Datascience Corona Statistic

Datascience Project by Yasin Sahin and Sven Oberwalder

Table of Contents

Covid Status

This Data Science project by Sven Oberwalder and Yasin Sahin analyzes the development of the corona pandemic based on input data provided by Statistik Austria

The Datasets can be found here:

Dataset 1

Dataset 2

Additional Info

This project was version controlled by GitHub and can be accessed through the public repo here

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import ListedColormap
```

Initial Data Analysis and Import

Dataset 1 has following attributes:

- **C-BEZIMST-0** Politischer Bezirk (PolBez)
- **C-ALTGRIMST-0** 10-years age group (Altersgr)
- **C-GLIMST-0** Country of Birth (GebLand)
- **C-C11-0** Sex (Geschl)
- **C-IMST-0** COVID-19 vaccinated-recovered-status (GeimpftGenesen)
- **F-DATA** Number of records (Anz)

Dataset 2 has following attributes:

- **C-B00-0** Federal country (Bundesland)
- **C-BILIMST-0** Education (Bildung)
- **C-ALTGRIMST-0** 10-years age group (Altersgr)
- **C-ESIMST-0** Economic status (ErwerbStatus)
- **C-IMST-0** COVID-19 vaccinated-recovered-status (GeimpftGenesen)
- **F-DATA** Number of records (Anz)

In below code, the datasets need to be imported and the attributes must be renamed to more meaningful names. Since the data is separated by semi-colons (;), we need to specify that when importing the csv-files. Furthermore, a simple Std-Analysis has to be conducted, to ensure the progress.

In [2]:
```
#import datasets
dataset1 = pd.read_csv("./data/dataset1.csv", sep=";")
dataset2 = pd.read_csv("./data/dataset2.csv", sep=";")

#rename attributes
dataset1.rename(columns={"C-BEZIMST-0": "PolBez",
                         "C-ALTGRIMST-0": "Altersgr",
                         "C-GLIMST-0": "GebLand",
                         "C-C11-0": "Geschl",
                         "C-IMST-0": "GeimpftGenesen",
                         "F-DATA": "Anz"}, inplace=True)
dataset2.rename(columns={"C-B00-0": "Bundesland",
                         "C-BILIMST-0": "Bildung",
                         "C-ALTGRIMST-0": "Altersgr",
                         "C-ESIMST-0": "ErwerbStatus",
                         "C-IMST-0": "GeimpftGenesen",
                         "F-DATA": "Anz"}, inplace=True)
```

Std-Analysis for Dataset 1

In [3]:
```
dataset1.sample(5)
```

Out[3]:

	PolBez	Altersgr	GebLand	Geschl	GeimpftGenesen	Anz
22946	BEZIMST-412	ALTGRIMST-3	GLIMST-2	C11-2	IMST-3	51
28589	BEZIMST-601	ALTGRIMST-3	GLIMST-2	C11-1	IMST-1	148
41287	BEZIMST-904	ALTGRIMST-9	GLIMST-2	C11-2	IMST-2	145
17993	BEZIMST-325	ALT10IMST-1	GLIMST-2	C11-1	IMST-4	2
3763	BEZIMST-201	ALT10IMST-3	GLIMST-2	C11-1	IMST-4	317

We immediately recognize that all values have a certain code (Except for Anz)

In [4]:
```
dataset1.head(5)
```

Out[4]:

	PolBez	Altersgr	GebLand	Geschl	GeimpftGenesen	Anz
0	BEZIMST-101	ALTGRIMST-1	GLIMST-1	C11-1	IMST-1	1
1	BEZIMST-101	ALTGRIMST-1	GLIMST-1	C11-1	IMST-3	80
2	BEZIMST-101	ALTGRIMST-1	GLIMST-1	C11-1	IMST-4	216
3	BEZIMST-101	ALTGRIMST-1	GLIMST-1	C11-2	IMST-1	2
4	BEZIMST-101	ALTGRIMST-1	GLIMST-1	C11-2	IMST-3	86

In [5]:
```
dataset1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53961 entries, 0 to 53960
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   PolBez          53961 non-null  object
 1   Altersgr        53961 non-null  object
 2   GebLand         53961 non-null  object
 3   Geschl          53961 non-null  object
 4   GeimpftGenesen  53961 non-null  object
 5   Anz             53961 non-null  int64 
dtypes: int64(1), object(5)
memory usage: 2.5+ MB
```

There are no null-values for this dataset, which is positive for future analysis.

In [6]:
```
dataset1.describe()
```

Out[6]:

	Anz
count	53961.000000
mean	642.049703
std	2171.148389
min	1.000000
25%	44.000000
50%	169.000000
75%	508.000000
max	70338.000000

Std-Analysis for Dataset 2

In [7]:
```
dataset2.sample(5)
```

Out[7]:

	Bundesland	Bildung	Altersgr	ErwerbStatus	GeimpftGenesen	Anz
1173	B00-8	BILIMST-3	ALT10IMST-6	ESIMST-2	IMST-2	98
1229	B00-8	BILIMST-99	ALT10IMST-5	ESIMST-2	IMST-2	3
705	B00-5	BILIMST-3	ALT10IMST-7	ESIMST-1	IMST-1	3609
81	B00-1	BILIMST-3	ALT10IMST-6	ESIMST-1	IMST-2	2025
1045	B00-7	BILIMST-4	ALT10IMST-6	ESIMST-1	IMST-3	1646

Same here: Every value is its own code and needs to be converted, to understand the dataset better.

In [8]:
```
dataset2.head(5)
```

Out[8]:

	Bundesland	Bildung	Altersgr	ErwerbStatus	GeimpftGenesen	Anz
0	B00-1	BILIMST-1	ALT10IMST-4	ESIMST-1	IMST-1	611
1	B00-1	BILIMST-1	ALT10IMST-4	ESIMST-1	IMST-2	285
2	B00-1	BILIMST-1	ALT10IMST-4	ESIMST-1	IMST-3	396
3	B00-1	BILIMST-1	ALT10IMST-4	ESIMST-1	IMST-4	472
4	B00-1	BILIMST-1	ALT10IMST-4	ESIMST-2	IMST-1	471

In [9]:
```
dataset2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1404 entries, 0 to 1403
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Bundesland      1404 non-null   object
 1   Bildung         1404 non-null   object
 2   Altersgr        1404 non-null   object
 3   ErwerbStatus    1404 non-null   object
 4   GeimpftGenesen  1404 non-null   object
 5   Anz             1404 non-null   int64 
dtypes: int64(1), object(5)
memory usage: 65.9+ KB
```

No null-values in this dataset too.

In [10]:
```
dataset2.describe()
```

Out[10]:

	Anz
count	1404.000000
mean	3414.423789
std	5746.310880
min	1.000000
25%	310.500000
50%	1375.500000
75%	3926.000000
max	55581.000000

Data Cleaning

In this section, the input data will be cleaned. For this manner, any wrong, missing, or irrelevant informations will be treated accordingly.

To Be marked The Column `Altersgr` uses an age gap interval of 10 years to inform us about the age of the adressed person. However, the first interval currently named `ALT10IMST-1` only has a 5 year gap (from 0 - 4 years) to especially adress babys and small children

Dataset 1

There is already a csv which helps us convert these codes into values we can understand. The reason we do not import these values from the csv directly, is that the values are too long. We hereby keep the values steady and simple.

- `GeimpftGenesen` is converted into more meaningful names
- For `Altersgr` we type in the exact age group
- For `GebLand` : `GLIMST-1` stands for people born in Austria, and `GLIMST-1` for people born outside of Austria

- `Geschl` is also being converted into m (male) and f (female)

In [11]:
```
bundeslandDict = {
    "B00-1": "Burgenland",
    "B00-2": "Kärnten",
    "B00-3": "Niederösterreich",
    "B00-4": "Oberösterreich",
    "B00-5": "Salzburg",
    "B00-6": "Steiermark",
    "B00-7": "Tirol",
    "B00-8": "Vorarlberg",
    "B00-9": "Wien"
}

geimpftGenesenDict = {"IMST-4": "NICHTS",  #nothing (NICHTS)
                       "IMST-1": "IMPF",  #vaccinated (geIMPFt)
                       "IMST-3": "GEN",  #recovered (GENesen)
                       "IMST-2": "IMPF + GEN"} #vaccinated + recovered (geIMPFt und

ageGapDict = {"ALT10IMST-1": "0 - 4",
              "ALT10IMST-2": "5 - 14",
              "ALT10IMST-3": "15 - 24",
              "ALT10IMST-4": "25 - 34",
              "ALT10IMST-5": "35 - 44",
              "ALT10IMST-6": "45 - 54",
              "ALT10IMST-7": "55 - 64",
              "ALT10IMST-8": "65 - 74",
              "ALT10IMST-9": "75 - 84",
              "ALT10IMST-10": "85+"}

dataset1["GebLand"] = dataset1["GebLand"].map({"GLIMST-1": "INLAND",
                                               "GLIMST-2": "AUSLAND"})

dataset1["Geschl"] = dataset1["Geschl"].map({"C11-1": "m",
                                             "C11-2": "f"})

dataset1["GeimpftGenesen"] = dataset1["GeimpftGenesen"].map(geimpftGenesenDict)

dataset1["Altersgr"] = dataset1["Altersgr"].map(ageGapDict)

dataset1.sample(10)
```

Out[11]:

	PolBez	Altersgr	GebLand	Geschl	GeimpftGenesen	Anz
1276	BEZIMST-104	85+	AUSLAND	m	IMPF	15
52618	B00-6	85+	INLAND	m	NICHTS	949
25388	BEZIMST-417	5 - 14	AUSLAND	f	IMPF	32
48053	BEZIMST-919	0 - 4	INLAND	m	IMPF	27
37148	BEZIMST-707	55 - 64	INLAND	f	NICHTS	622
34657	BEZIMST-702	NaN	AUSLAND	f	NICHTS	29
10794	BEZIMST-308	NaN	INLAND	f	IMPF + GEN	310
21580	BEZIMST-408	75 - 84	INLAND	f	IMPF	1717
7708	BEZIMST-210	75 - 84	INLAND	m	NICHTS	99
11632	BEZIMST-309	75 - 84	AUSLAND	f	NICHTS	5

Dataset 2

In [12]:
```
dataset2["Bundesland"] = dataset2["Bundesland"].map(bundeslandDict)

#uses the dictionary geimpftGenesenDict from the previous code block
dataset2["GeimpftGenesen"] = dataset2["GeimpftGenesen"].map(geimpftGenesenDict)

dataset2["Bildung"] = dataset2["Bildung"].map({"BILIMST-1": "Pflichtschule",
                                               "BILIMST-2": "Lehrabschluss/BMS",
                                               "BILIMST-3": "BHS/AHS/Kolleg",
                                               "BILIMST-4": "Akademie/Hochschule",
                                               "BILIMST-99": "Unknown"})

dataset2["ErwerbStatus"] = dataset2["ErwerbStatus"].map({"ESIMST-1": "aktiv",
                                                          "ESIMST-2": "inaktiv"})

#uses the dictionary ageGapDict from the previous code block
dataset2["Altersgr"] = dataset2["Altersgr"].map(ageGapDict)

dataset2.sample(10)
```

Out[12]:

	Bundesland	Bildung	Altersgr	ErwerbStatus	GeimpftGenesen	Anz
778	Steiermark	Pflichtschule	25 - 34	aktiv	NICHTS	3311
793	Steiermark	Pflichtschule	45 - 54	aktiv	GEN	2702
50	Burgenland	Lehrabschluss/BMS	45 - 54	aktiv	GEN	2350
1342	Wien	Akademie/Hochschule	25 - 34	aktiv	GEN	7803
606	Oberösterreich	Unknown	45 - 54	aktiv	IMPF + GEN	2
815	Steiermark	Lehrabschluss/BMS	35 - 44	aktiv	IMPF	25012
1013	Tirol	BHS/AHS/Kolleg	45 - 54	aktiv	GEN	1579
656	Salzburg	Lehrabschluss/BMS	25 - 34	inaktiv	NICHTS	1357
1107	Vorarlberg	Pflichtschule	45 - 54	aktiv	NICHTS	1556
152	Kärnten	Pflichtschule	25 - 34	aktiv	GEN	1254

Duplicate Data

In [13]:
```
dataset1["Anz"].sum()
```

Out[13]:
```
34645644
```

In the above code we can see that the sum of records is approximately 4 times the total population of Austria. This strongly suggests that there may be duplicate data. And exactly that is the case: In `PolBez` there are records for each province and also for each federal country. This duplicate is not necessary, since one province can easily be assigned to its federal country (e.g. `BEZIMST-304` (= Wiener Neustadt) must be in `B00-3` (=Niederoesterreich)). So the records for the federal countries can be removed.

In [14]:
```
dataset1 = dataset1.loc[dataset1["PolBez"].map(lambda val: "BEZIMST" in val)] #only
```

Instead, we can add a new feature providing information about the federal country.

In [15]:
```
dataset1["Bundesland"] = dataset1["PolBez"].map(lambda val: "B00-" + val[8]).map(bu

dataset1["Anz"].sum()
```

Out[15]:
```
17322822
```

The above code still returns an amount which is twice as big as Austria's population. Similarly, the column `Altersgr` has duplicate records for different age groups. Previously we mapped `Altersgr` to more readable strings, but left out the duplicate values that will be deleted in the next step. So we can simply drop the N/A records that emerge after mapping the column `Altersgr`.

In [16]:
```
dataset1 = dataset1.dropna()
dataset1["Anz"].sum()
```

Out[16]:
```
8661411
```

Now we have removed all duplicate values. Let's continue cleaning our data.

Null Values

In [17]:
```
dataset1.isnull().sum()
```

Out[17]:
```
PolBez            0
Altersgr          0
GebLand           0
Geschl            0
GeimpftGenesen    0
Anz               0
Bundesland        0
dtype: int64
```

In [18]:
```
dataset2.isnull().sum()
```

Out[18]:
```
Bundesland        0
Bildung           0
Altersgr          0
ErwerbStatus      0
GeimpftGenesen    0
Anz               0
dtype: int64
```

Since there are no null values, we are done cleaning our data and can save it into two datasets. `korr1.csv` for `dataset1.csv` and `korr2.csv` for `dataset2.csv`

In [19]:
```
dataset1.to_csv("./output/korr1.csv", sep=";", index=False)
dataset2.to_csv("./output/korr2.csv", sep=";", index=False)
```

Now, lets look at our corrected values...

In [20]:
```
korr1 = pd.read_csv("./output/korr1.csv", sep=";")
korr2 = pd.read_csv("./output/korr2.csv", sep=";")
```

