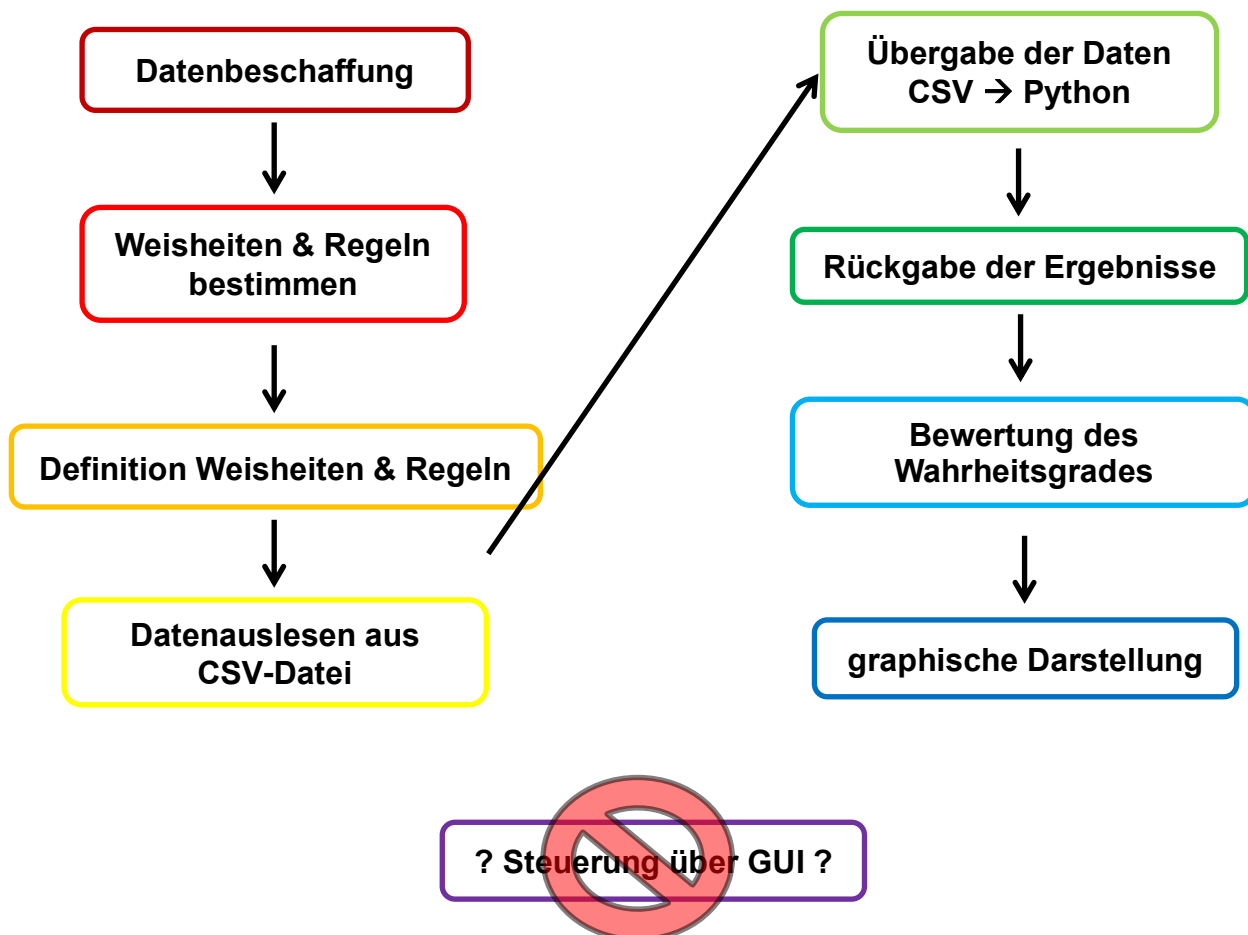
Darauf folgte die Übergabe der benötigten Wetterdaten an die jeweilige Funktion, sowie die entsprechende Analyse wie viele der Werte, den definierten Parametern entsprechen.

Anschließend wurde für jede Funktion ein additives Diagramm erstellt, welches die Werte der einzelnen Teilbereiche der entsprechenden Weisheit/Regel für jedes Jahr im Intervall von 1990-2015 darstellt.

Im letzten Schritt wurden die Weisheiten/Regeln anhand ihres Wahrheitsgrades geprüft, dabei wurde analysiert in wie vielen Jahren im Beobachtungszeitraum die entsprechende Weisheit/Regel zutraf oder nicht. Schließlich wurde mit den so ermittelten Werten für jede Weisheit/Regel ein Tortendiagramm erstellt, das den Wahrheitsgrad graphisch darstellt.

Auf die Umsetzung einer optionalen GUI wurde verzichtet, da auf Grund von Umfang und nicht benötigter Funktion diese nicht erforderlich war.

