RM_A02_Group

November 22, 2020

1 Research Methods UHH - Knowledge Technology Research Group - WiSe 2020/2021

1.1 Assignment #2 - Empirical Studies & EDA

1.1.1 Group: C

1.1.2 Names of members: Aida Usmanova, Emilio Brambilla and Navneet Singh Arora

---

1.1.3 Instructions:

Please answer the questions below. Copy this notebook and enter your answers underneath each task description, inserting cells as needed. You may use a combination of python 3, markdown, and LaTex to formulate your responses. In order to successfully complete the assignment, you will need the lecture material provided in the RM moodle course, especially L02 & L03. **Make sure to use only a copy of this notebook for your answers instead of a new/blank notebook.**

1.1.4 Grading Criteria:

In order to successfully pass this assignment, you will need **at least a total of 70 points out of 100 points**, and every task has to be tackled.

1.1.5 Submission:

Please upload the following two files **until Tuesday, 24 November 2020, 19:59 CET (Germany)** together in a .zip archive in moodle: 1. a (single) copy of this jupyter notebook containing your answers for all tasks (file extension: .ipynb) 2. an exported PDF document of the jupyter notebook (file extension: .pdf)

1.1.6 Presentation:

Make sure that each (!) group member takes part in solving this assignment and is prepared to answer questions and/or present solutions from your submitted notebook during our assignment revision meeting scheduled for **Wednesday, 2 December 2020, 12:00 - 13:30 CET (Germany)**.

1.1.7 File Naming:

Add the group letter to the file name prior to submission. For example, if your group letter is "A" (see group selection in moodle), you would use the following filename: 1. RM_A02_Group_A.ipynb 2. RM_A02_Group_A.pdf

---

Task 1 [10 points] Data Scales

1. For each of the 18 feature columns in the RKI COVID-19 database, identify all scales of data whose definition is valid for all column entries. Create a table using python code that contains all column headers as rows, data scales as columns, and binary table entries indicating whether the feature values (i.e. column entries in the database) correspond to the data scale or not.
2. Identify two feature columns and respective data scales for which a precise allocation is either not feasible or questionable and explain your decision.

1.2 —

1.2.1 Pre-Task Activities

- Importing Libraries in-order to successfully work around with various tasks.
  - pandas
  - matplotlib
  - seaborn
  - numpy
  - scipy
- Reading the **CSV** data from the **RKI Data URL** using `read_csv()` method.

  ```
  pd.read_csv("https://*****.csv")
  ```

- Storing the data as a Dataframe in a variable named **germany_covid_data** for the date when this file is executed.

  ```
  germany_covid_data = pd.read_csv("https://*****.csv")
  ```

- Data for **18th November 2020** is stored in a variable named **previous_day_data**
- Data for **19th November 2020** is stored in a variable named **current_day_data**
- For ease of use, the columns present in German Language are **renamed** to the ones in English Language using `rename()` method.
- To view if the data has been configured correctly or not, `head()` method is used to have a **view of the data**.

  ```
  germany_covid_data.head(5)
  ```

- To verify that no data has been clipped or no data has been changed in the configuration process, using the `shape` property, we **verify the total no of rows and columns**. Checking the `len()`, also verifies the no of columns on which we will be working on in the following tasks.

  - germany_covid_data.shape
  - len(germany_covid_data.columns)
- Last but not the least, we rename the columns from **German** to **English** for ease of use.

NOTE: *The data exported using url fetches data for the date whenever the file is executed*

NOTE: *The data exported using csv files for comparison belongs to **18th and 19th November** respectively.*

1.3 —

[1]:
```
# Importing Libraries to carry out various tasks in this assignment

import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from matplotlib.dates import AutoDateFormatter, AutoDateLocator
import seaborn as sns
import numpy as np
from scipy.stats import chi2_contingency
import scipy

# Importing data for the day when it is executed using url
# Importing data for 18th and 19th November respectively to compare the data in␣
↪tasks ahead

germany_covid_data = pd.read_csv("https://prod-hub-indexer.s3.amazonaws.com/
↪files/dd4580c810204019a7b8eb3e0b329dd6/0/full/4326/
↪dd4580c810204019a7b8eb3e0b329dd6_0_full_4326.csv");
previous_day_data = pd.read_csv("/Users/navneet/Downloads/RKI_COVID19.csv");
current_day_data = pd.read_csv("/Users/navneet/Downloads/RKI_COVID19-2.csv");
```

[2]:
```
# Verifying few details of the data before use

print("Data Type of Imported Data: ", type(germany_covid_data));
print("Length of Data Imported: ", len(germany_covid_data.columns));
germany_covid_data.shape
```

```
Data Type of Imported Data:  <class 'pandas.core.frame.DataFrame'>
Length of Data Imported:  18
```

[2]:
```
(526110, 18)
```

[3]:
```
# Renaming columns from German to English for ease of use
```

```
germany_covid_data = germany_covid_data.rename(
    columns = {
                "ObjectId": "Object_Id",
                "IdBundesland": "State_Id",
                "Bundesland": "State",
                "Landkreis": "District",
                "Altersgruppe": "Age_Group",
                "Geschlecht": "Sex",
                "AnzahlFall": "Number_of_Cases",
                "AnzahlTodesfall": "Number_of_Deaths",
                "Meldedatum": "Reporting_Date",
                "IdLandkreis": "District_Id",
                "Datenstand": "Last_Updated",
                "NeuerFall": "New_Case",
                "NeuerTodesfall": "New_Death",
                "Refdatum": "Reference_Date",
                "NeuGenesen": "New_Recovered",
                "AnzahlGenesen": "Number_of_Recovered",
                "IstErkrankungsbeginn": "Disease_Onset",
                "Altersgruppe2": "Age_Group2"
    }
);
previous_day_data = previous_day_data.rename(
    columns = {
                "ObjectId": "Object_Id",
                "IdBundesland": "State_Id",
                "Bundesland": "State",
                "Landkreis": "District",
                "Altersgruppe": "Age_Group",
                "Geschlecht": "Sex",
                "AnzahlFall": "Number_of_Cases",
                "AnzahlTodesfall": "Number_of_Deaths",
                "Meldedatum": "Reporting_Date",
                "IdLandkreis": "District_Id",
                "Datenstand": "Last_Updated",
                "NeuerFall": "New_Case",
                "NeuerTodesfall": "New_Death",
                "Refdatum": "Reference_Date",
                "NeuGenesen": "New_Recovered",
                "AnzahlGenesen": "Number_of_Recovered",
                "IstErkrankungsbeginn": "Disease_Onset",
                "Altersgruppe2": "Age_Group2"
    }
);
current_day_data = current_day_data.rename(
    columns = {
                "ObjectId": "Object_Id",
```

```
                  "IdBundesland": "State_Id",
                  "Bundesland": "State",
                  "Landkreis": "District",
                  "Altersgruppe": "Age_Group",
                  "Geschlecht": "Sex",
                  "AnzahlFall": "Number_of_Cases",
                  "AnzahlTodesfall": "Number_of_Deaths",
                  "Meldedatum": "Reporting_Date",
                  "IdLandkreis": "District_Id",
                  "Datenstand": "Last_Updated",
                  "NeuerFall": "New_Case",
                  "NeuerTodesfall": "New_Death",
                  "Refdatum": "Reference_Date",
                  "NeuGenesen": "New_Recovered",
                  "AnzahlGenesen": "Number_of_Recovered",
                  "IstErkrankungsbeginn": "Disease_Onset",
                  "Altersgruppe2": "Age_Group2"
     }
);
```

[4]:

```
# Previewing data to check before use

germany_covid_data.head(5)
```

[4]:

```
   Object_Id  State_Id               State      District Age_Group Sex  \
0          1         1  Schleswig-Holstein  SK Flensburg   A00-A04   M
1          2         1  Schleswig-Holstein  SK Flensburg   A00-A04   M
2          3         1  Schleswig-Holstein  SK Flensburg   A00-A04   M
3          4         1  Schleswig-Holstein  SK Flensburg   A00-A04   M
4          5         1  Schleswig-Holstein  SK Flensburg   A00-A04   W

   Number_of_Cases  Number_of_Deaths       Reporting_Date  District_Id  \
0                1                 0  2020/09/30 00:00:00         1001
1                1                 0  2020/10/29 00:00:00         1001
2                1                 0  2020/11/03 00:00:00         1001
3                1                 0  2020/11/20 00:00:00         1001
4                1                 0  2020/08/24 00:00:00         1001

            Last_Updated  New_Case  New_Death       Reference_Date  \
0  22.11.2020, 00:00 Uhr         0         -9  2020/09/30 00:00:00
1  22.11.2020, 00:00 Uhr         0         -9  2020/10/29 00:00:00
2  22.11.2020, 00:00 Uhr         0         -9  2020/11/03 00:00:00
3  22.11.2020, 00:00 Uhr         0         -9  2020/11/19 00:00:00
4  22.11.2020, 00:00 Uhr         0         -9  2020/08/24 00:00:00

   New_Recovered  Number_of_Recovered  Disease_Onset        Age_Group2
0              0                    1              0  Nicht übermittelt
```

```
1             0                    1              0  Nicht übermittelt
2             0                    1              0  Nicht übermittelt
3            -9                    0              1  Nicht übermittelt
4             0                    1              0  Nicht übermittelt
```

[5]:
```
# Previewing data to check before use

previous_day_data.head(5)
```

[5]:
```
   Object_Id  State_Id               State District Age_Group Sex  \
0          1         1  Schleswig-Holstein  SK Kiel   A15-A34   M
1          2         1  Schleswig-Holstein  SK Kiel   A15-A34   W
2          3         1  Schleswig-Holstein  SK Kiel   A15-A34   W
3          4         1  Schleswig-Holstein  SK Kiel   A15-A34   W
4          5         1  Schleswig-Holstein  SK Kiel   A15-A34   W

   Number_of_Cases  Number_of_Deaths       Reporting_Date  District_Id  \
0                3                 0  2020/11/17 00:00:00         1002
1                1                 0  2020/03/10 00:00:00         1002
2                1                 0  2020/03/12 00:00:00         1002
3                1                 0  2020/03/16 00:00:00         1002
4                1                 0  2020/03/16 00:00:00         1002

            Last_Updated  New_Case  New_Death       Reference_Date  \
0  18.11.2020, 00:00 Uhr         1         -9  2020/11/17 00:00:00
1  18.11.2020, 00:00 Uhr         0         -9  2020/03/06 00:00:00
2  18.11.2020, 00:00 Uhr         0         -9  2020/03/11 00:00:00
3  18.11.2020, 00:00 Uhr         0         -9  2020/03/10 00:00:00
4  18.11.2020, 00:00 Uhr         0         -9  2020/03/13 00:00:00

   New_Recovered  Number_of_Recovered  Disease_Onset         Age_Group2
0             -9                    0              0  Nicht übermittelt
1              0                    1              1  Nicht übermittelt
2              0                    1              1  Nicht übermittelt
3              0                    1              1  Nicht übermittelt
4              0                    1              1  Nicht übermittelt
```

[6]:
```
# Previewing data to check before use

current_day_data.head(5)
```

[6]:
```
   Object_Id  State_Id               State       District Age_Group Sex  \
0      183.0         1  Schleswig-Holstein  SK Neumünster   A15-A34   W
1      316.0         1  Schleswig-Holstein  SK Neumünster   A35-A59   W
2      547.0         1  Schleswig-Holstein        SK Kiel   A15-A34   M
3      551.0         1  Schleswig-Holstein        SK Kiel   A15-A34   M
4     1046.0         1  Schleswig-Holstein        SK Kiel   A15-A34   M
```

```
   Number_of_Cases  Number_of_Deaths Reporting_Date  District_Id  \
0              1.0               0.0  18/11/20 0:00       1004.0
1              1.0               0.0  18/11/20 0:00       1004.0
2              1.0               0.0  18/11/20 0:00       1002.0
3              1.0               0.0  18/11/20 0:00       1002.0
4              1.0               0.0  18/11/20 0:00       1002.0

            Last_Updated  New_Case  New_Death Reference_Date  New_Recovered  \
0  19.11.2020, 00:00 Uhr       1.0       -9.0  18/11/20 0:00           -9.0
1  19.11.2020, 00:00 Uhr       1.0       -9.0  11/11/20 0:00           -9.0
2  19.11.2020, 00:00 Uhr       1.0       -9.0  11/11/20 0:00           -9.0
3  19.11.2020, 00:00 Uhr       1.0       -9.0  13/11/20 0:00           -9.0
4  19.11.2020, 00:00 Uhr       1.0       -9.0  16/11/20 0:00           -9.0

   Number_of_Recovered  Disease_Onset         Age_Group2
0                  0.0            0.0  Nicht übermittelt
1                  0.0            1.0  Nicht übermittelt
2                  0.0            1.0  Nicht übermittelt
3                  0.0            1.0  Nicht übermittelt
4                  0.0            1.0  Nicht übermittelt
```

[7]:
```
print("Lenght of Data Imported: ", len(germany_covid_data.columns));
germany_covid_data.shape
```

```
Lenght of Data Imported:  18
```

[7]:
```
(526110, 18)
```

1.4 —

1.4.1 Task 1

Part 1

> For each of the 18 feature columns in the RKI COVID-19 database, identify all scales of data whose definition is valid for all column entries. Create a table using python code that contains all column headers as rows, data scales as columns, and binary table entries indicating whether the feature values (i.e. column entries in the database) correspond to the data scale or not.