```
korr1.head(5)
```

Out[20]:

	PolBez	Altersgr	GebLand	Geschl	GeimpftGenesen	Anz	Bundesland
0	BEZIMST-101	0 - 4	INLAND	m	IMPF	1	Burgenland
1	BEZIMST-101	0 - 4	INLAND	m	GEN	80	Burgenland
2	BEZIMST-101	0 - 4	INLAND	m	NICHTS	216	Burgenland
3	BEZIMST-101	0 - 4	INLAND	f	IMPF	2	Burgenland
4	BEZIMST-101	0 - 4	INLAND	f	GEN	86	Burgenland

In [21]:
```
korr2.head(5)
```

Out[21]:

	Bundesland	Bildung	Altersgr	ErwerbStatus	GeimpftGenesen	Anz
0	Burgenland	Pflichtschule	25 - 34	aktiv	IMPF	611
1	Burgenland	Pflichtschule	25 - 34	aktiv	IMPF + GEN	285
2	Burgenland	Pflichtschule	25 - 34	aktiv	GEN	396
3	Burgenland	Pflichtschule	25 - 34	aktiv	NICHTS	472
4	Burgenland	Pflichtschule	25 - 34	inaktiv	IMPF	471

Data Preparation

Numerical Values

Dataset 1

- For `Polbez` we take the number in the value code (so `BEZIMST-340` will be converted into `340`)
- For `Altersgr` we take the mean value from the min and max of the age group (except for 85, it will stay 85 since there is no max)
- For `GebLand` we take 0 for `INLAND` and 1 for `AUSLAND`
- For `Geschl` we take 0 for `m` and 1 for `f`
- For `GeimpftGenesen` we take 0 for `nothing` , 1 for vaccinated, 2 for recovered, and 3 (1+2) for both vaccinated and recovered
- For `Bundesland` we should consider looking at the `PolBez` once more: As we already know, the first digit of the PolBez corresponds to the `Bundesland` . Because of this, we should keep these numbers for the sake of having consistent values. (If we look closer, we can also see that the states are numbered alphabetically so that's also what we are going to do (:)

In [22]:
```
num1 = pd.DataFrame()

altersgrToNumeric = {"0 - 4": 2.0,
                     "5 - 14": 9.5,
                     "15 - 24": 19.5,
                     "25 - 34": 29.5,
                     "35 - 44": 39.5,
                         "45 - 54": 49.5,
                         "55 - 64": 59.5,
                         "65 - 74": 69.5,
                         "75 - 84": 79.5,
                         "85+": 85.0}

geimpftGenesenToNumeric = {"NICHTS": 0,  #nothing
                          "IMPF": 1,  #vacc
                          "GEN": 2,  #recov
                          "IMPF + GEN": 3} #vacc + recov

geblandToNumeric = dict((v, k) for k, v in enumerate(korr1["GebLand"].unique()))
geschlToNumeric = dict((v, k) for k, v in enumerate(korr1["Geschl"].unique()))
bundeslandToNumeric = dict((v, k + 1) for k, v in enumerate(np.sort(korr2["Bundesla

num1["PolBez"] = korr1["PolBez"].map(lambda val: int(val[-3:]))
num1["Altersgr"] = korr1["Altersgr"].map(altersgrToNumeric)
num1["GebLand"] = korr1["GebLand"].map(geblandToNumeric)
num1["Geschl"] = korr1["Geschl"].map(geschlToNumeric)
num1["GeimpftGenesen"] = korr1["GeimpftGenesen"].map(geimpftGenesenToNumeric)
num1["Bundesland"] = korr1["Bundesland"].map(bundeslandToNumeric)
num1["Anz"] = korr1["Anz"]

num1.sample(5)
```

Out[22]:

	PolBez	Altersgr	GebLand	Geschl	GeimpftGenesen	Bundesland	Anz
11600	620	79.5	0	1	1	6	2663
15267	908	49.5	1	0	1	9	212
8735	416	9.5	0	1	2	4	1537
1632	203	69.5	0	1	1	2	771
12972	706	69.5	1	1	1	7	142

Let's look if our values are truly numerical:

In [23]:
```
num1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17694 entries, 0 to 17693
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   PolBez          17694 non-null  int64  
 1   Altersgr        17694 non-null  float64
 2   GebLand         17694 non-null  int64  
 3   Geschl          17694 non-null  int64  
 4   GeimpftGenesen  17694 non-null  int64  
 5   Bundesland      17694 non-null  int64  
 6   Anz             17694 non-null  int64  
dtypes: float64(1), int64(6)
memory usage: 967.8 KB
```

So far so good: Everything's an int64 or float64, so we can do the same thing for dataset 2.

Dataset 2

For `Altersgr`, `Bundesland`, and `GeimpftGenesen` we used the same dict as above to map our values to numerical ones.

- For `ErwerbStatus` we take 0 for inactive and 1 for active (common values to symbolize true/false)
- For `Bildung` we are going to order the values from 0-4 (lowest graduation-highest graduation)

In [24]:
```
num2 = pd.DataFrame()


erwerbstatusToNumeric = {"inaktiv": 0, "aktiv": 1}

bildungToNumeric = {"Unknown": 0, "Pflichtschule": 1, "Lehrabschluss/BMS": 2, "BHS/

num2["Bundesland"] = korr2["Bundesland"].map(bundeslandToNumeric)
num2["Bildung"] = korr2["Bildung"].map(bildungToNumeric)
num2["Altersgr"] = korr2["Altersgr"].map(altersgrToNumeric)
num2["ErwerbStatus"] = korr2["ErwerbStatus"].map(erwerbstatusToNumeric)
num2["GeimpftGenesen"] = korr2["GeimpftGenesen"].map(geimpftGenesenToNumeric)
num2["Anz"] = korr2["Anz"]

num2.sample(5)
```

Out[24]:

	Bundesland	Bildung	Altersgr	ErwerbStatus	GeimpftGenesen	Anz
891	6	4	49.5	0	1	974
991	7	2	59.5	0	1	14677
1173	8	3	49.5	0	3	98
1276	9	2	29.5	1	1	18194
1036	7	4	39.5	1	3	5986

We need to check again if the values are all numeric:

In [25]:
```
num2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1404 entries, 0 to 1403
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Bundesland      1404 non-null   int64  
 1   Bildung         1404 non-null   int64  
 2   Altersgr        1404 non-null   float64
 3   ErwerbStatus    1404 non-null   int64  
 4   GeimpftGenesen  1404 non-null   int64  
 5   Anz             1404 non-null   int64  
dtypes: float64(1), int64(5)
memory usage: 65.9 KB
```

So let's save these numerical values into different csv-files.

In [26]:
```
num1.to_csv("./output/num1.csv", index=False)
num2.to_csv("./output/num2.csv", index=False)
```

Normalized Values

In this section, we will adjust our numerical values so that all of them are in the range of 0 - 1 (each column). First, we need to import the csv with the numerical data.

In [27]:
```
num1 = pd.read_csv("./output/num1.csv")
num2 = pd.read_csv("./output/num2.csv")
```

Let's apply our formula to get normalized values from 0 to 1:

In [28]:
```
norm1 = (num1 - num1.min()) / (num1.max() - num1.min())
norm2 = (num2 - num2.min()) / (num2.max() - num2.min())
```

Let's see if all of these values are within 0 and 1 with a simple pandas function:

In [29]:
```
norm1.describe()
```

Out[29]:

	PolBez	Altersgr	GebLand	Geschl	GeimpftGenesen	Bundesland	
count	17694.000000	17694.000000	17694.000000	17694.000000	17694.000000	17694.000000	17
mean	0.508211	0.525914	0.488584	0.501187	0.492012	0.510456	
std	0.318163	0.323962	0.499884	0.500013	0.372839	0.327328	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.256691	0.210843	0.000000	0.000000	0.000000	0.250000	
50%	0.383212	0.572289	0.000000	1.000000	0.333333	0.375000	
75%	0.738443	0.813253	1.000000	1.000000	0.666667	0.750000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

Looking good so far. Every min is 0 and every max is 1 (which was our goal). Now let's check norm2, too.

In [30]:
```
norm2.describe()
```

Out[30]:

	Bundesland	Bildung	Altersgr	ErwerbStatus	GeimpftGenesen	Anz
count	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000
mean	0.504006	0.512821	0.493115	0.499288	0.491690	0.061415
std	0.322200	0.348880	0.371470	0.500178	0.372293	0.103388
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.250000	0.250000	0.000000	0.000000	0.000000	0.005569
50%	0.500000	0.500000	0.333333	0.000000	0.333333	0.024730
75%	0.750000	0.750000	0.666667	1.000000	0.666667	0.070619
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Now, it's time to save these datasets once more as `norm1.csv` and `norm2.csv`.

In [31]:
```
norm1.to_csv("./output/norm1.csv", index=False)
norm2.to_csv("./output/norm2.csv", index=False)
```

Nominal Values

In this section we will convert the values to only ordinal or nominal values and therefore get rid of any metric values. First, let's look at dataset 1.

Dataset 1

We can actually keep the values in `PolBez`, `Altersgr`, `GebLand`, `Geschl`, `GeimpftGenesen`, and `Bundesland` as they are from korr1 dataframe. For `Anz` however, we need to define a limit. Everything under this limit will be considered "low" (`WENIG`) and everything above (incl. the limit) is considered "high" (`VIEL`). Let's take the median value as the limit, so that (approx.) 50% of the values are "low" and 50% are "high".

In [32]:
```
nom1 = pd.DataFrame()

nom1["PolBez"] = korr1.PolBez
nom1["Altersgr"] = korr1.Altersgr
nom1["GebLand"] = korr1.GebLand
nom1["Geschl"] = korr1.Geschl
nom1["GeimpftGenesen"] = korr1.GeimpftGenesen
nom1["Bundesland"] = korr1.Bundesland

anzLimit1 = korr1["Anz"].median()
nom1["Anz"] = korr1["Anz"].map(lambda x: "WENIG" if x < anzLimit1 else "VIEL")

nom1.sample(5)
```

Out[32]:

	PolBez	Altersgr	GebLand	Geschl	GeimpftGenesen	Bundesland	Anz
16093	BEZIMST-913	75 - 84	INLAND	m	NICHTS	Wien	WENIG
5917	BEZIMST-321	55 - 64	AUSLAND	f	IMPF + GEN	Niederösterreich	WENIG
13507	BEZIMST-801	15 - 24	AUSLAND	f	GEN	Vorarlberg	WENIG
1977	BEZIMST-205	85+	AUSLAND	f	GEN	Kärnten	WENIG
7135	BEZIMST-405	55 - 64	AUSLAND	f	IMPF	Oberösterreich	WENIG

Dataset 2

Like the previous dataframe, we can leave `Bundesland`, `Bildung`, `ErwerbStatus`, and `GeimpftGenesen` from korr2 dataframe unchanged:

In [33]:
```
nom2 = pd.DataFrame()

nom2["Bundesland"] = korr2.Bundesland
nom2["Bildung"] = korr2.Bildung
nom2["Altersgr"] = korr2.Altersgr
nom2["ErwerbStatus"] = korr2.ErwerbStatus
nom2["GeimpftGenesen"] = korr2.GeimpftGenesen

anzLimit2 = korr2["Anz"].median()
nom2["Anz"] = korr2["Anz"].map(lambda x: "WENIG" if x < anzLimit2 else "VIEL")

nom2.sample(5)
```

Out[33]:

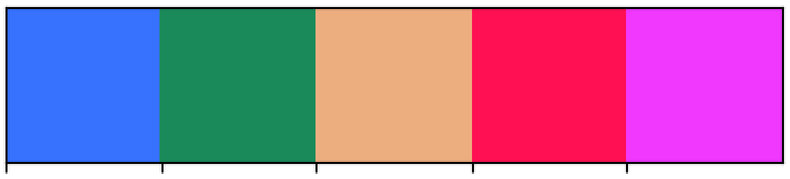
	Bundesland	Bildung	Altersgr	ErwerbStatus	GeimpftGenesen	Anz
660	Salzburg	Lehrabschluss/BMS	35 - 44	aktiv	NICHTS	VIEL
1313	Wien	BHS/AHS/Kolleg	25 - 34	inaktiv	IMPF + GEN	VIEL
308	Niederösterreich	Pflichtschule	25 - 34	aktiv	NICHTS	VIEL
539	Oberösterreich	BHS/AHS/Kolleg	35 - 44	inaktiv	GEN	WENIG
1341	Wien	Akademie/Hochschule	25 - 34	aktiv	IMPF + GEN	VIEL

Visualization with Seaborn

Color Palette

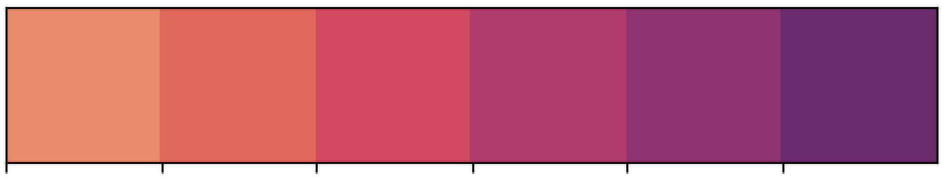
Color palettes are essential when it comes to visualization. We should define a color palette for categorical data (nominal).

In [34]:
```
colors = ["#3772FF", "#1b8a5a", "#edae7f", "#FF1053", "#F038FF"]
cat_palette = sns.color_palette(colors)
sns.palplot(cat_palette)
plt.show()
```



We should also define a palette for "continuous" data (such as when depicting a heatmap). I really like the predefined "flare" palette, which is also a gradient when defining `as_cmap=True`.

In [35]:
```
cont_palette = sns.color_palette("flare", as_cmap=True)
sns.palplot(sns.color_palette("flare"))
plt.show()
```



Correlation

Let us detect some correlation between our values. Since numerical values are better for clustering and finding correlation, we can use our num dataframes to detect some correlation. We should at least see a correlation between the features `PolBez` and `Bundesland` from num1, right?

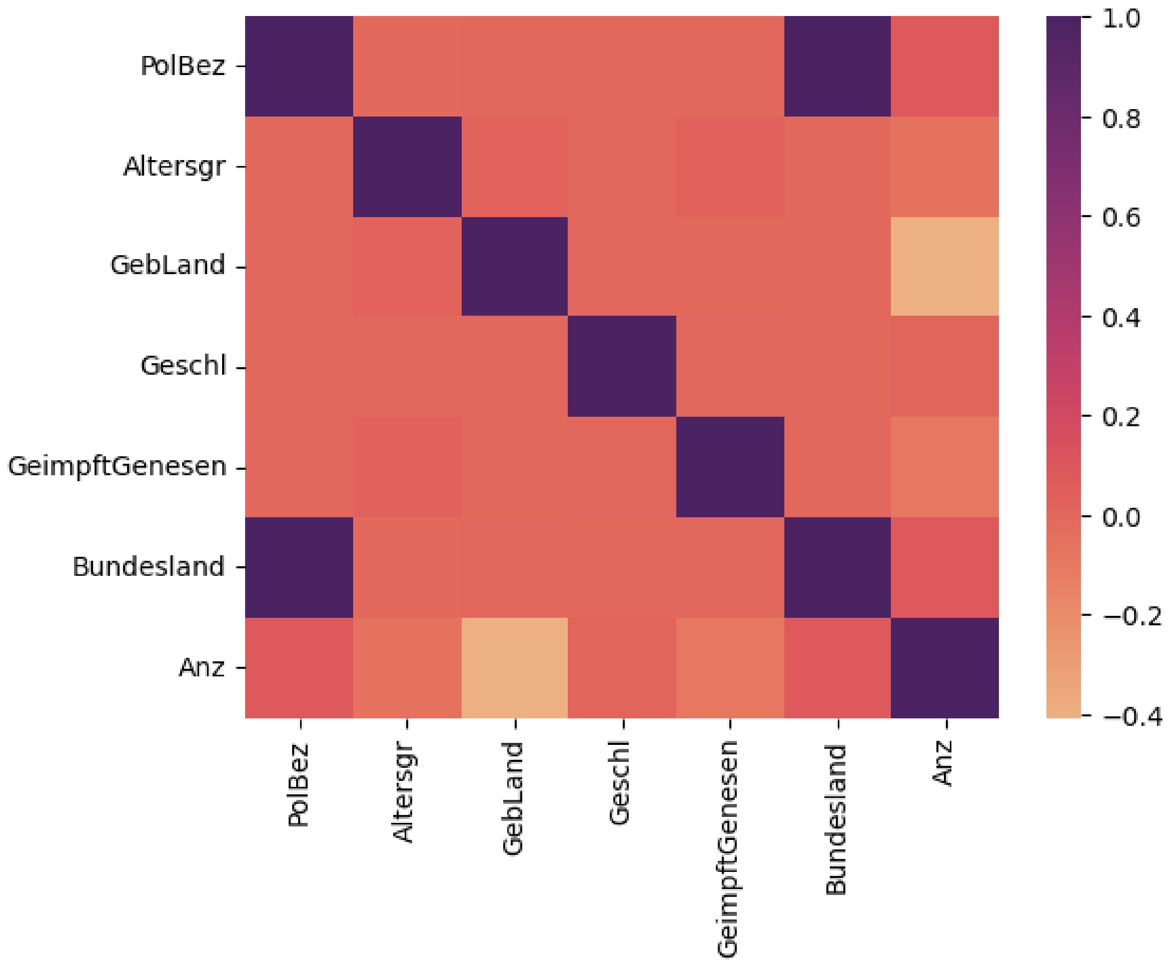
In [36]:
```
num1.corr()
```

Out[36]:

	PolBez	Altersgr	GebLand	Geschl	GeimpftGenesen	Bundesland	A
PolBez	1.000000	-0.007576	0.003521	-0.000854	0.004618	0.999188	0.0826
Altersgr	-0.007576	1.000000	0.024892	0.002802	0.029775	-0.007483	-0.0563
GebLand	0.003521	0.024892	1.000000	0.002542	-0.008020	0.003579	-0.4077
Geschl	-0.000854	0.002802	0.002542	1.000000	0.000051	-0.000896	0.0121
GeimpftGenesen	0.004618	0.029775	-0.008020	0.000051	1.000000	0.004582	-0.0911
Bundesland	0.999188	-0.007483	0.003579	-0.000896	0.004582	1.000000	0.0828
Anz	0.082690	-0.056352	-0.407700	0.012189	-0.091159	0.082894	1.0000

Well we were right with Bundesland, but with all these numbers, it sure looks complicated... Let's use colors instead!

In [37]:
```
sns.heatmap(num1.corr(), cmap=cont_palette)
plt.show()
```



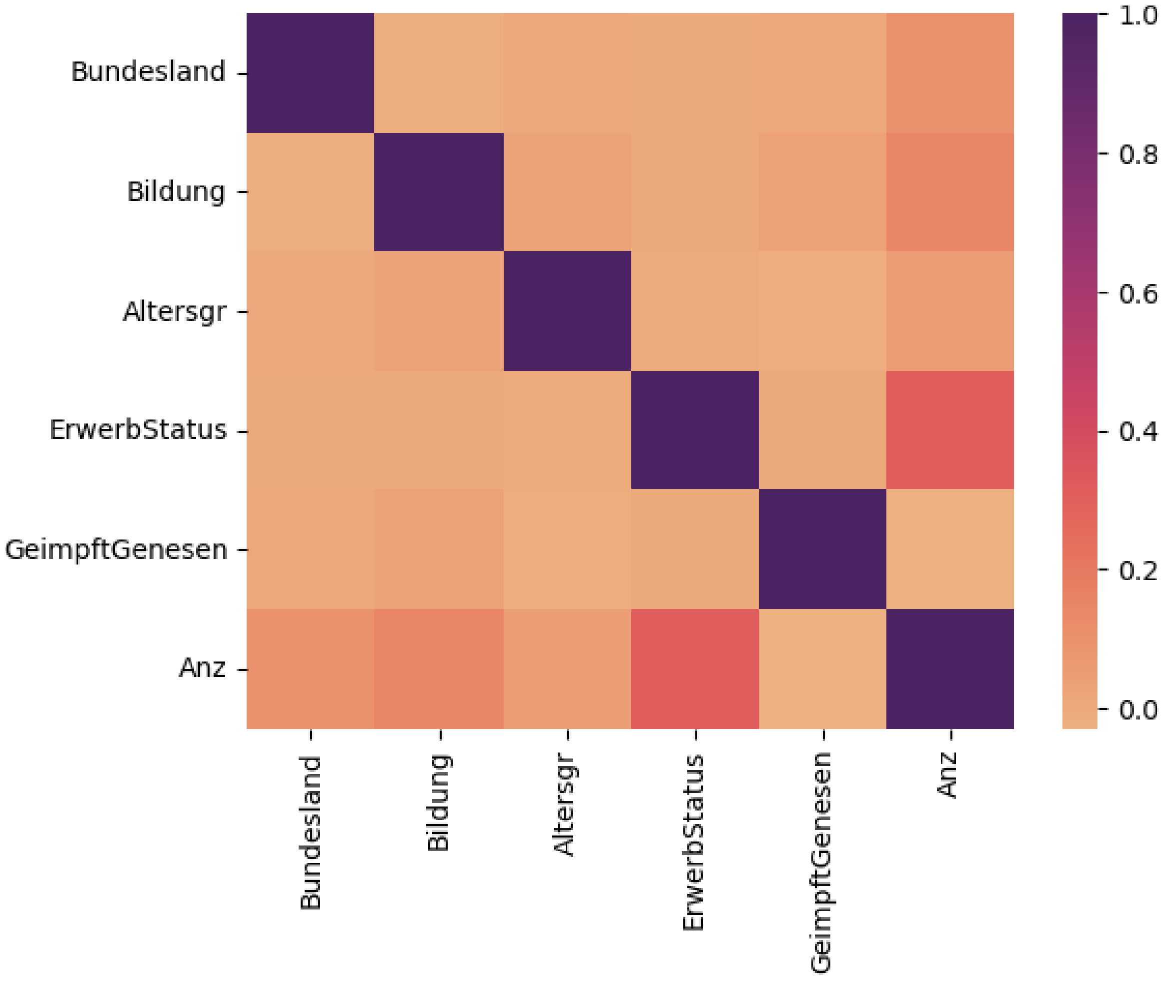
This suggests that there is no concrete correlation except for `Bundesland` and `PolBez`. Except that, there is also a slight "color change" between `Anz` and `GebLand`. Let's look at that more closely:

In [38]:
```
num1["Anz"].corr(num1["GebLand"])
```

Out[38]:
```
-0.40769994363327905
```

