In [1]:
```
import pandas as pd
import pyarrow.parquet as pq
import pyarrow as pa
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
import json
from typing import Union, List, Optional
import matplotlib.patches as mpatches
import scipy.stats as stats
import numpy as np
from scipy.stats import shapiro
import math
import matplotlib.ticker as ticker
import plotly.express as px

sns.set_palette("crest")
```

1. Laden der Datensätze

In der folgenden deskriptiven Analyse werden wir die aufbereiteten Ergebnisse der Umfrage verwenden. Zur vereinfachten Verarbeitung der Daten wurden die Fragen auf ihre Codes reduziert und die Antwortmöglichkeiten in numerischer Darstellung verwendet.

1.1 Utility Functions

Die Utility Funktionen werden verwendet, um die Fragen-Codes wieder in Text und die numerischen Antwortmöglichkeiten in den zugehörigen Text umzuwandeln.

In [2]:
```
def get_answer_text(question_code: str, answer_res: Union[float, int, str] =
    if isinstance(answer_res, str):
        _res = answer_res
    else:
        _res = int(answer_res) if answer_res != 0 else '<leer>'
    t_code = df_code[df_code["Variable"] == question_code]
    question_type = t_code["Fragetyp"].values[0]
    assert (answer_res is not None and question_type != 'Offene Frage') or (
    answer_dict = t_code["Werte"].values[0]
    out = answer_dict[str(_res)] if answer_res is not None else answer_dict
    return out
```

In [3]:
```
def add_new_line(i_str: str):
    lines = i_str.split(" ")
    print(lines)
    if len(lines) > 1:
        return " ".join(lines[:3]) + "\n" + " ".join(lines[3:])
    else:
        return lines[0]
```

In [4]:
```
add_new_line("4.5 Flache Hierarchien (auf Augenhöhe miteinander arbeiten)")
```

```
['4.5', 'Flache', 'Hierarchien', '(auf', 'Augenhöhe', 'miteinander', 'arbeite
n)']
```

Out[4]:
```
'4.5 Flache Hierarchien\n(auf Augenhöhe miteinander arbeiten)'
```

In [5]:
```
def get_question_text(question_code: str):
    return df_code[df_code["Variable"] == question_code]["Fragetext"].values
```

1.2 Daten

Laden des aufbereiteten Datensatzes im parquet Format.

In [6]:
```
results_pre_processed = pq.read_table('results_preprocessed.parquet')
```

In [7]:
```
df_yz = results_pre_processed.to_pandas()
```

In [8]:
```
df_yz.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 200 entries, 0 to 226
Data columns (total 56 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Bogen               200 non-null    int64
 1   60371597            200 non-null    int64
 2   60371598            200 non-null    int64
 3   60371599            200 non-null    int64
 4   60371600            200 non-null    int64
 5   60371601            200 non-null    int64
 6   60371602            200 non-null    int64
 7   60371603            200 non-null    int64
 8   60371604            200 non-null    int64
 9   60371605            200 non-null    int64
 10  60371606            200 non-null    int64
 11  60371607            200 non-null    int64
 12  60371608            200 non-null    int64
 13  60371609            200 non-null    int64
 14  60371612            200 non-null    object
 15  60371613            200 non-null    int64
 16  60371614            200 non-null    int64
 17  60371615            200 non-null    int64
 18  60371616            200 non-null    int64
 19  60371617            200 non-null    int64
 20  60371618            200 non-null    int64
 21  60371619            200 non-null    int64
 22  60371620            200 non-null    int64
 23  60371624            200 non-null    int64
 24  60371625            200 non-null    int64
 25  60371626            200 non-null    int64
 26  60371627            200 non-null    int64
 27  60371628            200 non-null    int64
 28  60371629            200 non-null    int64
 29  60371630            200 non-null    int64
 30  60371631            200 non-null    int64
 31  60371632            200 non-null    int64
 32  60371633            200 non-null    int64
 33  60371634            200 non-null    int64
 34  60371635            200 non-null    int64
 35  60371636            200 non-null    int64
 36  60371637            200 non-null    int64
 37  60371638            200 non-null    int64
 38  60371639            200 non-null    int64
 39  60371640            200 non-null    int64
 40  60371641            200 non-null    int64
 41  60371642            200 non-null    int64
 42  60371643            200 non-null    int64
 43  60371644            200 non-null    int64
 44  60371645            200 non-null    int64
 45  60371646            200 non-null    int64
 46  60371647            200 non-null    int64
 47  60371648            200 non-null    int64
 48  60371649            200 non-null    int64
 49  60371650            200 non-null    float64
 50  60371651            200 non-null    int64
```

```
 51  60371652            200 non-null    int64
 52  60371653            200 non-null    float64
 53  60371654            200 non-null    float64
 54  Zeitstempel         200 non-null    object
 55  Datensatz-Ursprung  200 non-null    object
dtypes: float64(3), int64(50), object(3)
memory usage: 89.1+ KB
```

In [9]:
```
df_yz.head()
```

Out[9]:

	Bogen	60371597	60371598	60371599	60371600	60371601	60371602	60371
0	1	4	4	5	4	4	5	
1	2	4	5	5	5	5	4	
2	3	4	2	5	4	5	4	
3	4	5	4	5	4	4	4	
4	5	5	4	5	3	5	5	

5 rows × 56 columns

Laden der Fragen-Code Zuordnung

In [10]:
```
with open('code.json', 'r') as f:
  code_data = json.load(f)
```

In [11]:
```
df_code = pd.DataFrame(code_data)
codes = df_code["Variable"].tolist()
```

1.3 Gliederung der Fragetypen

Die Umfrage enthält 3 Fragetypen.

- Skalafragen - Antwortmöglichkeiten werden mittels kategorischen numerischen Werten auf einer Ordinalskala dargestellt.
- 1 aus n - Antwortmöglichkeiten werden mittels kategorischen numerischen Werten auf einer Nominalskala dargestellt.
- Offene Frage - Antwort besteht aus einem Freitext

In [12]:
```
skala_ids = df_code[df_code["Fragetyp"] == "Skalafrage"]["Variable"].tolist(
one_of_n_ids = df_code[df_code["Fragetyp"] == "1 aus n"]["Variable"].tolist(
numeric_ids = skala_ids + one_of_n_ids
open_q_ids = df_code[df_code["Fragetyp"] == "Offene Frage"]["Variable"].toli
```

Neben den fachlichen Fragen der Umfrage, gibt es auch Fragetypen, welche zur Klassifierung der Umfrage Ergebnisse dienen.

- Bitte kreuze an, welcher Generation du zugehörig bist
- Welchem Geschlecht fühlst du dich zugehörig?
- Bitte wähle deinen höchsten formalen Bildungsabschluss aus:
- Bist du derzeit berufstätig?
- In welcher Beschäftigungsform befindest du dich?
- In welchem Wirtschaftssektor bist du beschäftigt?

In [13]:
```
generation_col = '60371649'
gender_col = '60371650'
academic_col = '60371651'
working_col = '60371652'
work_form_col = '60371653'
economic_sector_col = '60371654'
```

Zur weiteren Einschränkung der Fragetypen, können wir die Überschneidungen zwischen "fachlichen" Fragen und "gruppierenden" Fragen bestimmen und für weitere mögliche Auswertungen zwischen den beiden Typen unterscheiden.

In [15]:
```
skala_fach_ids = [x for x in skala_ids if int(x) <= 60371648]
one_of_n_fach_ids = [x for x in one_of_n_ids if int(x) <= 60371648]
num_fach_cols = skala_fach_ids + one_of_n_fach_ids

print(skala_fach_ids)
print(one_of_n_fach_ids)
```

```
['60371597', '60371598', '60371599', '60371600', '60371601', '60371602', '603
71603', '60371604', '60371605', '60371606', '60371607', '60371608', '6037160
9', '60371615', '60371616', '60371617', '60371618', '60371619', '60371620',
'60371624', '60371625', '60371626', '60371627', '60371628', '60371629', '6037
1630', '60371631', '60371632', '60371633', '60371634', '60371635', '6037163
6', '60371637', '60371638', '60371639', '60371640', '60371641', '60371642',
'60371643', '60371644', '60371645', '60371646', '60371647', '60371648']
['60371613', '60371614']
```

Darüberhinaus gliedert sich die Umfrage in 2 Fragen-Cluster:

- Fragen rund um die Erwartungen der Probanten an einen Arbeitsplatz sowie Arbeitgebermerkmalen
- Fragen an persönliche Wertevorstellungen der Probanten

In [16]:
```
erwartungen_ids = [x for x in codes if 60371597 <= int(x) and int(x) <= 6037
ag_merkmale_ids = [x for x in codes if 60371615 <= int(x) and int(x) <= 6037
print(erwartungen_ids)
```

```
['60371597', '60371598', '60371599', '60371600', '60371601', '60371602', '603
71603', '60371604', '60371605', '60371606', '60371607', '60371608', '6037160
9']
```

In [17]:
```
werte_ids = [x for x in codes if 60371624 <= int(x) and int(x) <= 60371648]
```

```
familie_ids = ["60371641", "60371640", "60371634", "60371635", "60371638", "
leben_ids = ["60371643", "60371646", "60371644", "60371642", "60371647", "60
tugend_ids = ["60371624", "60371628", "60371631", "60371638"]
toleranz_ids = ["60371633", "60371629", "60371632", "60371648", "60371627"]
macht_ids = ["60371626", "60371625", "60371630", "60371642"]
tradition_ids = ["60371637", "60371639", "60371636"]

print(werte_ids)
```

```
['60371624', '60371625', '60371626', '60371627', '60371628', '60371629', '603
71630', '60371631', '60371632', '60371633', '60371634', '60371635', '6037163
6', '60371637', '60371638', '60371639', '60371640', '60371641', '60371642',
'60371643', '60371644', '60371645', '60371646', '60371647', '60371648']
```

2. Deskriptive Beschreibung der Stichproben Gesamtheit

In einem ersten Schritt betrachten wir generelle statistsiche Kennzahlen wie mean, std, quantile,... zu unseren Umfrage Ergebnissen.

In [18]:
```
desc = df_yz.describe()
display(desc)
```

	Bogen	60371597	60371598	60371599	60371600	60371601	6
count	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200
mean	112.595000	3.990000	3.660000	4.485000	3.655000	4.695000	
std	67.286987	0.593008	0.958642	0.743329	0.932913	0.482846	
min	1.000000	2.000000	1.000000	1.000000	1.000000	3.000000	
25%	52.750000	4.000000	3.000000	4.000000	3.000000	4.000000	
50%	111.500000	4.000000	4.000000	5.000000	4.000000	5.000000	
75%	172.500000	4.000000	4.000000	5.000000	4.000000	5.000000	
max	227.000000	5.000000	5.000000	5.000000	5.000000	5.000000	

8 rows × 53 columns

In [19]:
```
desc_y = df_yz[df_yz[generation_col]==1].describe()
display(desc_y)
```

	Bogen	60371597	60371598	60371599	60371600	60371601	603
count	108.000000	108.000000	108.000000	108.000000	108.000000	108.000000	108.0
mean	89.037037	4.027778	3.518519	4.416667	3.583333	4.601852	4.(
std	57.690914	0.603081	0.980954	0.737830	0.958240	0.510448	0.8
min	2.000000	2.000000	1.000000	2.000000	1.000000	3.000000	2.(
25%	44.750000	4.000000	3.000000	4.000000	3.000000	4.000000	4.(
50%	81.000000	4.000000	4.000000	5.000000	4.000000	5.000000	4.(
75%	119.500000	4.000000	4.000000	5.000000	4.000000	5.000000	5.(
max	225.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.(

8 rows × 53 columns

In [20]:
```
desc_z = df_yz[df_yz[generation_col]==2].describe()
display(desc_z)
```

	Bogen	60371597	60371598	60371599	60371600	60371601	6037160
count	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.00000(
mean	140.250000	3.945652	3.826087	4.565217	3.739130	4.804348	4.34782(
std	67.495157	0.581097	0.909156	0.745748	0.900178	0.425535	0.60100
min	1.000000	2.000000	1.000000	1.000000	1.000000	3.000000	3.00000(
25%	106.750000	4.000000	3.000000	4.000000	3.000000	5.000000	4.00000(
50%	157.500000	4.000000	4.000000	5.000000	4.000000	5.000000	4.00000(
75%	192.250000	4.000000	4.000000	5.000000	4.000000	5.000000	5.00000(
max	227.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.00000(

8 rows × 53 columns

In [21]:
```
desc.to_excel("result_description.xlsx")
desc_y.to_excel("result_description_gen_y.xlsx")
desc_z.to_excel("result_description_gen_z.xlsx")
```

2.1 Aufbereitung der Umfrage Ergebnisse

Wir betrachten die genaue Verteilung der gegeben Antworten unserer Stichprobe.

In [23]:
```
df_yz_count = df_yz[df_yz.columns.difference(['Bogen', 'Zeitstempel', 'Daten
df_yz_percentage = df_yz[df_yz.columns.difference(['Bogen', 'Zeitstempel', '
```

In [24]:
```
df_y = df_yz[df_yz[generation_col]==1]
df_z = df_yz[df_yz[generation_col]==2]
```

In [25]:
```
df_y_count = df_y[df_y.columns.difference(['Bogen', 'Zeitstempel', 'Datensat
df_y_percentage = df_y[df_y.columns.difference(['Bogen', 'Zeitstempel', 'Dat

df_z_count = df_z[df_z.columns.difference(['Bogen', 'Zeitstempel', 'Datensat
df_z_percentage = df_z[df_z.columns.difference(['Bogen', 'Zeitstempel', 'Dat
```

In [26]:
```
df_out = pd.concat([df_yz_count, df_yz_percentage, df_y_count, df_y_percenta
df_out = df_out.rename(columns={0: "Anzahl Gesamt", 1: "Anteil Gesamt", 2: "
```

In [27]:
```
df_out.head()
```

Out[27]:

		Anzahl Gesamt	Anteil Gesamt	Anzahl Generation Y	Anteil Generation Y	Anzahl Generation Z	Anteil Generation Z
60371597	**2**	7.0	0.035	3.0	0.027778	4.0	0.043478
	3	15.0	0.075	9.0	0.083333	6.0	0.065217
	4	151.0	0.755	78.0	0.722222	73.0	0.793478
	5	27.0	0.135	18.0	0.166667	9.0	0.097826
60371598	**1**	3.0	0.015	2.0	0.018519	1.0	0.010870

In [27]:
```
df_out.to_excel("results_wo_missing.xlsx")
```

2.2 Auswertung der Durchschnitte

Wir bilden die Mittelwerte der Fragen und stellen diese grafisch dar.

Darüber hinaus bilden wir neue Fragencluster, welche die persönlichen Wertevorstellungen der Probanten feiner untergliedern. Dabei bilden wir pro Fragencluster wieder den Mittelwert über die zugehörigen Fragen zu diesem Cluster.

