Data Literacy

University of Tübingen, Winter Term 2021/22

Exercise Sheet 5

This sheet is **due on Monday 29 November 2021 at 10 am sharp (i.e. before the start of the lecture).**

Hypothesis Testing & *Hunting* for Significance

What is this week's tutorial about? In this week's tutorial, we will analyze data from the 1. Fußball-Bundesliga (the German premier soccer league). The goal is to investigate the effect of the Corona pandemic on the teams' performances. More precisely, we will try to find teams that achieved significantly worse results in the first "Corona-year" 2020 than in previous years (possibly caused by empty stadiums, etc.). For that purpose, we will conduct multiple hypotheses tests and discuss whether and how we should therefore adapt our strategy.

In [ ]:
```
# Make inline plots vector graphics
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
```

1. Load and prepare the Data

In this tutorial, we will use data provided by OpenLigaDB. The function `get_league_table` allows you to retrieve the data via the API interface and convert it into a Pandas data frame.

In [ ]:
```
import requests

API_ENDPOINT = "https://www.openligadb.de/api"

def get_league_table(league, year):
    """
    Get team rankings as Pandas data frame.

    Parameters
    ----------
    league : str
        'bl1' for 1. Bundesliga, see https://github.com/OpenLigaDB/OpenLigaDB-Samples
    year : int
        Get data for this year
    """
    response = requests.get(f"{API_ENDPOINT}/getbltable/{league}/{year}")
    data = response.json()
    return pd.DataFrame(data)

# Get and display data for the 1. Bundesliga ('bl1') for 2020
data = get_league_table(league="bl1", year=2020)
display(data.head())
```

	TeamInfoId	TeamName	ShortName	TeamIconUrl	Points	OpponentGoals	Goals	Matches	Won	Lost	Draw
0	40	FC Bayern München	Bayern	https://i.imgur.com/jJEsJrj.png	78	44	99	34	24	4	6
1	1635	RB Leipzig	Leipzig	https://i.imgur.com/Rpwsjz1.png	65	32	60	34	19	7	8
2	7	Borussia Dortmund	BVB	https://upload.wikimedia.org/wikipedia/commons...	64	46	75	34	20	10	4
3	131	VfL Wolfsburg	Wolfsburg	https://i.imgur.com/ucqKV4B.png	61	37	61	34	17	7	10
4	91	Eintracht Frankfurt	Frankfurt	https://i.imgur.com/X8NFkOb.png	60	53	69	34	16	6	12

In [ ]:
```
# construct dataframe pre 2020
years_pre_2020 = np.arange(2010, 2020, 1)
df_list = []

for year in years_pre_2020:
```

```
    df_list.append(get_league_table(league="bl1", year=year))

df_pre_2020 = pd.concat(df_list, axis=0)
df_pre_2020.drop(columns=['TeamInfoId', 'ShortName', 'TeamIconUrl', 'Points', 'OpponentGoals', 'Goals', 'GoalDif
df_pre_2020.rename({'Matches':'MatchesPre2020', 'Won':'WonPre2020', 'Lost':'LostPre2020', 'Draw':'DrawPre2020'},

df_pre_2020 = df_pre_2020.groupby(by='TeamName').sum()
df_pre_2020.head()
```

Out[ ]:

TeamName	MatchesPre2020	WonPre2020	LostPre2020	DrawPre2020
1. FC Kaiserslautern	68	17	33	18
1. FC Köln	238	67	108	63
1. FC Nürnberg	170	44	80	46
1. FC Union Berlin	34	12	17	5
1. FSV Mainz 05	340	118	141	81

In [ ]:

```
# construct 2020 dataframe
df_2020 = data[data['TeamName']!='Arminia Bielefeld'].reset_index(drop=True)
df_2020.drop(columns=['TeamInfoId', 'ShortName', 'TeamIconUrl', 'Points', 'OpponentGoals', 'Goals', 'GoalDiff'],
df_2020.rename({'Matches':'Matches2020', 'Won':'Won2020', 'Lost':'Lost2020', 'Draw':'Draw2020'}, axis=1, inplace=
df_2020.head()
```