Des Weiteren wurde zur graphischen Darstellung Matplotlib verwendet, da die Umsetzung mit D3 zwar visuell interessanter, aber diese aus zeitlichen Gründen nicht realisierbar war. Jedoch wäre beides eine sinnvolle Weiterentwicklung, besonders um größere Datensätze z.B. mit verschiedenen Städten, benutzerfreundlicher zu machen.

**Projektplan:****Programmiersprache:** Python**Tools:** CSV, Datetime, Matplotlib und NumPy**Ablaufplanung:**

**Umsetzung:** Umsetzungs idee / Probleme

**Vorbereitung:**

**1. Datenbeschaffung:**

- Wetterdaten in einem Austauschformat (z.B. „.csv“) beschaffen
- ✓ DWD Datenbank (WESTE)
- ❖ Datenbeschaffung kompliziert evtl. andere Datenquelle
- ❖ pro CSV-Datei max. 10.000 Einträge möglich

**2. Weisheiten und Regeln bestimmen:**

- Weisheiten bzw. Bauernregeln beschaffen und benötigte Werte prüfen
- ✓ Nachschlagen z.B. Wetter.de
- ❖ Umfang prüfen, Mittelmaß zwischen zu wenig und zu viel
- ❖ Welche Weisheiten/Regeln lassen sich mit den gegebenen Werten prüfen?

**Kernstruktur:**

**3. Definition der Weisheiten/Regeln:**

- Definieren der benötigten Werte jeder Weisheit/Regel
- Bestimmen der Referenzwerte, wann trifft Regel zu, wann nicht
- ✓ Übersetzung der Weisheiten/Regeln zu Python-Code

**4. Datenauslesen aus CSV-Datei:**

- Funktion zur Identifizierung der gesuchten Werte definieren
- ✓ Suchfunktion in Python für CSV-Datei (Zeile/Spalte) oder (Suchwort/Zeile/Spalte)
- ❖ Ordnung der Daten nicht kontinuierlich, evtl. sortieren
- ❖ Sonderzeichen

**5. Übergabe der Daten aus CSV-Datei zu den Python-Funktionen**

- Funktion zur Übergabe der benötigten Werte
- ✓ Übergabe direkt aus der CSV-Datei oder aus Zwischenformat
- ❖ Format und Vollständigkeit

**6. Rückgabe der Ergebnisse nach der Analyse der Daten**

- Rückgabe der Ergebnisse als Vorbereitung für graphische Darstellung
- ✓ Ergebnis mit Referenzwert und Namen der Weisheit/Regel wird zurückgegeben

- ❖ **Format der Ergebnisse muss graphisch darstellbar sein**

## 7. Bewertung des Wahrheitsgrades der Wetterweisheiten/Bauernregeln

- Funktion zur Bewertung, ob die Weisheiten/Regeln wahr oder falsch sind bzw. Grad der Abweichung vom Referenzwert
- ✓ **Ergebnisse mit Referenzwert vergleichen und bewerten**
- ❖ **Format der Ergebnisse muss graphisch darstellbar sein**

## 8. graphische Darstellung der Ergebnisse

- ~~Ergebnisse mit D3~~ oder notfalls mit Matplotlib darstellen
- ✓ **Ergebnisse und Abweichungsgrad übersichtlich darstellen**
- ❖ **geeigneter Maßstab und Darstellung der Abweichung, bei großen Unterschieden und kleinen Unterschieden**

### Erweiterungen:

## 9. Erstellung und Steuerung GUI

- Bedarf einer GUI prüfen
- Auswahl verschiedener Jahre und/oder Orte
- ✓ **GUI erstellen und Steuerung der Funktionen festlegen**
- ❖ **Aufwand/Nutzen der GUI abwägen**

### Beispiel:

Siebenschläfer = „*Wenn's am Siebenschläfer gießt, sieben Wochen Regen fließt.*“

Siebenschläfer ist der 27.Juni, wenn es an diesem Tag regnet, soll es gemäß dieser Weisheit sieben Wochen (bis zum 16. August) regnen. Um diese Weisheit zu prüfen, muss der Wert der Niederschlagshöhe in der CSV > 0 sein. Anschließend müssen die Niederschlagshöhen vom 28. Juni bis zum 16. August erfasst werden. Diese werden in einer Liste gespeichert und anschließend werden die Nullwerte gelöscht, sowie die Anzahl der Einträge gezählt. Um den Grad der Abweichung bestimmen zu können, wird der errechnete Wert vom Referenzwert (7 Tage x 7 Wochen = 49) abgezogen. Somit ergibt sich bei nur 19 Regentagen eine Abweichung von 30 Tagen → 61.2%.