In [28]:
```
df_yz["tradition"] = df_yz[tradition_ids].mean(axis=1)
df_yz["familie"] = df_yz[familie_ids].mean(axis=1)
df_yz["leben"] = df_yz[leben_ids].mean(axis=1)
df_yz["tugend"] = df_yz[tugend_ids].mean(axis=1)
df_yz["toleranz"] = df_yz[toleranz_ids].mean(axis=1)
df_yz["macht"] = df_yz[macht_ids].mean(axis=1)
```

In [29]:
```
mean_z=df_yz[df_yz[generation_col]==2].mean(numeric_only=True)
mean_y=df_yz[df_yz[generation_col]==1].mean(numeric_only=True)

df_mean = pd.concat([mean_y, mean_z], axis=1)
df_mean["mean_diff"] = df_mean[0] - df_mean[1]
df_mean["mean_diff_abs"] = abs(df_mean[0] - df_mean[1])
```

In [30]:
```
df_mean.sort_values(by=['mean_diff'])
df_mean = df_mean.rename(columns={0: "gen_y", 1: "gen_z"})
display(df_mean)
```

	gen_y	gen_z	mean_diff	mean_diff_abs
Bogen	89.037037	140.250000	-51.212963	51.212963
60371597	4.027778	3.945652	0.082126	0.082126
60371598	3.518519	3.826087	-0.307568	0.307568
60371599	4.416667	4.565217	-0.148551	0.148551
60371600	3.583333	3.739130	-0.155797	0.155797
60371601	4.601852	4.804348	-0.202496	0.202496
60371602	4.037037	4.347826	-0.310789	0.310789
60371603	4.074074	4.282609	-0.208535	0.208535
60371604	3.018519	3.423913	-0.405395	0.405395
60371605	3.833333	3.891304	-0.057971	0.057971
60371606	3.268519	3.793478	-0.524960	0.524960
60371607	4.037037	4.336957	-0.299919	0.299919
60371608	4.435185	4.391304	0.043881	0.043881
60371609	4.240741	3.760870	0.479871	0.479871
60371613	6.342593	6.478261	-0.135668	0.135668
60371614	7.074074	7.858696	-0.784622	0.784622
60371615	2.472222	2.304348	0.167874	0.167874
60371616	3.481481	3.597826	-0.116345	0.116345
60371617	3.962963	4.032609	-0.069646	0.069646
60371618	3.370370	3.510870	-0.140499	0.140499
60371619	3.916667	4.086957	-0.170290	0.170290
60371620	3.379630	3.510870	-0.131240	0.131240
60371624	3.898148	3.869565	0.028583	0.028583
60371625	3.907407	3.902174	0.005233	0.005233
60371626	2.416667	2.586957	-0.170290	0.170290
60371627	3.416667	3.684783	-0.268116	0.268116
60371628	3.981481	4.054348	-0.072866	0.072866
60371629	3.185185	3.250000	-0.064815	0.064815
60371630	3.111111	2.836957	0.274155	0.274155
60371631	3.666667	4.021739	-0.355072	0.355072
60371632	3.425926	3.597826	-0.171900	0.171900
60371633	2.601852	2.510870	0.090982	0.090982
60371634	4.194444	4.380435	-0.185990	0.185990

	gen_y	gen_z	mean_diff	mean_diff_abs
60371635	4.481481	4.478261	0.003221	0.003221
60371636	2.009259	1.978261	0.030998	0.030998
60371637	2.018519	1.858696	0.159823	0.159823
60371638	4.416667	4.369565	0.047101	0.047101
60371639	2.148148	2.195652	-0.047504	0.047504
60371640	4.620370	4.782609	-0.162238	0.162238
60371641	4.564815	4.804348	-0.239533	0.239533
60371642	3.324074	3.478261	-0.154187	0.154187
60371643	3.907407	3.956522	-0.049114	0.049114
60371644	3.500000	3.663043	-0.163043	0.163043
60371645	3.981481	4.195652	-0.214171	0.214171
60371646	3.111111	3.097826	0.013285	0.013285
60371647	1.703704	1.934783	-0.231079	0.231079
60371648	4.138889	4.423913	-0.285024	0.285024
60371649	1.000000	2.000000	-1.000000	1.000000
60371650	1.472222	1.706522	-0.234300	0.234300
60371651	5.129630	4.750000	0.379630	0.379630
60371652	1.018519	1.173913	-0.155395	0.155395
60371653	1.333333	1.065217	0.268116	0.268116
60371654	2.546296	2.239130	0.307166	0.307166
tradition	2.058642	2.010870	0.047772	0.047772
familie	4.376543	4.501812	-0.125268	0.125268
leben	3.280864	3.425725	-0.144860	0.144860
tugend	3.990741	4.078804	-0.088064	0.088064
toleranz	3.353704	3.493478	-0.139775	0.139775
macht	3.189815	3.201087	-0.011272	0.011272

In [31]:
```
df_mean.to_excel("result_mean_diff.xlsx")
```

Für die weitere Verarbeitung wird die Spalte "Bogen" aus dem Datensatz entfernt. Diese enthält keine sinnvoll auszuwertenden Daten.

In [32]:
```
df_mean = df_mean.drop('Bogen')
```

Die Mittelwert Berechungen werden wieder in die Fragen Cluster Werte und AG-

Erwartungen aufgeteilt.

In [33]:
```
df_mean_werte = df_mean.filter(items=werte_ids, axis=0)
```

In [34]:
```
df_mean_erwartungen = df_mean.filter(items=erwartungen_ids, axis=0)
```

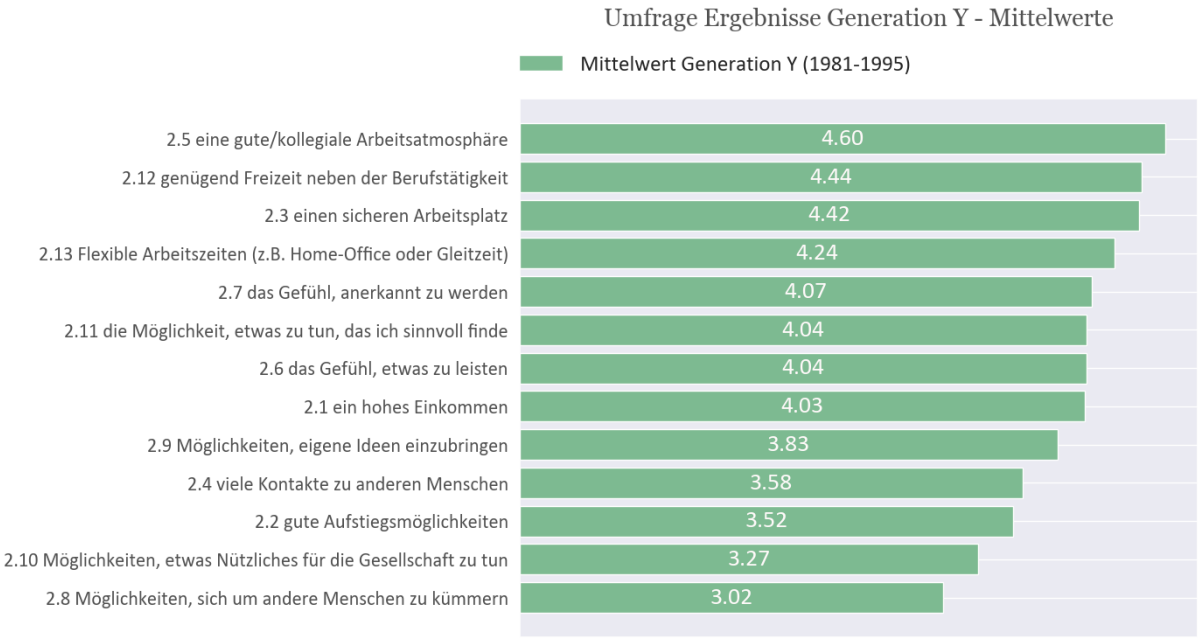
In [39]:
```
df_mean_ag_merkmale = df_mean.filter(items=ag_merkmale_ids, axis=0)
```

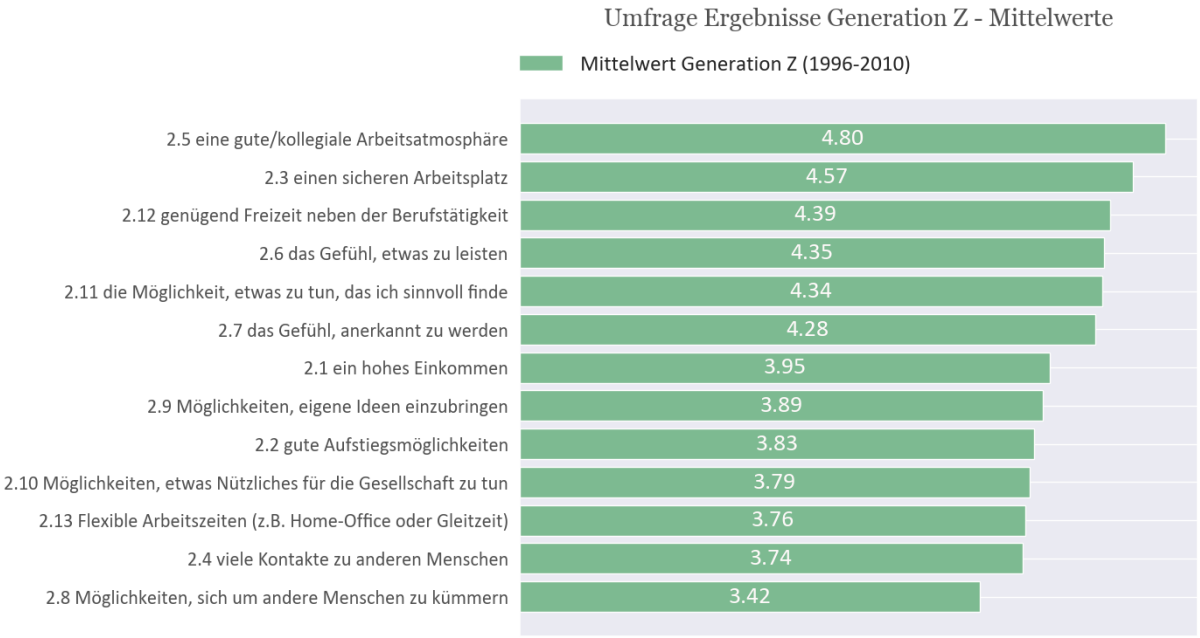
Die Ergebnisse werden grafisch aufbereitet

In [44]:
```
def plot_means(input_df: pd.DataFrame, gen: str, title: str, fig_size:tuple,
    fig, ax = plt.subplots(figsize=fig_size)

    gen_matching = {"gen_y": "Generation Y (1981-1995)",
                    "gen_z": "Generation Z (1996-2010)"}

    font_color = '#525252'
    csfont = {'fontname':'Georgia'} # title font
    hfont = {'fontname':'Calibri'} # main font
    sns.set()
    sns.set_palette("crest")


    title = plt.title(title, pad=60, fontsize=20, color=font_color, **csfont

    input_df.sort_values(by=[gen], inplace=True)

    ax.barh(input_df.index, input_df[gen], align='center', label=f"Mittelwer

    plt.xticks(color=font_color, **hfont)
    plt.yticks(color=font_color, **hfont)

    ax.axes.get_xaxis().set_visible(False)
    if not agg:
        ax.set_yticks(input_df.index, labels=[get_question_text(code) for co
    else:
        ax.set_yticks(input_df.index, labels=input_df.index, fontsize=16)
    ax.bar_label(ax.containers[0], label_type="center", color='snow', fmt='{

    # Get the first two and last y-tick positions.
    miny, nexty, *_, maxy = ax.get_yticks()

    # Compute half the y-tick interval (for example).
    eps = (nexty - miny) / 2  # <-- Your choice.

    legend = plt.legend(loc='center',
                frameon=False,
                bbox_to_anchor=legend_anchor,
                mode='expand',
                ncol=3,
                borderaxespad=-.46,
                prop={'size': 18, 'family':'Calibri'})
```

In [45]:
```
plot_means(df_mean_erwartungen,
```

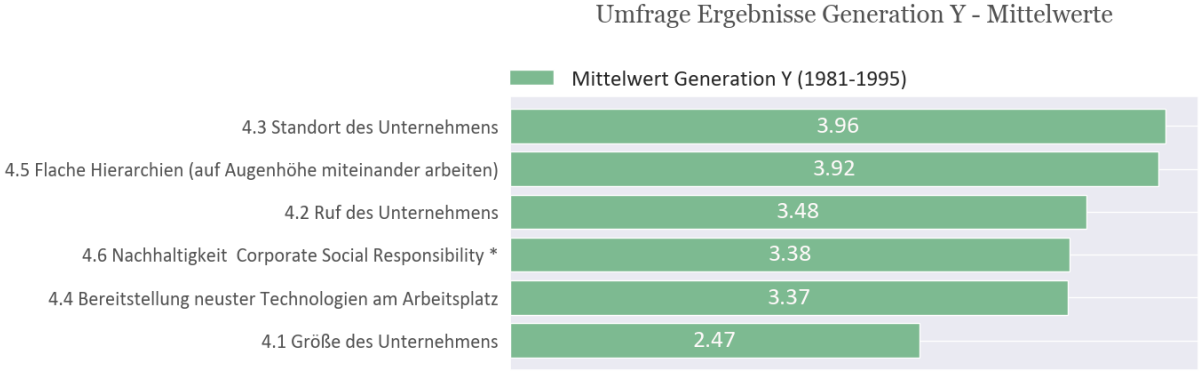
```
            "gen_y",
            "Umfrage Ergebnisse Generation Y – Mittelwerte",
            (10,8),
            (0., 1.012, 1., .102))
plot_means(df_mean_erwartungen,
            "gen_z",
            "Umfrage Ergebnisse Generation Z – Mittelwerte",
            (10,8),
           (0., 1.012, 1., .102))
```

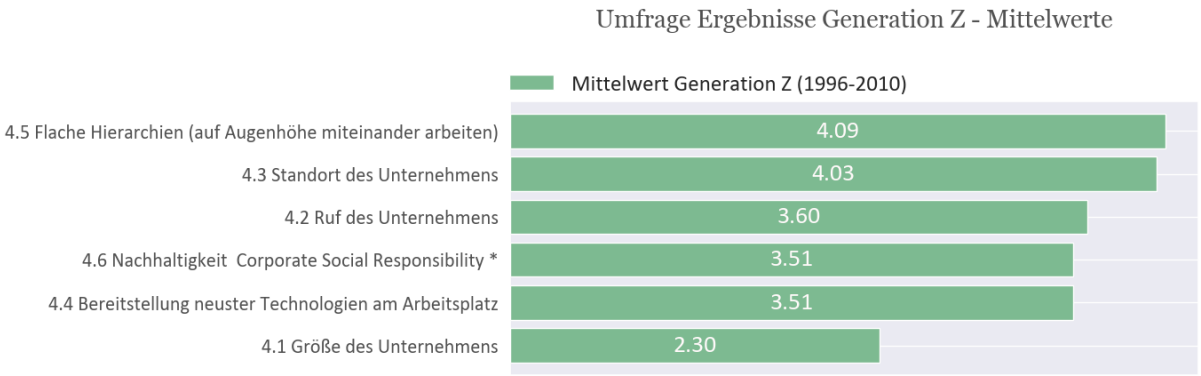




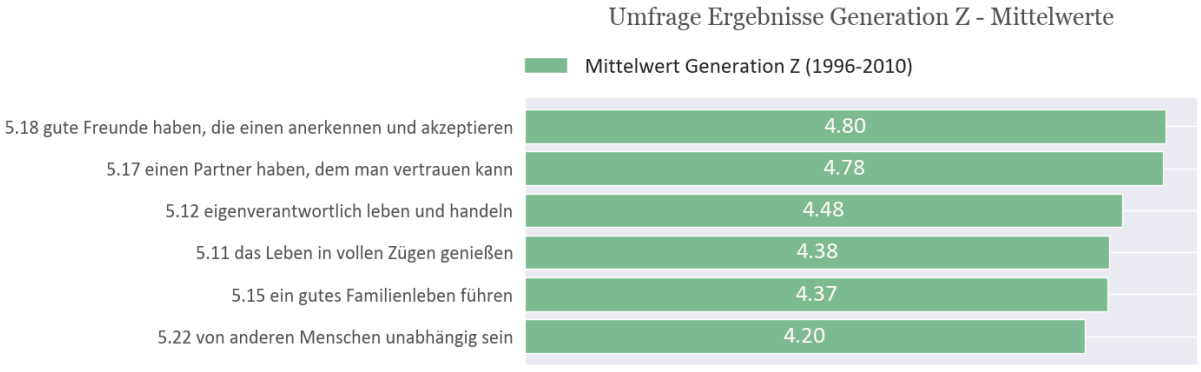
In [46]:
```
plot_means(df_mean_ag_merkmale,
            "gen_y",
            "Umfrage Ergebnisse Generation Y – Mittelwerte",
            (10,4),
            (0., 1.012, 1., .102))
plot_means(df_mean_ag_merkmale,
            "gen_z",
```