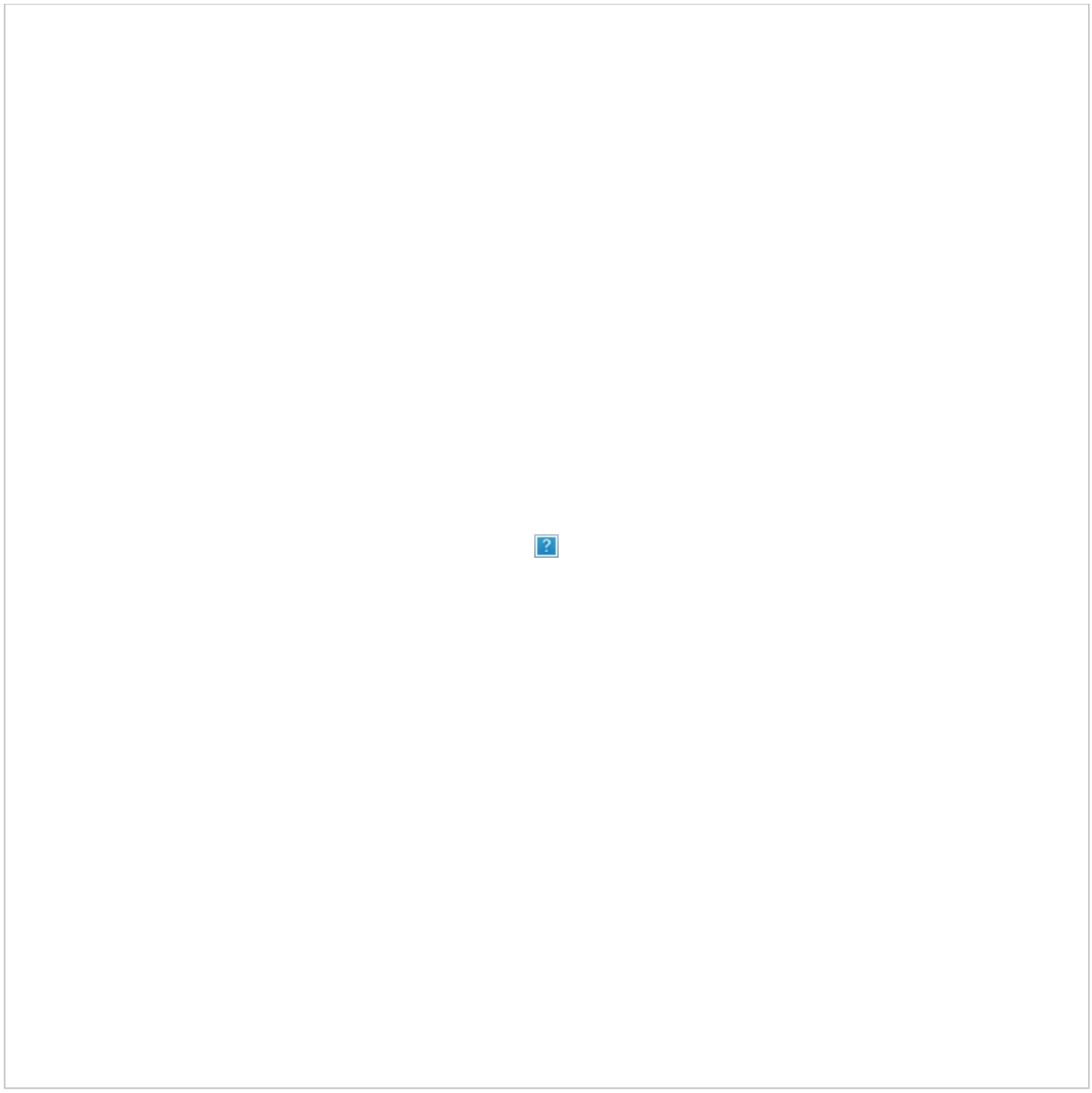
Out[ ]:

	TeamName	Matches2020	Won2020	Lost2020	Draw2020
0	FC Bayern München	34	24	4	6
1	RB Leipzig	34	19	7	8
2	Borussia Dortmund	34	20	10	4
3	VfL Wolfsburg	34	17	7	10
4	Eintracht Frankfurt	34	16	6	12

In [ ]:

```
# merge dataframes
merged_df = df_2020.merge(df_pre_2020, on='TeamName')
```

Task: Your first task is to essentially recreate the following table:



In [ ]:
```
merged_df
```

Out[ ]:

	TeamName	Matches2020	Won2020	Lost2020	Draw2020	MatchesPre2020	WonPre2020	LostPre2020	DrawPre2020
0	FC Bayern München	34	24	4	6	340	255	38	47
1	RB Leipzig	34	19	7	8	136	72	28	36
2	Borussia Dortmund	34	20	10	4	340	203	65	72
3	VfL Wolfsburg	34	17	7	10	340	127	121	92
4	Eintracht Frankfurt	34	16	6	12	306	105	126	75
5	Bayer Leverkusen	34	14	10	10	340	171	96	73
6	1. FC Union Berlin	34	12	8	14	34	12	17	5
7	Borussia Mönchengladbach	34	13	11	10	340	152	113	75
8	VfB Stuttgart	34	12	13	9	272	87	128	57
9	SC Freiburg	34	12	13	9	306	96	125	85
10	TSG 1899 Hoffenheim	34	11	13	10	340	120	120	100
11	1. FSV Mainz 05	34	10	15	9	340	118	141	81
12	FC Augsburg	34	10	18	6	306	91	131	84
13	Hertha BSC	34	8	15	11	272	88	115	69
14	1. FC Köln	34	8	17	9	238	67	108	63
15	Werder Bremen	34	7	17	10	340	105	142	93
16	FC Schalke 04	34	3	24	7	340	140	119	81

2. Compute p-Values

Our goal is to find out if there are teams that achieved significantly worse results in 2020 than in previous years. In the lecture, you already

learned about a statistical test for this purpose:

- First, we put a beta-prior on f_0 (the winning probability before 2020) which is based on m_0 (the number of wins before 2020) in n_0 matches (the number of matches before 2020).
- Under the null hypothesis $H_0: f_1 = f_0$, the number of wins in 2020 m_1 (given the number of matches in 2020 n_1) follows a binomial distribution.
- Putting these building blocks together, we obtain a beta-binomial distribution

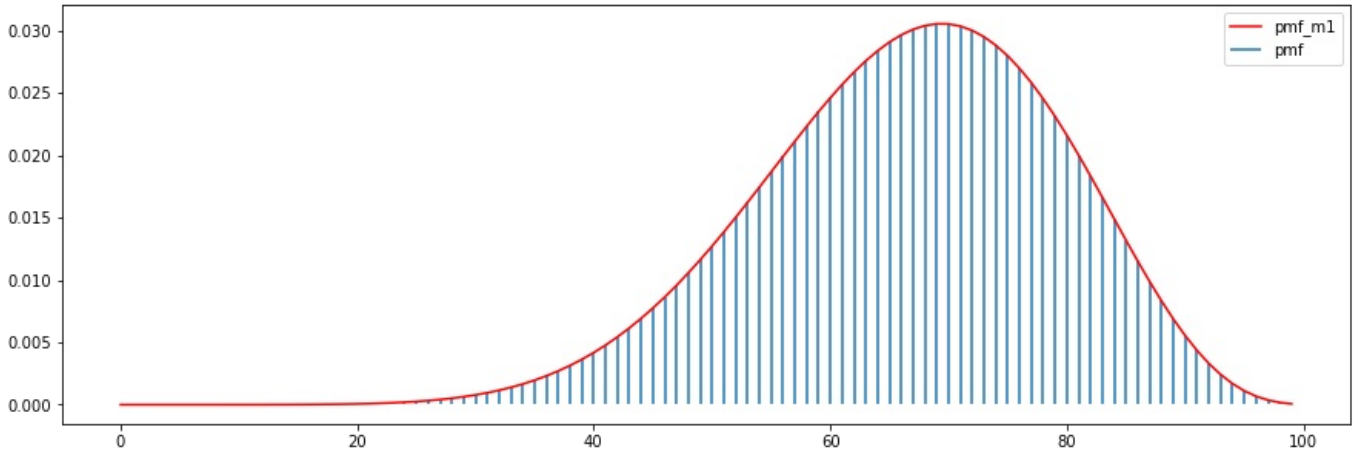
  $$p(m_1 \vert n_1, m_0, n_0) = {n_1\choose m_1} \frac{\text{B}(m_0 + m_1 + 1, (n_0-m_0) + (n_1-m_1) + 1)}{\text{B}(m_0 + 1, n_0 - m_0 + 1)}.$$