Summary So in this task, we go **Column By Column**, in order to decide in what all measurement scale categories does a particular column belongs and then categorize them into certain columns.

The allocations have been done by clearly going through the data, first by checking the data type for all the columns, to check which column is object, numeric and so on and to check if the data contains any null value or not.

```
germany_covid_data.info()
```

And then getting data insights using describe method

```
germany_covid_data.describe()
```

And finally, manually deciding the scale for each column

Measurement Scales:

1. Categorical Scale / Nominal Scale
2. Ordinal Scale
3. Interval Scale
4. Ratio Scale

1.5 —

[8]:
```
# Get an data overview for all the columns of the dataset

germany_covid_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 526110 entries, 0 to 526109
Data columns (total 18 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   Object_Id            526110 non-null  int64
 1   State_Id             526110 non-null  int64
 2   State                526110 non-null  object
 3   District             526110 non-null  object
 4   Age_Group            526110 non-null  object
 5   Sex                  526110 non-null  object
 6   Number_of_Cases      526110 non-null  int64
 7   Number_of_Deaths     526110 non-null  int64
 8   Reporting_Date       526110 non-null  object
 9   District_Id          526110 non-null  int64
 10  Last_Updated         526110 non-null  object
 11  New_Case             526110 non-null  int64
 12  New_Death            526110 non-null  int64
 13  Reference_Date       526110 non-null  object
 14  New_Recovered        526110 non-null  int64
 15  Number_of_Recovered  526110 non-null  int64
 16  Disease_Onset        526110 non-null  int64
 17  Age_Group2           526110 non-null  object
dtypes: int64(10), object(8)
memory usage: 72.3+ MB
```

[9]:
```
# Describing the data to further get insigt into the data

germany_covid_data.describe()
```

[9]:

```
             Object_Id       State_Id  Number_of_Cases  Number_of_Deaths  \
count  526110.000000  526110.000000    526110.000000     526110.000000
mean   263055.500000       7.502724         1.745186          0.026648
std    151875.019399       3.233262         2.913915          0.177354
min         1.000000       1.000000        -2.000000         -1.000000
25%    131528.250000       5.000000         1.000000          0.000000
50%    263055.500000       8.000000         1.000000          0.000000
75%    394582.750000       9.000000         1.000000          0.000000
max    526110.000000      16.000000       147.000000          8.000000

         District_Id       New_Case      New_Death  New_Recovered  \
count  526110.000000  526110.000000  526110.000000  526110.000000
mean     7832.903427       0.011047      -8.776963      -1.883479
std      3201.898329       0.106451       1.400059       3.677506
min      1001.000000      -1.000000      -9.000000      -9.000000
25%      5515.000000       0.000000      -9.000000       0.000000
50%      8118.000000       0.000000      -9.000000       0.000000
75%      9474.000000       0.000000      -9.000000       0.000000
max     16077.000000       1.000000       1.000000       1.000000

       Number_of_Recovered  Disease_Onset
count        526110.000000  526110.000000
mean              1.147534       0.664502
std               1.621040       0.472165
min              -1.000000       0.000000
25%               1.000000       0.000000
50%               1.000000       1.000000
75%               1.000000       1.000000
max             147.000000       1.000000
```

[10]:

```python
# Creating a dictionary and then assigning values to columns in binary terms␣
↪(Yes/No)

tabular_scaling_data = {};
tabular_scaling_data["Categorical Scale / Nominal Scale"] = ["No", "Yes",␣
↪"Yes", "Yes", "Yes", "Yes", "No", "No", "No", "Yes", "No", "Yes", "Yes",␣
↪"No", "Yes", "No", "Yes", "No"]
tabular_scaling_data["Ordinal Scale"] = ["No", "No", "No", "No", "Yes", "No",␣
↪"No", "No", "Yes", "No", "No", "No", "No", "Yes", "No", "No", "No", "No"]
tabular_scaling_data["Interval Scale"] = ["No", "No", "No", "No", "Yes", "No",␣
↪"No", "No", "Yes", "No", "No", "No", "No", "Yes", "No", "No", "No", "No"]
tabular_scaling_data["Ratio Scale"] = ["No", "No", "No", "No", "No", "No",␣
↪"Yes", "Yes", "No", "No", "No", "No", "No", "No", "Yes", "Yes", "No", "No"]
```

[11]:

```python
# Display the dictionary in a tabular form
```

```
df = pd.DataFrame.from_dict(tabular_scaling_data, columns=germany_covid_data.
 ↪columns, orient='index')
df.transpose()
```

[11]:

```
                     Categorical Scale / Nominal Scale Ordinal Scale  \
Object_Id                                           No            No
State_Id                                           Yes            No
State                                              Yes            No
District                                           Yes            No
Age_Group                                          Yes           Yes
Sex                                                Yes            No
Number_of_Cases                                     No            No
Number_of_Deaths                                    No            No
Reporting_Date                                      No           Yes
District_Id                                        Yes            No
Last_Updated                                        No            No
New_Case                                           Yes            No
New_Death                                          Yes            No
Reference_Date                                      No           Yes
New_Recovered                                      Yes            No
Number_of_Recovered                                 No            No
Disease_Onset                                      Yes            No
Age_Group2                                          No            No

                    Interval Scale Ratio Scale
Object_Id                       No          No
State_Id                        No          No
State                           No          No
District                        No          No
Age_Group                      Yes          No
Sex                             No          No
Number_of_Cases                 No         Yes
Number_of_Deaths                No         Yes
Reporting_Date                 Yes          No
District_Id                     No          No
Last_Updated                    No          No
New_Case                        No          No
New_Death                       No          No
Reference_Date                 Yes          No
New_Recovered                   No         Yes
Number_of_Recovered             No         Yes
Disease_Onset                   No          No
Age_Group2                      No          No
```

1.6 —

Part 2

Identify two feature columns and respective data scales for which a precise allocation is either not feasible or questionable and explain your decision.

Summary **Feature Columns and respective Data Scales** for which precise allocation is not feasible or questionable is: - **Object_Id** — **Interval Scale** : The data does not represent any category and changing its order does not really affect the data but it does have a constant interval of 1 but that interval does not have any significance. Hence, the allocation of ***Interval Scale*** as no is ***Questionable***. - **Age_Group** — **Interval Scale / Ratio Scale** : The data does represent category and it does have a specific ranking which does affect the data but it does have a constant interval but that interval does have some significance. Hence, the allocation of ***Interval Scale*** as yes is ***Questionable***. Similarly, it does have a Meaningful Zero as Age 0, but in context to data, Age 0 does not signify anything. Hence, the allocation of ***Ratio Scale*** as no is ***Questionable***. - **Last_Updated** — **All Scales** : The data has a constant value throughout the data. Even if we change the value, it will have zero or no effect. Hence, allocating ***Any Scale*** is ***Not Feasible***. - **Age_Group2** — **All Scales** : The data has a constant value throughout the data. Even if we change the value, it will have zero or no effect. Hence, allocating ***Any Scale*** is ***Not Feasible***.

1.7 —

[12]:
```python
# Creating a dictionary and then assigning values to columns with respective␣
↪questionable scales

tabular_questionable_scaling_data = {}
tabular_questionable_scaling_data['Columns'] = ['Object_Id', 'Age_Group',␣
↪'Last_Updated', 'Age_Group2']
tabular_questionable_scaling_data['Scales'] = ['Interval Scale', 'Interval␣
↪Scale, Ratio Scale', 'All the Scales', 'All the scales']
```

[13]:
```python
# Display the dictionary in a tabular form

df = pd.DataFrame.from_dict(tabular_questionable_scaling_data, orient='index')
df.transpose()
```

[13]:

	Columns	Scales
0	Object_Id	Interval Scale
1	Age_Group	Interval Scale, Ratio Scale
2	Last_Updated	All the Scales
3	Age_Group2	All the scales

Task 2 [10 points] Data Collection With respect to the COVID-19 database, give four different examples of potential measurement errors/inaccuracies that can occur during data collection and, for each example, identify a column whose entries can be negatively impacted by that error. Explain your choices briefly.

1.8 —

1.8.1 Task 2

> With respect to the COVID-19 database, give four different examples of potential measurement errors/inaccuracies that can occur during data collection and, for each example, identify a column whose entries can be negatively impacted by that error. Explain your choices briefly.

Summary

Example 1

Error Column:

```
- New_Case
- New_Death
- New_Recovered
```

Impacted Column:

```
- Number_of_Cases
- Number_of_Deaths
- Number_of_Recovered
```

Summary

- New Cases are identified using three Categories: **0, 1 and -1**
  - 0 Contains data for Today's and Yesterday's Publication
  - 1 Contains data for Today's Publication
  - -1 Contains data for yesterday's Publication
- Category(0): Total Cases in Yesterday's Publication = 4,83,697 Cases
- Category(1): Total Cases in Yesterday's Publication = 6,943 Cases
- Category(0): Total Cases in Today's Publication = 4,92,381 Cases
- Category(-1): Total Cases in Today's Publication = 247 Cases

So if we **sum** yesterday's Category(1) cases and yesterday's Category(0) cases, **Total = 4,90,640**
Now, this is **1,988** less cases than Category(0) + Category(-1) cases in Today's Publication

> ***Note:*** *Category(0) + Category(1) cases of yesterday should be equal to Category(0) + Category(-1) cases of today*
>
> ***Note:*** *All calculations are done using data for 18th and 19th November*

1.9 —

[14]:
```python
# Initializing variables to calculate specific sums of data

yesterday_sum = 0
today_sum = 0

# Calculating Category(0) and Category(1) data for 18th November 2020

previous_new_case_data = previous_day_data['New_Case'] == 0
previous_new_case_filtered_data = previous_day_data[previous_new_case_data]
yesterday_sum = yesterday_sum + len(previous_new_case_filtered_data)
print("Publication: 18th November 2020: Category(0) : Total Cases: ",␣
↪len(previous_new_case_filtered_data))

previous_new_case_data = previous_day_data['New_Case'] > 0
previous_new_case_filtered_data = previous_day_data[previous_new_case_data]
yesterday_sum = yesterday_sum + len(previous_new_case_filtered_data)
print("Publication: 18th November 2020: Category(1) : Total Cases: ",␣
↪len(previous_new_case_filtered_data))

print("Publication: 18th November 2020: Total Cases: ", yesterday_sum)

# Calculating Category(0) and Category(-1) data for 19th November 2020

current_new_case_data = current_day_data['New_Case'] == 0
current_new_case_filtered_data = current_day_data[current_new_case_data]
today_sum = today_sum + len(current_new_case_filtered_data)
print("\nPublication: 19th November 2020: Category(0) : Total Cases: ",␣
↪len(current_new_case_filtered_data))

current_new_case_data = current_day_data['New_Case'] < 0
current_new_case_filtered_data = current_day_data[current_new_case_data]
today_sum = today_sum + len(current_new_case_filtered_data)
print("Publication: 19th November 2020: Category(-1): Total Cases: ",␣
↪len(current_new_case_filtered_data))

print("Publication: 19th November 2020: Total Cases: ", today_sum)

# Getting Inconsistent Data between two publications

print("\nInconsistent Data: ", today_sum - yesterday_sum)
```

```
Publication: 18th November 2020: Category(0) : Total Cases:  483697
Publication: 18th November 2020: Category(1) : Total Cases:  6943
Publication: 18th November 2020: Total Cases:  490640

Publication: 19th November 2020: Category(0) : Total Cases:  492381
```

```
Publication: 19th November 2020: Category(-1): Total Cases:  247
Publication: 19th November 2020: Total Cases:  492628

Inconsistent Data:  1988
```

Example 2

Error Column:

```
- Reference_Date
- Reporting_Date
```

Impacted Column:

```
- Number_of_Cases
```

Summary

- Refernce Date is said to be date of Illness as well as Reporting data.
- The possible error can be that Date of Illness is greater than the reporting date. Which means a person got ill on 28th October 2020 but his illness was reported on 17th November 2020.
- Therefore the data is incorrect and negatively impacts the **Number_of_Cases** reported.