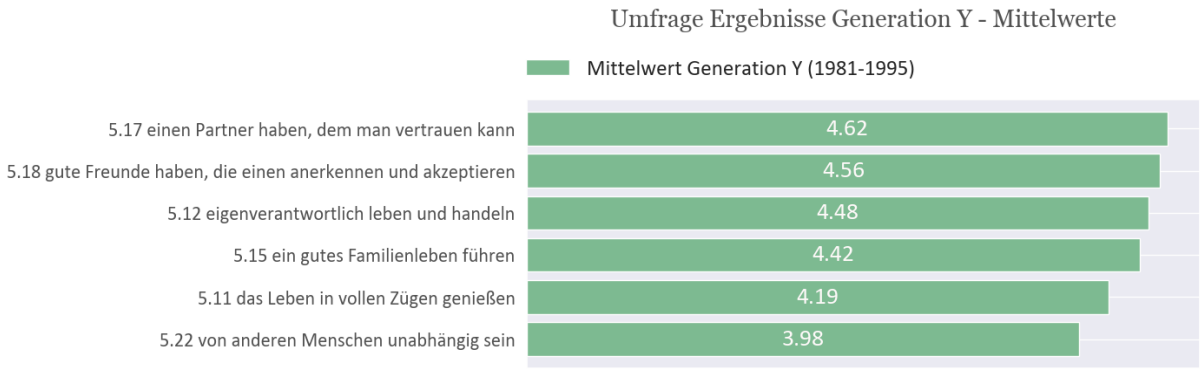
```
            "Umfrage Ergebnisse Generation Z – Mittelwerte",
            (10,4),
            (0., 1.012, 1., .102))
```



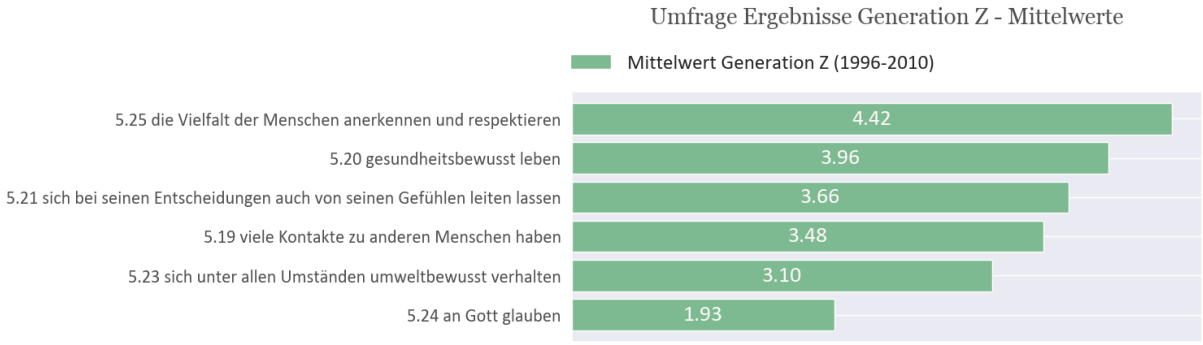


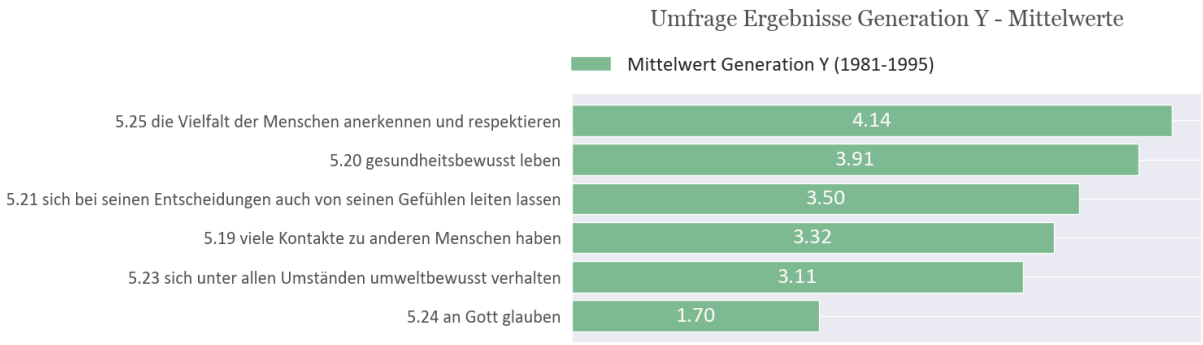
In [47]:
```
plot_means(df_mean.filter(items=familie_ids, axis=0),
            "gen_z",
            "Umfrage Ergebnisse Generation Z – Mittelwerte",
            (10,4),
            (0., 1.062, 1., .102))
plot_means(df_mean.filter(items=familie_ids, axis=0),
            "gen_y",
            "Umfrage Ergebnisse Generation Y – Mittelwerte",
            (10,4),
            (0., 1.062, 1., .102))
```



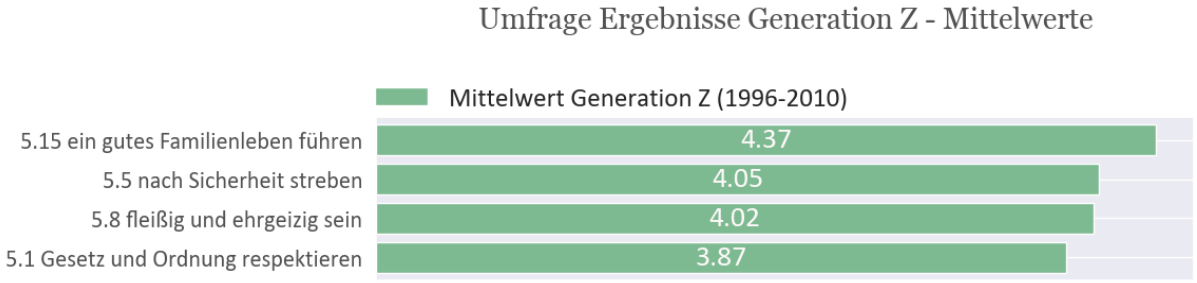


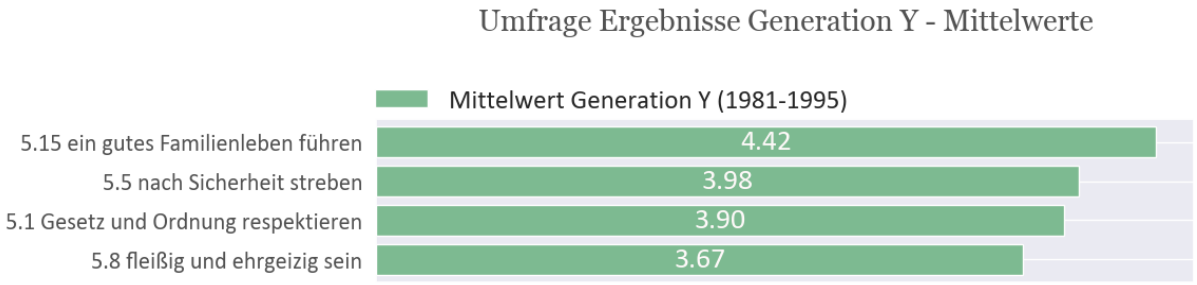
In [48]:
```
plot_means(df_mean.filter(items=leben_ids, axis=0),
           "gen_z",
           "Umfrage Ergebnisse Generation Z - Mittelwerte",
           (10,4),
           (0., 1.062, 1., .102))
plot_means(df_mean.filter(items=leben_ids, axis=0),
           "gen_y",
           "Umfrage Ergebnisse Generation Y - Mittelwerte",
           (10,4),
           (0., 1.062, 1., .102))
```



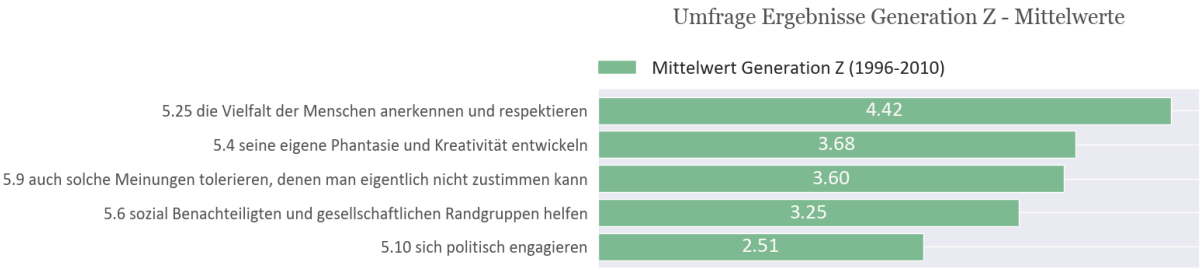


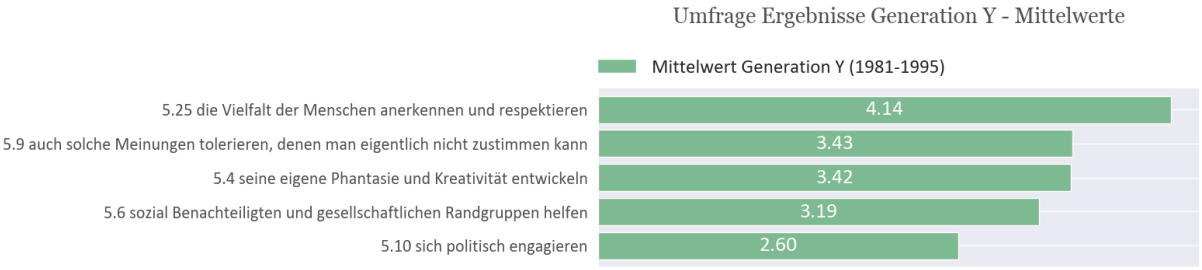
In [49]:
```
plot_means(df_mean.filter(items=tugend_ids, axis=0),
           "gen_z",
           "Umfrage Ergebnisse Generation Z - Mittelwerte",
           (10,2),
           (0., 1.062, 1., .102))
plot_means(df_mean.filter(items=tugend_ids, axis=0),
           "gen_y",
           "Umfrage Ergebnisse Generation Y - Mittelwerte",
           (10,2),
           (0., 1.062, 1., .102))
```





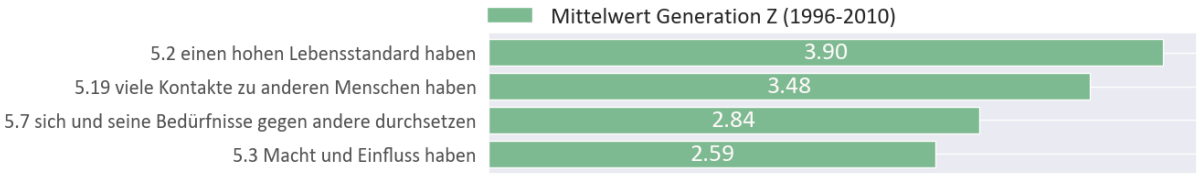
In [50]:
```
plot_means(df_mean.filter(items=toleranz_ids, axis=0),
           "gen_z",
           "Umfrage Ergebnisse Generation Z – Mittelwerte",
           (10,3),
           (0., 1.062, 1., .102))
plot_means(df_mean.filter(items=toleranz_ids, axis=0),
           "gen_y",
           "Umfrage Ergebnisse Generation Y – Mittelwerte",
           (10,3),
           (0., 1.062, 1., .102))
```

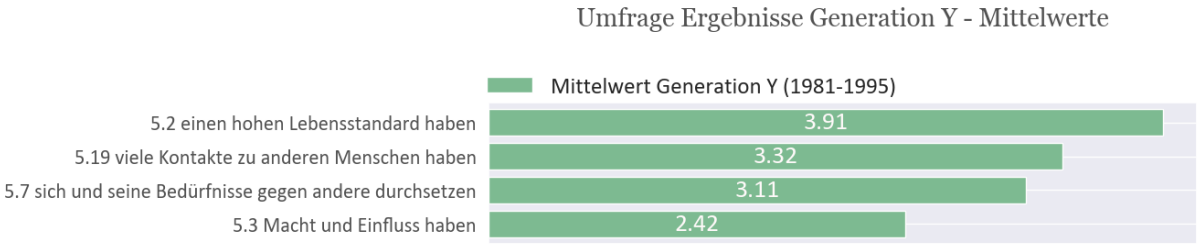




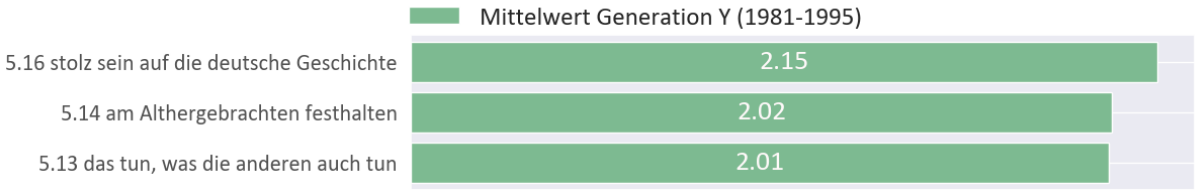
In [51]:
```
plot_means(df_mean.filter(items=macht_ids, axis=0),
           "gen_z",
           "Umfrage Ergebnisse Generation Z – Mittelwerte",
           (10,2),
           (0., 1.062, 1., .102))
plot_means(df_mean.filter(items=macht_ids, axis=0),
           "gen_y",
           "Umfrage Ergebnisse Generation Y – Mittelwerte",
```

```
            (10,2),
            (0., 1.062, 1., .102))
```