  This tells us the probability to observe m_1 wins in 2020, given the number of matches in 2020 n_1 and the statistics m_0, n_0 for the years before.

Task: Plot $p(m_1 \vert n_1, m_0, n_0)$ for $m_1 \in \{0, 1, ..., n_1\}$ for some arbitrary numbers n_1, m_0 and n_0. Play around with these parameters to gain some intuition how this distribution behaves.

In [ ]:

```
from scipy.stats import betabinom

def plot_beta_binom(n0, n1, m0):

    p = betabinom(n1, m0, n0)

    x1 = np.arange(n1)
    plt.figure(figsize=(15, 5))
    plt.vlines(x1,0,p.pmf(x1), label='pmf')
    plt.plot(x1, p.pmf(x1), color='red', label='pmf_m1')
    plt.legend();

plot_beta_binom(5, 100, 10)
```



- The p-value represents the probability to observe a certain number of wins or *more extreme* results. Let's assume, the team we consider has won 3 times in 2020. Since we are interested in teams that have played particularly badly, we sum $p(m_1 \vert n_1, m_0, n_0)$ over $m_1 = 0, 1, 2, 3$. Evaluating the cumulative distribution function `betabinom.cdf` performs this summation for us.
- If this probability is small, then it is very unlikely that the observed data has been generated from the winning probability f_0. Or, in other words, the team has played particularly badly in 2020. We thus reject H_0 if $p \leq \alpha := 5\,\%$.

Task: Compute the p-values for every team and add the results as a new column `"p-value (won)"`. Check if there are teams whose p-value falls below the $5\,\%$ threshold.

In [ ]:

```
def p_val_won(m_1, n_1, m_0, n_0):
    """
    Compute p-value by summing the evidence p(m_1 | n_1, m_0, n_0) over the
    observed number of wins and 'more extreme' (i.e. smaller) results.

    Parameters
    ----------
    m_1 : int
        Number of wins in 2020 (0 <= m_1 <= n_1)
    n_1 : int
        Number of matches in 2020 (n_1 > 0)
    m_0 : int
        Number of wins before 2020 (0 <= m_0 <= n_0)
    n_0 : int
```

```
        Number of matches before 2020 (n_0 > 0)
    """
    return betabinom.cdf(m_1, n_1, m_0 + 1, n_0 - m_0 + 1)  # betabinom link, alpha = m_0+1, beta = n0-m0+1
```

In [ ]:
```
merged_df['p-value (won)'] = merged_df.apply(lambda x: p_val_won(x[2], x[1], x[6], x[5]), axis=1)
merged_df
```

Out[ ]:

	TeamName	Matches2020	Won2020	Lost2020	Draw2020	MatchesPre2020	WonPre2020	LostPre2020	DrawPre2020	p-value (won)
0	FC Bayern München	34	24	4	6	340	255	38	47	0.348167
1	RB Leipzig	34	19	7	8	136	72	28	36	0.677643
2	Borussia Dortmund	34	20	10	4	340	203	65	72	0.523893
3	VfL Wolfsburg	34	17	7	10	340	127	121	92	0.944680
4	Eintracht Frankfurt	34	16	6	12	306	105	126	75	0.947118
5	Bayer Leverkusen	34	14	10	10	340	171	96	73	0.197757
6	1. FC Union Berlin	34	12	8	14	34	12	17	5	0.536227
7	Borussia Mönchengladbach	34	13	11	10	340	152	113	75	0.288884
8	VfB Stuttgart	34	12	13	9	272	87	128	57	0.713822
9	SC Freiburg	34	12	13	9	306	96	125	85	0.739930
10	TSG 1899 Hoffenheim	34	11	13	10	340	120	120	100	0.436111
11	1. FSV Mainz 05	34	10	15	9	340	118	141	81	0.330397
12	FC Augsburg	34	10	18	6	306	91	131	84	0.560026
13	Hertha BSC	34	8	15	11	272	88	115	69	0.191480
14	1. FC Köln	34	8	17	9	238	67	108	63	0.354132
15	Werder Bremen	34	7	17	10	340	105	142	93	0.139969
16	FC Schalke 04	34	3	24	7	340	140	119	81	0.000065

In [ ]:
```
merged_df[merged_df['p-value (won)']<=0.05]['TeamName']
```

Out[ ]:
```
16    FC Schalke 04
Name: TeamName, dtype: object
```

We reject the 0-hypothesis for Schalke since the p-value is below 5%. Nevertheless, due to performing multiple tests, this might just be an outlier.

Our goal is to find teams that played significantly worse in 2020 compared to previous years. We did this by checking whether the number of games won this year is "surprisingly" low. Now, we will use an alternative approach: We check whether the number of games *lost* this year is particularly *high*. For this, we can re-use the statistical beta-binomial model from above by simply plugging in the number of *lost* games for m_0 and m_1.

Task: Compute the corresponding p-values and store the results in a new column `"p-value (lost)"`. Note that you cannot simply use the `p_val_won` function from above. This time, the question is: What is the probability for observing m_1 or *more* lost matches.

In [ ]:
```
def p_val_lost(m_1, n_1, m_0, n_0):
    """
    Compute p-value by summing the evidence p(m_1 | n_1, m_0, n_0) over the
    observed number of lost matches and 'more extreme' (i.e. larger) results.

    Parameters
    ----------
    m_1 : int
        Number of lost matches in 2020 (0 <= m_1 <= n_1)
    n_1 : int
        Number of matches in 2020 (n_1 > 0)
    m_0 : int
        Number of lost matches before 2020 (0 <= m_0 <= n_0)
    n_0 : int
        Number of matches before 2020 (n_0 > 0)
    """
    return 1 - betabinom.cdf(m_1, n_1, m_0 + 1, n_0 - m_0 + 1)
```

In [ ]:
```
merged_df['p-value (lost)'] = merged_df.apply(lambda x: p_val_lost(x[3], x[1], x[7], x[5]), axis=1)
merged_df
```

Out[ ]:

	TeamName	Matches2020	Won2020	Lost2020	Draw2020	MatchesPre2020	WonPre2020	LostPre2020	DrawPre2020	p-value (won)	p-valu (lost
0	FC Bayern München	34	24	4	6	340	255	38	47	0.348167	0.34758
1	RB Leipzig	34	19	7	8	136	72	28	36	0.677643	0.42564
2	Borussia Dortmund	34	20	10	4	340	203	65	72	0.523893	0.05811
3	VfL Wolfsburg	34	17	7	10	340	127	121	92	0.944680	0.94756
4	Eintracht Frankfurt	34	16	6	12	306	105	126	75	0.947118	0.99538
5	Bayer Leverkusen	34	14	10	10	340	171	96	73	0.197757	0.36744
6	1. FC Union Berlin	34	12	8	14	34	12	17	5	0.536227	0.98443
7	Borussia Mönchengladbach	34	13	11	10	340	152	113	75	0.288884	0.46773
8	VfB Stuttgart	34	12	13	9	272	87	128	57	0.713822	0.79109
9	SC Freiburg	34	12	13	9	306	96	125	85	0.739930	0.54887
10	TSG 1899 Hoffenheim	34	11	13	10	340	120	120	100	0.436111	0.30254
11	1. FSV Mainz 05	34	10	15	9	340	118	141	81	0.330397	0.32026
12	FC Augsburg	34	10	18	6	306	91	131	84	0.560026	0.09882
13	Hertha BSC	34	8	15	11	272	88	115	69	0.191480	0.35561
14	1. FC Köln	34	8	17	9	238	67	108	63	0.354132	0.25273
15	Werder Bremen	34	7	17	10	340	105	142	93	0.139969	0.13839
16	FC Schalke 04	34	3	24	7	340	140	119	81	0.000065	0.00001