Now, this is proved by following steps: - Firstly, we check the datatype of **Reference_Date** and **Reporting_Date**, which is **object - dtype('O')** - Secondly, we convert the datatype to compatible timestamp of **datetimestamp dtype('<M8[ns]')**

```
reference_date = pd.to_datetime(germany_covid_data["Reference_Date"],
format='%Y%m%d %H:%M:%S')
```

```
reporting_date = pd.to_datetime(germany_covid_data["Reporting_Date"],
format='%Y%m%d %H:%M:%S')
```

- Thirdly, we **substract Reporting_Date and Reference_Date** to form **early_reporting_data**

  ```
  early_reporting = (reporting_date - reference_date).dt.days
  ```

- Fourthly, we view the sample data to check if the data computed is correct or not

  ```
  early_reporting_data = germany_covid_data[early_reporting_true];
  ```

  ```
  early_reporting_data[['Reference_Date','Reporting_Date']].head(5)
  ```

- Lastly, we find out that there are 4551 Entries where the infection was reported before the onset of the illness.

1.10 > *Note: This count is subject to change as the data will be re-calculated on the day of execution. Mentioned count was taken on 22st November 2020*

[15]:
```python
# Verifying data type for Reference_Date column

germany_covid_data["Reference_Date"].dtype
```

[15]: dtype('O')

[16]:
```python
# Verifying data type for Reporting_Date column

germany_covid_data["Reporting_Date"].dtype
```

[16]: dtype('O')

[17]:
```python
# Converting column to compatible data type by formating and confirming the␣
↪data type

reference_date = pd.to_datetime(germany_covid_data["Reference_Date"],␣
↪format='%Y%m%d %H:%M:%S')
reference_date.dtype
```

[17]: dtype('<M8[ns]')

[18]:
```python
# Converting column to compatible data type by formating and confirming the␣
↪data type

reporting_date = pd.to_datetime(germany_covid_data["Reporting_Date"],␣
↪format='%Y%m%d %H:%M:%S')
reporting_date.dtype
```

[18]: dtype('<M8[ns]')

[19]:
```python
# Retreiving sample data where the Reporting_Date is less than Reference Date

early_reporting = (reporting_date - reference_date).dt.days;
early_reporting_true = early_reporting < 0;

print("\nSample Data where the Reporting_Date is less than Reference_Date")

early_reporting_data = germany_covid_data[early_reporting_true];
early_reporting_data[['Reference_Date','Reporting_Date']].head(5)
```

```
Sample Data where the Reporting_Date is less than Reference_Date
```

[19]:
```
          Reference_Date       Reporting_Date
23   2020/03/16 00:00:00  2020/03/14 00:00:00
144  2020/03/16 00:00:00  2020/03/14 00:00:00
193  2020/03/20 00:00:00  2020/03/19 00:00:00
402  2020/07/02 00:00:00  2020/07/01 00:00:00
577  2020/10/12 00:00:00  2020/10/07 00:00:00
```

[20]:
```python
# Getting final value of incorrect data

print("Incorrect Entries where Infection was Reported Earlier than Onset of the␣
↪Illness: ", len(early_reporting[early_reporting_true]));
```

```
Incorrect Entries where Infection was Reported Earlier than Onset of the
Illness:  4551
```

Example 3

Error Column:

- `Disease_Onset`

Impacted Column:

- `Reference_Date`
- `Reporting_Date`
- `Number_of_Cases`
- `Number_of_Recovered`

Summary

- As per the data publication, the **No_of_Recovered** is being done based on an estimation that a normal person recovers in 14 days and elderly or people with severe condition recover in 28 Days. > The duration of the illness is estimated for each case on the basis of the detailed information on a case of illness sent by the health authorities to the RKI. For cases in which only symptoms are indicated that suggest a mild course of the disease, a duration of the disease of 14 days is assumed. In hospitalized cases or cases with symptoms that indicate a severe course (e.g. pneumonia), the duration of the illness is assumed to be 28 days. Based on the beginning of the illness or, if this is not known, the reporting date, there is an estimated recovery date for each case. Since in individual cases significantly longer disease courses are possible, or the information used here is not transmitted to the RKI in all cases, the data calculated in this way are only rough estimates of the number of people recovered and should therefore only be used taking these limitations into account.
- But this data does not present the actual data of people who are getting recovered.
- Therefore the data is incorrect and negatively impacts the **Number_of_Cases** reported.
- While proving this impact, we found **data inconsistency** in the data for **Disease_Onset**.

- Data is inconsistent and everyday update in data also changes the data belonging to older date.

Now, this is proved by following steps: - Firstly, we bring in the data for **19th November 2020** - Secondly, we bring in the data for **18th November 2020**

Now as per the publication, **Disease_Onset = 1** represents **Reference_Date** as the onset date.

Therefore, we try to compare the data from **two consecutive days** and we see that there is clearly a difference between data published yesterday and data published today.

Hence, we confirm data inconsistency/error in the Disease_Onset Data

---

[21]:
```
# Calculating data for Disease_Onset=1 for 18th November 2020

previous_day_data_with_onset = previous_day_data['Disease_Onset'] > 0
previous_day_data_filtered = previous_day_data[previous_day_data_with_onset]
previous_day_reporting_data = pd.
 ↪to_datetime(previous_day_data_filtered['Reference_Date'], format="%Y/%m/%d␣
 ↪%H:%M:%S").dt.month_name().to_list()
previous_day_reporting_dataframe = pd.DataFrame(previous_day_reporting_data)

# Calculating data for Disease_Onset=1 for 19th November 2020

current_day_data_with_onset = current_day_data['Disease_Onset'] > 0
current_day_data_filtered = current_day_data[current_day_data_with_onset]
current_day_reporting_data = pd.
 ↪to_datetime(current_day_data_filtered['Reference_Date'], format="%d/%m/%y %H:
 ↪%M").dt.month_name().to_list()
current_day_reporting_dataframe = pd.DataFrame(current_day_reporting_data)
```

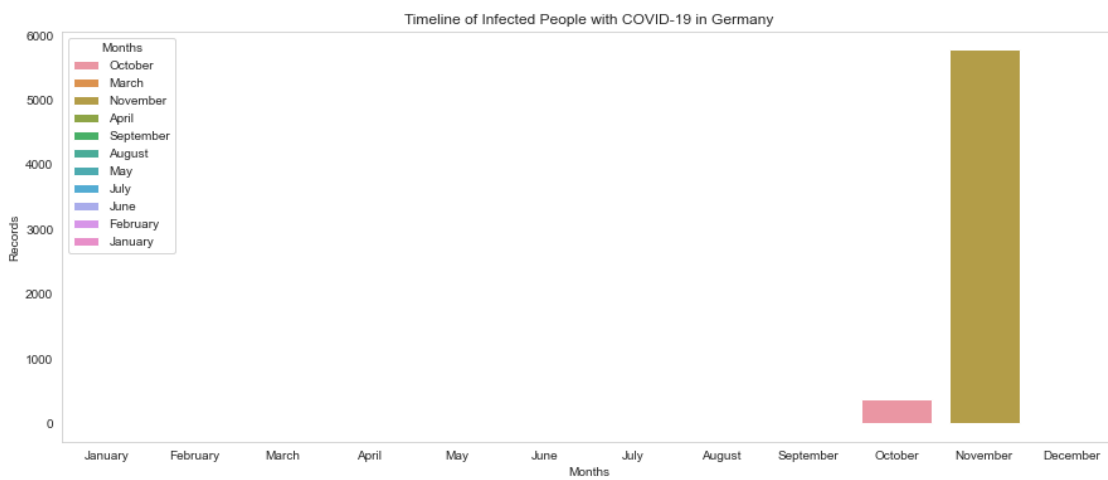
[22]:
```
# Reset of the pandas dataframe

previous_monthly_data = previous_day_reporting_dataframe.value_counts().
 ↪rename_axis('Months').reset_index(name='Records')
current_monthly_data = current_day_reporting_dataframe.value_counts().
 ↪rename_axis('Months').reset_index(name='Records')
```

[23]:
```
# Now, calculating the monthly difference of the Data Onset values
monthly_data_difference = current_monthly_data['Records'] -␣
 ↪previous_monthly_data['Records']
monthly_data_difference_dataframe = pd.DataFrame(monthly_data_difference)
monthly_data_difference_dataframe['Months'] = current_monthly_data['Months']
```

- In the Visualization below, it is clearly visible that data for **October** is also changing whereas the only expected change was in the data for **November**

[77]:
```python
# Plotting the data for better understanding the issue

plt.figure(figsize=(15,6))
plt.title('Timeline of Infected People with COVID-19 in Germany')
sns.barplot(x="Months", y="Records", data=monthly_data_difference_dataframe,␣
 ↪order=['January','February','March','April','May','June','July','August','September','Octob
 ↪hue='Months', dodge=False)
```

[77]:
```
<AxesSubplot:title={'center':'Timeline of Infected People with COVID-19 in
Germany'}, xlabel='Months', ylabel='Records'>
```



Example 4

Error Column:

```
- Age_Group
- Sex
```

Impacted Column:

```
- Age_Group
- Sex
- Entire Dataset
```

Summary Unknown - unbekannt values in both **Age_Group** and **Sex** data column are errorsome. - This **impacts** the **Entire Dataset** as the categorization gets affected throughout the data. - This shows **Errorsome Data**, because if the infection was reported, then not knowing the Age and Gender is questionable.

[25]:
```python
# Filtering data based on the data value

unknown_data = germany_covid_data['Age_Group'] == 'unbekannt'
germany_covid_data[unknown_data].head(5)
```

[25]:
```
      Object_Id  State_Id               State                District  \
2942       2943         1  Schleswig-Holstein        LK Dithmarschen
2943       2944         1  Schleswig-Holstein        LK Dithmarschen
2944       2945         1  Schleswig-Holstein        LK Dithmarschen
3217       3218         1  Schleswig-Holstein       LK Nordfriesland
4219       4220         1  Schleswig-Holstein  LK Herzogtum Lauenburg

      Age_Group Sex  Number_of_Cases  Number_of_Deaths       Reporting_Date  \
2942  unbekannt   M                1                 0  2020/10/31 00:00:00
2943  unbekannt   W                3                 0  2020/10/31 00:00:00
2944  unbekannt   W                1                 0  2020/11/01 00:00:00
3217  unbekannt   M                1                 0  2020/11/20 00:00:00
4219  unbekannt   M                1                 0  2020/11/21 00:00:00

      District_Id           Last_Updated  New_Case  New_Death  \
2942         1051  22.11.2020, 00:00 Uhr         0         -9
2943         1051  22.11.2020, 00:00 Uhr         0         -9
2944         1051  22.11.2020, 00:00 Uhr         0         -9
3217         1054  22.11.2020, 00:00 Uhr         0         -9
4219         1053  22.11.2020, 00:00 Uhr         1         -9

           Reference_Date  New_Recovered  Number_of_Recovered  Disease_Onset  \
2942  2020/10/31 00:00:00             -9                    0              0
2943  2020/10/31 00:00:00             -9                    0              0
2944  2020/11/01 00:00:00             -9                    0              0
3217  2020/11/20 00:00:00             -9                    0              0
4219  2020/11/21 00:00:00             -9                    0              0

             Age_Group2
2942  Nicht übermittelt
2943  Nicht übermittelt
2944  Nicht übermittelt
3217  Nicht übermittelt
4219  Nicht übermittelt
```