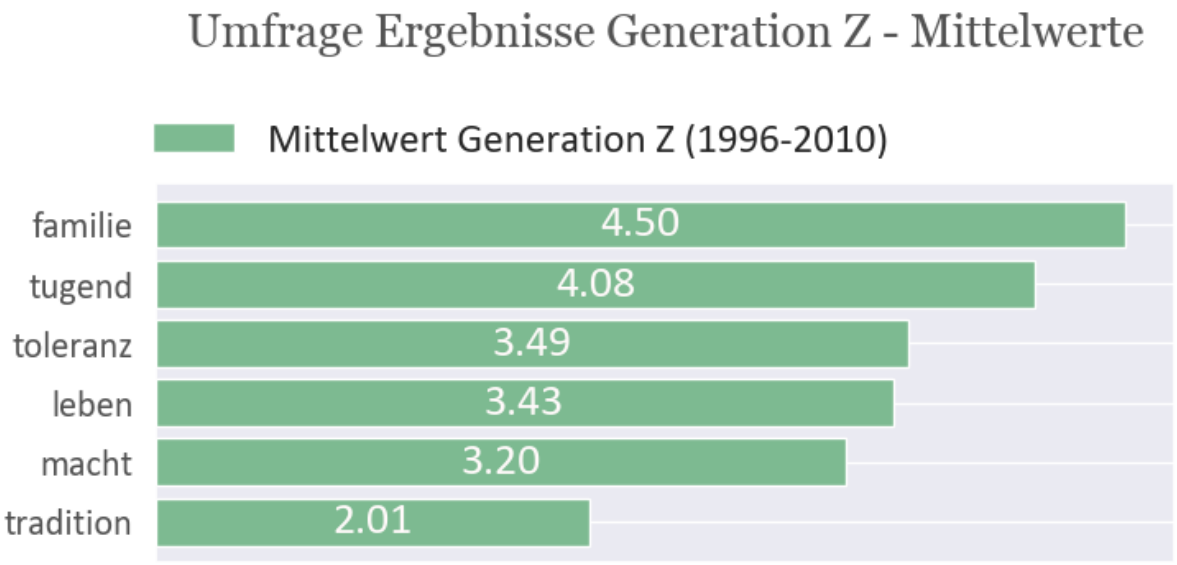


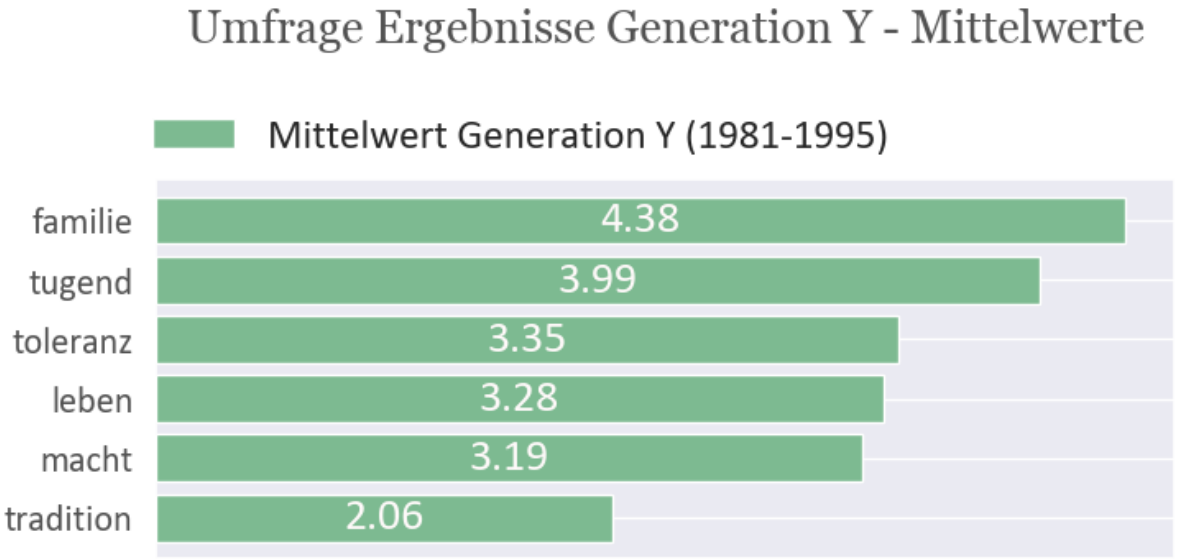
In [52]:
```
plot_means(df_mean.filter(items=tradition_ids, axis=0),
           "gen_z",
           "Umfrage Ergebnisse Generation Z – Mittelwerte",
           (10,2),
           (0., 1.062, 1., .102))
plot_means(df_mean.filter(items=tradition_ids, axis=0),
           "gen_y",
           "Umfrage Ergebnisse Generation Y – Mittelwerte",
           (10,2),
           (0., 1.062, 1., .102))
```



In [53]:
```
plot_means(df_mean.filter(items=["tradition", "familie", "leben", "tugend",
           "gen_z",
           "Umfrage Ergebnisse Generation Z – Mittelwerte",
           (8,3),
           (0., 1.062, 1., .102),
          True)
plot_means(df_mean.filter(items=["tradition", "familie", "leben", "tugend",
           "gen_y",
```

```
            "Umfrage Ergebnisse Generation Y - Mittelwerte",
            (8,3),
            (0., 1.062, 1., .102),
           True)
```





In [54]:
```
df_mean.filter(items=tradition_ids, axis=0)
```

Out[54]:

	gen_y	gen_z	mean_diff	mean_diff_abs
60371637	2.018519	1.858696	0.159823	0.159823
60371639	2.148148	2.195652	-0.047504	0.047504
60371636	2.009259	1.978261	0.030998	0.030998

In [55]:
```
df_mean_werte
```

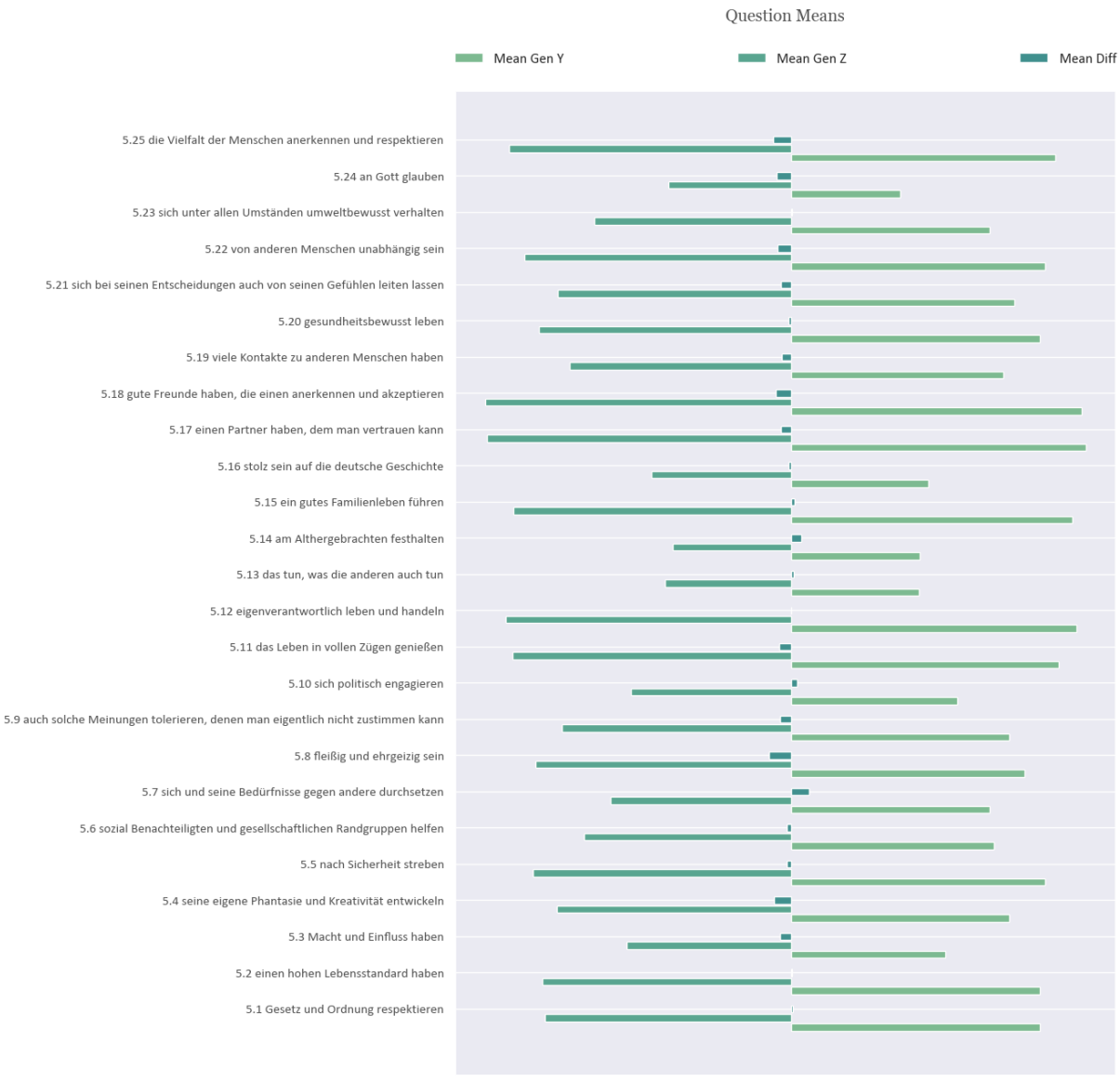
Out[55]:

	gen_y	gen_z	mean_diff	mean_diff_abs
60371624	3.898148	3.869565	0.028583	0.028583
60371625	3.907407	3.902174	0.005233	0.005233
60371626	2.416667	2.586957	-0.170290	0.170290
60371627	3.416667	3.684783	-0.268116	0.268116
60371628	3.981481	4.054348	-0.072866	0.072866
60371629	3.185185	3.250000	-0.064815	0.064815
60371630	3.111111	2.836957	0.274155	0.274155
60371631	3.666667	4.021739	-0.355072	0.355072
60371632	3.425926	3.597826	-0.171900	0.171900
60371633	2.601852	2.510870	0.090982	0.090982
60371634	4.194444	4.380435	-0.185990	0.185990
60371635	4.481481	4.478261	0.003221	0.003221
60371636	2.009259	1.978261	0.030998	0.030998
60371637	2.018519	1.858696	0.159823	0.159823
60371638	4.416667	4.369565	0.047101	0.047101
60371639	2.148148	2.195652	-0.047504	0.047504
60371640	4.620370	4.782609	-0.162238	0.162238
60371641	4.564815	4.804348	-0.239533	0.239533
60371642	3.324074	3.478261	-0.154187	0.154187
60371643	3.907407	3.956522	-0.049114	0.049114
60371644	3.500000	3.663043	-0.163043	0.163043
60371645	3.981481	4.195652	-0.214171	0.214171
60371646	3.111111	3.097826	0.013285	0.013285
60371647	1.703704	1.934783	-0.231079	0.231079
60371648	4.138889	4.423913	-0.285024	0.285024

In [57]:

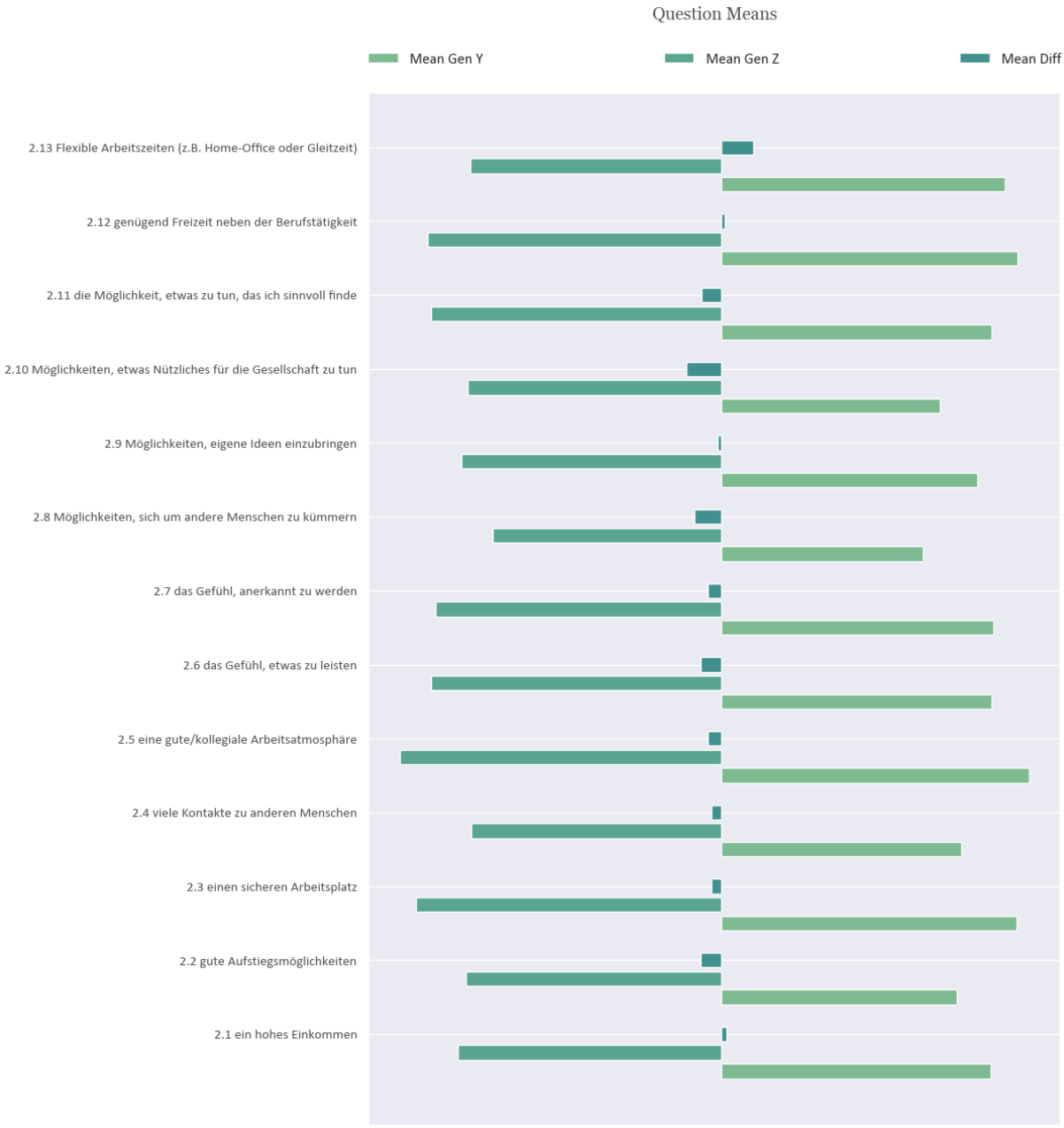
```
fig, ax = plt.subplots(figsize=(10, 15))

font_color = '#525252'
csfont = {'fontname':'Georgia'} # title font
hfont = {'fontname':'Calibri'} # main font
colors = ['#f47e7a', '#b71f5c', '#621237', '#dbbaa7', '#df4382', '#c58f71']

title = plt.title("Question Means", pad=60, fontsize=14, color=font_color, *

y_pos_y = np.arange(len(df_mean_werte.index)) * 4
```

```
y_pos_z = np.arange(len(df_mean_werte.index)) * 4 + 1

y_pos_diff = np.arange(len(df_mean_werte.index)) * 4 + 2

ax.barh(y_pos_y, df_mean_werte["gen_y"], align='center', label="Mean Gen Y")
ax.barh(y_pos_z, -1 * df_mean_werte["gen_z"], align='center', label="Mean Ge
ax.barh(y_pos_diff, df_mean_werte["mean_diff"], align='center', label="Mean

plt.xticks(color=font_color, **hfont)
plt.yticks(color=font_color, **hfont)

ax.axes.get_xaxis().set_visible(False)
ax.set_yticks(y_pos_diff, labels=[get_question_text(code) for code in df_mea

legend = plt.legend(loc='center',
           frameon=False,
           bbox_to_anchor=(0., .982, 1., .102),
           mode='expand',
           ncol=3,
           borderaxespad=-.46,
           prop={'size': 12, 'family':'Calibri'})
```