In [ ]:
```
merged_df[merged_df['p-value (lost)']<=0.05]['TeamName']
```

Out[ ]:
```
16    FC Schalke 04
Name: TeamName, dtype: object
```

3. Bonferroni Correction

By now, we conducted 2 \cdot 17 = 34 hypotheses tests at the significance level \alpha = 5 \%. \alpha corresponds to the probability of a type I error, i.e. rejecting the null hypothesis given that it is true. However, the more tests we perform, the higher the chance of observing a rare event simply due to chance. For example, if we assume that H_0 holds for every team, the chance of falsely rejecting at least one out of 34 hypothesis is 1 - (1-0.05)^{34} \approx 83 \% (assuming independent tests). Thus, it is quite likely that one of the results we found is a type I error. The Bonferroni correction is one possibility for compensating for that effect by decreasing the significance level. The significance level is defined as the original one divided by the total number of hypotheses.

Task: Define the new significance level and see whether you can (still) find significant results.

In [ ]:
```
new_alpha = 0.05/34
merged_df[(merged_df['p-value (won)']<=new_alpha) | (merged_df['p-value (lost)']<=new_alpha)]['TeamName']
```

Out[ ]:
```
16    FC Schalke 04
Name: TeamName, dtype: object
```

The Bonferroni method tends to be too *conservative*, i.e. the significance level might be too restrictive. This is especially the case when the tests are dependent.

Task: Think about if the tests we performed are dependent or independent and give a short explanation.

Solution: We think the tests are dependent since the wins of one team influence the number of its losses and the number of wins of all other teams.

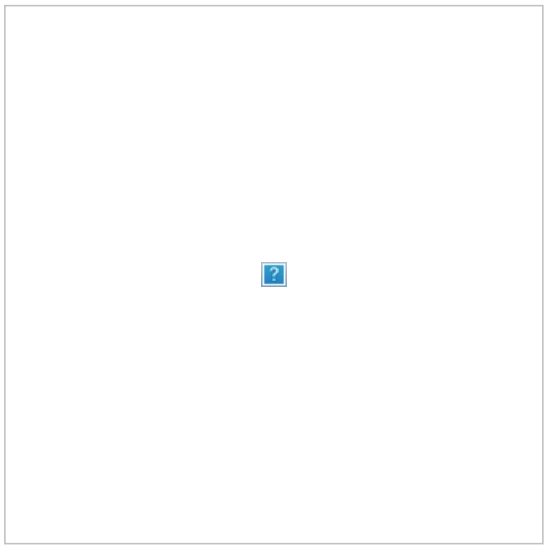
4. Q-Q-Plots

We conducted multiple hypothesis tests at significance level \alpha = 5 \%. \alpha corresponds to the probability of a type I error, i.e. \alpha = P(\text{type 1 error}) = P(p \leq \alpha\,|\,H_0\,\text{ is true}) = \text{cdf}_p(\alpha). So, under H_0, the cumulative distribution function of the p-value at \alpha is \text{cdf}_p(\alpha) = \alpha. That implies that p is uniformly distributed under H_0. Let us visually explore, if the observed p-values are likely to be drawn from a uniform distribution. This can be done by a so-called Q-Q plot.

The idea of a Q-Q Plot is to compare the empirical quantiles of the empirical distribution of the p-values with the quantiles of the theoretical distribution (the uniform distribution, as explained above). Let \beta \in (0, 1).

- The theoretical \beta-quantile of the uniform distribution is q_{\beta} = \beta.
- The empirical \beta-quantile of the *ascendingly ordered* sample (p_1, ..., p_n) is q_{\beta} = p_{\lfloor n\cdot\beta + 1\rfloor}

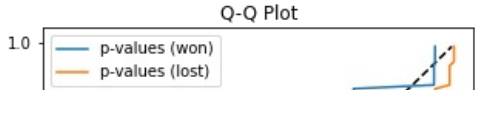
Task: Complement the two functions below such that they return the quantiles defined above. Do not use `numpy.quantile` or similar. Generate a vector that discretizes the variable \beta and compute the corresponding quantiles. Plot the theoretical quantiles over the empirical quantiles. If the distributions are *similar*, the points will lie on the 45° line y = x. The result should look like this:

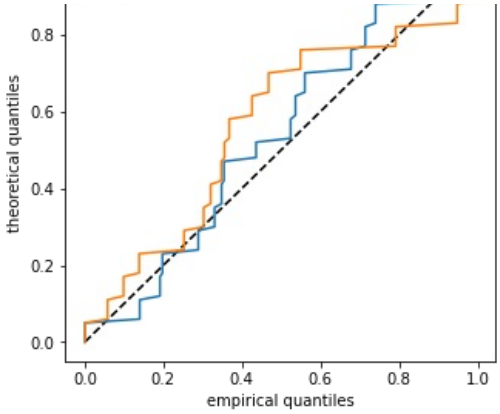


In [ ]:
```
import scipy.stats
from math import floor

def q_theoretical(beta):
    """
    Compute theoretical beta-quantile of uniform distribution.

    Parameters
    ----------
    beta : array-like, shape=(n,)
    """
    return scipy.stats.uniform.cdf(beta)

def q_empirical(beta, p):
    """
    Compute empirical beta-quantile of sample p.

    Parameters
    ----------
    beta : array-like, shape=(n,)
    p : array-like, shape=(n,)
        Unordered sample
    """
    n = len(p)
    quantile_idx = np.floor(n * beta+1).astype(int) -1
    return p[quantile_idx]
```

In [ ]:
```
discretized_beta = np.arange(0, 1, 0.01)
p_won_sorted = np.sort(merged_df['p-value (won)'].values)
p_lost_sorted = np.sort(merged_df['p-value (lost)'].values)
q_theo = q_theoretical(discretized_beta)
q_emp_won = q_empirical(discretized_beta, p_won_sorted)
q_emp_lost = q_empirical(discretized_beta, p_lost_sorted)
```

In [ ]:
```
plt.figure(figsize=(5,5))
plt.plot(q_theo, q_theo, color='black', linestyle='dashed')
plt.plot(q_emp_won, q_theo, label='p-values (won)')
plt.plot(q_emp_lost, q_theo, label='p-values (lost)')
plt.xlabel('empirical quantiles')
plt.ylabel('theoretical quantiles')
plt.legend()
plt.title('Q-Q Plot');
```





Of course, the two generated lines do not coincide perfectly with the 45° line $y = x$. That raises the question, what deviation from that line we would expect if the p-values were actually drawn from a uniform distribution. One way to approach this question visually is to generate multiple samples of a uniform distribution (each sample consisting of 17 numbers, like `p-values (won)` and `p-values (lost)`) and make a Q-Q Plot for each sample. So, we basically sample Q-Q plots under H_0.

Task: Generate multiple (e.g. 1000) samples as described above and show the corresponding Q-Q Plots together with the two lines from the previous task.

In [ ]:

```
plt.figure(figsize=(6,6))

for i in range(100):

    p_sample = np.sort(scipy.stats.uniform.rvs(size=17))
    q_emp = q_empirical(discretized_beta, p_sample)
    plt.plot(q_emp, q_theo, color='grey')

plt.plot(q_theo, q_theo, color='black', linestyle='dashed')
plt.plot(q_emp_won, q_theo, label='p-values (won)')
plt.plot(q_emp_lost, q_theo, label='p-values (lost)')
plt.xlabel('empirical quantiles')
plt.ylabel('theoretical quantiles')
plt.legend()
plt.title('Q-Q Plot');
```