This value suggests that the amount of people getting vaccinated/recovered/or both depends on where you have been born. But to be sure, you have to look at the causality here: We do not think that the mere amount (--> without any information whether the number of people is vaccinated/recovered/both/none) is not dependent on where you are coming from. Maybe this value is being produced, since there are more people born in Austria and currently living in Austria. This would lead to a bigger number in the column `Anz` for people who are born in Austria (and vice versa). The fact that the value is negative even supports my argumentation (`INLAND` has smaller value (0), `Anz` having higher value --> negative correlation)

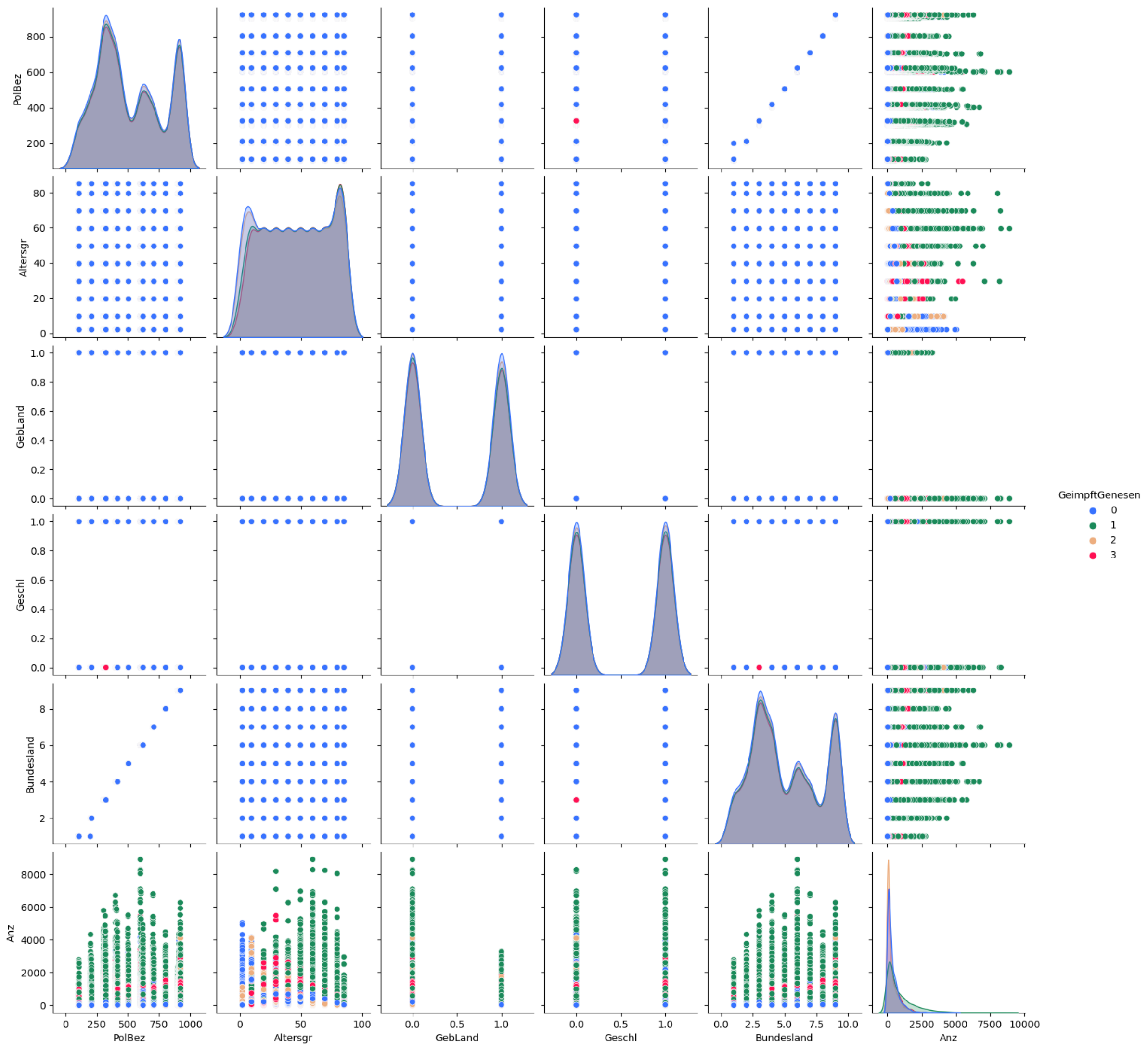
In [39]:
```
sns.heatmap(num2.corr(), cmap=cont_palette)
plt.show()
```



Well we see more saturated cells than above when it comes to other features (but not that obvious). We sure can see in both graphs very slight correlations (<0.1). For example between `Bildung` and `GeimpftGenesen`. This might suggest that the education might have something to do with people getting vaccinated. There is also the fact that `Bildung` correlates with `Anz`, too, which also supports the argument that there might be a connection. There is also a higher saturation between `Anz` and `Erwerbstatus` which might suggest that there is a connection, too. But as above, without any additional information, this correlation lacks causality.

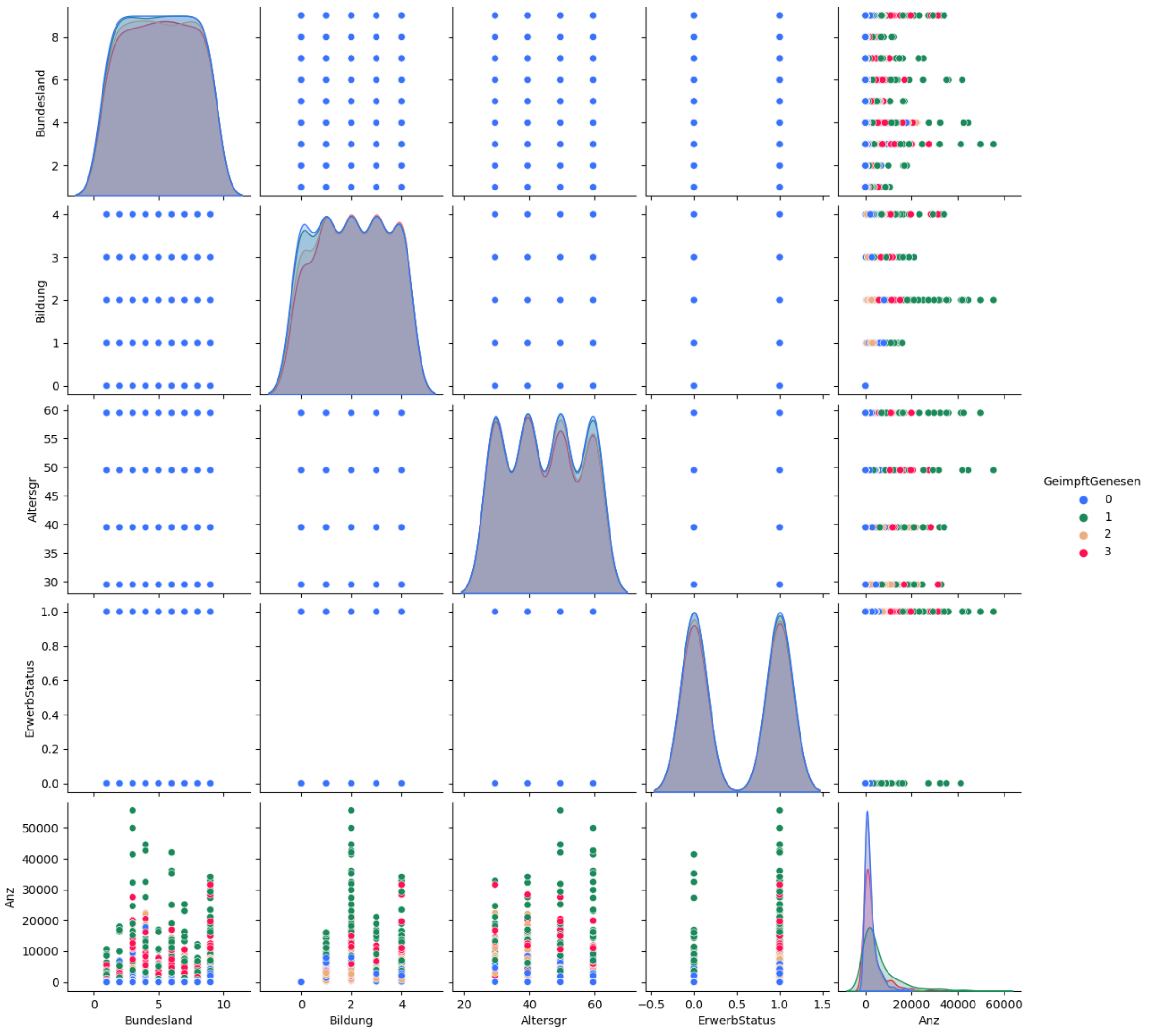
Pairplot

In [40]:
```
sns.pairplot(num1, hue="GeimpftGenesen", palette=cat_palette[0:4])
plt.show()
```



Well this is the backlash you get when your dataset has too many nominal/ordinal values. In the above diagram you cannot really see certain groupings, clusters, or anything similar. This is because there are too many nominal/ordinal values and pairplots (scatterplots to be exact) work much better between two metric features (e.g. Length and Width).

In [41]:
```
sns.pairplot(num2, hue="GeimpftGenesen", palette=cat_palette[0:4])
plt.show()
```



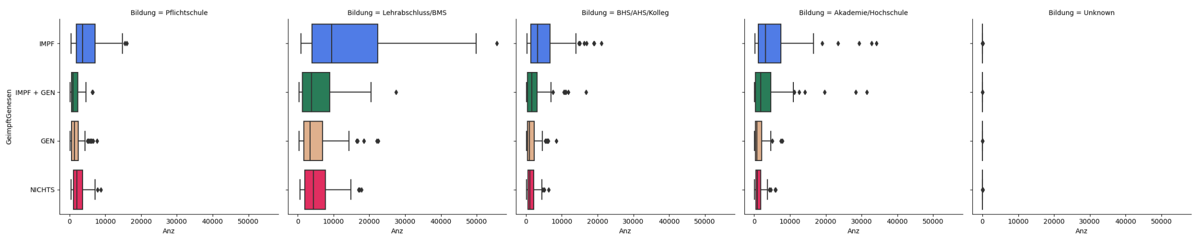
Same goes here, too many nominal/ordinal values. Well we could look at `Bundesland` and `Anz`, too. We can see that federal states 3 (Niederösterreich), 4 (Oberösterreich), and 6 (Steiermark) much more green at the top. That means the mere number is higher when it comes to these federal countries. Maybe because the population there is bigger? (P.S. most of these graphs just show blue dots because the dots overlap)

Boxplots and Violinplots

Let's see if we were right, when we argued that the education has something to do with vaccination cases:

In [42]:
```
sns.catplot(data=korr2, kind="box", x="Anz", y="GeimpftGenesen", col="Bildung", ori
```

Out[42]:
```
<seaborn.axisgrid.FacetGrid at 0x1e4b3bb99d0>
```



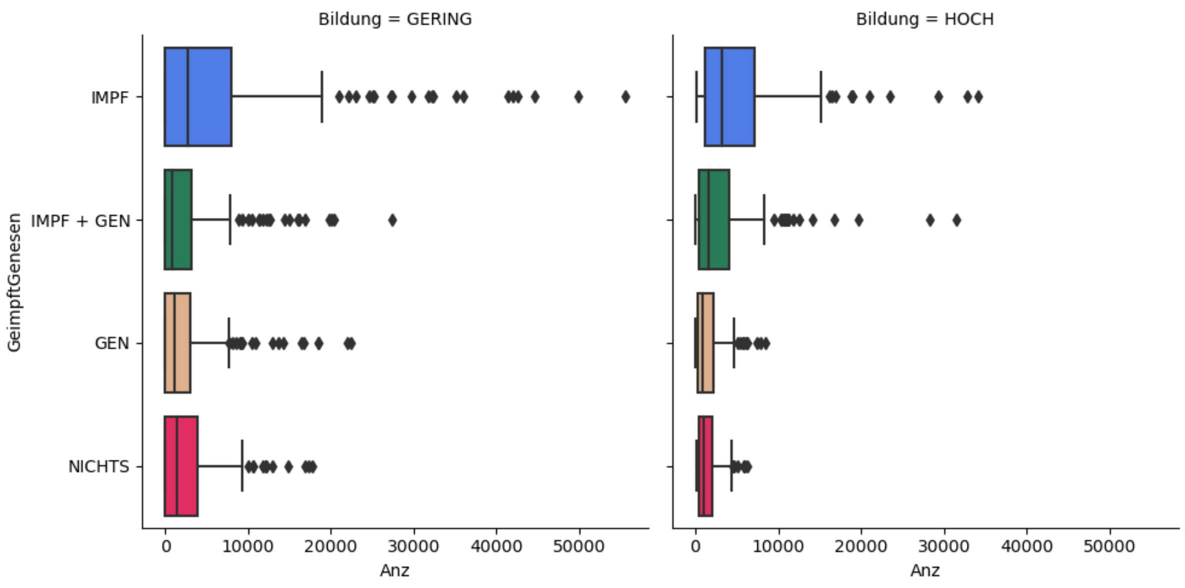
With all of these boxplots one can really get overwhelmed. Instead, we should simplify the feature `Bildung`:

In [43]:
```
bildungDictToTemp = {"Unknown": "GERING", "Pflichtschule": "GERING", "Lehrabschluss
                     "BHS/AHS/Kolleg": "HOCH", "Akademie/Hochschule": "HOCH"}

temp2 = pd.DataFrame(korr2) #new instance
temp2["Bildung"] = temp2["Bildung"].map(bildungDictToTemp)

sns.catplot(data=temp2, kind="box", x="Anz", y="GeimpftGenesen", col="Bildung", ori
```

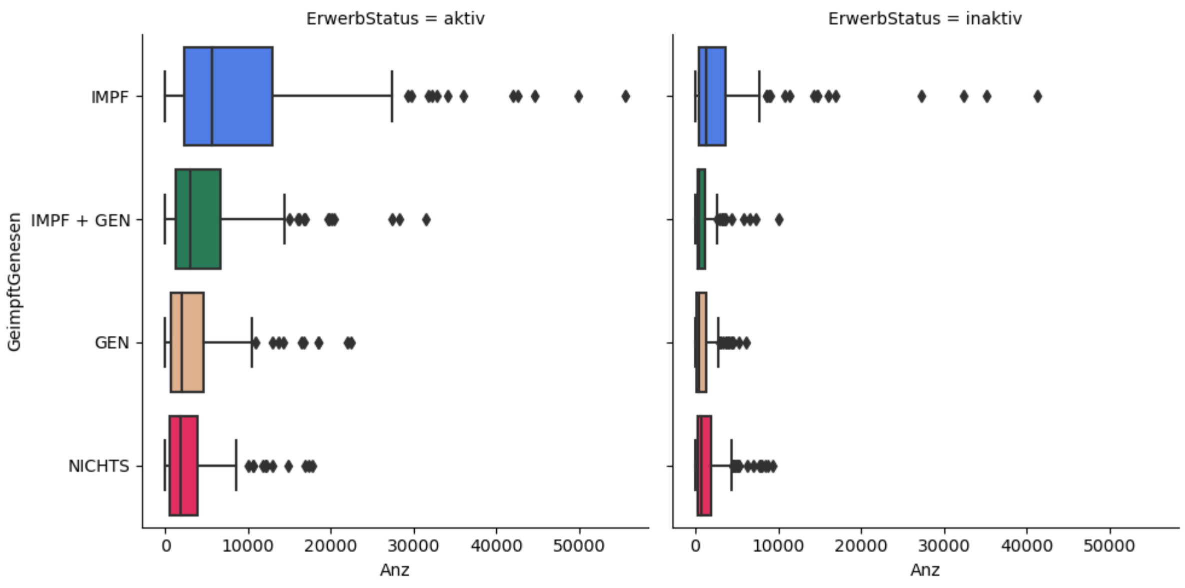
Out[43]:
```
<seaborn.axisgrid.FacetGrid at 0x1e4af1c3690>
```



Well maybe our theory was wrong, These ranges look pretty evened out. Let's look at another theory of mine: Maybe the amount of vaccinated people is dependent on the economic status of a person.

In [44]:
```
sns.catplot(data=korr2, kind="box", x="Anz", y="GeimpftGenesen", col="ErwerbStatus'
```

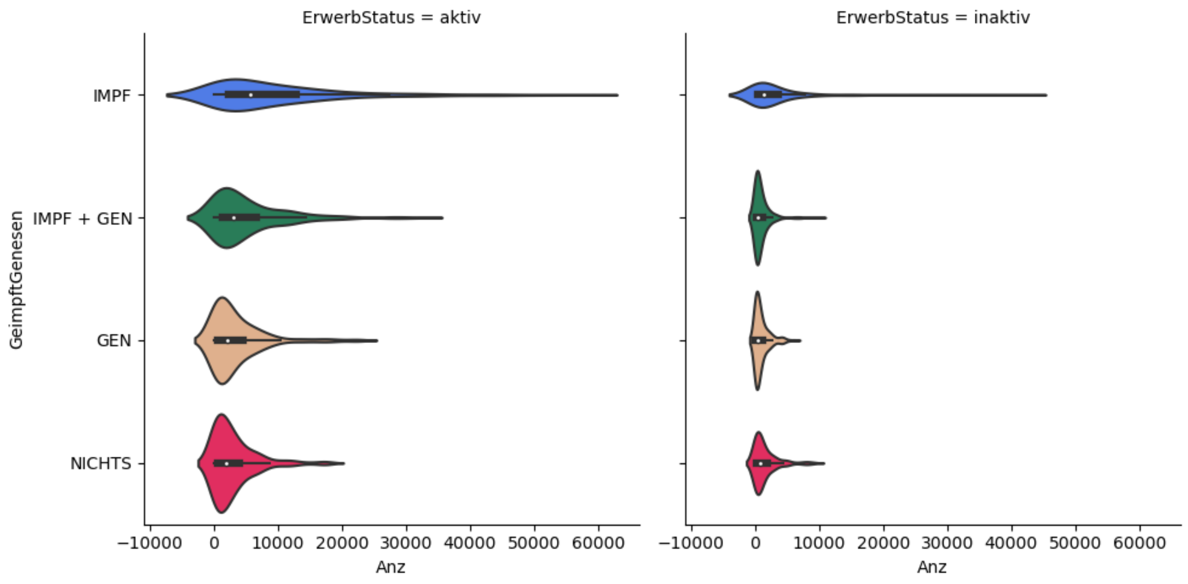
Out[44]:
```
<seaborn.axisgrid.FacetGrid at 0x1e4b54d1ad0>
```



We can see here that more economically active people go vaccinating, are recovering, or both than economically inactive people. (There might also be a chance that there are more economically active people alltogether). What I really found interesting is, is that the range of `NICHTS` is slightly bigger than `GEN` when it comes to economically inactive people. To be a hundred percent sure whether there is a connection between vaccination and economic status, we can look at a violin plot.

```
In [45]: sns.catplot(data=korr2, kind="violin", x="Anz", y="GeimpftGenesen", col="ErwerbStat
```

```
Out[45]: <seaborn.axisgrid.FacetGrid at 0x1e4b038b690>
```

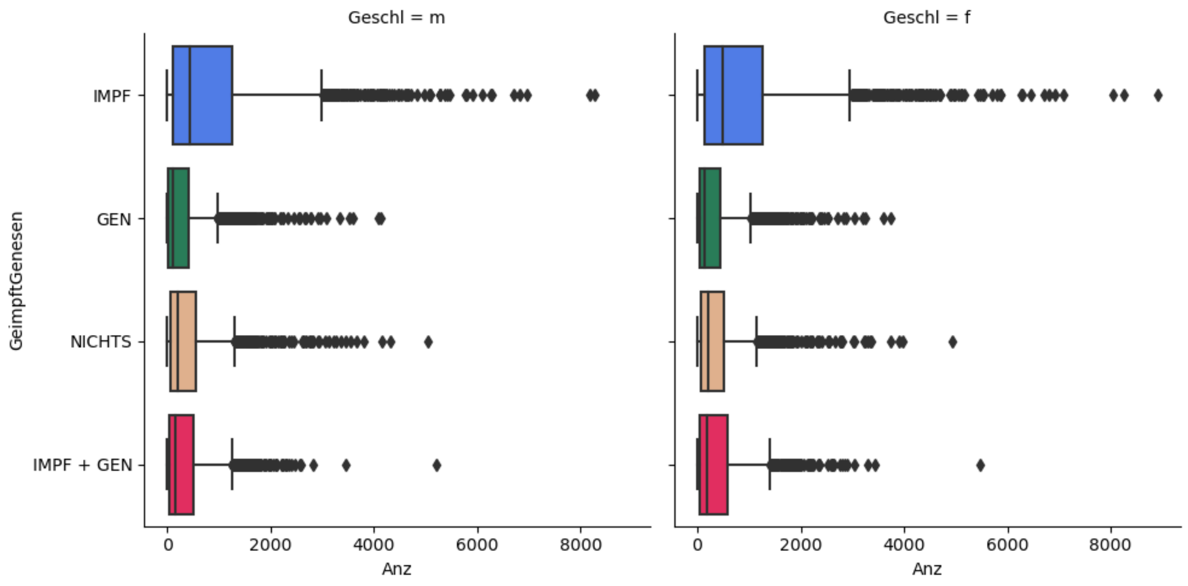


If there was a higher density on the Blue violin plot when it comes to economically inactive people, we could say that there is an equal number of people but inactive ones choose not to vaccinate. Since the density is roughly the same, we can just say that there are more economically active people and have therefore a wider range in the boxplot.

Lastly, we would like to look at the ranges of the male/female population:

```
In [46]: sns.catplot(data=korr1, kind="box", x="Anz", y="GeimpftGenesen", col="Geschl", orie
```

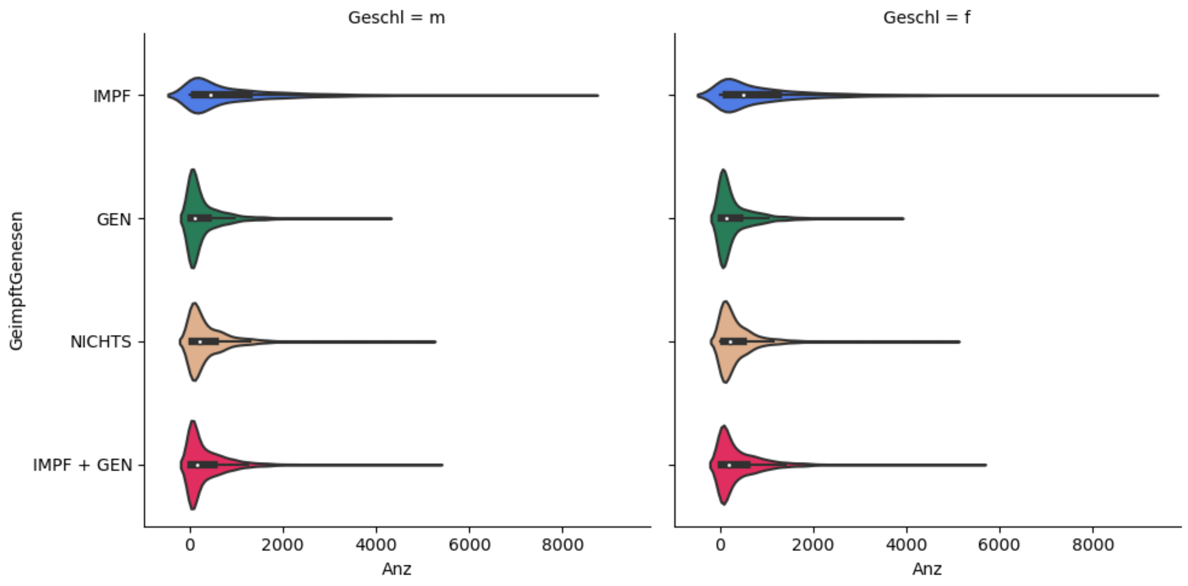
```
Out[46]: <seaborn.axisgrid.FacetGrid at 0x1e4aeefb690>
```



Well these ranges are evenly distributed amongst each group, too (boring...). But notice that there are so many outliers! Let's look at the density, too.

In [47]:
```
sns.catplot(data=korr1, kind="violin", x="Anz", y="GeimpftGenesen", col="Geschl", c
```

Out[47]:
```
<seaborn.axisgrid.FacetGrid at 0x1e4b39bfa50>
```

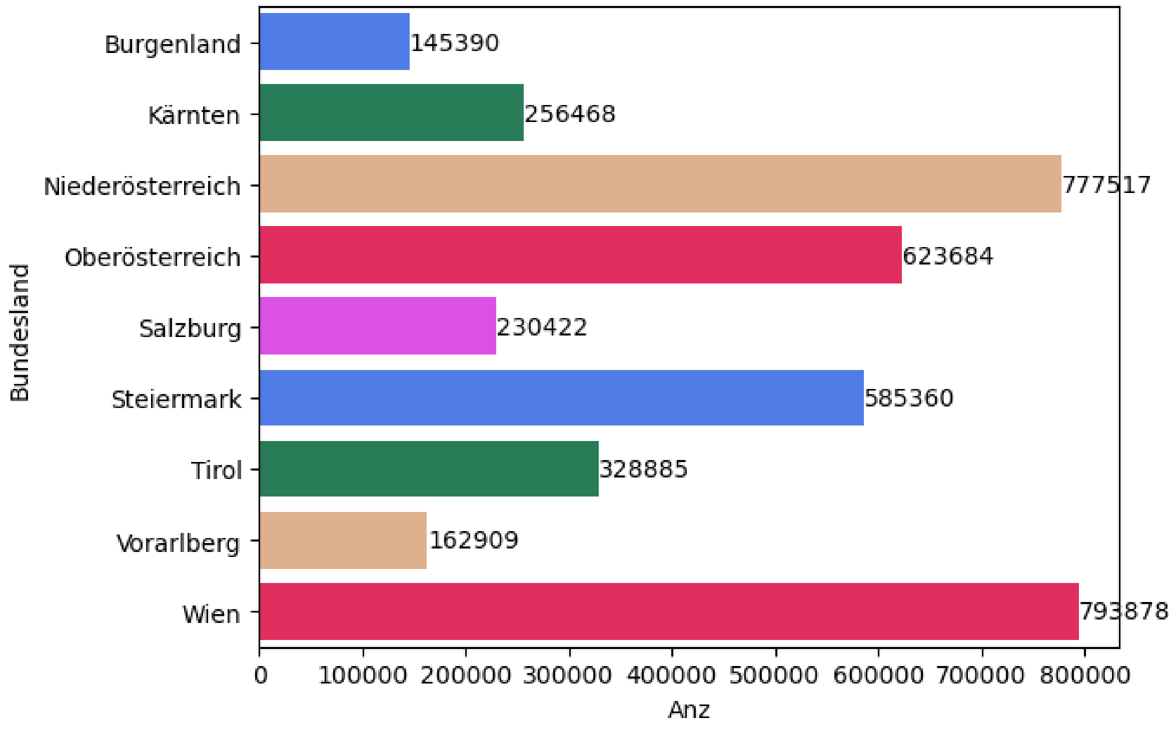


Well we can say that there is an equal range regarding the two genders. Even the median values do not vary between these two categories!

Barchart

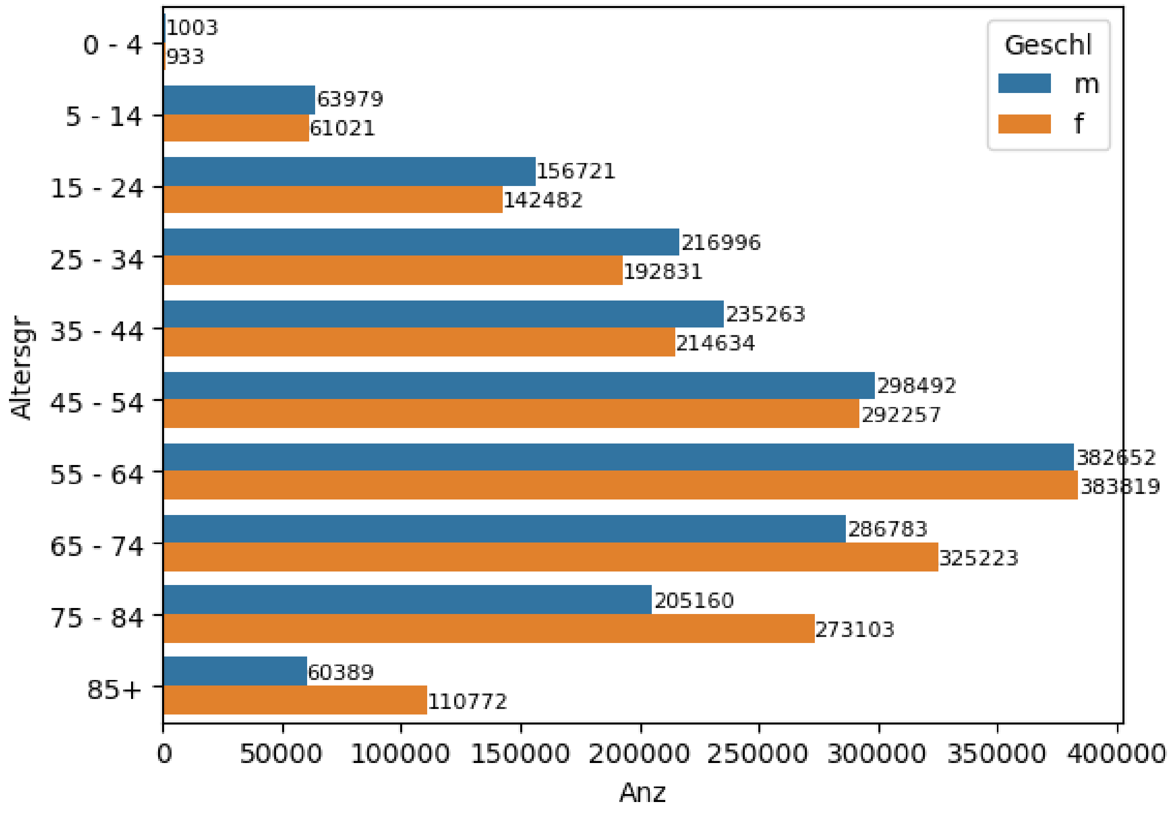
Suppose we want to know how many people in each federal country is vaccinated:

In [48]:
```
ax = sns.barplot(korr1.loc[korr1["GeimpftGenesen"] == "IMPF"], x="Anz", y="Bundesla
ax.bar_label(ax.containers[0], fontsize=10)
plt.show()
```



Alright so we see that Vienna has the most vaccinated people and Burgenland has the least vaccinated people. Let's look at the age groups now:

```
In [49]: ax = sns.barplot(korr1.loc[korr1["GeimpftGenesen"] == "IMPF"], x="Anz", y="Altersgr
         ax.bar_label(ax.containers[0], fontsize=8)
         ax.bar_label(ax.containers[1], fontsize=8)
         plt.show()
```



What is interesting is as the age groups get bigger (older), there are more female people getting vaccinated (Maybe because women live longer than men?).

Transformation for future graphs

For our next analysis, we should transform our data so we have multiple values in one record. I think instead of one number `Anz` we should have number of vaccinated `ImpfAnz`, number of recovered `GenAnz`, number of vaccinated and recovered `ImpfGenAnz`, and number of people that are not vaccinated nor recovered `NichtsAnz`.

In [50]:
```
def transform_df2(row):
    result_row = row
    temp_df = korr2.loc[(korr2["Bundesland"] == row["Bundesland"]) &
                        (korr2["Bildung"] == row["Bildung"]) &
                        (korr2["Altersgr"] == row["Altersgr"]) &
                        (korr2["ErwerbStatus"] == row["ErwerbStatus"])]

    result_row["ImpfAnz"] = temp_df.loc[temp_df["GeimpftGenesen"] == "IMPF"]["Anz"]
    result_row["GenAnz"] = temp_df.loc[temp_df["GeimpftGenesen"] == "GEN"]["Anz"].s
    result_row["ImpfGenAnz"] = temp_df.loc[temp_df["GeimpftGenesen"] == "IMPF + GEN
    result_row["NichtsAnz"] = temp_df.loc[temp_df["GeimpftGenesen"] == "NICHTS"]["A

    result_row.drop(["GeimpftGenesen", "Anz"], inplace=True)
    return result_row

transKorr2 = korr2.apply(transform_df2, axis=1).drop_duplicates()
transKorr2
```

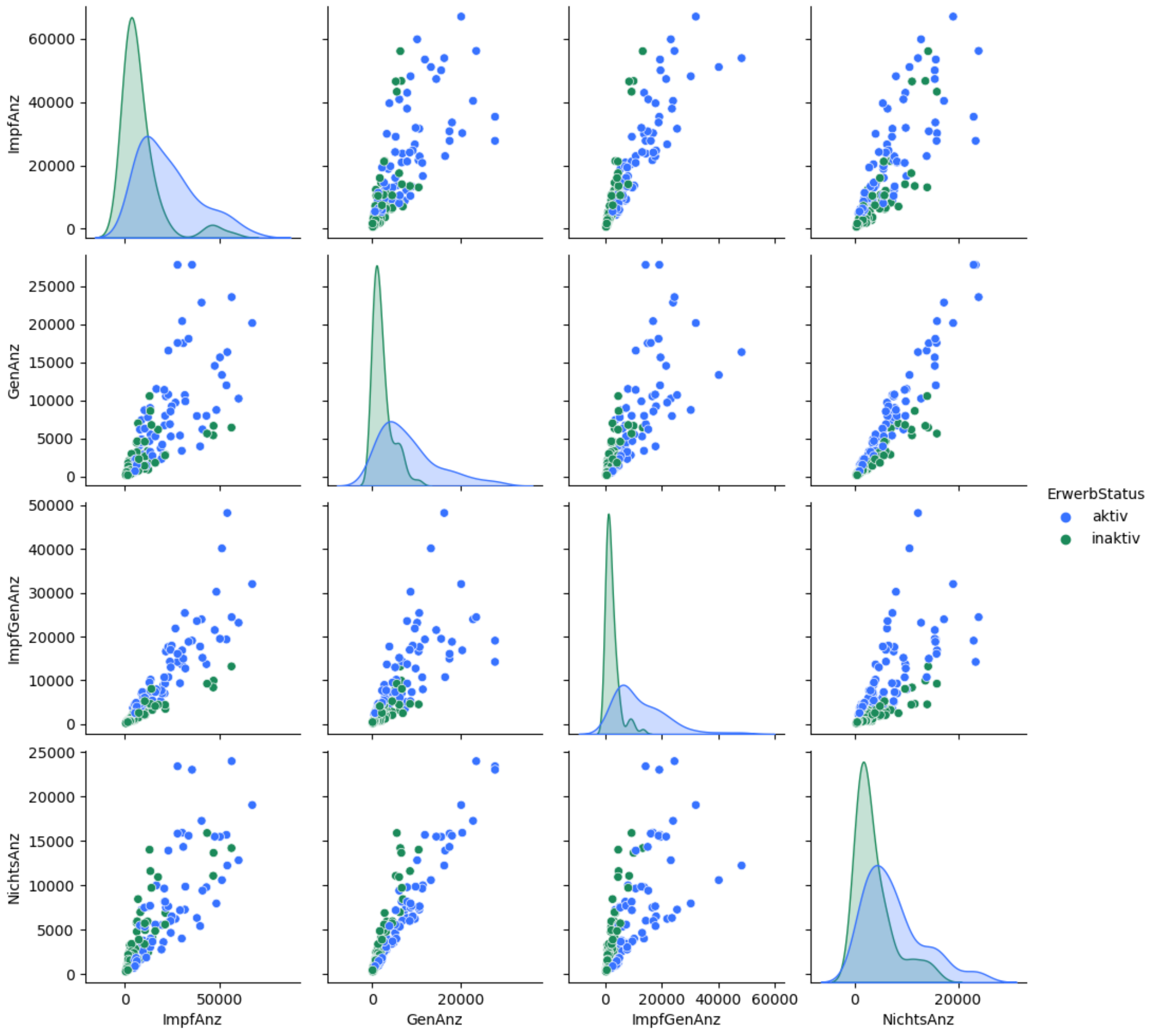
Out[50]:

	Bundesland	Bildung	Altersgr	ErwerbStatus	ImpfAnz	GenAnz	ImpfGenAnz	NichtsAnz
0	Burgenland	GERING	25 - 34	aktiv	4805	2573	2763	2301
4	Burgenland	GERING	25 - 34	inaktiv	1347	809	514	1107
8	Burgenland	GERING	35 - 44	aktiv	7527	3246	4595	2718
12	Burgenland	GERING	35 - 44	inaktiv	1530	765	558	1102
16	Burgenland	GERING	45 - 54	aktiv	12622	2928	6426	2962
...	...	...	...	...	...	...	...	...
1320	Wien	HOCH	35 - 44	inaktiv	10547	4555	5240	5717
1324	Wien	HOCH	45 - 54	aktiv	48166	8720	30227	7952
1328	Wien	HOCH	45 - 54	inaktiv	7433	2267	2572	3874
1332	Wien	HOCH	55 - 64	aktiv	39637	3940	17722	5402
1336	Wien	HOCH	55 - 64	inaktiv	16032	1781	4099	4847

144 rows × 8 columns

Very well lets look at our pairplot once more:

In [51]:
```
sns.pairplot(transKorr2, hue="ErwerbStatus", palette=cat_palette[0:2])
plt.show()
```

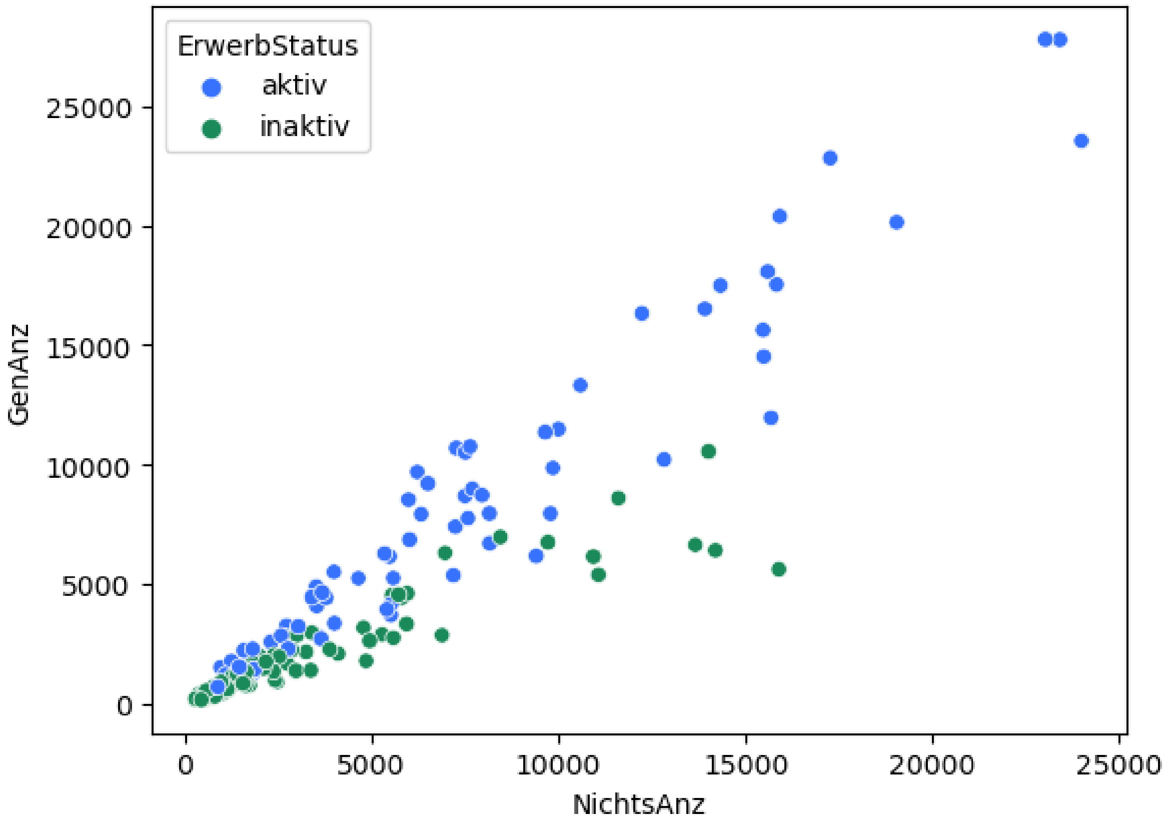


What really caught my attention here is the scatterplot between `GenAnz` and `NichtsAnz`. Let's take a closer look:

Scatterplots

```
In [52]: sns.scatterplot(data=transKorr2, x="NichtsAnz", y="GenAnz", hue="ErwerbStatus", pal
```