In [46]:

```
fig, ax = plt.subplots(figsize=(10, 15))

font_color = '#525252'
csfont = {'fontname':'Georgia'} # title font
hfont = {'fontname':'Calibri'} # main font
colors = ['#f47e7a', '#b71f5c', '#621237', '#dbbaa7', '#df4382', '#c58f71']


title = plt.title("Question Means", pad=60, fontsize=14, color=font_color, *

y_pos_y = np.arange(len(df_mean_erwartungen.index)) * 4

y_pos_z = np.arange(len(df_mean_erwartungen.index)) * 4 + 1

y_pos_diff = np.arange(len(df_mean_erwartungen.index)) * 4 + 2

ax.barh(y_pos_y, df_mean_erwartungen["gen_y"], align='center', label="Mean G
ax.barh(y_pos_z, -1 * df_mean_erwartungen["gen_z"], align='center', label="M
ax.barh(y_pos_diff, df_mean_erwartungen["mean_diff"], align='center', label=

plt.xticks(color=font_color, **hfont)
plt.yticks(color=font_color, **hfont)

ax.axes.get_xaxis().set_visible(False)
ax.set_yticks(y_pos_diff, labels=[get_question_text(code) for code in df_mea

legend = plt.legend(loc='center',
           frameon=False,
           bbox_to_anchor=(0., .982, 1., .102),
           mode='expand',
           ncol=3,
           borderaxespad=-.46,
           prop={'size': 12, 'family':'Calibri'})
```



In [58]:
```
fig, ax = plt.subplots(figsize=(10, 15))

font_color = '#525252'
csfont = {'fontname':'Georgia'} # title font
hfont = {'fontname':'Calibri'} # main font
colors = ['#f47e7a', '#b71f5c', '#621237', '#dbbaa7', '#df4382', '#c58f71']

title = plt.title("Question Means", pad=60, fontsize=14, color=font_color, *

y_pos_y = np.arange(len(df_mean_erwartungen.index)) * 4

y_pos_z = np.arange(len(df_mean_erwartungen.index)) * 4 + 1

y_pos_diff = np.arange(len(df_mean_erwartungen.index)) * 4 + 2

ax.barh(y_pos_y, df_mean_erwartungen["gen_y"], align='center', label="Mean G
ax.barh(y_pos_z, -1 * df_mean_erwartungen["gen_z"], align='center', label="M
```

```
ax.barh(y_pos_diff, df_mean_erwartungen["mean_diff"], align='center', label=

plt.xticks(color=font_color, **hfont)
plt.yticks(color=font_color, **hfont)

ax.axes.get_xaxis().set_visible(False)
ax.set_yticks(y_pos_diff, labels=[get_question_text(code) for code in df_mea

legend = plt.legend(loc='center',
           frameon=False,
           bbox_to_anchor=(0., .982, 1., .102),
           mode='expand',
           ncol=3,
           borderaxespad=-.46,
           prop={'size': 12, 'family':'Calibri'})
```

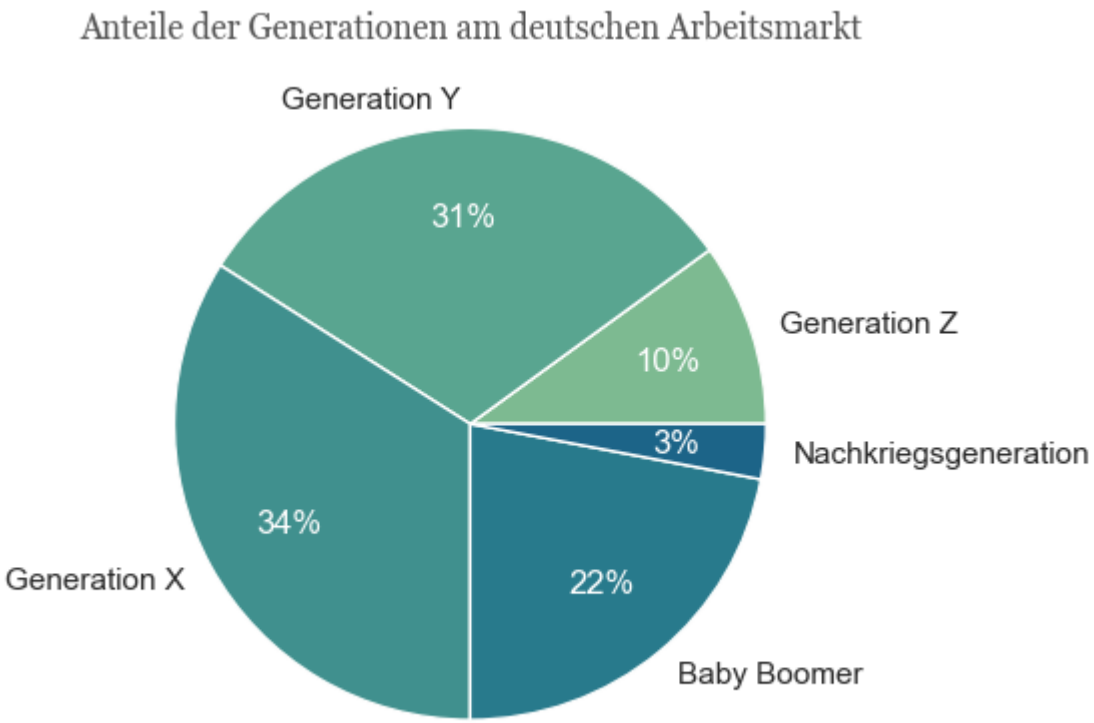
2.3 Visualisierung der Grundgesamtheit

Die Verteilung der Generationen am deutschen Arbeitsmarkt, sowie die Grundgesamtheit der Stichprobe wird grafisch dargestellt.

In [60]:
```
gen_count = df_yz[generation_col].value_counts()
print(gen_count)
```

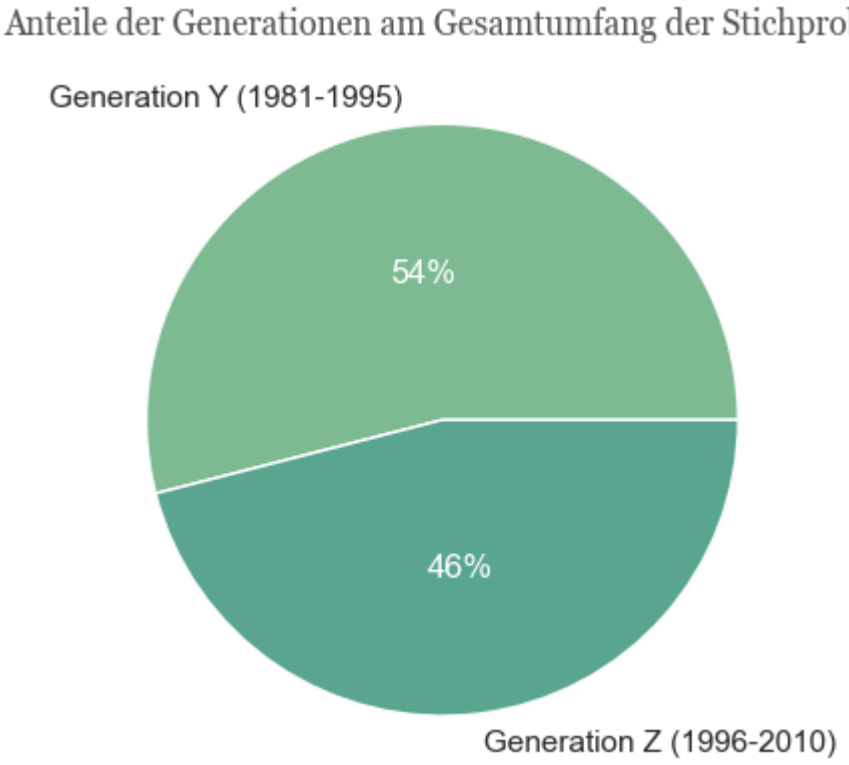
```
60371649
1    108
2     92
Name: count, dtype: int64
```

In [61]:
```
percent_work = [10, 31, 34, 22, 3]
generation_work = ["Generation Z", "Generation Y", "Generation X", "Baby Boo


fig, ax = plt.subplots()
patches, texts, autotexts  = ax.pie(percent_work,
                                    labels=generation_work,
                                    autopct='%.0f%%',
                                    pctdistance=0.7)

[autotext.set_color('white') for autotext in autotexts]
plt.title("Anteile der Generationen am deutschen Arbeitsmarkt", color=font_c
plt.show()
```



In [62]:
```
fig, ax = plt.subplots()
patches, texts, autotexts  = ax.pie(gen_count,
                                    labels=["Generation Y (1981–1995)", "Ger
                                    autopct='%.0f%%',
                                    pctdistance=0.5)
```

```
[autotext.set_color('white') for autotext in autotexts]
plt.title("Anteile der Generationen am Gesamtumfang der Stichprobe", color=f
plt.show(fig)
```

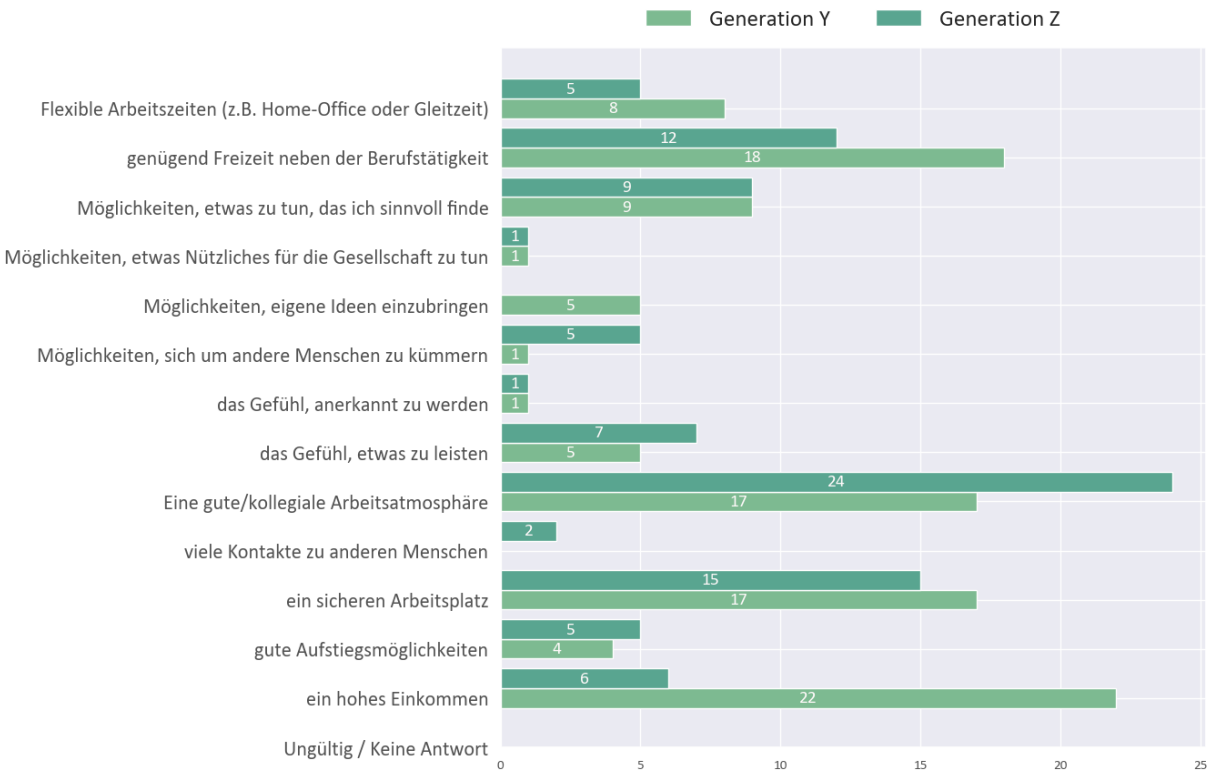


2.3.1 Auswertung Freitextfragen - 3.1 Welches der Kriterien ist für dich am wichtigsten? - 3.2 Welches der Kriterien ist für dich am wenigsten wichtig?

In [63]:
```
frei_text_wichtig = "60371613"
frei_text_unwichtig = "60371614"
```

In [64]:
```
fig, ax = plt.subplots(figsize=(10,10))
title = plt.title("3.1 Welches der Kriterien ist für dich am wichtigsten?",

wichtig_y = df_yz[frei_text_wichtig][df_yz[generation_col] == 1].value_count
wichtig_z = df_yz[frei_text_wichtig][df_yz[generation_col] == 2].value_count

y_pos_y = wichtig_y.index * 2
y_pos_z = wichtig_z.index * 2 + 0.8

ax.barh(y_pos_y, wichtig_y,height=0.8, align='center', label=f"Generation Y"
ax.barh(y_pos_z, wichtig_z,height=0.8, align='center', label=f"Generation Z"

plt.xticks(color=font_color, **hfont)
plt.yticks(color=font_color, **hfont)

ax.set_yticks(range(0, 28, 2), labels=[get_answer_text(frei_text_wichtig, co
ax.bar_label(ax.containers[0], label_type="center", color='snow', fmt='{:,.0
ax.bar_label(ax.containers[1], label_type="center", color='snow', fmt='{:,.0
```

```
# Get the first two and last y-tick positions.
miny, nexty, *_, maxy = ax.get_yticks()

# Compute half the y-tick interval (for example).
eps = (nexty - miny) / 2  # <-- Your choice.

# Adjust the limits.
legend = plt.legend(loc='upper center',
                    frameon=False,
                    bbox_to_anchor=(0., .55, 1., .502),
                    ncol=3,
                    borderaxespad=-.46,
                    prop={'size': 18, 'family':'Calibri'})
```



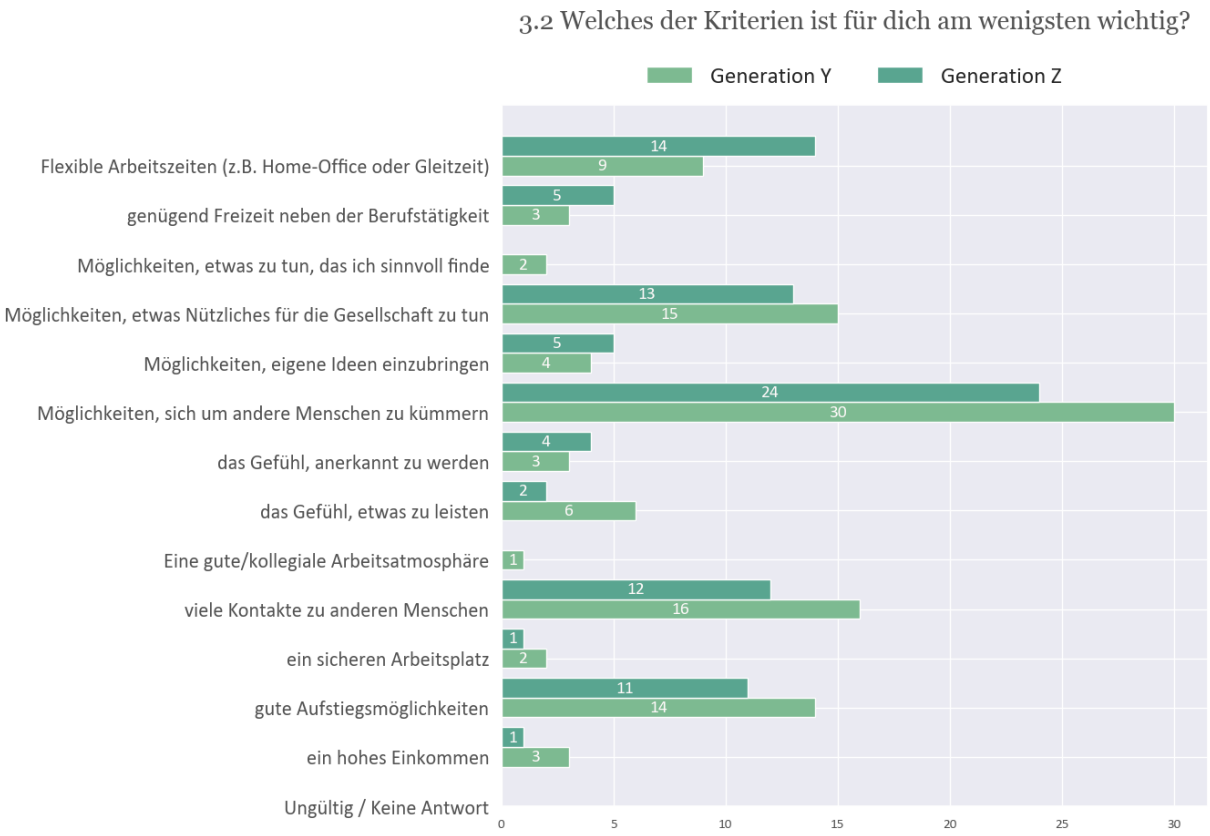
In [65]:
```
wichtig_y
```

Out[65]:
```
60371613
1     22
12    18
3     17
5     17
11     9
13     8
9      5
6      5
2      4
8      1
7      1
10     1
Name: count, dtype: int64
```