## Vorbereitung:

1. **Datenbeschaffung:** Der Deutsche Wetterdienst (DWD) betreibt verschiedene Datenbanken, dazu gehört WESTE XL. Mit dieser Datenbank ist es möglich für einen bestimmten Zeitraum unterschiedliche Werte (Lufttemperatur Tagesmittel, Lufttemperatur Tagesminimum, Lufttemperatur Tagesmaximum, Niederschlagshöhe, Schneehöhe, Neuschneehöhe, Sonnenscheindauer, Windspitze und relative Luftfeuchte), entweder stündlich, oder täglich nachzuschlagen. Da jede „Kategorie“ und jede Wetterstation einzeln ausgewählt werden muss, kann die Beschaffung viel Zeit in Anspruch nehmen. Die gewählten Daten können in einem Austauschformat (z.B. als CSV-Datei) gespeichert werden. Dabei ist zu beachten, dass pro Datei maximal 10.000 Einträge möglich sind, weshalb evtl. mehrere CSV-Dateien nötig sein werden. Anschließend sollten die Daten „bereinigt“ werden, wodurch nicht verwendete Spalten gelöscht werden können.

Element	Messstation	Datum	Wert	Einheit	Geo-Breite	Geo-Länge
Lufttempera	DÄ%sseldorf	01.09.2001	11,5	Grad C	51,289	6,781
Lufttempera	DÄ%sseldorf	02.09.2001	14,6	Grad C	51,289	6,781
Niederschlag	DÄ%sseldorf	01.09.2001	0	mm	51,289	6,781
Niederschlag	DÄ%sseldorf	02.09.2001	0	mm	51,289	6,781
Schneehöhe	DÄ%sseldorf	01.09.2001	0	cm	51,289	6,781
Schneehöhe	DÄ%sseldorf	02.09.2001	0	cm	51,289	6,781
Neuschneeh	DÄ%sseldorf	01.09.2001	0	cm	51,289	6,781
Neuschneeh	DÄ%sseldorf	02.09.2001	0	cm	51,289	6,781
Sonnenschei	DÄ%sseldorf	01.09.2001	5,5	hour	51,289	6,781
Sonnenschei	DÄ%sseldorf	02.09.2001	3,3	hour	51,289	6,781
Windspitze	DÄ%sseldorf	01.09.2001	3,9	m/s	51,289	6,781
Windspitze	DÄ%sseldorf	02.09.2001	5,7	m/s	51,289	6,781
Lufttempera	DÄ%sseldorf	01.09.2001	18,1	Grad C	51,289	6,781
Lufttempera	DÄ%sseldorf	02.09.2001	18,2	Grad C	51,289	6,781
Lufttempera	DÄ%sseldorf	01.09.2001	4,6	Grad C	51,289	6,781
Lufttempera	DÄ%sseldorf	02.09.2001	8,4	Grad C	51,289	6,781
Relative Luft	DÄ%sseldorf	01.09.2001	85	%	51,289	6,781
Relative Luft	DÄ%sseldorf	02.09.2001	86	%	51,289	6,781

Abb.1 Beispiel für CSV-Datei von WESTE XL

2. **Weisheiten und Regeln bestimmen:** Wetter.de bietet eine Aufzählung von Wetterweisheiten bzw. Bauernregeln an. Dabei gibt es tages- bzw. monatspezifische Besonderheiten. Da es eine sehr große Menge an Regeln gibt, sollte ein Mittelmaß ausgewählt werden. Dabei ist auf die benötigten meteorologischen Daten zu achten.



**wetter.de**

Suche nach PLZ, Stadt oder Land

Deutschland Europa Welt Gesundheit Reise Videos Klima Freizeit Premium Anmelden

HOME > BAUERNREGELN > MÄRZ

## Bauernregeln März

Januar Februar März April Mai Juni Juli August September Oktober November Dezember

Regnet's stark zu Albinus, macht's dem Bauern viel Verdruß.  
Erklärung: Der 1. März ist der Tag des heiligen Albinus von Angers (um 496-554). Er wurde zunächst Abt im Kloster Tincillacense in Westfrankreich und im Jahre 529 Bischof von Angers.

Wenn es an Sankt Albin regnet, gibt es weder Heu noch Stroh.  
Erklärung: Der 1. März ist der Tag des heiligen Albinus von Angers (um 496-554). Er wurde zunächst Abt im Kloster Tincillacense in Westfrankreich und im Jahre 529 Bischof von Angers.

So wie der erste März, so der Frühling. So wie der 2. März, so der Sommer. So wie der 3. März, so der Herbst.

Märzenstaub und Märzenwind, guten Sommers Vorboten sind.

Ist Kunigunde tränenschwer, bleibt oft die Scheune leer.

### Eine kühle und trockene Nacht

Der Wolken- und Regenfilm für Deutschland für die nächsten 48 Stunden.