[26]:
```python
# Filtering data based on the data value

unknown_data = germany_covid_data['Sex'] == 'unbekannt'
germany_covid_data[unknown_data].head(5)
```

```
[26]:       Object_Id  State_Id               State               District  \
2043         2044         1  Schleswig-Holstein                SK Kiel
2736         2737         1  Schleswig-Holstein        LK Dithmarschen
3348         3349         1  Schleswig-Holstein         LK Ostholstein
3687         3688         1  Schleswig-Holstein         LK Ostholstein
4195         4196         1  Schleswig-Holstein  LK Herzogtum Lauenburg

     Age_Group        Sex  Number_of_Cases  Number_of_Deaths  \
2043   A15-A34  unbekannt                1                 0
2736   A35-A59  unbekannt                1                 0
3348   A15-A34  unbekannt                1                 0
3687      A80+  unbekannt                1                 0
4195      A80+  unbekannt               -1                 0

           Reporting_Date  District_Id           Last_Updated  New_Case  \
2043  2020/11/21 00:00:00         1002  22.11.2020, 00:00 Uhr         1
2736  2020/11/07 00:00:00         1051  22.11.2020, 00:00 Uhr         0
3348  2020/11/08 00:00:00         1055  22.11.2020, 00:00 Uhr         0
3687  2020/10/29 00:00:00         1055  22.11.2020, 00:00 Uhr         0
4195  2020/11/20 00:00:00         1053  22.11.2020, 00:00 Uhr        -1

      New_Death       Reference_Date  New_Recovered  Number_of_Recovered  \
2043         -9  2020/11/21 00:00:00             -9                    0
2736         -9  2020/11/07 00:00:00              0                    1
3348         -9  2020/11/02 00:00:00             -9                    0
3687         -9  2020/10/29 00:00:00              0                    1
4195         -9  2020/11/20 00:00:00             -9                    0

      Disease_Onset         Age_Group2
2043              0  Nicht übermittelt
2736              0  Nicht übermittelt
3348              1  Nicht übermittelt
3687              0  Nicht übermittelt
4195              0  Nicht übermittelt
```

Task 3 [30 points] Visualization Plot the following fact sheets using suitable python packages. Make sure to use appropriate plot types for visualization (e.g. histogram, scatter plot, ...) and proper axis labelling/scaling. Add a legend to each plot to facilitate the viewers understanding. *Note: Active cases are defined as patients whose COVID-19 infection did not have an outcome yet (neither died nor recovered).*

1. Timeline of accumulated number of people infected with COVID-19 in Germany.
2. Number of active cases **per age group** as of October 1st, 2020.
3. Overall average of daily active cases **per county**.
4. Timeline of daily deaths from COVID-19 in Germany *(i)* overall and *(ii)* by gender.

1.11 —

1.11.1 Task 3

Part 1

Timeline of accumulated number of people infected with COVID-19 in Germany.

Summary To get the timeline of the accumulated infected people, `cumsum()` method was used which calculates the cumulated sum and it was calculated based on the **Reporting_Date**

```
cumulated_data = germany_covid_data.groupby('Reporting_Date')['Number_of_Cases'].sum().cum
```

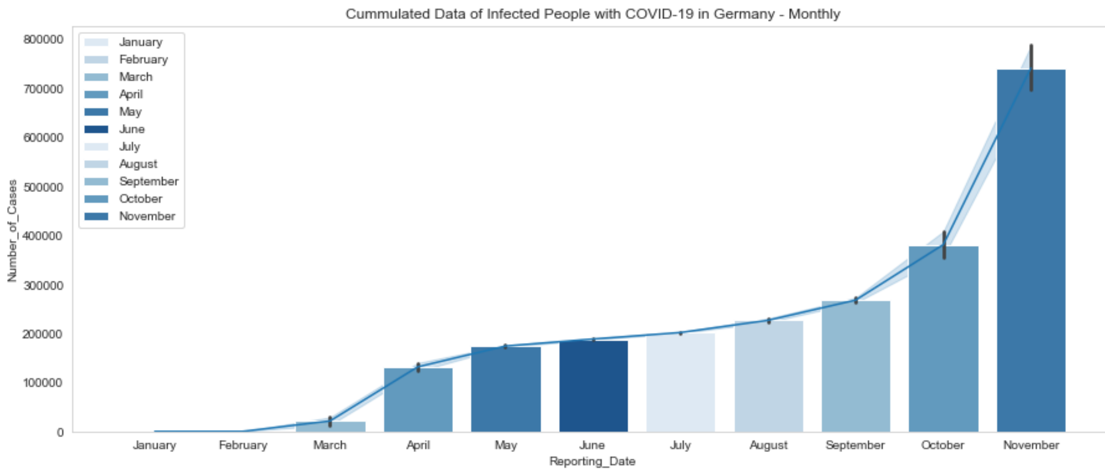
After the calculation, the same was plotted: - As a **Bar Plot** grouped based on **Monthly Data**. This gives a very clear picture of how exactly the infections kept on increasing every. Also, to show the increase, used **Line Plot** on the same graph. - As a **Bar Plot** grouped based on **Daily Data**. This gives the overall picture where a lot of days, the count of infections was very constant and then it suddenly starts to rise from the month of **August**

1.12 —

[71]:
```
# Calculating the cumulated sum of 'Number_of_Cases' by grouping the data based␣
↪on 'Reporting_Date'
# Reshaping the Dataframe to plot correctly
# Further grouping the data based on Months

cumulated_data = germany_covid_data.
↪groupby('Reporting_Date')['Number_of_Cases'].sum().cumsum()
cumulated_data_dataframe = pd.DataFrame(cumulated_data).
↪rename_axis('Reporting_Date').reset_index()
cumulated_data_dataframe['Reporting_Date'] = pd.
↪to_datetime(cumulated_data_dataframe['Reporting_Date'], format="%Y/%m/%d %H:
↪%M:%S").dt.month_name()
```

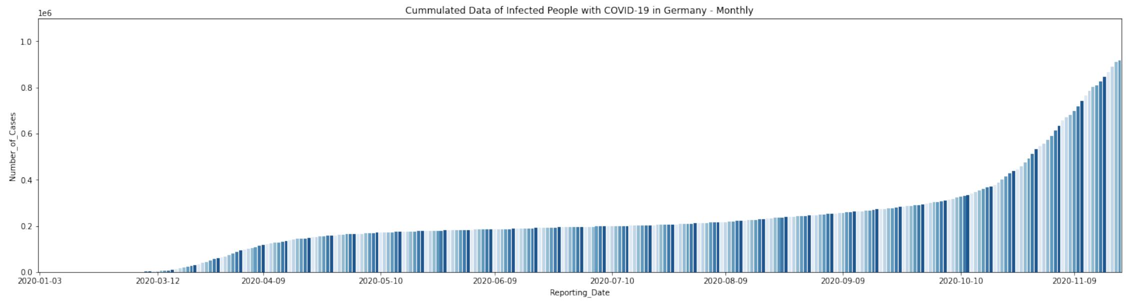
[76]:
```
# Plotting the data with 'Reporting_Date' (Months) as x-axis and␣
↪'Number_of_Cases' as y-axis
# Subplotting Bar Plot and Line Plot to show the gradual rise

plt.figure(figsize=(15,6))
plt.title('Cummulated Data of Infected People with COVID-19 in Germany -␣
↪Monthly')
sns.barplot(x="Reporting_Date", y="Number_of_Cases",␣
↪data=cumulated_data_dataframe, palette = sns.color_palette("Blues"),␣
↪hue='Reporting_Date', dodge=False)
sns.lineplot(x="Reporting_Date", y="Number_of_Cases",␣
↪data=cumulated_data_dataframe)
```

[76]:
```
<AxesSubplot:title={'center':'Cummulated Data of Infected People with COVID-19
in Germany - Monthly'}, xlabel='Reporting_Date', ylabel='Number_of_Cases'>
```



[29]:
```
# Calculating the cumulated sum of 'Number_of_Cases' by grouping the data based␣
↪on 'Reporting_Date'
# Reshaping the Dataframe to plot correctly
# Further grouping the data based on Date

cumulated_data = germany_covid_data.
↪groupby('Reporting_Date')['Number_of_Cases'].sum().cumsum()
cumulated_data_dataframe = pd.DataFrame(cumulated_data).
↪rename_axis('Reporting_Date').reset_index()
cumulated_data_dataframe['Reporting_Date'] = pd.
↪to_datetime(cumulated_data_dataframe['Reporting_Date']).dt.
↪strftime('%Y-%m-%d')
```

[30]:
```
# Plotting the data with 'Reporting_Date' (Months) as x-axis and␣
↪'Number_of_Cases' as y-axis
# Subplotting Bar Plot and Line Plot to show the gradual rise
# Formatting the x-axis as there is too much data to show, so removed the␣
↪clutter in the axis

plt.figure(figsize=(25,6))
plt.title('Cummulated Data of Infected People with COVID-19 in Germany -␣
↪Monthly')
ax = sns.barplot(x="Reporting_Date", y="Number_of_Cases",␣
↪data=cumulated_data_dataframe, palette = sns.color_palette("Blues"))

xtick_locator = AutoDateLocator(maxticks=11)
ax.xaxis.set_major_locator(xtick_locator)

plt.ylim(0, 1100000)
plt.show()
```



Part 2

Number of active cases per age group as of October 1st, 2020.

Summary In order to calculate the **Active Cases** per **Age Group** as of **October 1st, 2020**, - Firstly, filter out the data based on **Reporting_Date less than equal to October 1st, 2020** into **data_till_october**. - The filtering is being done by first converting the **Reporting_Date** to a compatible format and then comparing it with **October 1st, 2020** and this returns the index of the values where the condition matches. > `dates_till_october = pd.to_datetime(pd.to_datetime(germany_covid_data['Reporting_Date']).dt.strftime('%Y-%m-%d')) <= pd.to_datetime('2020-10-01')`

`- Once the indexes are fetched, using these indices, filter out the data.`

`` > `data_till_october = germany_covid_data[dates_till_october]` ``

- Secondly, the data is filtered in a similar fashion for
  - Filtering based on `data_till_october['New_Case'] >= 0` into **data_till_october_infected** because we need the count for infected
  - Filtering based on `data_till_october_infected['New_Death'] == -9` into **data_till_october_not_died** because we need the count for people who are still alive
  - Filtering based on `data_till_october_not_died['New_Recovered'] == -9` into **data_till_october_filtered** because we need the count for infected but not recovered.
- Finally, we group the data by **Age_Group** > `active_data_by_age = data_till_october_filtered.groupby('Age_Group')['Number_of_Cases'].sum()`

---

[31]:
```
# Comparing and filtering data based on October 1st, 2020

dates_till_october = pd.to_datetime(pd.
 ↪to_datetime(germany_covid_data['Reporting_Date']).dt.strftime('%Y-%m-%d'))␣
 ↪<= pd.to_datetime('2020-10-01')
data_till_october = germany_covid_data[dates_till_october]
```

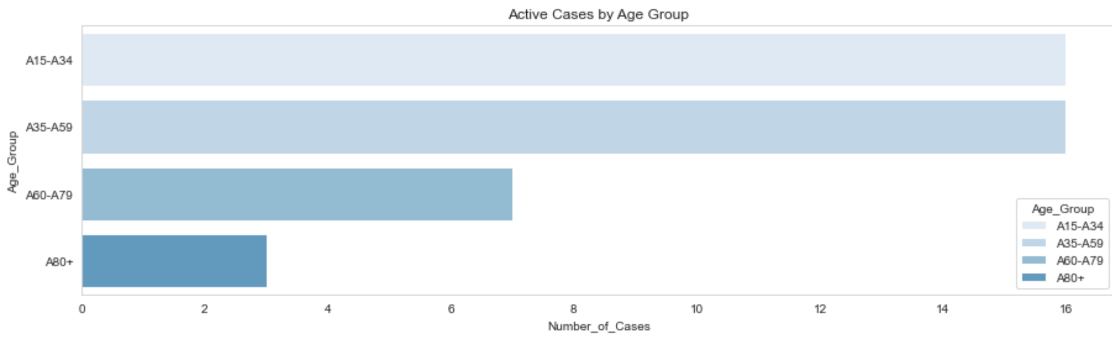
```
# Filtering data for infected cases

data_till_october_index = data_till_october['New_Case'] >= 0
data_till_october_infected = data_till_october[data_till_october_index]

# Filtering data for Non-Death cases

data_till_october_not_died_index = data_till_october_infected['New_Death'] == -9
data_till_october_not_died =␣
↪data_till_october_infected[data_till_october_not_died_index]

# Filtering data for Not-Recovered cases

data_till_october_filtered_index = data_till_october_not_died['New_Recovered']␣
↪== -9
data_till_october_filtered =␣
↪data_till_october_not_died[data_till_october_filtered_index]

# Grouping data by Age Group

active_data_by_age = data_till_october_filtered.
↪groupby('Age_Group')['Number_of_Cases'].sum()
print("Active Cases by Age Group are : \n", active_data_by_age)
```

```
Active Cases by Age Group are :
 Age_Group
A15-A34    16
A35-A59    16
A60-A79     7
A80+        3
Name: Number_of_Cases, dtype: int64
```

[32]:
```
active_data_by_age = pd.DataFrame(active_data_by_age).rename_axis('Age_Group').
↪reset_index()
```

[67]:
```
# Plotting the data with 'Number_of_Cases' as x-axis and 'Age_Group' as y-axis

plt.figure(figsize=(15,4))
plt.title('Active Cases by Age Group')
sns.barplot(x="Number_of_Cases", y="Age_Group", data=active_data_by_age,␣
↪palette = sns.color_palette("Blues"), hue='Age_Group', dodge=False)
```

[67]:
```
<AxesSubplot:title={'center':'Active Cases by Age Group'},
xlabel='Number_of_Cases', ylabel='Age_Group'>
```



Part 3

Overall average of daily active cases per county.

Summary In order to calculate the daily average of active cases per State, we follow the same way like we did to calculate the active cases till October 1st, 2020, except that this time we do not filter it by Reporting_Date but we group by **State** to achieve the outcome.

Also, we round-off the values and convert them to int for

---

[34]:
```
# Filtering data for infected cases

data_index = germany_covid_data['New_Case'] >= 0
data_infected = germany_covid_data[data_index]

# Filtering data for Non-Death cases

data_not_died_index = data_infected['New_Death'] == -9
data_not_died = data_infected[data_not_died_index]

# Filtering data for Not-Recovered cases

data_filtered_index = data_not_died['New_Recovered'] == -9
data_filtered = data_not_died[data_filtered_index]

# Grouping data by States

data_filtered_by_states = data_filtered.groupby('State')['Number_of_Cases'].
 ↪mean().round().astype(int)
print(data_filtered_by_states)
```

```
State
Baden-Württemberg          3
```

```
Bayern                     3
Berlin                     4
Brandenburg                1
Bremen                     5
Hamburg                   11
Hessen                     4
Mecklenburg-Vorpommern     2
Niedersachsen              2
Nordrhein-Westfalen        4
Rheinland-Pfalz            2
Saarland                   2
Sachsen                    3
Sachsen-Anhalt             2
Schleswig-Holstein         2
Thüringen                  1
Name: Number_of_Cases, dtype: int64
```

[35]:
```
data_filtered_by_states = pd.DataFrame(data_filtered_by_states).
↪rename_axis('State').reset_index()
```

[36]:
```
data_filtered_by_states = data_filtered_by_states.
↪sort_values(by=['Number_of_Cases'])
```

[37]:
```
data_filtered_by_states
```

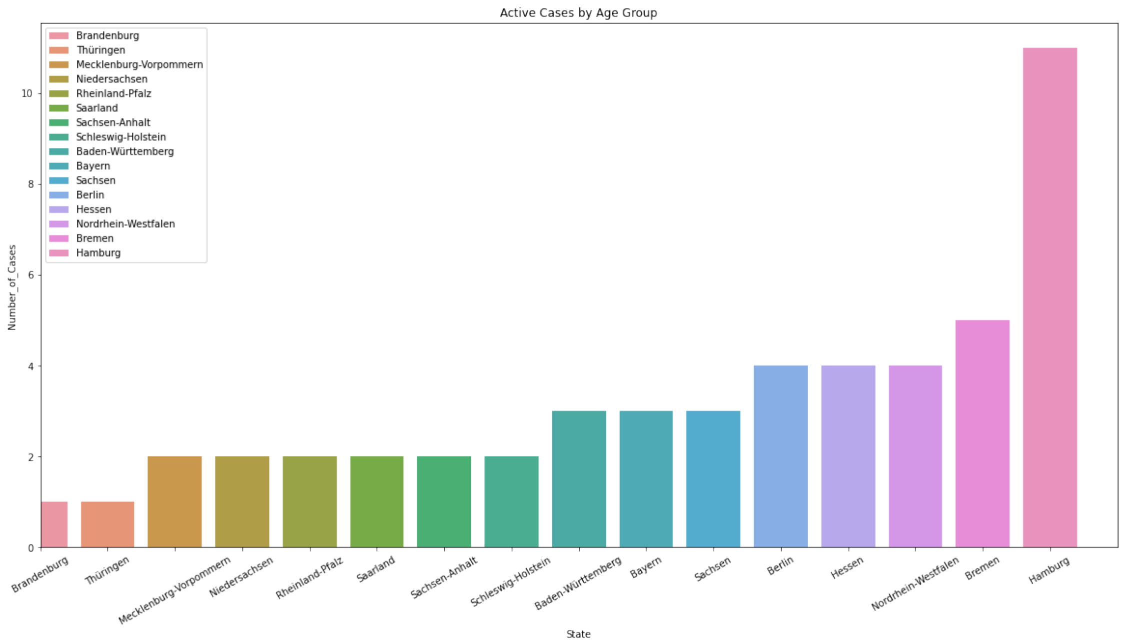
[37]:

	State	Number_of_Cases
3	Brandenburg	1
15	Thüringen	1
7	Mecklenburg-Vorpommern	2
8	Niedersachsen	2
10	Rheinland-Pfalz	2
11	Saarland	2
13	Sachsen-Anhalt	2
14	Schleswig-Holstein	2
0	Baden-Württemberg	3
1	Bayern	3
12	Sachsen	3
2	Berlin	4
6	Hessen	4
9	Nordrhein-Westfalen	4
4	Bremen	5
5	Hamburg	11

[38]:
```
# Plotting the data with 'Number_of_Cases' as x-axis and 'Age_Group' as y-axis

plt.figure(figsize=(20,10))
plt.title('Active Cases by Age Group')
```

```
sns.barplot(y="Number_of_Cases", x="State", data=data_filtered_by_states,␣
 ↪hue='State', dodge=False)
plt.xticks(rotation=30)
plt.xlim(0,16)
plt.legend(loc='upper left')
```

[38]: <matplotlib.legend.Legend at 0x12999a9d0>



Part 4

Timeline of daily deaths from COVID-19 in Germany (i) overall and (ii) by gender.

Summary In order to get the timeline of **Overall Daily Deaths**, - Firstly, we filter the data by `germany_covid_data['New_Death'] >= 0` as we need count only for no of deaths. This returns the index of the values where the condition is valid. Now using this index, we filter out the data.

> `data_filtered_for_death = germany_covid_data[data_filtered_for_death_index]`

- Secondly, we group the data by **Reporting_Date**

```
overall_deaths = data_filtered_for_death.groupby('Reporting_Date')['Number_of_Deaths']
```

In order to get the timeline of **Daily Deaths by Gender**, - Firstly, we filter the data by `data_filtered_for_death['Sex'] == 'M'` for **Male** and `data_filtered_for_death['Sex'] == 'W'` for **Female**. This returns the index of the values where the condition is valid. Now using this index, we filter out the data.

> `male_deaths = data_filtered_for_death[male_deaths_index]`
>
> `female_deaths = data_filtered_for_death[female_deaths_index]`

- Secondly, we group the data by **Reporting_Date**

```
male_deaths_filtered = male_deaths.groupby('Reporting_Date')['Number_of_Deaths'].sum(
```

```
female_deaths_filtered = female_deaths.groupby('Reporting_Date')['Number_of_Deaths'].s
```

Finally, we plot the timeline of Overall and Gender based data simultaneously

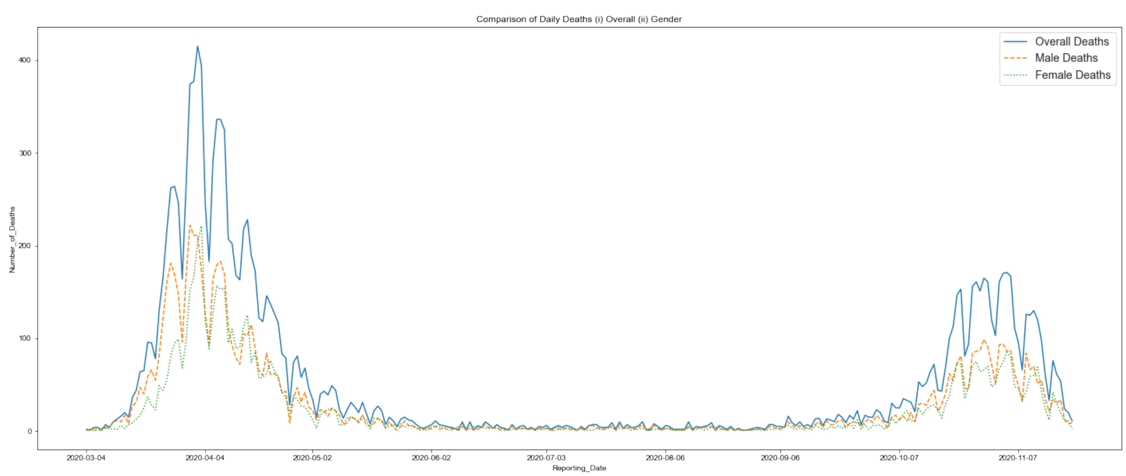
---

[39]:
```
# Filtering data for confirmed deaths

data_filtered_for_death_index = germany_covid_data['New_Death'] >= 0
data_filtered_for_death = germany_covid_data[data_filtered_for_death_index]

# Grouping data by Reporting_Date for Overall Deaths timeline

overall_deaths = data_filtered_for_death.
 ↪groupby('Reporting_Date')['Number_of_Deaths'].sum()
overall_deaths_dataframe = pd.DataFrame(overall_deaths).
 ↪rename_axis('Reporting_Date').reset_index()
overall_deaths_dataframe['Reporting_Date'] = pd.
 ↪to_datetime(overall_deaths_dataframe['Reporting_Date']).dt.
 ↪strftime('%Y-%m-%d')

# Filtering Data for confirmed deaths (Male)

male_deaths_index = data_filtered_for_death['Sex'] == 'M'
male_deaths = data_filtered_for_death[male_deaths_index]

# Grouping data by Reporting_Date for Male Deaths timeline

male_deaths_filtered = male_deaths.
 ↪groupby('Reporting_Date')['Number_of_Deaths'].sum()
male_deaths_dataframe = pd.DataFrame(male_deaths_filtered).
 ↪rename_axis('Reporting_Date').reset_index()
male_deaths_dataframe['Reporting_Date'] = pd.
 ↪to_datetime(male_deaths_dataframe['Reporting_Date']).dt.strftime('%Y-%m-%d')

# Filtering Data for confirmed deaths (Female)

female_deaths_index = data_filtered_for_death['Sex'] == 'W'
female_deaths = data_filtered_for_death[female_deaths_index]

# Grouping data by Reporting_Date for Female Deaths timeline
```

```
female_deaths_filtered = female_deaths.
 ↪groupby('Reporting_Date')['Number_of_Deaths'].sum()
female_deaths_dataframe = pd.DataFrame(female_deaths_filtered).
 ↪rename_axis('Reporting_Date').reset_index()
female_deaths_dataframe['Reporting_Date'] = pd.
 ↪to_datetime(female_deaths_dataframe['Reporting_Date']).dt.
 ↪strftime('%Y-%m-%d')
```

[40]:

```
# Plotting all the three timelines of Overall, Male and Female simultaneously
# Formatting the axes so that the dates are readible
# Resizing the labels for ease of view

plt.figure(figsize=(25,10))
plt.title('Comparison of Daily Deaths (i) Overall (ii) Gender')
sns.set_style("whitegrid", {'axes.grid' : False})
sns.lineplot(x="Reporting_Date", y="Number_of_Deaths",␣
 ↪data=overall_deaths_dataframe, label="Overall Deaths", linestyle="solid")
sns.lineplot(x="Reporting_Date", y="Number_of_Deaths",␣
 ↪data=male_deaths_dataframe, label="Male Deaths", linestyle="dashed")
ax = sns.lineplot(x="Reporting_Date", y="Number_of_Deaths",␣
 ↪data=female_deaths_dataframe, label="Female Deaths", linestyle="dotted")

xtick_locator = AutoDateLocator(maxticks=11)
ax.xaxis.set_major_locator(xtick_locator)

plt.legend(fontsize=20) # using a size in points
plt.legend(fontsize="x-large") # using a named size
plt.show()
```



Task 4 [50 points] EDA Following the Titanic example from the lecture, we want to gain first insights into multivariate EDA. For this purpose, take the following steps using python to answer the question **whether COVID-19 is more lethal to men than to women in Germany and whether there exists a difference between different age groups per gender.**.