```
Out[52]: <Axes: xlabel='NichtsAnz', ylabel='GenAnz'>
```



What can we see here? Well, first of all, there is quite a bit of "density" in the bottom left corner. We can also see that there are clusters forming. So, can we say that, a recovered person is more likely to be economically active since all the blue dots are at the top? Well, not really. Economic status describes a "personal" feature. The problem is that GenAnz and NichtsAnz are both the number of people recovered, and so on. So both of these features are not personal features and therefore do not have an influence whether one is economically active or not. This is the case in all other scatterplots above

But we can estimate if a **person group** is economically active/inactive alltogether, depending on the amount of people that are recovered and the amount of people that are not vaccinated nor recovered (of course given that all of them have the same economic status). The question is of course, is there a reason we would like to know that (use case).

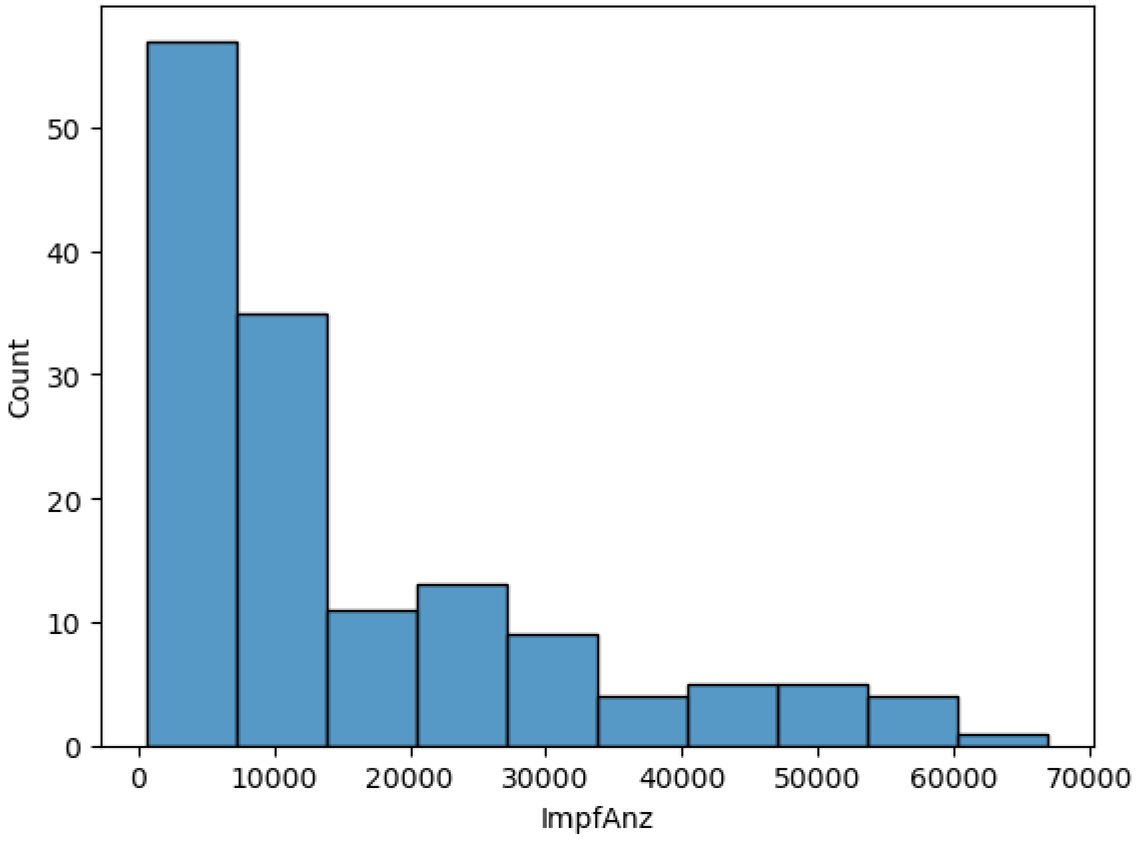
A **person group** is a group of people with the same personal features (economic status, educational status, federal country, ... except the features which count the amount of people in these person groups (e.g. `ImpfAnz`))

Histograms and KDEs (Kernel Density Estimation)

Histograms are often used to see how often a certain value-range occurs. So suppose we want to know what is the most likely value of vaccinated people given there is a certain *person group*. To estimate that, we need a histogram. As we have already seen in the scatterplots above, there is quite a density in some corners. Let's see if we can detect it with our histogram, too:

In [53]:
```
sns.histplot(data=transKorr2, x="ImpfAnz")
```

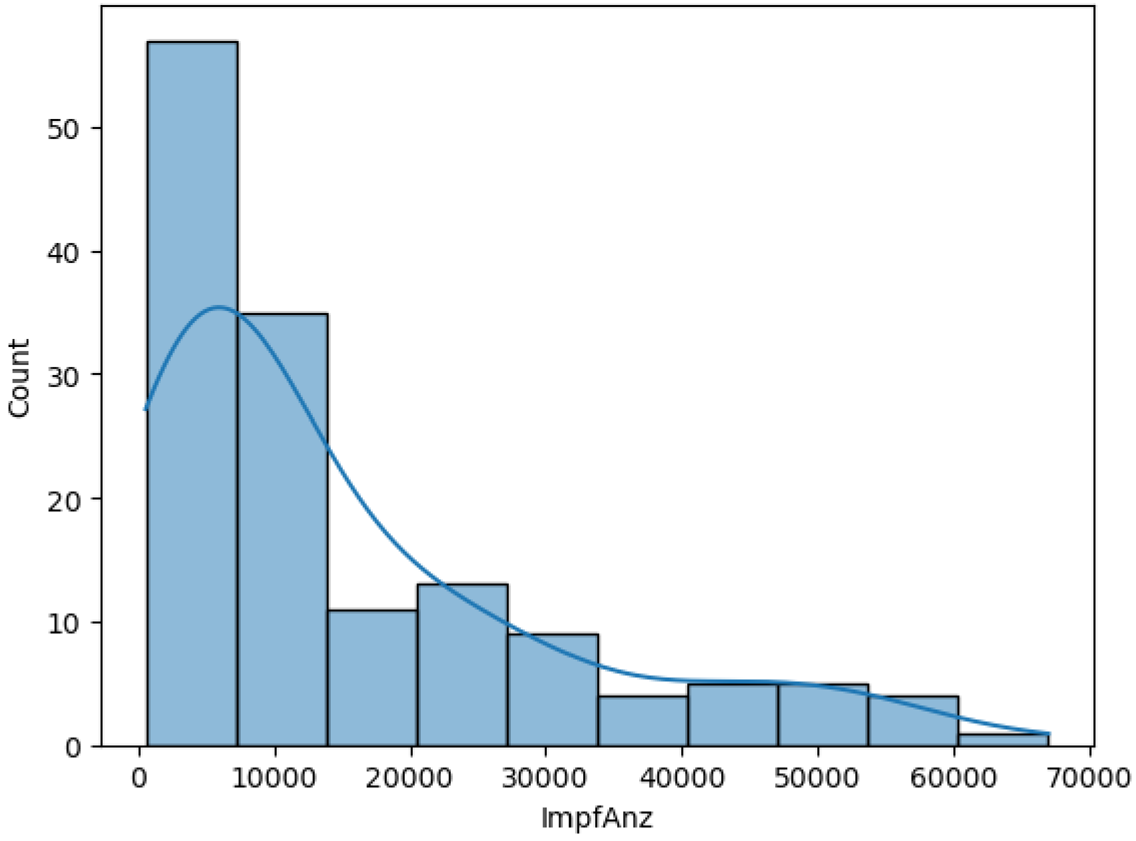
Out[53]:
```
<Axes: xlabel='ImpfAnz', ylabel='Count'>
```



Wow, that's a huuge bump! What does this tell us? The amount of vaccinated people for a certain person group is pretty likely to be quite low. Now, let's also add a KDE to this graph:

In [54]:
```python
sns.histplot(data=transKorr2, x="ImpfAnz", kde=True)
```

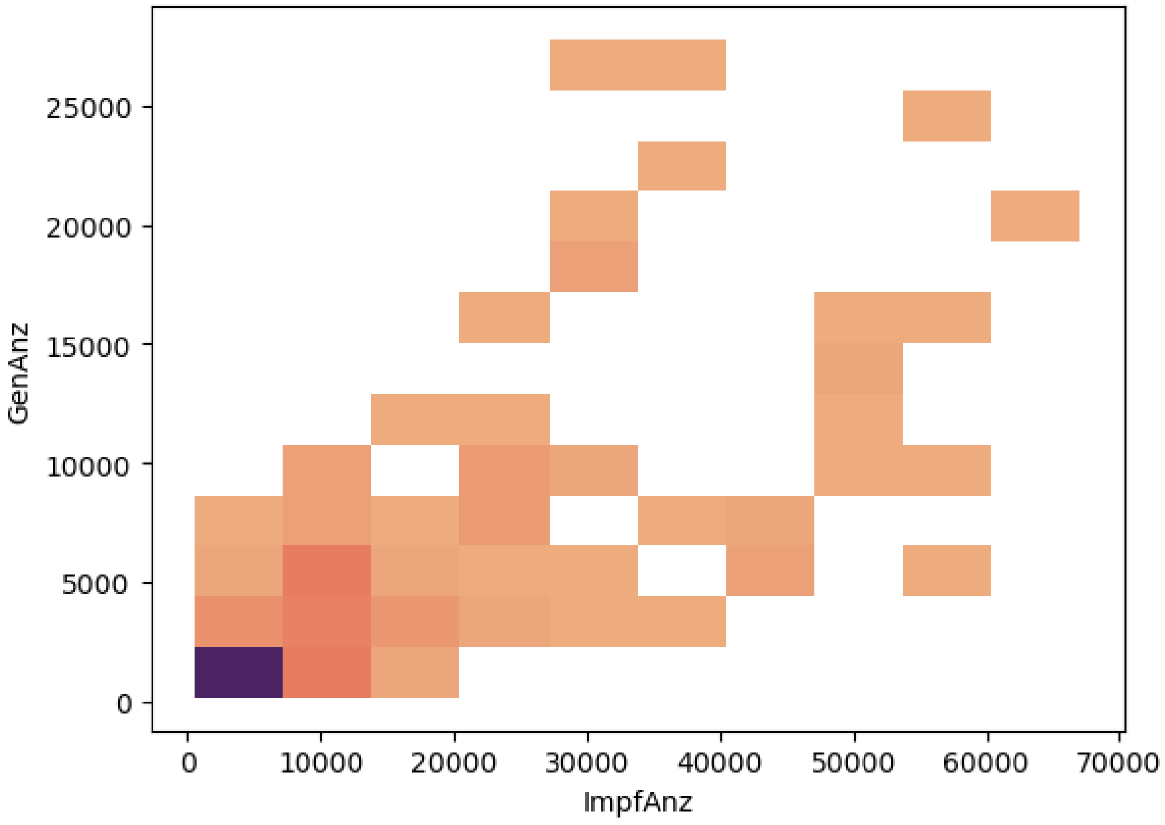
Out[54]:
```
<Axes: xlabel='ImpfAnz', ylabel='Count'>
```



We can also see the distribution of two features combined. This will look something like a heatmap but in reality it's just a fancy histogram:

```
In [55]: sns.histplot(data=transKorr2, x="ImpfAnz", y="GenAnz", cmap=cont_palette)
```

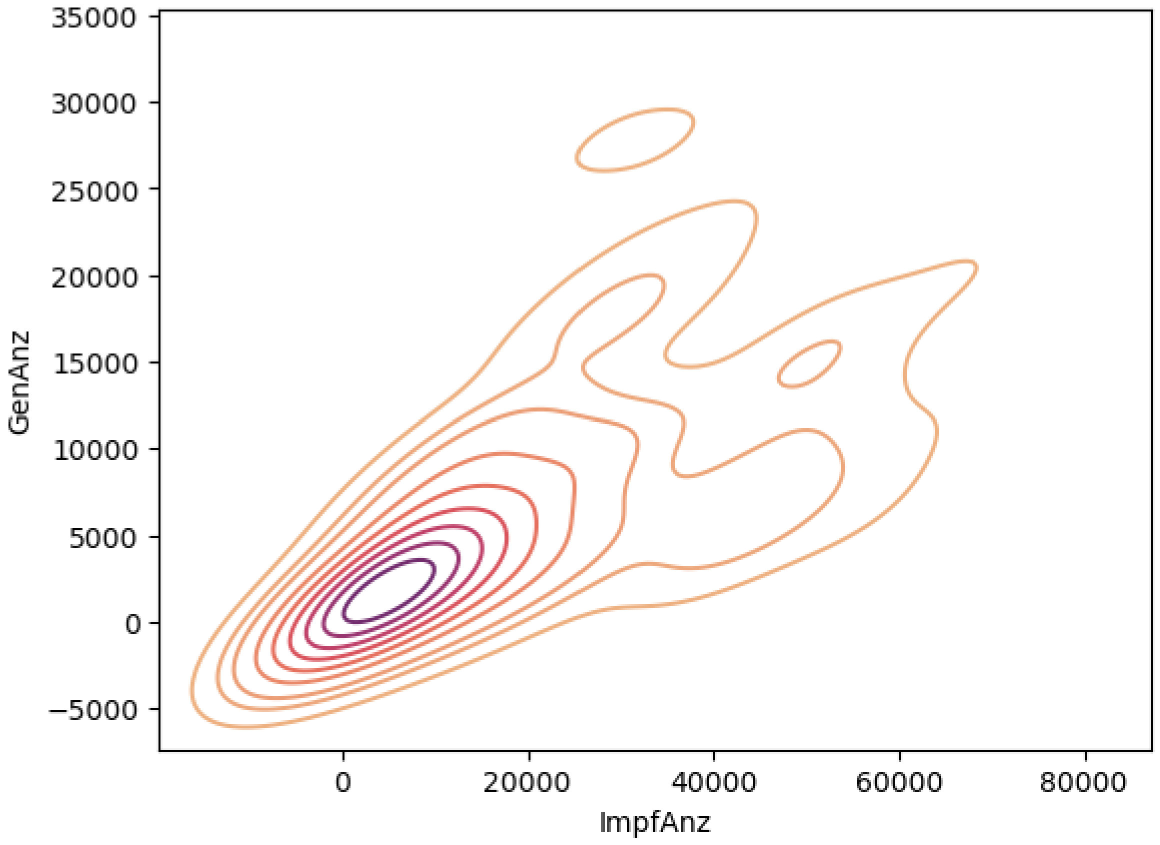
```
Out[55]: <Axes: xlabel='ImpfAnz', ylabel='GenAnz'>
```



This surely looks like our scatterplot in our last pairplot! Let's add a KDE to this one and see what it looks like

```
In [56]: sns.kdeplot(data=transKorr2, x="ImpfAnz", y="GenAnz", cmap=cont_palette)
```

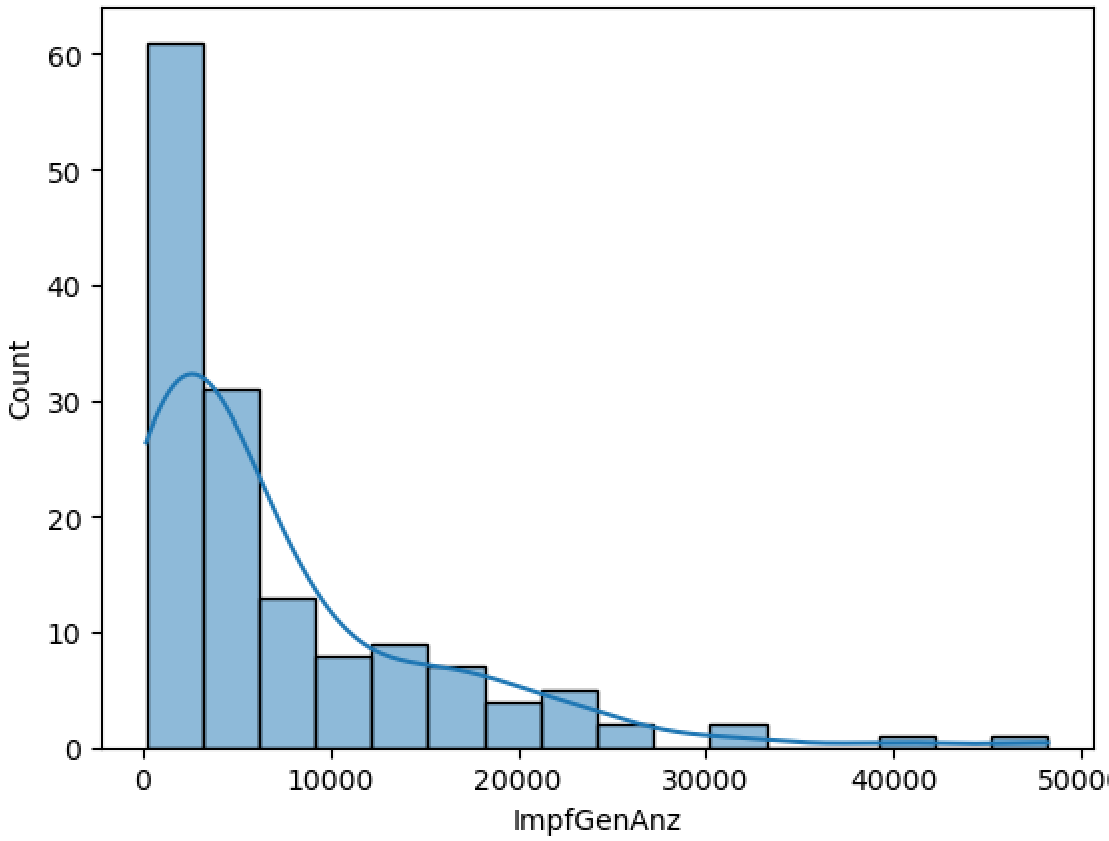
```
Out[56]: <Axes: xlabel='ImpfAnz', ylabel='GenAnz'>
```



Let's compare these graphs with the distribution of `ImpfGenAnz`. We should get approximately the same result, rigth?

In [57]:
```
sns.histplot(data=transKorr2, x="ImpfGenAnz", kde=True)
```

Out[57]:
```
<Axes: xlabel='ImpfGenAnz', ylabel='Count'>
```



Looking good!

Correlation (again)

I am not pleased with the correlation chapter above, since our values were so separated. Should we create another heatmap now? But first we need to transform our num2 dataframe as we did for korr2:

In [58]:
```
def transform_num2(row):
    result_row = row
    temp_df = num2.loc[(num2["Bundesland"] == row["Bundesland"]) &
                       (num2["Bildung"] == row["Bildung"]) &
                       (num2["Altersgr"] == row["Altersgr"]) &
                       (num2["ErwerbStatus"] == row["ErwerbStatus"])]

    result_row["ImpfAnz"] = temp_df.loc[temp_df["GeimpftGenesen"] == 1]["Anz"].sum(
    result_row["GenAnz"] = temp_df.loc[temp_df["GeimpftGenesen"] == 2]["Anz"].sum()
    result_row["ImpfGenAnz"] = temp_df.loc[temp_df["GeimpftGenesen"] == 3]["Anz"].s
    result_row["NichtsAnz"] = temp_df.loc[temp_df["GeimpftGenesen"] == 0]["Anz"].su

    result_row.drop(["GeimpftGenesen", "Anz"], inplace=True)
    return result_row

transNum2 = num2.apply(transform_num2, axis=1)
transNum2.drop_duplicates(inplace=True)
transNum2
```

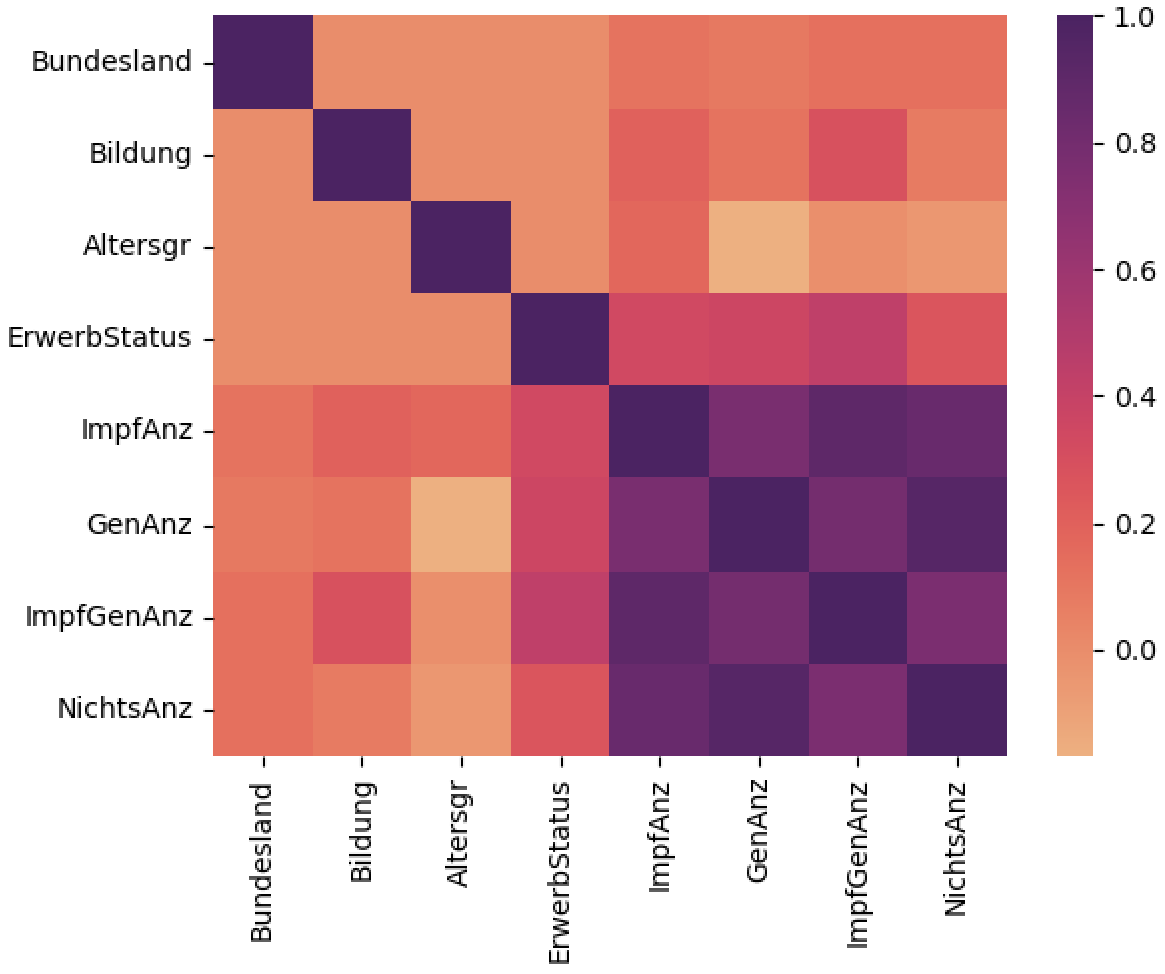
Out[58]:

	Bundesland	Bildung	Altersgr	ErwerbStatus	ImpfAnz	GenAnz	ImpfGenAnz	NichtsAnz
0	1.0	1.0	29.5	1.0	611.0	396.0	285.0	472.0
4	1.0	1.0	29.5	0.0	471.0	303.0	173.0	456.0
8	1.0	1.0	39.5	1.0	1170.0	532.0	597.0	531.0
12	1.0	1.0	39.5	0.0	549.0	253.0	199.0	425.0
16	1.0	1.0	49.5	1.0	2010.0	575.0	932.0	688.0
...	...	...	...	...	...	...	...	...
1384	9.0	0.0	39.5	0.0	46.0	23.0	15.0	47.0
1388	9.0	0.0	49.5	1.0	13.0	2.0	5.0	18.0
1392	9.0	0.0	49.5	0.0	37.0	12.0	6.0	44.0
1396	9.0	0.0	59.5	1.0	5.0	1.0	2.0	14.0
1400	9.0	0.0	59.5	0.0	31.0	4.0	1.0	16.0

360 rows × 8 columns

Looking sharp! Now create the heatmap for correlation and...

In [59]:
```
sns.heatmap(transNum2.corr(), cmap=cont_palette)
plt.show()
```



Haha! I knew it. We got something. Well the correlation between all the `Anz` -Features is logical: The more people there is in a certain group, the more our values are going to get bigger. Therefore, when a certain `Anz` -Feature is bigger, the othres are also bigger.

What I really found interesting here is, is that there is a slight correlation between `Bildung` (education) and `ImpfAnz` as well as `ImpfGenAnz`. The interesting factor that connects both of these features is that both of these person groups are vaccinated (or chose not to vaccinate because of various reasons). What if we add up both of these features? Will we get a bigger correlation? Let's find out!

In [60]:
```
transNum2["Bildung"].corr(transNum2["ImpfAnz"])
```
Out[60]:
```
0.19783150785477005
```

So this is the correlation between these two.

In [61]:
```
transNum2["Bildung"].corr(transNum2["ImpfGenAnz"])
```
Out[61]:
```
0.28602770613668754
```

Alright, now let's see the result we get when we combine these two:

In [62]:
```
transNum2["Bildung"].corr(transNum2["ImpfGenAnz"] + transNum2["ImpfAnz"])
```
Out[62]:
```
0.23316936946347674
```

Well that's kind of disappointing... Well at least I tried to prove our theory. Now we will look at something different: There is also a stronger correlation between `ErwerbStatus` and the `Anz` -Features (especially `ImpfGenAnz`). This could also be a hint that one is more likely to get vaccinated when they are economically active. We should inspect the certain correlation value:

```
In [63]: transNum2["ErwerbStatus"].corr(transNum2["ImpfGenAnz"])

Out[63]: 0.4331320158379753
```

And I don't think this correlation does not lack causality. To be able to be in the office without having to conduct COVID-tests every day, you needed to be either vaccinated or recovered from this disease.

Summary

We hereby are finished with our Corona-Data-Analysis. First, we introduced our data, which was published by Statistik Austria. Then, we cleaned and prepared our data for the future analysis. Through various visualization techniques, we analyzed our data and discussed results. After we realized that our data is not optimal for some visualization techniques, we transformed our data, which was actually not a part of the task assignment. But for the sake of having clear results, we did that, too and applied critical analysis, discussions, and assumptions.