In [66]:
```
wichtig_z
```

Out[66]:
```
60371613
5     24
3     15
12    12
11     9
6      7
1      6
2      5
13     5
8      5
4      2
7      1
10     1
Name: count, dtype: int64
```

In [67]:
```
fig, ax = plt.subplots(figsize=(10,10))
title = plt.title("3.2 Welches der Kriterien ist für dich am wenigsten wicht

unwichtig_y = df_yz[frei_text_unwichtig][df_yz[generation_col] == 1].value_c
unwichtig_z = df_yz[frei_text_unwichtig][df_yz[generation_col] == 2].value_c

y_pos_y = unwichtig_y.index * 2
y_pos_z = unwichtig_z.index * 2 + 0.8

ax.barh(y_pos_y, unwichtig_y, height=0.8, align='center', label=f"Generation
ax.barh(y_pos_z, unwichtig_z, height=0.8, align='center', label=f"Generation

plt.xticks(color=font_color, **hfont)
plt.yticks(color=font_color, **hfont)

ax.set_yticks(range(0, 28, 2), labels=[get_answer_text(frei_text_unwichtig,
ax.bar_label(ax.containers[0], label_type="center", color='snow', fmt='{:,.0
ax.bar_label(ax.containers[1], label_type="center", color='snow', fmt='{:,.0


# Get the first two and last y-tick positions.
miny, nexty, *_, maxy = ax.get_yticks()

# Compute half the y-tick interval (for example).
eps = (nexty - miny) / 2  # <-- Your choice.

# Adjust the limits.
#ax.set_ylim(maxy+eps, miny-eps)
legend = plt.legend(loc='upper center',
                    frameon=False,
                    bbox_to_anchor=(0., .55, 1., .502),
                    ncol=3,
                    borderaxespad=-.46,
                    prop={'size': 18, 'family':'Calibri'})
```



In [68]:
```
unwichtig_y
```

Out[68]:
```
60371614
8     30
4     16
10    15
2     14
13     9
6      6
9      4
12     3
1      3
7      3
11     2
3      2
5      1
Name: count, dtype: int64
```

In [69]:
```
unwichtig_z
```

Out[69]:
```
60371614
8     24
13    14
10    13
4     12
2     11
9      5
12     5
7      4
6      2
1      1
3      1
Name: count, dtype: int64
```

2.3.2 Auswertung Skalafragen

Im folgenden werden die Ergebnisse der Fragencluster AN-Erwartungen & AG-Erwartungen sowie persönliche Wertevorstellungen grafisch aufbereitet. Zuerst werden die Daten dafür vorbereitet.

In [70]:
```
def transpose_count(count_df, index_str):
    count_df.columns = count_df.iloc[0]
    count_df.drop(count_df.index[0], inplace=True)
    count_df.rename(index={'proportion':index_str}, inplace=True)
    return count_df

def get_count_df(input_df, question_code):
    temp_y = input_df[question_code][input_df[generation_col] == 1].value_co
    temp_z = input_df[question_code][input_df[generation_col] == 2].value_co
    out_df = pd.concat([pd.DataFrame(columns=[0,1,2,3,4,5]), transpose_count
    out_df.fillna(0, inplace=True)
    out_df = out_df.round(10)
    out_df.rename(columns={col: int(col) for col in out_df.columns}, inplace
    #out_df['sum'] = out_df[list(out_df.columns)].sum(axis=1)
    return out_df
```

In [71]:
```
get_count_df(df_yz, "60371626")
```

Out[71]:

	0	1	2	3	4	5
Gen Y	0	0.166667	0.379630	0.361111	0.055556	0.037037
Gen Z	0	0.130435	0.336957	0.358696	0.163043	0.010870

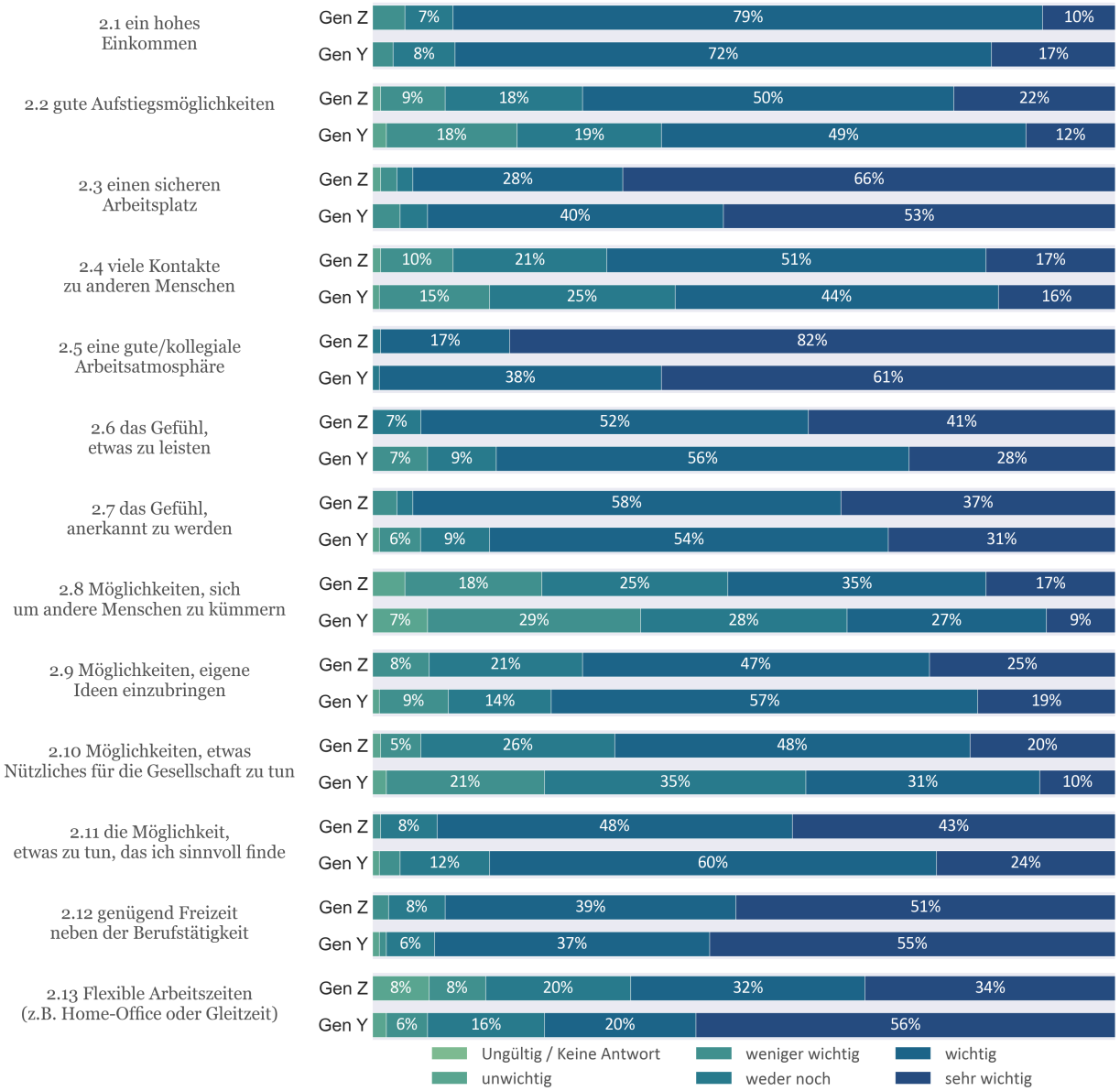
In [72]:
```
def plot_question_results(questions: List[str],
                          fig_size: tuple,
                          label_pos: tuple,
                          bar_height: float,
                          legend_anchor: tuple):
    fig, axs = plt.subplots(nrows=len(questions), figsize=fig_size)

    sns.set_palette("crest")

    for code, ax in zip(questions, axs.ravel()):
```

```
        count_df = get_count_df(df_yz, code)

        left = np.zeros(2)
        y_pos = [0, bar_height + bar_height/2]

        for col in count_df.columns:
            ax.barh(y_pos, count_df[col], height=bar_height, label=col, left
            left += count_df[col]


        # Adjust the subplot so that the title would fit
        plt.subplots_adjust(top=65)

        for label in (ax.get_xticklabels() + ax.get_yticklabels()):
            label.set_fontsize(40)

        ax.axes.get_xaxis().set_visible(False)

        ax.set_yticks(y_pos, labels=count_df.index)
        ax.set_ylabel(add_new_line(get_question_text(code)), labelpad=0, fon
        ax.yaxis.set_label_coords(*label_pos)
        ax.set_xlim(0, 1)

        for p in ax.patches:
            width, height = p.get_width(), p.get_height()
            x, y = p.get_xy()
            if width > 0.05:
                ax.text(x+width/2,
                        y+height/2,
                        f"{width*100:.0f}%",
                        horizontalalignment='center',
                        verticalalignment='center',
                        color='white',
                        fontsize=40,
                        **hfont)

    labels_handles = {
      label: handle for ax in fig.axes for handle, label in zip(*ax.get_lege
    }

    legend = fig.legend(labels_handles.values(),
                        labels_handles.keys(),
                        bbox_to_anchor=legend_anchor,
                        loc='lower center',
                        frameon=False,
                        ncol=3,
                        borderaxespad=-3.75,
                        prop={'size': 40, 'family':'Calibri'})

    for text in legend.get_texts():
        text.set(text=get_answer_text(code,int(text.get_text())))
        plt.setp(text, color=font_color) # legend font color
```

2.3.2a Darstellung Umfrageergebnisse - Erwartungen an die Beschäftigung

In [73]:
```
plot_question_results(erwartungen_ids, (30, 0.5), (-0.3, 0.25), 0.15, (0., .
```

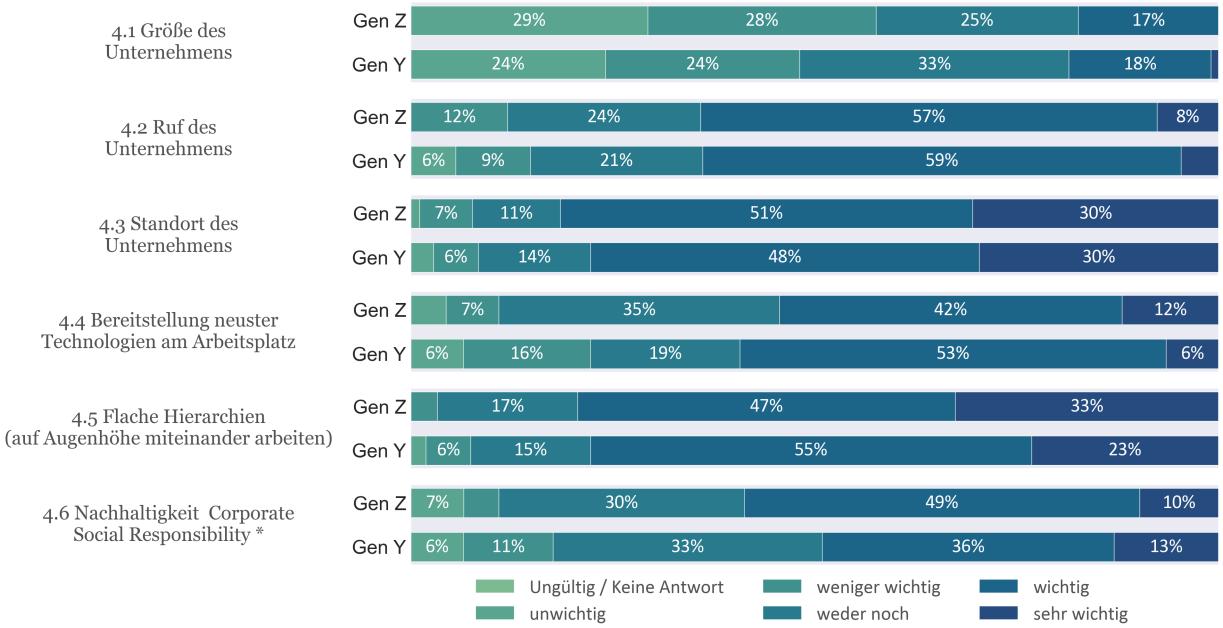
```
['2.1', 'ein', 'hohes', 'Einkommen']
['2.2', 'gute', 'Aufstiegsmöglichkeiten']
['2.3', 'einen', 'sicheren', 'Arbeitsplatz']
['2.4', 'viele', 'Kontakte', 'zu', 'anderen', 'Menschen']
['2.5', 'eine', 'gute/kollegiale', 'Arbeitsatmosphäre']
['2.6', 'das', 'Gefühl,', 'etwas', 'zu', 'leisten']
['2.7', 'das', 'Gefühl,', 'anerkannt', 'zu', 'werden']
['2.8', 'Möglichkeiten,', 'sich', 'um', 'andere', 'Menschen', 'zu', 'kümmer
n']
['2.9', 'Möglichkeiten,', 'eigene', 'Ideen', 'einzubringen']
['2.10', 'Möglichkeiten,', 'etwas', 'Nützliches', 'für', 'die', 'Gesellschaf
t', 'zu', 'tun']
['2.11', 'die', 'Möglichkeit,', 'etwas', 'zu', 'tun,', 'das', 'ich', 'sinnvol
l', 'finde']
['2.12', 'genügend', 'Freizeit', 'neben', 'der', 'Berufstätigkeit']
['2.13', 'Flexible', 'Arbeitszeiten', '(z.B.', 'Home-Office', 'oder', 'Gleitze
it)']
```



2.3.2b Darstellung Umfrageergebnisse - Merkmale des Arbeitgebers

In [74]:
```
plot_question_results(ag_merkmale_ids, (30, 0.25), (-0.3, 0.25), 0.05, (0.,
```

