# Python Komplett

## 5-Tages Seminar

### Tag 5

Timm Gieger

GFU Cyrus AG

# Agenda – Tag 5

GUI Programmierung mit tkinter

---

Testing (docstring- , unit- und pytest)

---

Git Einführung

---

Ausblick Data Science

---

Feedback & Fragerunde

---

# GUI Programmierung mit Python

# Testing

## Modultests

- Allgemein als Unittests bekannt
- Einzelne Teile eines Programms werden getestet
- Bessere Testmöglichkeiten als bei kompletten Softwaretests
- In Python durch docstring- und unittest-Modul möglich

→ Meist  effizienter als gesamte Software zu testen

## Testing

### Modultests

docstring vs. unittest:

- unittest kann in eigenem Skript realisiert werden
- docstring wird in Funktion selbst realisiert
- unittest deutlich flexibler und besser anzupassen

## Testing

### Modultests

Pytest

- Testframework für Python-Entwicklung
- Auto-discovery of tests Feature
- „rich assertion introspection“

→ einfache Testdefinition mit der Hilfe von „assert“-Abfragen

- Bestehende Unittests können in pytest eingebunden werden




Source: https://docs.pytest.org/en/8.2.x/index.html

# Testing mit Python

# Git-Einführung

# Was ist Git?

Versionierungssoftware (distributed Version Control System)

Jeder Entwickler hat eine vollständige Kopie sowie die gesamte Historie des Repositories (Arbeiten auch ohne Netzwerkverbindung möglich

Branching & Merging

Ermöglicht effizientes Arbeiten in Teams wie z.B.: parallele Bearbeitung von Features oder Bugfixes

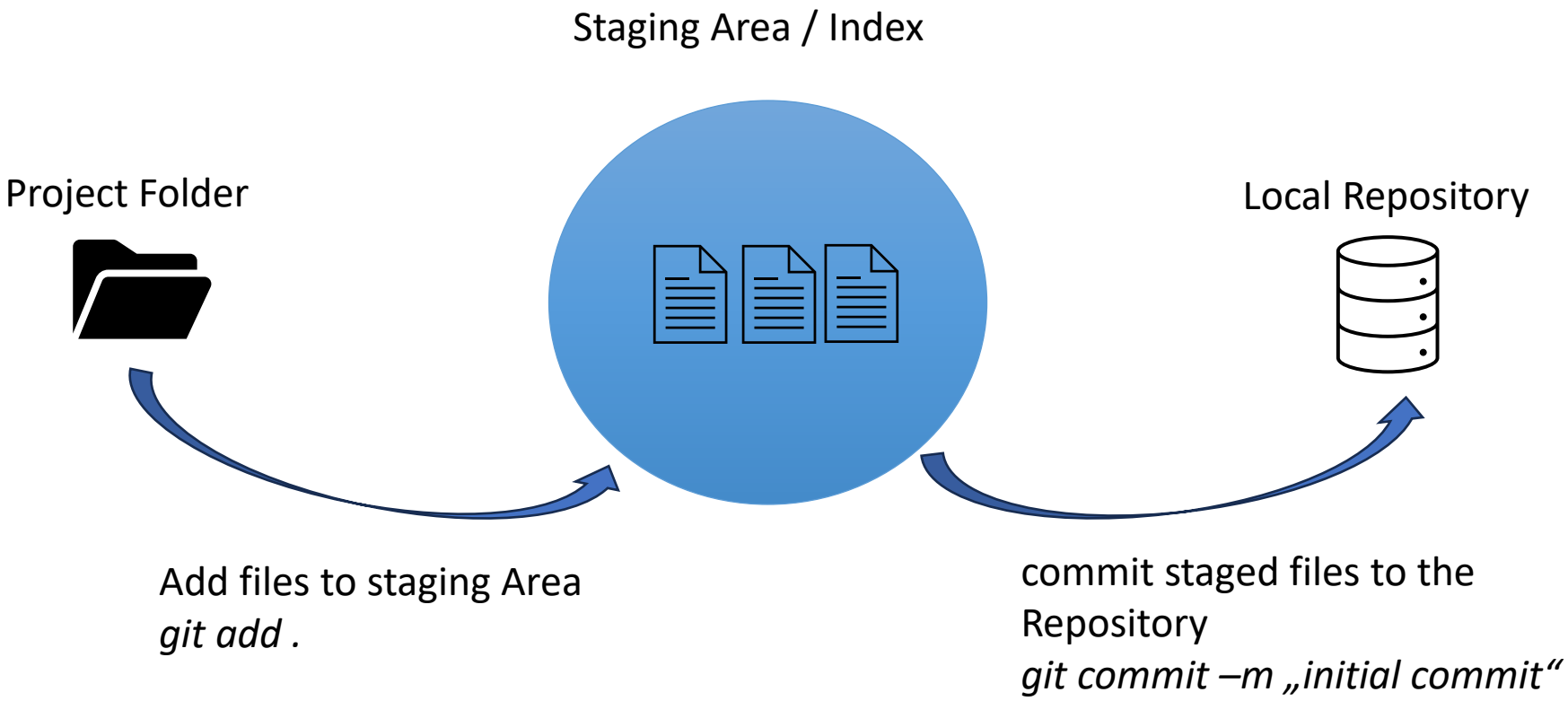
Snapshoting

Im Vergleich zu anderen Versionskontrollsystemen speichert git die Änderungen als komplette Snapshots und nicht nur Deltas. → Effiziente und schnelle Nachverfolgung und Verwaltung von Änderungen möglich

Effizient und schnelle Performance

Durch hautpsächlich lokales Arbeiten werden Operationen wie Branches, Comitten von Änderungen etc. sehr schnell durchgeführt

# Git Workflow



## Was ist Git?

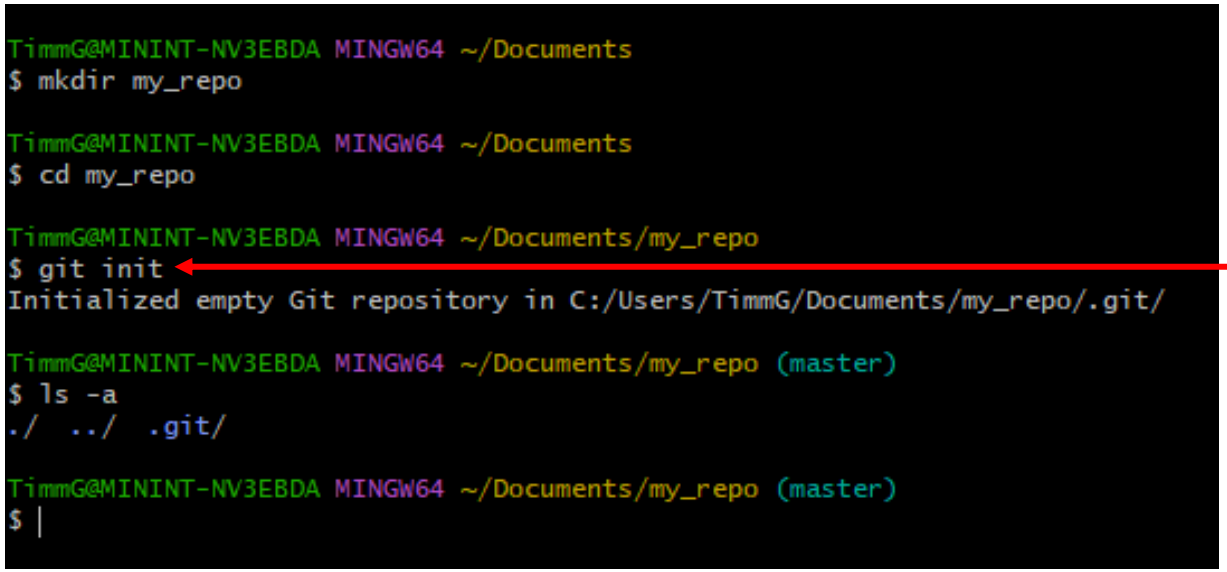
### Commit

- Erstellt einen Snapshot der aktuellen Version
- Speichert gewisse Metadaten: ID, Message, Date/time, author
- Enthält eine komplette Abbildung des Snapshots (möglich durch effiziente Datenkomprimierung)

## Git Step-by-Step Guide

### Repository anlegen

1. Sicherstellen dass git installiert ist. Wenn dies der Fall ist, sollte bei einem Rechtsklick in einen Ordner die Eigenschaft „Git Bash Here“ auswählbar sein.
2. Öffnen einer Git Bash über den „Git Bash Here“ Befehl (am besten unter Documents)
3. Eingabe folgender Befehle:




Hier wird das Repo angelegt

## Git Step-by-Step Guide

### Files hinzufügen

4. Legen Sie 2 Text files mit dem Befehl „echo hello > file1.txt“ und „echo hello > file2.txt“ an
5. Lassen Sie sich den status über „git status“ ausgeben

```
TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$ echo hello > file1.txt

TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$ echo hello > file2.txt

TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt
        file2.txt

nothing added to commit but untracked files present (use "git add" to track)

TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$
```

Unstaged Files

## Git Step-by-Step Guide

### Files hinzufügen

6. Fügen Sie alle files der Staging Area hinzu mit dem „git add . “ -Befehl
7. Lassen Sie sich den status über „git status“ ausgeben
8. Ändern Sie den ersten File mit folgendem Befehl „echo world >> file1.txt“
9. Fügen Sie die geänderte Datei der Staging Area hinzu,indem Sie den  add-Befehl erneut ausführen

```
TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$ git add .
warning: in the working copy of 'file1.txt', LF will be replaced by CRLF the n
t time Git touches it
warning: in the working copy of 'file2.txt', LF will be replaced by CRLF the n
t time Git touches it

TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt
```

Modifizierung erkannt

```
TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$ echo world >> file1.txt

TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt
```

## Git Step-by-Step Guide

### Änderungen commiten

10. Fügen Sie die Änderungen Ihrem lokalen Repository hinzu, indem Sie den Befehl *git commit –m "Message"* verwenden
11. Lassen Sie sich danach wieder den Status anzeigen über „git status". Hier sollte nun der working tree als clean angezeigt werden

```
TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$ git commit -m "initial commit"
[master (root-commit) 3b2ca6f] initial commit
 2 files changed, 3 insertions(+)
 create mode 100644 file1.txt
 create mode 100644 file2.txt

TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$ git status
On branch master
nothing to commit, working tree clean

TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$
```

→ Guidelines für commits: So viel wie nötig, so wenig wie möglich. (nicht zu viel und nicht zu wenig)

## Git Step-by-Step Guide

### gitignore

1. Erstellen Sie einen Ordner namens „Files“ in Ihrem Projektordner (mkdir Files)
2. Fügen Sie dem Ordner eine Datei hinzu.
3. Geben Sie sich den status mit „git status“ aus.
4. Erstellen Sie nun eine Datei in Ihrem Projektordner mit dem Eintrag „Files/“ namens .gitignore mit folgendem Befehl:

„echo Files/ > .gitignore“

5. Fügen Sie den File der Staging Area hinzu mit dem Befehl „git add .gitignore“ und commiten Sie die Änderungen mit *git commit –m “Message“*

# Git Step-by-Step Guide

## gitignore

```
TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$ echo Files/ > .gitignore

TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git add

TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$ git add .gitignore
warning: in the working copy of '.gitignore', LF will be replaced
xt time Git touches it

TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$ git commit -m "added gitignore"
[master 45d4918] added gitignore
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore

TimmG@MININT-NV3EBDA MINGW64 ~/Documents/my_repo (master)
$
```

# Git-Tutorial

## Branching

- Jedes Repository ist in branches eingeteilt

- Unterteilung in lokale und remote-branches (remote → GitHub, Gitlab etc…)

- der primäre Branch ist immer als „master/main“ bezeichnet

Vorteil → jeder Branch kann eigenständig arbeiten

Nachteil → wenn zu lange in eigenem Branch gearbeitet wird ist das delta zwischen master und branch meist recht groß

**Git-Tutorial**

## Merging

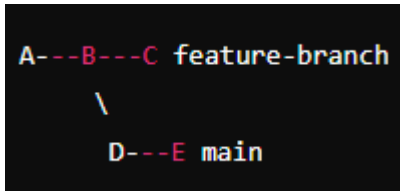
- Auf Branch wechseln der die Changes empfangen soll

- Ausführen des merge commands mit dem Namen des „features“-Branches

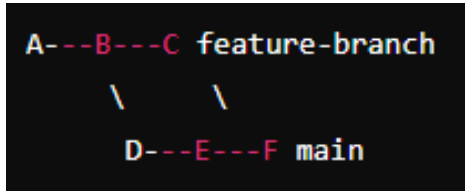
*git merge branchname*

*Beispiel: main soll mit feature-branch gemerged werden*

*vorher*




*nachher*




Änderungen beider Branches werden zusammengeführt

## Git-Tutorial

### Rebasing

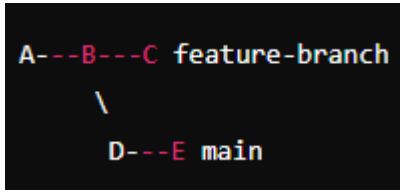
- Auf Branch wechseln der die Changes empfangen soll

- Ausführen des merge commands mit dem Namen des „features“-Branches

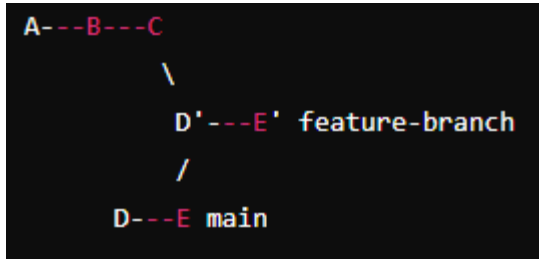
*git rebase branchname*

*Beispiel: feature-branch soll auf main rebased werden*

*vorher*




*nachher*




Commit Historie wird linearisiert
als ob die Arbeit immer auf dem neuesten Stand des Zielbranches stattgefunden hätte

# Git-Befehle

Wichtigste Git-Befehle:

Erstellen eines neuen Repositories

```
git init                #Erstellen eines Repositories
git clone               #Kopieren eines bestehenden Repositories
```

Staging von Dateien

```
git add .               #fügt alle Dateien im aktuellen Ordner der Staging Area hinzu oder löscht diese
git add file.txt        #hinzufügen oder entfernen einer bestimmten Datei
git ls-files            #Anzeigen aller files in der Staging Area
git diff --staged       #Anzeigen der Änderungen in der Staging Area
```

Abrufen des Status eine Repositories

```
git status              #kompletter Status
git status -s           #kurzer Status
```

# Git-Befehle

Wichtigste Git-Befehle:

Branching

```
git branch                  #Anzeigen aller Branches
git branch branchname       #Erstellen eines neuen branches
git checkout branchname     #Wechseln oder Erstellen eines neuen branches
```

oder

```
git switch branchname
```

Commiten von Änderungen

```
git commit –m „Nachricht“   # commiten mit kurzer Nachricht
git log                     #anschauen der commit history
```

Veröffentlichen & Updaten

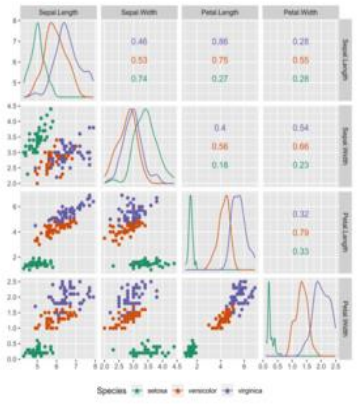
```
git push                    # Publishen auf einen remote-Branch
git pull                    # Pull des letzten Standes des Remote-Repositories
```

# Exkurs Data Science

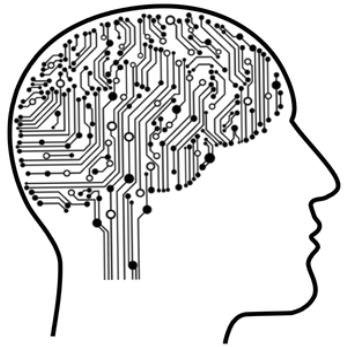
# Einführung pandas / numpy

# Was ist Data Science?


Prozess

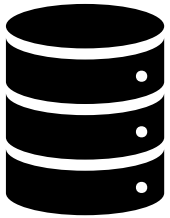


EDA



Machine Learning / AI

Data Science



Daten

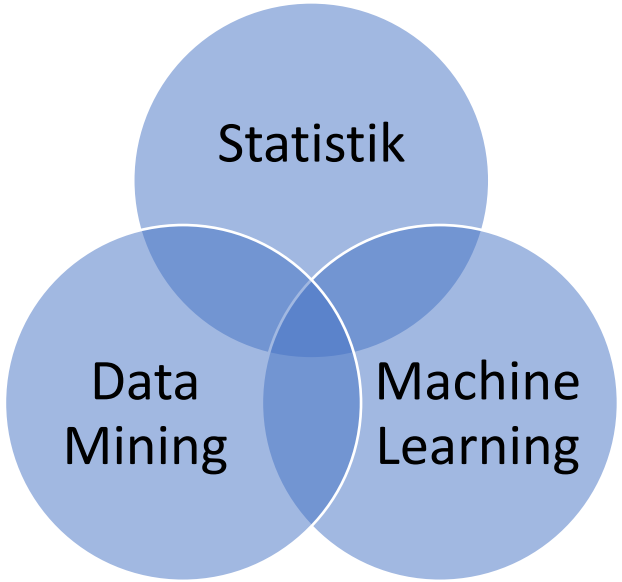
...

Analytics

# Was ist Data Science

## Data Science Definition

*„…Extraktion nützlichen Wissens und aussagekräftiger Informationen aus großen Datenmengen, um geschäftliche Entscheidungsfindungen zu verbessern.“*[1]




Hauptgebiete von Data Science, welche nicht eindeutig voneinander separierbar sind und sich teils überlappen.



[1] Provost, Foster, Tom Fawcett, Data Science für Unternehmen, 2017, S. 41

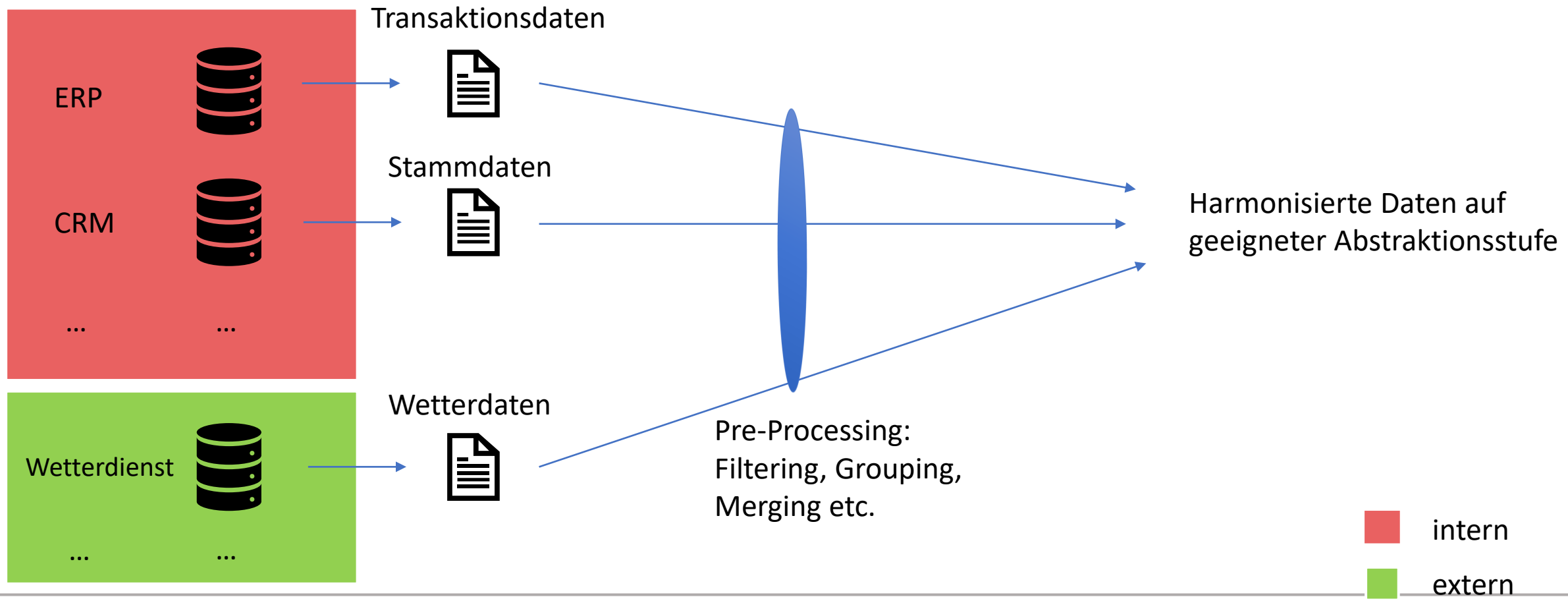
# Ausblick Data Science mit Python

- Grouping
- Merging
- Filtering
- Sorting

# Warum Merging/Grouping/Filtering…?

- Daten werden selten (eher nie) in geeigneter Abstraktionsstufe ausgeliefert
- Aufgabe des Data Scientist ist mit Auswahl geeigneter Methoden passende Datenform für spätere Analyse zu wählen
- Auch für EDA kann eine höhere/niedrigere Abstraktionsstufe größeren Einblick in die vorliegenden Daten geben
- Teil des Pre-Processing neben anderen notwendigen Schritten wie beispielsweise der Umgang mit Fehlwerten.

# Warum Merging/Grouping/Filtering…?



# Beispiel Corona-Daten

| IdBundesland | Bundesland | Landkreis | Altersgruppe | Geschlecht | AnzahlFall | AnzahlTode... | Meldedatum | IdLandkreis | Datenstand | NeuerFall | NeuerTodes... | Refdatum |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Schleswig-H... | SK Flensburg | A00-A04 | M | 1 | 0 | Sep 30, 2020 | 1001 | 06.04.2021, 0... | 0 | -9 | Sep 30, 2020 |
| 1 | Schleswig-H... | SK Flensburg | A00-A04 | M | 1 | 0 | Oct 29, 2020 | 1001 | 06.04.2021, 0... | 0 | -9 | Oct 29, 2020 |
| 1 | Schleswig-H... | SK Flensburg | A00-A04 | M | 1 | 0 | Nov 3, 2020 | 1001 | 06.04.2021, 0... | 0 | -9 | Nov 3, 2020 |
| 1 | Schleswig-H... | SK Flensburg | A00-A04 | M | 1 | 0 | Nov 20, 2020 | 1001 | 06.04.2021, 0... | 0 | -9 | Nov 19, 2020 |
| 1 | Schleswig-H... | SK Flensburg | A00-A04 | M | 1 | 0 | Nov 23, 2020 | 1001 | 06.04.2021, 0... | 0 | -9 | Nov 18, 2020 |
| 1 | Schleswig-H... | SK Flensburg | A00-A04 | M | 1 | 0 | Dec 18, 2020 | 1001 | 06.04.2021, 0... | 0 | -9 | Dec 14, 2020 |
| 1 | Schleswig-H... | SK Flensburg | A00-A04 | M | 2 | 0 | Jan 6, 2021 | 1001 | 06.04.2021, 0... | 0 | -9 | Jan 6, 2021 |
| 1 | Schleswig-H... | SK Flensburg | A00-A04 | M | 1 | 0 | Jan 8, 2021 | 1001 | 06.04.2021, 0... | 0 | -9 | Jan 6, 2021 |

→ Daten auf niedrigster Abstraktionsebene sind oftmals nicht geeignet für prädiktives Modell



Source: Robert Koch-Institut, presse@rki.de, https://www.rki.de

# Beispiel Corona Daten

Problemstellung:

Extraktion von Features für prädiktives Modell zur Vorhersage von Knappheit/Engpässen für Intensivbetten pro Stadt/Landkreis

- Sie wollen die Todesrate (prozentual) und die aktuelle ICU-Bettenauslastung (prozentual) bestimmen.

Wie sieht die Formel zur Berechnung dieser aus?

Formel Todesrate:

Formel ICU-Bettenauslastung:

# Data Science mit Python

# Feedback & Fragerunde

- Haben Sie Fragen zu den behandelten Themen?
- Welche Themen haben Sie vermisst?
- Feedback an mich?
- Bewertungsformular: https://ratings.gfu.cloud/form.html?h=a3b48d3008d71faad54a535e1c4deeaf739f1edda4d5ef97c71606b26bfced5b&type=trainer

# Ihr Ansprechpartner

Timm Gieger

mobile: +49176 40566154

mail: timm.gieger@geekit-ds.de