1. Create two contingency tables of **total death and survived counts per gender** (one containing the absolute counts and the second one containing row and column proportions). *Note that currently infected people do not have any dead/survived outcome yet, so ignore the respective entries in the database.*
2. Create another two contingency tables of **total death and survived counts per age group** (one containing the absolute counts and the second one containing row and column proportions).
3. Plot a histogram or bar chart that shows the total number of people who have died from COVID-19 to date **by age group and gender**. *Hint: The usage of different colors might help a lot!*
4. Now combine the contingency tables of task 1. and 2. (see Titantic example on slide 27 of the EDA lecture), so that you have a subdivision into gender categories by age group, with absolute counts and row/column proportions.
5. Calculate the expected frequencies f_e for each conjunct event in the contingency table from task 4 and create a new table with the proportions in the table from task 4 being replaced by the f_e values.
6. Calculate ${\chi^2}_{male}$ and ${\chi^2}_{female}$.
7. What does a small χ^2 value mean? What if it's zero? Explain.

1.13 —

1.13.1 Task 4

Part 1

> Create two contingency tables of total death and survived counts per gender (one containing the absolute counts and the second one containing row and column proportions). Note that currently infected people do not have any dead/survived outcome yet, so ignore the respective entries in the database.

Summary To create the **Contingency Table** for **Total Deaths and Survived by Gender**, - Firstly, we create a column **Status** to categorize, which column belongs to **Dead** and which to **Survived**. We apply this condition using a function

```
germany_covid_data['Status'] = germany_covid_data.apply(condition,
axis=1)
```

Function: `def condition(s): if s['New_Recovered'] >= 0: return "Survived" elif s['New_Death'] >= 0: return "Dead"`

- Secondly, we apply another condition, to calculate the exact count of the dead and survived. We calculate the **Sum of 'Number_of_Deaths' and 'Number_of_Recovered'**. This is done because:
  - Whenever the count of 'Number_of_Deaths' > 0, count of 'Number_of_Recovered' = 0

- Whenever the count of 'Number_of_Recovered' > 0, count of 'Number_of_Deaths' = 0
- Thus, instead of filtering the data for each column, we just calculate the sum as it does not affect the exact count.

`germany_covid_data['Count'] = germany_covid_data.apply(condition2, axis=1)`

Function: `def condition2(s): if s['Number_of_Deaths'] >= 0 and s['Number_of_Recovered'] >= 0: return s['Number_of_Deaths'] + s['Number_of_Recovered'] elif s['Number_of_Deaths'] >= 0: return s['Number_of_Deaths'] else: return s['Number_of_Recovered']`

- Finally, we create the **Contingency Table** using `pd.crosstab()` from pandas.

  `first_contingency_table = pd.crosstab(germany_covid_data['Sex'], germany_covid_data['Status'], values=germany_covid_data['Count'], aggfunc='sum', margins=True, margins_name="Total")`

Now, to create **Contingency Table** with **Row and Column Proportions**, we just use the attribute `normalize='all'` which creates the proportions and then just format it using `.applymap(lambda x: "{0:.2f}%".format(100*x))` to format it to percentage.

`second_contingency_table = pd.crosstab(germany_covid_data['Sex'], germany_covid_data['Status'], values=germany_covid_data['Count'], aggfunc='sum', margins=True, margins_name="Total", normalize='all').applymap(lambda x: "{0:.2f}%".format(100*x))`

1.14 —

[41]:
```python
# Method to check condition to categorize, which value belongs to Dead and␣
↪which belongs to Survived

def condition(s):
    if s['New_Recovered'] >= 0:
        return "Survived"
    elif s['New_Death'] >= 0:
        return "Dead"

# Method to check condition based on the category we just assigned,
# Calculating the sum of 'Number_of_Deaths' and 'Number_of_Recovered'


def condition2(s):
    if s['Number_of_Deaths'] >= 0 and s['Number_of_Recovered'] >= 0:
        return s['Number_of_Deaths'] + s['Number_of_Recovered']
    elif s['Number_of_Deaths'] >= 0:
        return s['Number_of_Deaths']
    else:
        return s['Number_of_Recovered']
```

```
germany_covid_data['Status'] = germany_covid_data.apply(condition, axis=1)
germany_covid_data['Count'] = germany_covid_data.apply(condition2, axis=1)
first_contingency_table = pd.crosstab(germany_covid_data['Sex'],␣
↪germany_covid_data['Status'], values=germany_covid_data['Count'],␣
↪aggfunc='sum', margins=True, margins_name="Total")
```

[42]:
```
first_contingency_table
```

[42]:

Status	Dead	Survived	Total
Sex			
M	7765	298160	305925
W	6244	302949	309193
unbekannt	13	2691	2704
Total	14022	603800	617822

[43]:
```
# Creating contingency table with row and column proportions using␣
↪"normalize='all'" attribute

second_contingency_table = pd.crosstab(germany_covid_data['Sex'],␣
↪germany_covid_data['Status'], values=germany_covid_data['Count'],␣
↪aggfunc='sum', margins=True, margins_name="Total", normalize='all').
↪applymap(lambda x: "{0:.2f}%".format(100*x))
second_contingency_table
```

[43]:

Status	Dead	Survived	Total
Sex			
M	1.26%	48.26%	49.52%
W	1.01%	49.03%	50.05%
unbekannt	0.00%	0.44%	0.44%
Total	2.27%	97.73%	100.00%

1.15 —

Part 2

> Create another two contingency tables of total death and survived counts per age group (one containing the absolute counts and the second one containing row and column proportions).

Summary To create the **Contingency Table** for **Total Deaths and Survived by Age Group**, - Firstly, we reuse the data calulated for gender based contingency table. - Finally, we create the **Contingency Table** using `pd.crosstab()` from pandas.

```
first_age_group_contingency_table = pd.crosstab(germany_covid_data['Age_Group'],
germany_covid_data['Status'], values=germany_covid_data['Count'],
aggfunc='sum', margins=True, margins_name="Total")
```

Now, to create **Contingency Table** with **Row and Column Proportions**, we just use the attribute `normalize='all'` which creates the proportions and then just format it using `.applymap(lambda x: "{0:.2f}%".format(100*x))` to format it to percentage.

```
second_age_group_contingency_table = pd.crosstab(germany_covid_data['Age_Group'],
germany_covid_data['Status'], values=germany_covid_data['Count'],
aggfunc='sum', margins=True, margins_name="Total",
normalize='all').applymap(lambda x: "{0:.2f}%".format(100*x))
```

1.16 —

[44]:

```python
# Creating contingency table with count

first_age_group_contingency_table = pd.
↪crosstab(germany_covid_data['Age_Group'], germany_covid_data['Status'],␣
↪values=germany_covid_data['Count'], aggfunc='sum', margins=True,␣
↪margins_name="Total")
first_age_group_contingency_table
```

[44]:

Status Age_Group	Dead	Survived	Total
A00-A04	4	11859	11863
A05-A14	1	35246	35247
A15-A34	31	206155	206186
A35-A59	626	238848	239474
A60-A79	4273	80601	84874
A80+	9084	30173	39257
unbekannt	3	918	921
Total	14022	603800	617822

[45]:

```python
# Creating contingency table with row and column proportions using␣
↪"normalize='all'" attribute

second_age_group_contingency_table = pd.
↪crosstab(germany_covid_data['Age_Group'], germany_covid_data['Status'],␣
↪values=germany_covid_data['Count'], aggfunc='sum', margins=True,␣
↪margins_name="Total", normalize='all').applymap(lambda x: "{0:.2f}%".
↪format(100*x))
second_age_group_contingency_table
```

[45]:

Status Age_Group	Dead	Survived	Total
A00-A04	0.00%	1.92%	1.92%
A05-A14	0.00%	5.70%	5.71%
A15-A34	0.01%	33.37%	33.37%
A35-A59	0.10%	38.66%	38.76%
A60-A79	0.69%	13.05%	13.74%
A80+	1.47%	4.88%	6.35%

```
unbekannt  0.00%    0.15%    0.15%
Total      2.27%   97.73%  100.00%
```

1.17 —

Part 3

> Plot a histogram or bar chart that shows the total number of people who have died from COVID-19 to date by age group and gender. Hint: The usage of different colors might help a lot!

Summary Plot a histogram or bar chart that shows the total number of people who have died from COVID-19 to date by age group and gender. Hint: The usage of different colors might help a lot!

- Calculating **Total Deaths** based on **Gender** and **Age Group**, we
  - Firstly, filter out data for deaths based on index which match the condition `germany_covid_data['New_Death'] >= 0` `total_dead_index = germany_covid_data['New_Death'] >= 0`
  - Then, just group the data based on 'Sex' and 'Age_Group' `total_dead_by_gender = total_dead.groupby('Sex')['Number_of_Deaths'].sum()` `total_dead_by_age_group = total_dead.groupby('Age_Group')['Number_of_Deaths'].sum()`

1.18 —

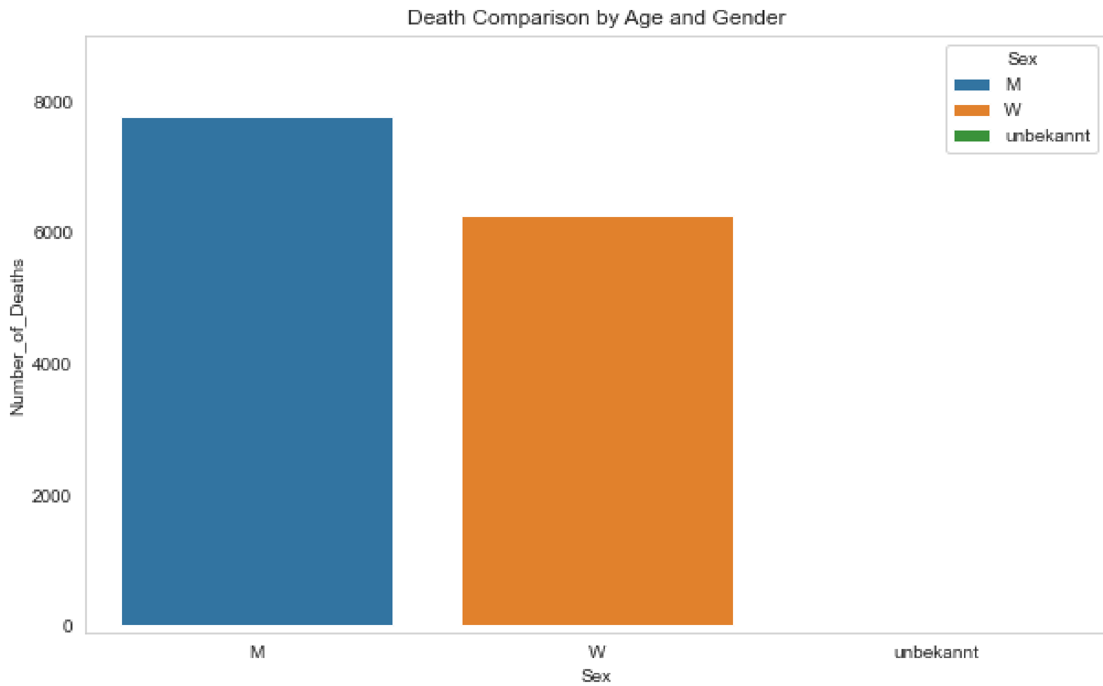
[46]:
```python
# Filtering data only for valid deaths based on index

total_dead_index = germany_covid_data['New_Death'] >= 0
total_dead = germany_covid_data[total_dead_index]

# Grouping data for Gender and Age Group

total_dead_by_gender = total_dead.groupby('Sex')['Number_of_Deaths'].sum()
total_dead_by_age_group = total_dead.groupby('Age_Group')['Number_of_Deaths'].
 ↪sum()
```

[47]:
```python
# Reformatting the Datafram for correct plotting

total_dead_by_gender = total_dead_by_gender.rename_axis('Sex').reset_index()
total_dead_by_age_group = total_dead_by_age_group.rename_axis('Age_Group').
 ↪reset_index()
```

[65]:
```python
# Plotting the data based on Gender

plt.figure(figsize=(10,6))
plt.title('Death Comparison by Age and Gender')
sns.barplot(x="Sex", y="Number_of_Deaths", data=total_dead_by_gender,␣
 ↪hue='Sex', dodge=False)
```

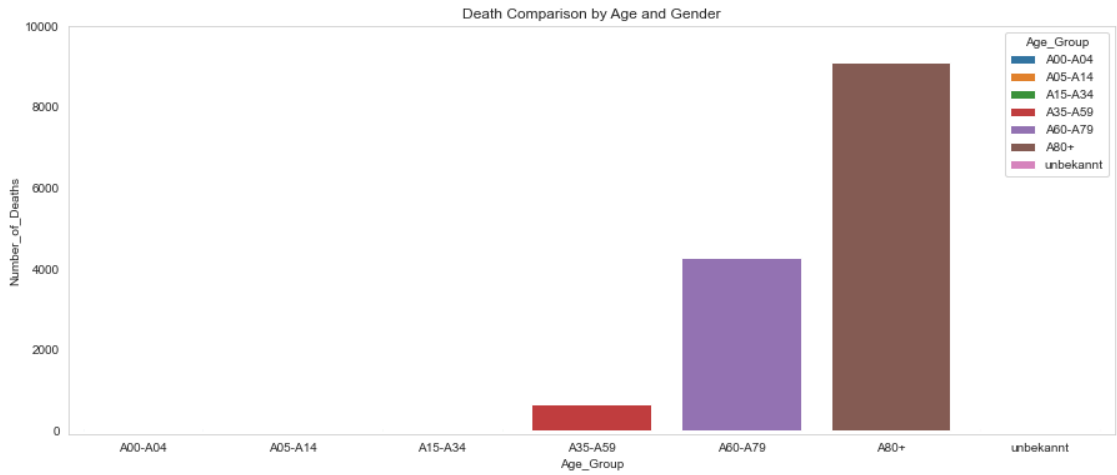
```
plt.ylim(-100,9000)
```

[65]: (-100.0, 9000.0)



[64]:
```
# Plotting Data based on Age Group
plt.figure(figsize=(15,6))
plt.title('Death Comparison by Age and Gender')
sns.barplot(x="Age_Group", y="Number_of_Deaths", data=total_dead_by_age_group,␣
↪hue="Age_Group", dodge=False)
plt.ylim(-100,10000)
```

[64]: (-100.0, 10000.0)



1.19 —

Part 4

> Now combine the contingency tables of task 1. and 2. (see Titantic example on slide 27 of the EDA lecture), so that you have a subdivision into gender categories by age group, with absolute counts and row/column proportions.

Summary In this task, we are just combining the the two contingency tables we created in previous tasks

1) Contingency Table with count

```
combined_contingency_table = pd.crosstab([germany_covid_data['Sex'],germany_covid_data['Ag
germany_covid_data['Status'], values=germany_covid_data['Count'],
aggfunc='sum', margins=True, margins_name="Total")
```

2) Contingency Table with percentage

```
combined_proprtions_contingency_table = pd.crosstab([germany_covid_data['Sex'],germany_cov
germany_covid_data['Status'], values=germany_covid_data['Count'],
aggfunc='sum', margins=True, margins_name="Total",
normalize='all').applymap(lambda x: "{0:.2f}%".format(100*x))
```

3) Contingency Table with proportions

```
combined_proprtions_contingency_table = pd.crosstab([germany_covid_data['Sex'],germany_cov
germany_covid_data['Status'], values=germany_covid_data['Count'],
aggfunc='sum', margins=True, margins_name="Total", normalize='all')
```

4) Cleaning the Contingency Table, by removing NaN values for better calculations and ease of use by using function as a condition

   Funtion: `def apply_value(value): if np.isnan(value): return 0 else: return value`

```
combined_contingency_table = pd.crosstab([germany_covid_data['Sex'],germany_covid_data['Ag
germany_covid_data['Status'], values=germany_covid_data['Count'],
aggfunc='sum', margins=True, margins_name="Total").apply(lambda x:
x.apply(lambda d: apply_value(d)), axis=1)
```

1.20 —

[50]:
```
# Contingency Table with count

combined_contingency_table = pd.
 ↪crosstab([germany_covid_data['Sex'],germany_covid_data['Age_Group']],␣
 ↪germany_covid_data['Status'], values=germany_covid_data['Count'],␣
 ↪aggfunc='sum', margins=True, margins_name="Total")
combined_contingency_table
```

[50]:
```
Status                   Dead  Survived   Total
Sex       Age_Group
M         A00-A04         1.0    6182.0    6183
          A05-A14         1.0   18093.0   18094
          A15-A34        22.0  105979.0  106001
          A35-A59       448.0  117276.0  117724
          A60-A79      2940.0   40239.0   43179
          A80+         4353.0    9976.0   14329
          unbekannt       NaN     415.0     415
W         A00-A04         2.0    5574.0    5576
          A05-A14         NaN   16910.0   16910
          A15-A34         9.0   99210.0   99219
          A35-A59       177.0  120628.0  120805
          A60-A79      1327.0   40151.0   41478
          A80+         4726.0   20144.0   24870
          unbekannt       3.0     332.0     335
unbekannt A00-A04         1.0     103.0     104
          A05-A14         NaN     243.0     243
          A15-A34         NaN     966.0     966
          A35-A59         1.0     944.0     945
          A60-A79         6.0     211.0     217
          A80+            5.0      53.0      58
          unbekannt       NaN     171.0     171
Total                 14022.0  603800.0  617822
```

[51]:
```
# Contingency Table with percentage

combined_proprtions_contingency_table = pd.
 ↪crosstab([germany_covid_data['Sex'],germany_covid_data['Age_Group']],␣
 ↪germany_covid_data['Status'], values=germany_covid_data['Count'],␣
 ↪aggfunc='sum', margins=True, margins_name="Total", normalize='all').
 ↪applymap(lambda x: "{0:.2f}%".format(100*x))
```

```
combined_proprtions_contingency_table
```

[51]:

```
Status                  Dead Survived    Total
Sex       Age_Group
M         A00-A04      0.00%    1.00%    1.00%
          A05-A14      0.00%    2.93%    2.93%
          A15-A34      0.00%   17.15%   17.16%
          A35-A59      0.07%   18.98%   19.05%
          A60-A79      0.48%    6.51%    6.99%
          A80+         0.70%    1.61%    2.32%
          unbekannt    0.00%    0.07%    0.07%
W         A00-A04      0.00%    0.90%    0.90%
          A05-A14      0.00%    2.74%    2.74%
          A15-A34      0.00%   16.06%   16.06%
          A35-A59      0.03%   19.52%   19.55%
          A60-A79      0.21%    6.50%    6.71%
          A80+         0.76%    3.26%    4.03%
          unbekannt    0.00%    0.05%    0.05%
unbekannt A00-A04      0.00%    0.02%    0.02%
          A05-A14      0.00%    0.04%    0.04%
          A15-A34      0.00%    0.16%    0.16%
          A35-A59      0.00%    0.15%    0.15%
          A60-A79      0.00%    0.03%    0.04%
          A80+         0.00%    0.01%    0.01%
          unbekannt    0.00%    0.03%    0.03%
Total                  2.27%   97.73%  100.00%
```

[52]:

```
# Contingency Table with Row and Column Proportions

combined_proprtions_contingency_table = pd.
 ↪crosstab([germany_covid_data['Sex'],germany_covid_data['Age_Group']],␣
 ↪germany_covid_data['Status'], values=germany_covid_data['Count'],␣
 ↪aggfunc='sum', margins=True, margins_name="Total", normalize='all')
combined_proprtions_contingency_table
```

[52]:

```
Status                     Dead  Survived     Total
Sex       Age_Group
M         A00-A04      0.000002  0.010006  0.010008
          A05-A14      0.000002  0.029285  0.029287
          A15-A34      0.000036  0.171536  0.171572
          A35-A59      0.000725  0.189822  0.190547
          A60-A79      0.004759  0.065130  0.069889
          A80+         0.007046  0.016147  0.023193
          unbekannt    0.000000  0.000672  0.000672
W         A00-A04      0.000003  0.009022  0.009025
          A05-A14      0.000000  0.027370  0.027370
          A15-A34      0.000015  0.160580  0.160595
```

```
            A35-A59    0.000286  0.195247  0.195534
            A60-A79    0.002148  0.064988  0.067136
            A80+       0.007649  0.032605  0.040254
            unbekannt  0.000005  0.000537  0.000542
unbekannt   A00-A04    0.000002  0.000167  0.000168
            A05-A14    0.000000  0.000393  0.000393
            A15-A34    0.000000  0.001564  0.001564
            A35-A59    0.000002  0.001528  0.001530
            A60-A79    0.000010  0.000342  0.000351
            A80+       0.000008  0.000086  0.000094
            unbekannt  0.000000  0.000277  0.000277
Total                  0.022696  0.977304  1.000000
```

[53]:

```
# Formatting data by replacing NaN with 0 for ease of use

def apply_value(value):
    if np.isnan(value):
        return 0
    else:
        return value
combined_contingency_table = pd.
 ↪crosstab([germany_covid_data['Sex'],germany_covid_data['Age_Group']],␣
 ↪germany_covid_data['Status'], values=germany_covid_data['Count'],␣
 ↪aggfunc='sum', margins=True, margins_name="Total").apply(lambda x: x.
 ↪apply(lambda d: apply_value(d)), axis=1)
combined_contingency_table
```

[53]:

```
Status                 Dead  Survived     Total
Sex        Age_Group
M          A00-A04      1.0    6182.0    6183.0
           A05-A14      1.0   18093.0   18094.0
           A15-A34     22.0  105979.0  106001.0
           A35-A59    448.0  117276.0  117724.0
           A60-A79   2940.0   40239.0   43179.0
           A80+      4353.0    9976.0   14329.0
           unbekannt    0.0     415.0     415.0
W          A00-A04      2.0    5574.0    5576.0
           A05-A14      0.0   16910.0   16910.0
           A15-A34      9.0   99210.0   99219.0
           A35-A59    177.0  120628.0  120805.0
           A60-A79   1327.0   40151.0   41478.0
           A80+      4726.0   20144.0   24870.0
           unbekannt    3.0     332.0     335.0
unbekannt  A00-A04      1.0     103.0     104.0
           A05-A14      0.0     243.0     243.0
           A15-A34      0.0     966.0     966.0
           A35-A59      1.0     944.0     945.0
```

```
          A60-A79         6.0     211.0     217.0
          A80+            5.0      53.0      58.0
          unbekannt       0.0     171.0     171.0
Total                 14022.0  603800.0  617822.0
```

1.21 —

Part 5

Calculate the expected frequencies for each conjunct event in the contingency table from task 4 and create a new table with the proportions in the table from task 4 being replaced by the values.

Summary

- In this task, to get the **Expected Frequencies ( )**, we use the scipy method `chi2_contingency()` to get the complete statistics of the data along with the expected frequencies using **Contingency Table with Count**.

  `stat, p, dof, expected = chi2_contingency(combined_contingency_table)`

- Now, we replace the contingency table with the expected values.