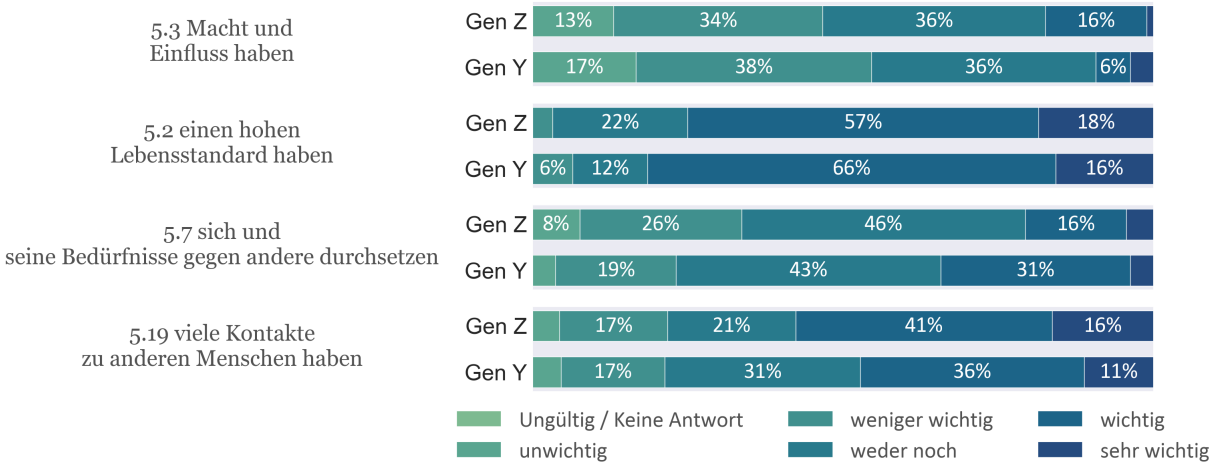
```
['4.1', 'Größe', 'des', 'Unternehmens']
['4.2', 'Ruf', 'des', 'Unternehmens']
['4.3', 'Standort', 'des', 'Unternehmens']
['4.4', 'Bereitstellung', 'neuster', 'Technologien', 'am', 'Arbeitsplatz']
['4.5', 'Flache', 'Hierarchien', '(auf', 'Augenhöhe', 'miteinander', 'arbeite
n)']
['4.6', 'Nachhaltigkeit\xa0', 'Corporate', 'Social', 'Responsibility', '*']
```



2.3.2c Darstellung Umfrageergebnisse - Persönliche Wertvorstellungen

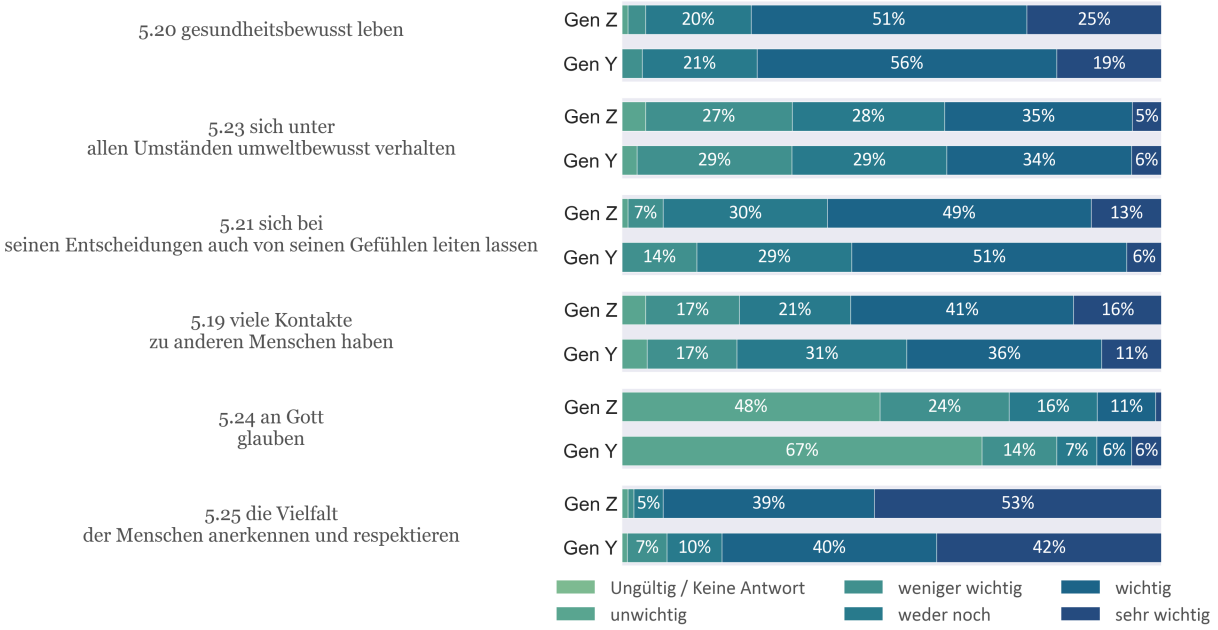
In [75]:
```
plot_question_results(macht_ids, (20, 0.15), (-0.5, 0.25), 0.25, (0., .90, 1
```

```
['5.3', 'Macht', 'und', 'Einfluss', 'haben']
['5.2', 'einen', 'hohen', 'Lebensstandard', 'haben']
['5.7', 'sich', 'und', 'seine', 'Bedürfnisse', 'gegen', 'andere', 'durchsetze
n']
['5.19', 'viele', 'Kontakte', 'zu', 'anderen', 'Menschen', 'haben']
```



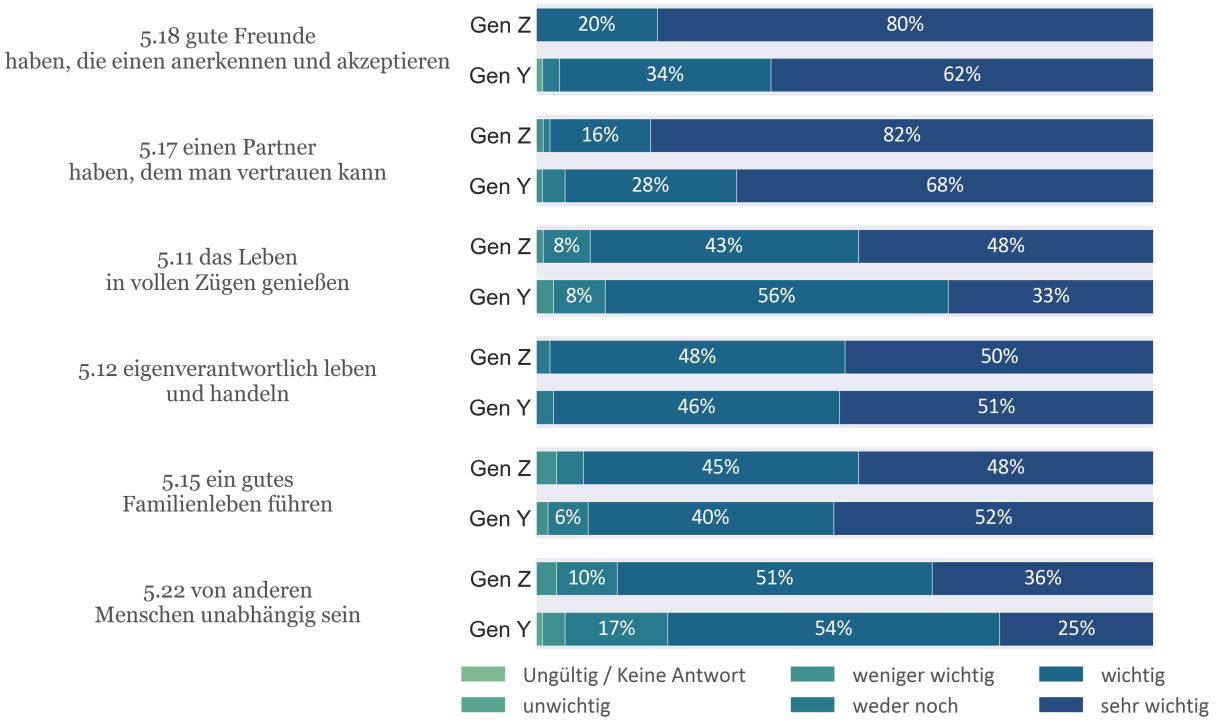
In [76]:
```
plot_question_results(leben_ids, (20, 0.25), (-0.65, 0.25), 0.15, (0., .90,
```

```
['5.20', 'gesundheitsbewusst', 'leben']
['5.23', 'sich', 'unter', 'allen', 'Umständen', 'umweltbewusst', 'verhalten']
['5.21', 'sich', 'bei', 'seinen', 'Entscheidungen', 'auch', 'von', 'seinen',
'Gefühlen', 'leiten', 'lassen']
['5.19', 'viele', 'Kontakte', 'zu', 'anderen', 'Menschen', 'haben']
['5.24', 'an', 'Gott', 'glauben']
['5.25', 'die', 'Vielfalt', 'der', 'Menschen', 'anerkennen', 'und', 'respekti
eren']
```



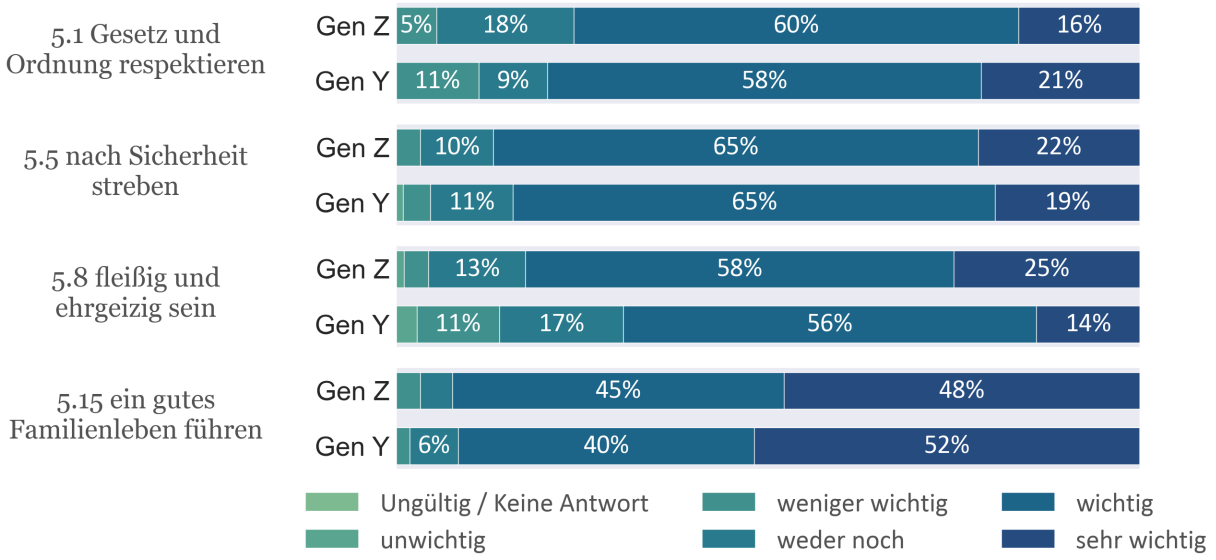
In [77]:
```
plot_question_results(familie_ids, (20, 0.25), (-0.5, 0.25), 0.15, (0., .90,
```

```
['5.18', 'gute', 'Freunde', 'haben,', 'die', 'einen', 'anerkennen', 'und', 'a
kzeptieren']
['5.17', 'einen', 'Partner', 'haben,', 'dem', 'man', 'vertrauen', 'kann']
['5.11', 'das', 'Leben', 'in', 'vollen', 'Zügen', 'genießen']
['5.12', 'eigenverantwortlich', 'leben', 'und', 'handeln']
['5.15', 'ein', 'gutes', 'Familienleben', 'führen']
['5.22', 'von', 'anderen', 'Menschen', 'unabhängig', 'sein']
```



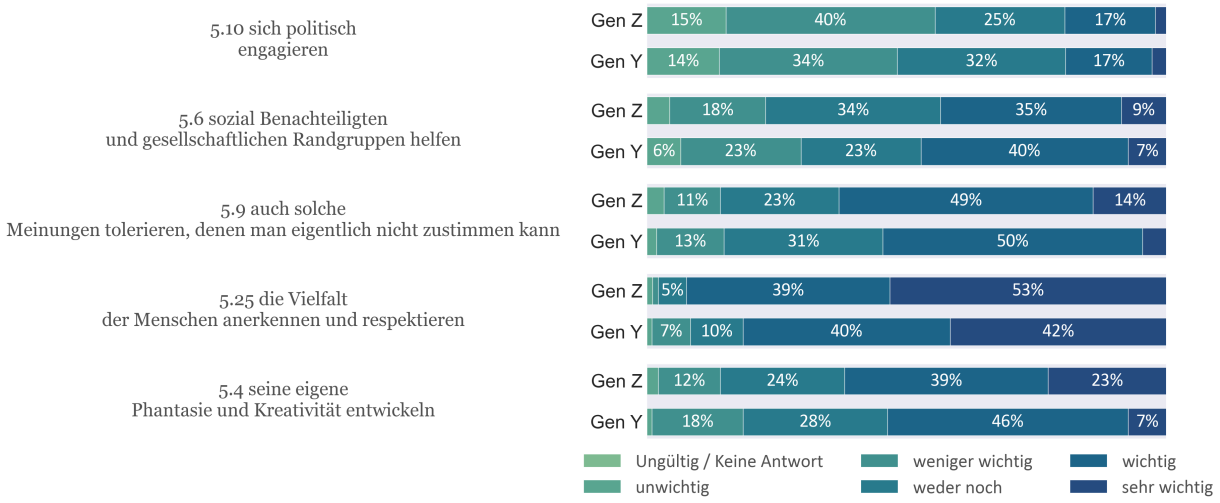
In [78]:
```
plot_question_results(tugend_ids, (20, 0.15), (-0.35, 0.25), 0.2, (0., .90,
```

```
['5.1', 'Gesetz', 'und', 'Ordnung', 'respektieren']
['5.5', 'nach', 'Sicherheit', 'streben']
['5.8', 'fleißig', 'und', 'ehrgeizig', 'sein']
['5.15', 'ein', 'gutes', 'Familienleben', 'führen']
```



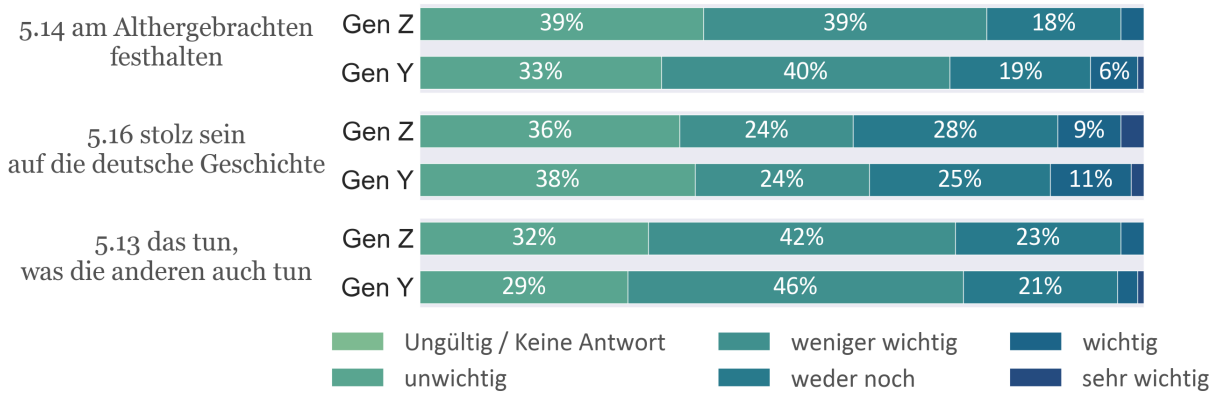
In [79]:
```
plot_question_results(toleranz_ids, (20, 0.2), (-0.7, 0.25), 0.15, (0., .90,
```

```
['5.10', 'sich', 'politisch', 'engagieren']
['5.6', 'sozial', 'Benachteiligten', 'und', 'gesellschaftlichen', 'Randgruppe
n', 'helfen']
['5.9', 'auch', 'solche', 'Meinungen', 'tolerieren,', 'denen', 'man', 'eigent
lich', 'nicht', 'zustimmen', 'kann']
['5.25', 'die', 'Vielfalt', 'der', 'Menschen', 'anerkennen', 'und', 'respekti
eren']
['5.4', 'seine', 'eigene', 'Phantasie', 'und', 'Kreativität', 'entwickeln']
```