Anzeige

Abb.2 Beispiel für Bauernregeln (März) von Wetter.de

## Kernstruktur:

3. **Definition der Weisheiten und Regeln:** Die ausgewählten Regeln müssen in Python-Code „übersetzt“ werden. Hierbei ist zu beachten, welche Werte die Regeln wirklich brauchen und was als Ergebnis erwartet wird. Dabei muss ein Referenzwert für jede Regel festgelegt werden, damit später der Grad der Abweichung bestimmt werden kann.
4. **Datenauslesen aus CSV-Datei:** Allgemeine Suchfunktion für CSV-Datei definieren. Hierbei ist auf die Struktur der Datei zu achten. WESTE XL trägt die verschiedenen ausgewählten Werte zeilenweise ein. Aus diesem Grund muss die Suchfunktion erst die Zeile und dann die betreffende Spalte erkennen können. Es wäre auch möglich eine Suchfunktion über das NLTK zu definieren, hierbei müsste man die gesuchte Spalte mit einem Suchwort bzw. regulären Ausdruck definieren. Hierbei muss besonders auf Sonderzeichen geachtet werden.
5. **Übergabe der Daten aus CSV-Datei zu den Python-Funktionen:** Beim Aufruf der Funktionen müssen die benötigten Daten aus der CSV-Datei übergeben werden. Dies erfolgt entweder durch die in 4. definierte Suchfunktion oder direkt aus der CSV-Datei. Hier sollte auf Vollständigkeit und Format der Daten geachtet werden.

- 6. Rückgabe der Ergebnisse nach Analyse der Daten:** Nachdem das Ergebnis der Analyse ermittelt wurde, wird dies zusammen mit dem Referenzwert und dem Namen der Weisheit/Regel zurückgegeben. Diese Werte werden später für die graphische Darstellung benötigt. Aus diesem Grund sollte schon hier auf das Format der Ergebnisse geachtet werden.
- 7. Bewertung des Wahrheitsgrades der Wetterweisheit/Bauernregel:** Um bestimmen zu können, ob die Weisheit/Regel zutrifft, müssen diese Werte verglichen werden. Falls es zu einer Abweichung kommt, wird der Grad der Abweichung bestimmt. Wie im 6. Schritt muss auch hier das Format für die graphische Darstellung eingehalten werden.
- 8. graphische Darstellung der Ergebnisse:** Die ermittelten Ergebnisse sollen mit D3 graphisch dargestellt werden. Eine mögliche Darstellung wäre ein gedoppeltes Balkendiagramm als Übersicht aller Weisheiten/Regeln, sowie Vektordiagramme für einzelne Regeln. Ob beide Versionen realisierbar sind, hängt vom Gesamtaufwand ab. Falls die Darstellung mit D3 misslingt, könnte diese auch durch Matplotlib erstellt werden.

#### **Erweiterungen (wurden nicht umgesetzt):**

- 9. Erstellung und Steuerung einer GUI:** Je nach Aufwand der Kernstruktur (Schritt 1.-8.) wäre es möglich das Analysetool über eine GUI zu steuern, anstelle einer Python-Shell innerhalb der CMD. Dies wäre sinnvoll, wenn mehrere Jahre oder unterschiedliche Orte auswählbar sein sollen. Hierbei ist zu beachten, dass durch diese Erweiterungen auch der Datenpool vergrößert werden muss, wodurch der Aufwand für die Kernstruktur ebenfalls wächst.



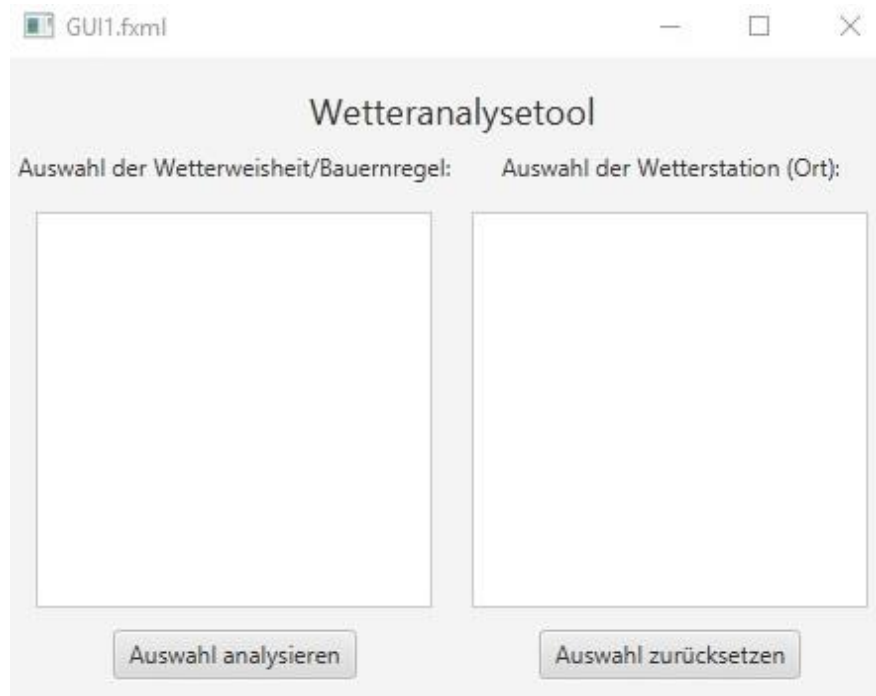


Abb.3 Beispiel für GUI zur Tool Steuerung

## Projektaufbau:

Im Folgenden wird der Projektaufbau anhand einer Beispiel Wetterregel (1. Januar Regel: „Ist der Januar hell und weiß, wird der Sommer sicher heiß.“) dargestellt.

1. **CSV-Werte einlesen:** Als erstes werden die benötigten Daten der jeweiligen CSV-Datei aus den entsprechenden Spalten eingelesen und in einem Dictionary gespeichert. Dabei dient das Datum als Key und die entsprechenden Werte werden als Value angehängt. Dieses Werte-Dictionary „wertedict“ wird zurückgegeben.

```
# Hier werden die Werte aus der .CSV gelesen und für die weitere Verarbeitung gespeichert.
def csv_werte (datei, spalte1, spalte2):
    liste0=[]

    wertedict={}
    with open (datei, 'r') as data_file:
        reader = csv.reader(data_file, delimiter=',')
        for row in reader:
            liste0.append(row)
        for element in liste0:
            wertedict.update({element[spalte1]:element[spalte2]})

#
# Im wertedict sind die Datumanangaben als Schlüssel und die spezifischen Daten als Value enthalten und werden zurückgegeben.
return wertedict
```

Abb.4 CSV-Werte einlesen

**2. Datumsintervall generieren:** Da jede Wetterweisheit/Bauernregel spezifische Datumsintervalle untersucht, müssen diese für jede Weisheit/Regel einzeln generiert werden. Laut Pieters (2012) lässt sich eine Liste mit Datumsintervallen am Besten auf folgende Art erzeugen:

```
from datetime import date, datetime, timedelta

def perdelta(start, end, delta):
    curr = start
    while curr < end:
        yield curr
        curr += delta

>>> for result in perdelta(date(2011, 10, 10), date(2011, 12, 12), timedelta(days=4)):
...     print result
...
2011-10-10
2011-10-14
2011-10-18
2011-10-22
2011-10-26
2011-10-30
2011-11-03
2011-11-07
2011-11-11
2011-11-15
2011-11-19
2011-11-23
2011-11-27
2011-12-01
2011-12-05
2011-12-09
```

Abb.5 Liste mit Datumsintervallen erzeugen

Mit Hilfe dieser Intervalle können nur die benötigten Wetterdaten aus dem „*wortedict*“ gewonnen werden, diese werden anschließend in einer Liste „*liste2*“ gespeichert.

```
#Jan: 1. Regel: Ist der Januar hell und weiß, wird der Sommer sicher heiß.

#1.Teil der Jan: 1. Regel
def jan1_hell(csv_werte):

#Zuerst wird das benötigte Datumintervall erzeugt

    def intervall(start, ende, delta):
        while start < ende:
            yield start
            start += delta
    datumliste=[]

    liste2=[]
    ergebnis=[]
    year=1990

    ylist=[]
    vlist=[]

#In der folgenden while Schleife werden die gleichen Intervalle in unterschiedlichen Jahren max. 1990-2015 erzeugt.
    while year<2016:
        datumliste=[]
        liste2=[]
        for result in intervall(date(year, 1, 1), date(year, 2, 1), timedelta(days=1)):
            datumliste.append(str(result))
        for element in datumliste:

#Es gibt manche Werte die nicht gemessen wurden z.B. 20.01.1993 Sonnenscheindauer
            if element in csv_werte.iterkeys():
                liste2.append(csv_werte[element])
```

Abb.6 Datumsintervall

- 3. Umwandlung der Strings, Analyse und Ergebnissrückgabe:** Im nächsten Schritt werden die in „liste2“ gespeicherten Strings für die Umwandlung in Float-Zahlen vorbereitet. Dabei werden die Kommas durch Punkte ersetzt. Im Anschluss werden die Werte analysiert, ob diese den gewünschten Parametern entsprechen (hier: Werte > 0.0), dabei wird gezählt, wie viele Werte zutreffen. Am Ende wird die Jahreszahl, die Anzahl an zutreffenden Werten und deren maximale Anzahl in einer Liste „ergebnis“ gespeichert. Dieser Vorgang wird für jeden Tag in dem definierten Intervall und für jedes Jahr wiederholt.

```
#Hier wird das Komma in einen Punkt umgewandelt, damit später die Strings durch Floats ersetzt werden.
counter=0
position=0
for element in liste2:
    element_neu=element.replace(",", ".")
    liste2[position]=element_neu
    position=position+1

#Hier wird gezählt, wie viele Werte den Anforderungen entsprechen.
position=0
while position < len(liste2):
    if float(liste2[position]) > 0.0:
        counter=counter + 1
        position=position+1
    else:
        position=position+1

#Nun werden in der Liste "ergebnis" das Jahr, die Anzahl an entsprechenden Werten und die max. Anzahl der möglichen Werte gespeichert.
ergebnis.append(year)
ergebnis.append(counter)
ergebnis.append(len(liste2))

year=year+1
return ergebnis
```

Abb.7 Umwandlung, Analyse & Ergebnis

**Wiederholung:** Schritt 1-3 wird für jeden Teil einer Wetterweisheit/Bauernregel einzeln durchgeführt, da unterschiedliche Werte oder auch Intervalle benötigt werden.

- 4. Vorbereitung zur Generierung des additiven Diagrammes:** Die Ergebnisse der einzelnen Teilstücke der Wetterweisheit/Bauernregel werden nun zur Darstellung vorbereitet. Dabei werden zuerst die Werte der Teilstücke übergeben, in einer Liste pro Teilstück zwischengespeichert und der Referenz Maximalwert bestimmt.

```

#Diagramm mit drei Werten:
def addidia3 (werte1, werte2, werte3, namen, titel):
    ywerte1=[]
    ywerte2=[]
    ywerte3=[]
    xwerte=[]
    x=0
    y=1
    z=2

#maxwert als Skala der y-Achsen Werte (werte1[z], werte2[z] und werte3[z] sind die maximalen Werte von werte1, werte2 und werte3)
    maxwert=werte1[z]+werte2[z]+werte3[z]

#in dieser Schleife werden die Werte der einzelnen Teile eine Weisheit/Regel für jedes Jahr zusammengeführt.
    while x < len(werte1):
        xwerte.append(werte1[x])
        ywerte1.append(werte1[y])
        ywerte2.append(werte2[y])
        ywerte3.append(werte3[y])
        x=x+3
        y=y+3
        z=z+3

    print ywerte1,ywerte2,ywerte3

```

Abb.8 Vorbereitung für additives Diagramm

- 5. Generierung des additiven Diagrammes:** In diesem Schritt wird das additive Diagramm erstellt. Dabei wird für jedes Teilstück der entsprechenden Wetterweisheit/Bauernregel eine Teilsäule aus den spezifischen Werten für jedes Jahr generiert. Anschließend werden die Teilsäulen für jedes Jahr aufeinander summiert und bilden so die Werte der einzelnen Teilstück für die Weisheit/Regel ab.

```

#nun wird das additive Diagramm erstellt, indem die Teilwerte für jedes Jahr als Säulen aufeinander gestellt werden
    ind = np.arange(len(xwerte))
    width=0.35

    p1=plt.bar(ind, ywerte1, width, color='r')
    p2=plt.bar(ind, ywerte2, width, color='b', bottom=ywerte1)
    p3=plt.bar(ind, ywerte3, width, color='y', bottom=[i+j for i, j in zip(ywerte1,ywerte2)])

    plt.legend((p1[0],p2[0],p3[0]), namen)
    plt.ylabel('Zutreffende Werte (Anzahl Tage)')
    plt.title(titel)
    plt.xticks(ind+width/2., xwerte, rotation=45)
    plt.yticks(np.arange(0,maxwert+1,10))
    plt.axhline(y=maxwert/2)

    plt.show()

```

Abb.9 Funktion zur Generierung des additiven Diagrammes

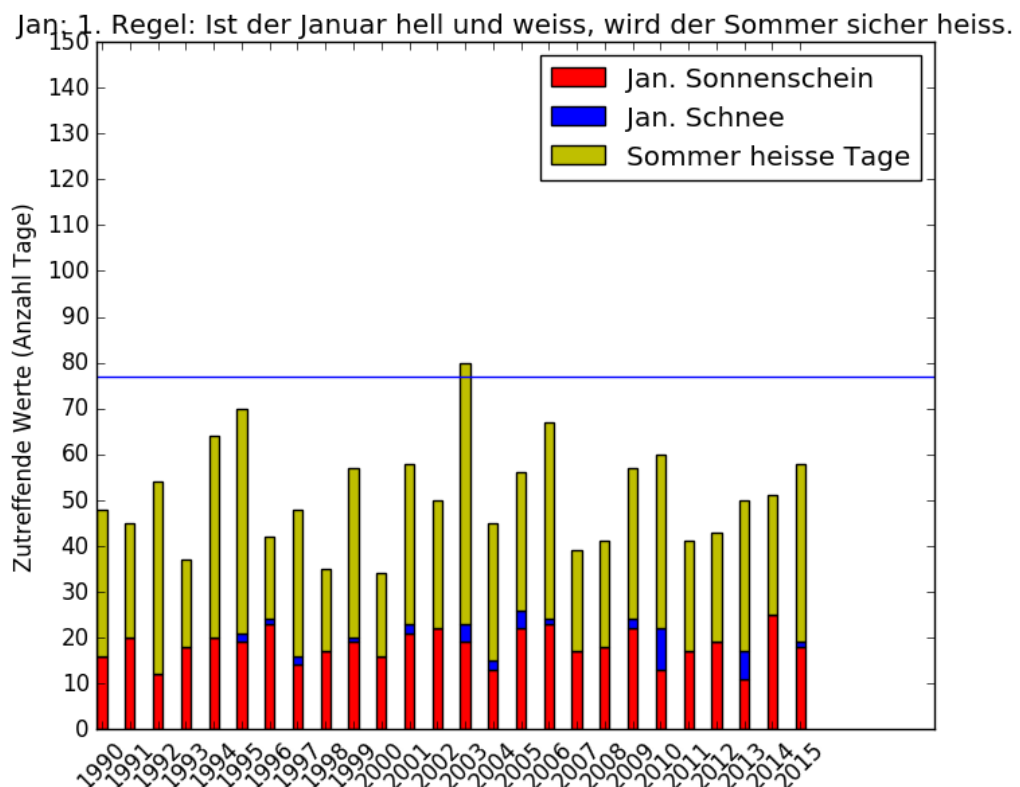


Abb.10 Beispiel für additives Diagramm

- 6. Bewertung der Wetterweisheit/Bauernregel:** Um die Aussagekraft der Wetterweisheit/Bauernregel bestimmen zu können, müssen die einzelnen Teilbereiche der Weisheit/Regel in jedem Jahr analysiert werden. Dabei wird geprüft, ob diese Werte über dem Referenzwert liegen. Dabei wurden die Referenzwerte teilweise so angepasst, dass sie für die Region Düsseldorf sinnvoll sind, da es sonst zu keiner vergleichbaren Aussage kommen kann. Besonders bei Weisheiten/Regeln, die mit Frost oder Schnee im Frühjahr zu tun haben, wurden die Werte angepasst. Dies gilt nicht nur für die Referenzwerte, sondern auch für die booleschen Operatoren. Ob die Weisheit/Regel in dem entsprechenden Jahr wahr oder falsch ist, wird durch den Vergleich der Teilwerte mit den Referenzwerten und in Verbindung mit den booleschen Operatoren bewertet und gespeichert.

```
def bewertung_jan1 (wertel,werte2,werte3,titel):
    ywertel=[]
    ywerte2=[]
    ywerte3=[]
    xwerte=[]
    x=0
    y=1
    z=2

    #60% als Referenzwert festgelegt um mehr als Durchschnitt 50% zu definieren.
    refwert1=float(wertel[z])/100*60
    refwert2=float(werte2[z])/100*60
    refwert3=float(werte3[z])/100*60

    #in dieser Schleife werden die Werte der einzelnen Segmente der Weisheit/Regel in verschiedenen Listen gespeichert.
    while x < len(wertel):
        xwerte.append(wertel[x])
        ywertel.append(float(wertel[y]))
        ywerte2.append(float(werte2[y]))
        ywerte3.append(float(werte3[y]))
        x=x+3
        y=y+3
        z=z+3

    x=0
    count_wahr=0
    count_falsch=0

    #in dieser Schleife wird geprüft, ob die Weisheit/Regel in den einzelnen Jahren zu treffen, oder nicht.
    #Dabei muss der jeweilige Wert über dem Referenzwert liegen.
    while x < len(xwerte):
        if (ywertel[x] >= refwert1 or ywerte2[x] >= refwert2) and ywerte3[x] >= refwert3:
            print xwerte[x],refwert1,ywertel[x],refwert2,ywerte2[x],refwert3,ywerte3[x]
            print xwerte[x], "trifft zu"
            count_wahr=count_wahr+1
            x=x+1
        else:
            print xwerte[x],refwert1,ywertel[x],refwert2,ywerte2[x],refwert3,ywerte3[x]
            print xwerte[x], "trifft nicht zu"
            count_falsch=count_falsch+1
            x=x+1
```

Abb.11 Bewertung der Wetterweisheit/Bauernregel

**7. Generierung der Bewertung als Tortendiagramm:** Um den in 6. ermittelten Wert der Aussagekraft graphisch darzustellen, werden die Werte für die Wetterweisheit/Bauernregel, in wie vielen Jahren diese richtig oder falsch waren als Tortendiagramm dargestellt. Dabei werden die Werte in Prozentangaben umgewandelt und innerhalb der entsprechenden Farbe (richtig=blau & falsch=rot) angezeigt.

```
#Tortendiagramm zur visuellen Darstellung wie zutreffend die Weisheiten bzw. egeln sind.
def tortendia (wertel,werte2,titel):
    labels = 'trifft zu','trifft nicht zu'
    sizes = [wertel,werte2]
    colors = ['b','r']
    plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=180)
    plt.title(titel)
    plt.axis('equal')
    plt.show()
```

Abb.12 Funktion zur Generierung des Tortendiagramms

#Jan: 1. Regel: Ist der Januar hell und weiss, wird der Sommer sicher heiss.

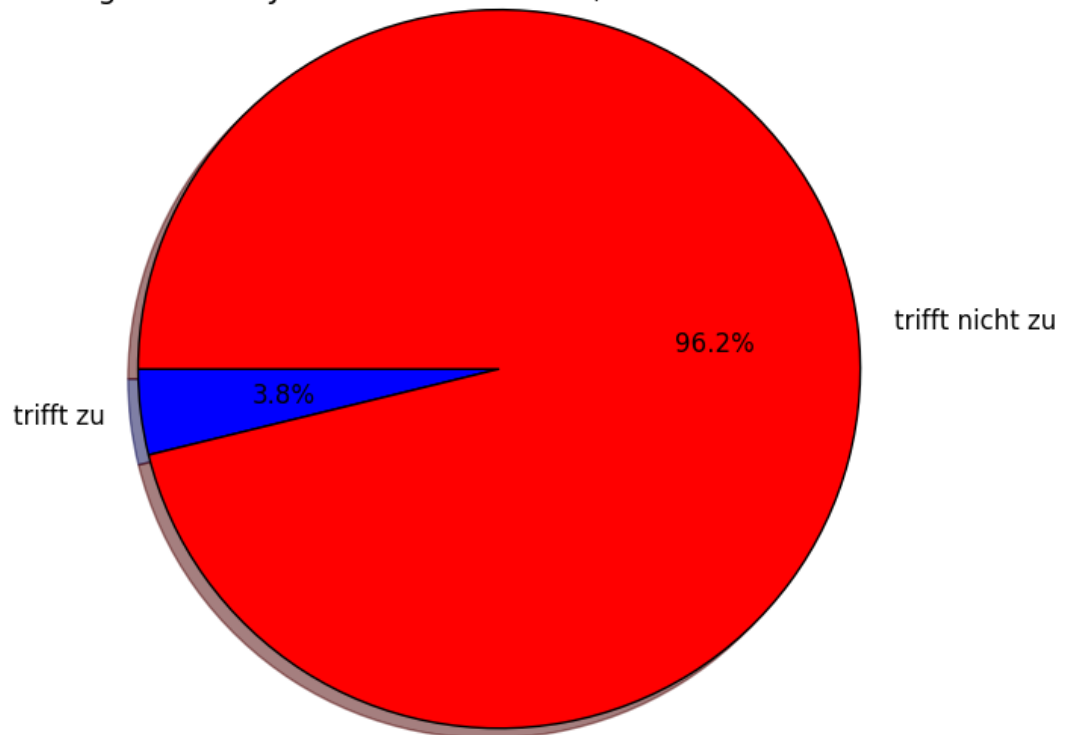


Abb.13 Beispiel für Tortendiagramm

### Anleitung zur Ausführung/Benutzung:

Das Programm ist mit Python 2.7.11 – Anaconda 4.0.0 (64-bit) erstellt worden. Zur problemlosen Ausführung muss mindestens diese Version installiert sein. Um das Programm starten zu können, sollte die `wetter.py` Datei mit einem Editor wie z.B. Geany geöffnet und ausgeführt werden. Anschließend werden die Diagramme für jede der Wetterweisheiten nacheinander dargestellt.

### Reflexion:

Insgesamt funktioniert das Programm so wie es soll. Die Wetterdaten aus den CSV-Dateien werden eingelesen, aufbereitet, analysiert und in den einzelnen Wetterweisheiten/Bauernregeln verarbeitet. Jedoch gibt es ein Problem mit genau diesen Weisheiten/Regeln.

Die Aussagekraft dieser Weisheiten/Regeln ist für den Bereich Düsseldorf relativ ungenau. Der Wahrheitswert der einzelnen Weisheiten/Regeln ist sehr gering. Dies könnte auf zwei unterschiedliche Gründe zurückzuführen sein. Einerseits ist es möglich, dass der Raum Düsseldorf nicht repräsentativ zur Analyse der Daten ist.

Andererseits könnten die verwendeten Weisheiten/Regeln generell zu ungenau sein oder aber zumindest für den Raum Düsseldorf nicht zutreffen. Es wäre sinnvoll als nächstes eine weitere Stadt, vorzugsweise in einer anderen Region z.B. München, für die Analyse der Weisheiten/Regeln zu verwenden. Dadurch könnten die Werte von Düsseldorf und München verglichen werden. Des Weiteren sollten noch mehr Wetterweisheiten/Regeln eingearbeitet und analysiert werden. Durch diese Schritte würden aussagekräftigere Ergebnisse entstehen.

### **Literaturverzeichnis:**

Pieters, M. (2012). *Generate a list of datetimes between an interval [42]*. Abgerufen 23. August 2016, von <http://stackoverflow.com/questions/10688006/generate-a-list-of-datetimes-between-an-interval>

### **Datenverzeichnis:**

Wetterdaten: Deutscher Wetterdienst (DWD)

Wetterweisheiten:

tz.de (2014). *Wetter-Weisheiten: Welche Sprüche wirklich stimmen*. Abgerufen 07. August 2016, von <http://www.tz.de/welt/wetter-weisheiten-welche-sprueche-wirklich-stimmen-fotostrecke-zr-1130003.html>

wetter.de. *Bauernregeln*. Abgerufen 07. August 2016, von <http://www.wetter.de/bauernregeln.html>