  Function: `def apply_value(value, key_index): global iteration global loop desired_value = expected[iteration][loop] loop = loop + 1 if loop == 3: loop = 0 iteration = iteration + 1 return desired_value`

  `combined_expected_contingency_table = pd.crosstab([germany_covid_data['Sex'],germany_c germany_covid_data['Status'], values=germany_covid_data['Count'], aggfunc='sum', margins=True, margins_name="Total").apply(lambda x: x.apply(lambda d: apply_value(d, x.loc[:])), axis=1)`

- Similarly, we follow the same steps to get the complete statistics of the data along with the expected frequencies using **Contingency Table with Proportions**.

  `stat, p, dof, expected = chi2_contingency(combined_proprtions_contingency_table)`

  Function: `def apply_value(value, key_index): global iteration global loop desired_value = expected[iteration][loop] loop = loop + 1 if loop == 3: loop = 0 iteration = iteration + 1 return desired_value`

  ## 1.22 `combined_expected_contingency_table = pd.crosstab([germany_covid_data['Sex'],germany_covid_data['Age_Group']], germany_covid_data['Status'], values=germany_covid_data['Count'], aggfunc='sum', margins=True, margins_name="Total", normalize='all').apply(lambda x: x.apply(lambda d: apply_value(d, x.loc[:])), axis=1)`

---

[54]:
```
# Using scipy method to fetch the expected frquencies
```

```
stat, p, dof, expected = chi2_contingency(combined_contingency_table)
expected
```

[54]:

```
array([[1.40328486e+02, 6.04267151e+03, 6.18300000e+03],
       [4.10658843e+02, 1.76833412e+04, 1.80940000e+04],
       [2.40578358e+03, 1.03595216e+05, 1.06001000e+05],
       [2.67184711e+03, 1.15052153e+05, 1.17724000e+05],
       [9.79984426e+02, 4.21990156e+04, 4.31790000e+04],
       [3.25208940e+02, 1.40037911e+04, 1.43290000e+04],
       [9.41878081e+00, 4.05581219e+02, 4.15000000e+02],
       [1.26552101e+02, 5.44944790e+03, 5.57600000e+03],
       [3.83786948e+02, 1.65262131e+04, 1.69100000e+04],
       [2.25186027e+03, 9.69671397e+04, 9.92190000e+04],
       [2.74177305e+03, 1.18063227e+05, 1.20805000e+05],
       [9.41378773e+02, 4.05366212e+04, 4.14780000e+04],
       [5.64445973e+02, 2.43055540e+04, 2.48700000e+04],
       [7.60311222e+00, 3.27396888e+02, 3.35000000e+02],
       [2.36036917e+00, 1.01639631e+02, 1.04000000e+02],
       [5.51509334e+00, 2.37484907e+02, 2.43000000e+02],
       [2.19241982e+01, 9.44075802e+02, 9.66000000e+02],
       [2.14475852e+01, 9.23552415e+02, 9.45000000e+02],
       [4.92500105e+00, 2.12074999e+02, 2.17000000e+02],
       [1.31635973e+00, 5.66836403e+01, 5.80000000e+01],
       [3.88099161e+00, 1.67119008e+02, 1.71000000e+02],
       [1.40220000e+04, 6.03800000e+05, 6.17822000e+05]])
```

[55]:

```
#  Replacing the currecnt contingency table with the expected values

iteration = 0
loop = 0

def apply_value(value, key_index):
    global iteration
    global loop
    desired_value = expected[iteration][loop]
    loop = loop + 1
    if loop == 3:
        loop = 0
        iteration = iteration + 1
    return desired_value


combined_expected_contingency_table = pd.
 ↪crosstab([germany_covid_data['Sex'],germany_covid_data['Age_Group']],␣
 ↪germany_covid_data['Status'], values=germany_covid_data['Count'],␣
 ↪aggfunc='sum', margins=True, margins_name="Total").apply(lambda x: x.
 ↪apply(lambda d: apply_value(d, x.loc[:])), axis=1)
```

```
combined_expected_contingency_table
```

[55]:
```
Status                      Dead       Survived     Total
Sex       Age_Group
M         A00-A04     140.328486    6042.671514    6183.0
          A05-A14     410.658843   17683.341157   18094.0
          A15-A34    2405.783578  103595.216422  106001.0
          A35-A59    2671.847115  115052.152885  117724.0
          A60-A79     979.984426   42199.015574   43179.0
          A80+        325.208940   14003.791060   14329.0
          unbekannt     9.418781     405.581219     415.0
W         A00-A04     126.552101    5449.447899    5576.0
          A05-A14     383.786948   16526.213052   16910.0
          A15-A34    2251.860274   96967.139726   99219.0
          A35-A59    2741.773051  118063.226949  120805.0
          A60-A79     941.378773   40536.621227   41478.0
          A80+        564.445973   24305.554027   24870.0
          unbekannt     7.603112     327.396888     335.0
unbekannt A00-A04       2.360369     101.639631     104.0
          A05-A14       5.515093     237.484907     243.0
          A15-A34      21.924198     944.075802     966.0
          A35-A59      21.447585     923.552415     945.0
          A60-A79       4.925001     212.074999     217.0
          A80+          1.316360      56.683640      58.0
          unbekannt     3.880992     167.119008     171.0
Total               14022.000000  603800.000000  617822.0
```

[56]:
```
# Using scipy method to fetch the expected frquencies

stat, p, dof, expected = chi2_contingency(combined_proprtions_contingency_table)
expected
```

[56]:
```
array([[2.27134168e-04, 9.78060269e-03, 1.00077369e-02],
       [6.64687958e-04, 2.86220645e-02, 2.92867525e-02],
       [3.89397525e-03, 1.67678096e-01, 1.71572071e-01],
       [4.32462281e-03, 1.86222169e-01, 1.90546792e-01],
       [1.58619218e-03, 6.83028697e-02, 6.98890619e-02],
       [5.26379670e-04, 2.26663846e-02, 2.31927643e-02],
       [1.52451366e-05, 6.56469370e-04, 6.71714507e-04],
       [2.04835860e-04, 8.82041737e-03, 9.02525323e-03],
       [6.21193399e-04, 2.67491495e-02, 2.73703429e-02],
       [3.64483666e-03, 1.56949962e-01, 1.60594799e-01],
       [4.43780418e-03, 1.91095861e-01, 1.95533665e-01],
       [1.52370549e-03, 6.56121362e-02, 6.71358417e-02],
       [9.13606141e-04, 3.93407066e-02, 4.02543127e-02],
       [1.23063151e-05, 5.29921058e-04, 5.42227373e-04],
       [3.82046798e-06, 1.64512806e-04, 1.68333274e-04],
```

```
        [8.92667037e-06, 3.84390499e-04, 3.93317169e-04],
        [3.54862699e-05, 1.52807087e-03, 1.56355714e-03],
        [3.47148292e-05, 1.49485194e-03, 1.52956677e-03],
        [7.97155338e-06, 3.43262297e-04, 3.51233851e-04],
        [2.13064560e-06, 9.17475264e-05, 9.38781720e-05],
        [6.28173100e-06, 2.70497018e-04, 2.76778749e-04],
        [2.26958574e-02, 9.77304143e-01, 1.00000000e+00]])
```

[57]:

```
#  Replacing the currecnt contingency table with the expected values

iteration = 0
loop = 0

def apply_value(value, key_index):
    global iteration
    global loop
    desired_value = expected[iteration][loop]
    loop = loop + 1
    if loop == 3:
        loop = 0
        iteration = iteration + 1
    return desired_value


combined_expected_contingency_table = pd.
 ↪crosstab([germany_covid_data['Sex'],germany_covid_data['Age_Group']],␣
 ↪germany_covid_data['Status'], values=germany_covid_data['Count'],␣
 ↪aggfunc='sum', margins=True, margins_name="Total", normalize='all').
 ↪apply(lambda x: x.apply(lambda d: apply_value(d, x.loc[:])), axis=1)
combined_expected_contingency_table
```

[57]:

Status		Dead	Survived	Total
Sex	Age_Group			
M	A00-A04	0.000227	0.009781	0.010008
	A05-A14	0.000665	0.028622	0.029287
	A15-A34	0.003894	0.167678	0.171572
	A35-A59	0.004325	0.186222	0.190547
	A60-A79	0.001586	0.068303	0.069889
	A80+	0.000526	0.022666	0.023193
	unbekannt	0.000015	0.000656	0.000672
W	A00-A04	0.000205	0.008820	0.009025
	A05-A14	0.000621	0.026749	0.027370
	A15-A34	0.003645	0.156950	0.160595
	A35-A59	0.004438	0.191096	0.195534
	A60-A79	0.001524	0.065612	0.067136
	A80+	0.000914	0.039341	0.040254
	unbekannt	0.000012	0.000530	0.000542

```
unbekannt A00-A04    0.000004  0.000165  0.000168
          A05-A14    0.000009  0.000384  0.000393
          A15-A34    0.000035  0.001528  0.001564
          A35-A59    0.000035  0.001495  0.001530
          A60-A79    0.000008  0.000343  0.000351
          A80+       0.000002  0.000092  0.000094
          unbekannt  0.000006  0.000270  0.000277
Total                0.022696  0.977304  1.000000
```

1.23 —

Part 6

Calculate ^{2} and ^{2} .

Summary To find out the ^{2} for **Male** and **Female** buy finding out the **Squared Difference** between **Observed** and **Expected** values.

```
chi2 = (((combined_contingency_table - combined_expected_contingency_table)**2)/combined_e
```

1.24 —

[58]:
```python
# Calculating the Chi2 value

chi2 = (((combined_contingency_table - combined_expected_contingency_table)**2)/
 ↪combined_expected_contingency_table).sum()
chi2
```

[58]:
```
Status
Dead         7.578407e+10
Survived     7.474370e+11
Total        7.634056e+11
dtype: float64
```

[59]:
```python
# Re-formatting the table for finding out the expected Chi2

def apply_value(value):
    if np.isnan(value):
        return 0
    else:
        return value

combined_contingency_table = pd.crosstab(germany_covid_data['Status'],␣
 ↪germany_covid_data['Sex'], values=germany_covid_data['Count'],␣
 ↪aggfunc='sum', margins=True, margins_name="Total").apply(lambda x: x.
 ↪apply(lambda d: apply_value(d)), axis=1)
combined_contingency_table
```

[59]:
```
Sex              M       W  unbekannt   Total
Status
Dead          7765    6244         13   14022
Survived    298160  302949       2691  603800
Total       305925  309193       2704  617822
```

[60]:
```
stat, p, dof, expected = chi2_contingency(combined_contingency_table)
expected
```

[60]:
```
array([[6.94323017e+03, 7.01740023e+03, 6.13695984e+01, 1.40220000e+04],
       [2.98981770e+05, 3.02175600e+05, 2.64263040e+03, 6.03800000e+05],
       [3.05925000e+05, 3.09193000e+05, 2.70400000e+03, 6.17822000e+05]])
```

[61]:
```
# Re-calculating the contingency table

iteration = 0
loop = 0

def apply_value(value, key_index):
    global iteration
    global loop
    desired_value = expected[iteration][loop]
    loop = loop + 1
    if loop == 4:
        loop = 0
        iteration = iteration + 1
    return desired_value


combined_expected_contingency_table = pd.crosstab(␣
 ↪germany_covid_data['Status'], germany_covid_data['Sex'],␣
 ↪values=germany_covid_data['Count'], aggfunc='sum', margins=True,␣
 ↪margins_name="Total").apply(lambda x: x.apply(lambda d: apply_value(d, x.
 ↪loc[:])), axis=1)
combined_expected_contingency_table
```

[61]:
```
Sex                    M              W    unbekannt     Total
Status
Dead          6943.23017    7017.400232    61.369598   14022.0
Survived    298981.76983  302175.599768  2642.630402  603800.0
Total       305925.00000  309193.000000  2704.000000  617822.0
```

[62]:
```
# Calculating the Chi2 value

chi2 = (((combined_contingency_table - combined_expected_contingency_table)**2)/
 ↪combined_expected_contingency_table).sum()
chi2
```

[62]:

```
Sex
M            99.519706
W            87.217294
unbekannt    39.008742
Total         0.000000
dtype: float64
```

1.25 —

Part 7

> What does a small ^{2} value mean? What if it's zero? Explain.

Summary

1) Small χ^2 means that the observed data fits with expected data extremely well
2) If χ^2 equals to 0, this means that the observed and expected data are identical. This can also mean that the model is overfitting.

1.26 —

[ ]:

```

```