In [80]:
```
plot_question_results(tradition_ids, (20, 0.1), (-0.35, 0.25), 0.15, (0., .9
```

```
['5.14', 'am', 'Althergebrachten', 'festhalten']
['5.16', 'stolz', 'sein', 'auf', 'die', 'deutsche', 'Geschichte']
['5.13', 'das', 'tun,', 'was', 'die', 'anderen', 'auch', 'tun']
```

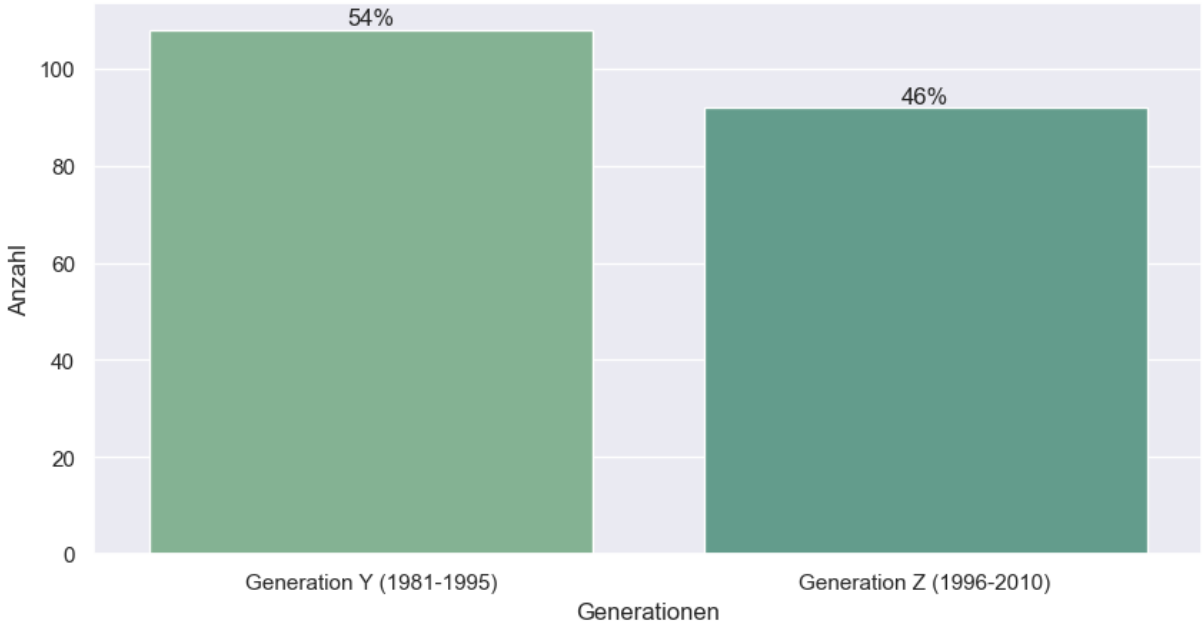


2.4 Anhang

Die folgenden Grafiken wurden im Laufe der Analyse erstellt, werden aber im Weiteren nicht mehr verwenden. Sie können deshalb lediglich als ein MVP (minimum viable product) angesehen werden.

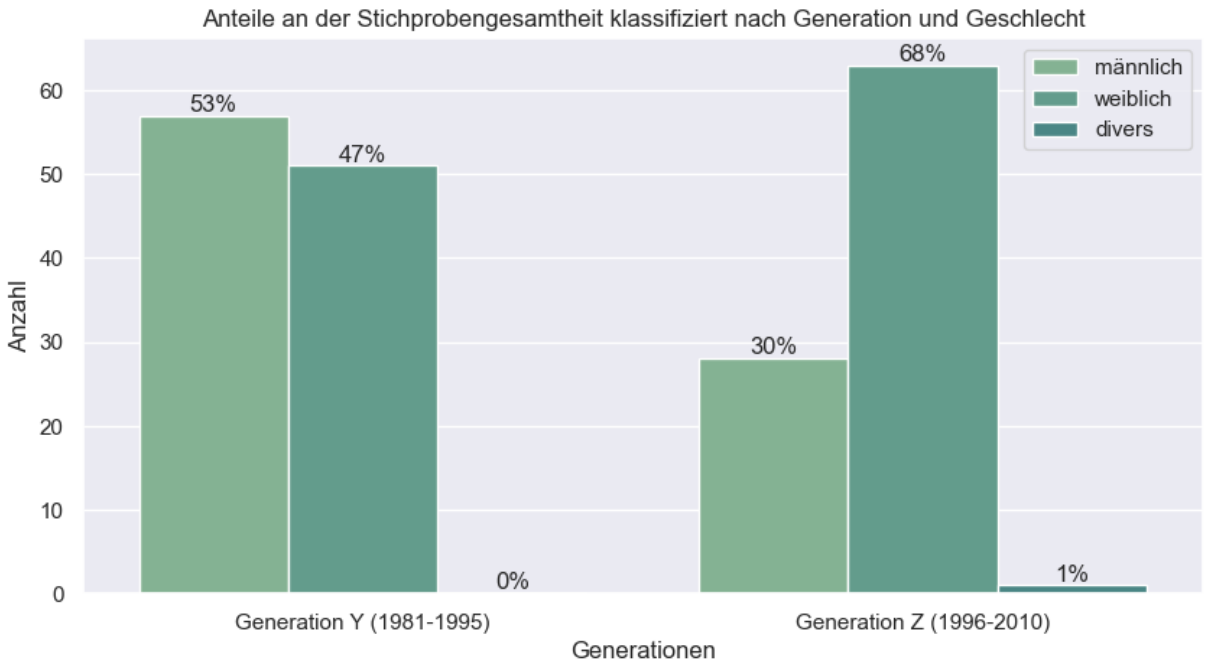
In [81]:
```
total = df_yz[gender_col].count()
total_y = 108
total_z = 92
```

In [82]:
```
plt.figure(figsize=(10,5))
fig = sns.countplot(data=df_yz, x=generation_col)
plt.xlabel("Generationen")
fig.axes.set_xticklabels(labels=[get_answer_text(generation_col, x) for x in
plt.ylabel("Anzahl")
# add annotations
for c in fig.containers:
    fig.bar_label(c, fmt=lambda v: f'{(v/total)*100:0.0f}%')
plt.show(fig)
```

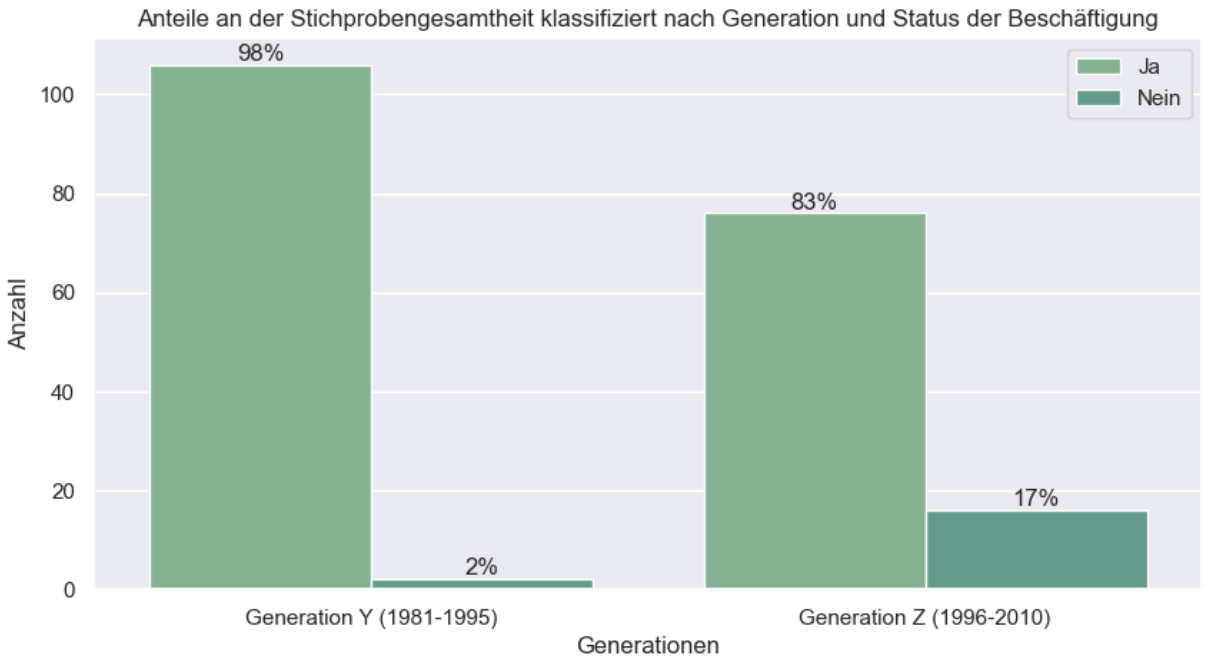


In [83]:
```
def plt_classes_bar(input_df: pd.DataFrame, col_code: str, title: str):
    plt.figure(figsize=(10,5))
    ax = sns.countplot(data=input_df, x=generation_col, hue=col_code)
    plt.xlabel("Generationen")
    ax.set_xticklabels(labels=[get_answer_text(generation_col, x) for x in s
    plt.ylabel("Anzahl")
    plt.title(title)
    plt.legend(labels=[get_answer_text(col_code, x) for x in sorted(input_df

    for rect in ax.patches:
        height = rect.get_height()
        x, y = rect.get_xy()
        if x < 0.6:
            tmp_total = total_y
        else:
            tmp_total = total_z
        ax.text(
        rect.get_x() + rect.get_width() / 2, height, f"{(height/tmp_total)*1
        )

        #fig.bar_label(c, fmt=lambda v: f'{(v/total)*100:0.1f}%')
    plt.show(fig)
```

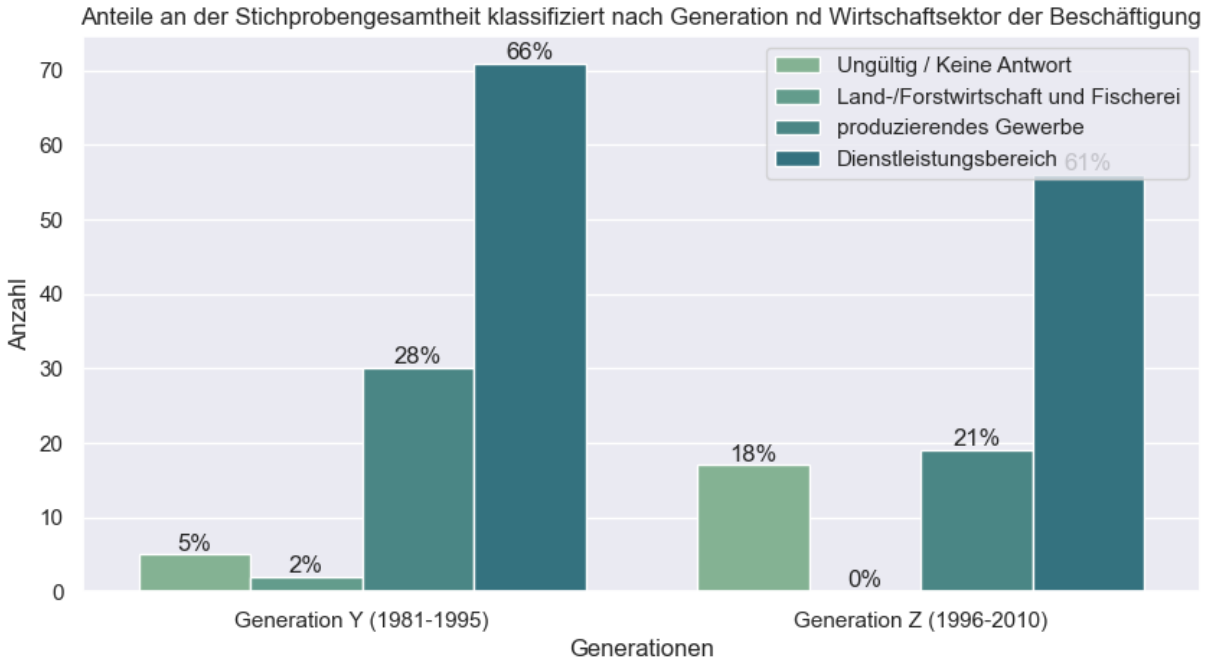
In [84]:
```
plt_classes_bar(df_yz, gender_col, "Anteile an der Stichprobengesamtheit kla
```



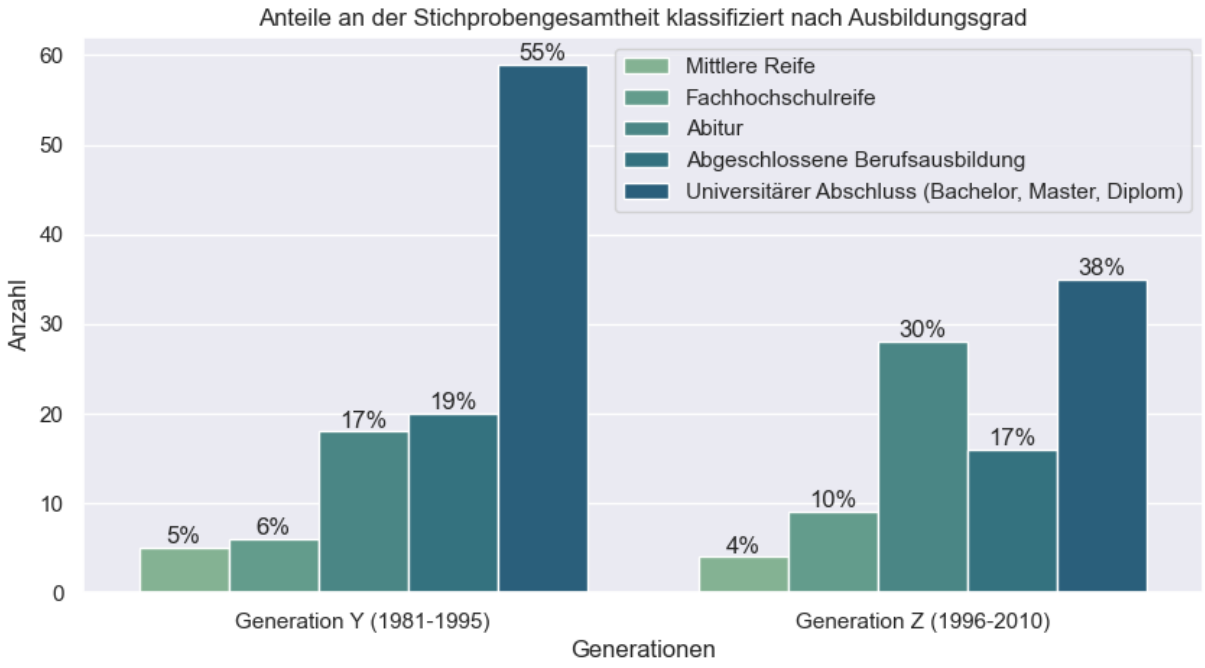
In [85]:
```
plt_classes_bar(df_yz, working_col, "Anteile an der Stichprobengesamtheit kl
```



In [86]:
```
plt_classes_bar(df_yz, economic_sector_col, "Anteile an der Stichprobengesam
```



In [87]:
```
plt_classes_bar(df_yz, academic_col, "Anteile an der Stichprobengesamtheit k
```



In [ ]:

In [ ]: