Data Literacy

University of Tübingen, Winter Term 2021/22

Exercise Sheet 2

This sheet is **due on Monday, November 8, 2021 at 10am sharp (i.e. before the start of the lecture).**

Randomized Testing

In this week we will take a shallow dive into experimental design. We will work with the data obtained from the RKI about COVID-19 infections in Germany again. Our aim will be to design a randomized study to determine the rate of COVID-19 cases in Germany.

In [ ]:
```
# Make inline plots vector graphics
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("pdf", "svg")

# Plotting setup
import matplotlib.pyplot as plt

# Package imports
import numpy as np
import pandas as pd
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
#import fiona
#import geopandas
```

```
<ipython-input-79-b515736391a4>:4: DeprecationWarning: `set_matplotlib_formats` is deprecated since IPython 7.23,
directly use `matplotlib_inline.backend_inline.set_matplotlib_formats()`
  set_matplotlib_formats("pdf", "svg")
```

COVID-19: Relative Incidence in Germany

We will begin by computing the relative incidence (new cases normalized by population size) on a county (Landkreis) level for Germany.

Task: Load the most recent data from the RKI and find the cumulative cases per county (Landkreis) over time.

> Data Description of the RKI Covid-19-Dashboard (https://corona.rki.de)
>
> The data has the following features:
>
> - ...
> - Landkreis: Name of the county
> - ...
> - AnzahlFall: Number of cases in the respective population group.
> - ...
> - NeuerFall:
>   - 0: Case is contained in the data of today and the previous day
>   - 1: Case is only contained in today's data
>   - -1: Case is only contained in the previous day's data

Source (in German): https://www.arcgis.com/home/item.html?id=f10774f1c63e40168479a1feb6c7ca74

In [ ]:
```
# Link to current data of the RKI
url = "https://www.arcgis.com/sharing/rest/content/items/f10774f1c63e40168479a1feb6c7ca74/data"

# Read CSV data from URL
data_rki = pd.read_csv(url)
```

In [ ]:
```
# Create new dataframe and sort by date
data_rki['Meldedatum'] = pd.to_datetime(data_rki['Meldedatum'])
sorted_data_rki = data_rki.sort_values(by="Meldedatum")
sorted_data_rki.head()
```

Out[ ]:

	FID	IdBundesland	Bundesland	Landkreis	Altersgruppe	Geschlecht	AnzahlFall	AnzahlTodesfall	Meldedatum	IdLandkreis	Da
833261	833262	5	Nordrhein-Westfalen	LK Märkischer Kreis	A80+	M	1	0	2020-01-02	5962	05
2089736	2089737	10	Saarland	LK Stadtverband Saarbrücken	A80+	M	1	0	2020-01-23	10041	05
1679388	1679389	9	Bayern	LK Landsberg a.Lech	A15-A34	M	1	0	2020-01-28	9181	05
1723338	1723339	9	Bayern	LK Starnberg	A35-A59	M	1	0	2020-01-28	9188	05
1597810	1597811	9	Bayern	SK München	A15-A34	W	1	0	2020-01-29	9162	05

In [ ]:

```
# Cumulative case numbers over time
lk = sorted_data_rki.groupby(['Landkreis', 'IdLandkreis', 'Meldedatum'])['AnzahlFall'].sum().reset_index()
lk['cumsum'] = lk.groupby('Landkreis')['AnzahlFall'].transform(pd.Series.cumsum)
lk
```

Out[ ]:

	Landkreis	IdLandkreis	Meldedatum	AnzahlFall	cumsum
0	LK Ahrweiler	7131	2020-03-12	6	6
1	LK Ahrweiler	7131	2020-03-13	3	9
2	LK Ahrweiler	7131	2020-03-14	1	10
3	LK Ahrweiler	7131	2020-03-16	4	14
4	LK Ahrweiler	7131	2020-03-17	6	20
...	...	...	...	...	...
193203	Städteregion Aachen	5334	2021-10-31	16	32066
193204	Städteregion Aachen	5334	2021-11-01	9	32075
193205	Städteregion Aachen	5334	2021-11-02	73	32148
193206	Städteregion Aachen	5334	2021-11-03	228	32376
193207	Städteregion Aachen	5334	2021-11-04	80	32456

193208 rows × 5 columns

Our aim is to visualize the relative incidence as a colored map of Germany. For this we will use the package `geopandas`.

Task: Load the provided shapefile `data/Kreisgrenzen_2017_mit_Einwohnerzahl.shp`. Geopandas will return a dataframe that contains population numbers ("EWZ") and a column called "geometry" which defines the polygons making up the map of counties.

In [ ]:

```
import geopandas

# Geometric data and population numbers
germany_geo_df = geopandas.read_file("data/data_ex2/Kreisgrenzen_2017_mit_Einwohnerzahl.shp")
germany_geo_df.head()
```

Out[ ]:

	FID	RS	AGS	SDV_RS	GEN	BEZ	IBZ	BEM	SN_L	SN_R	...	FK_S3	NUTS	WSK	EWZ	KFL	Kennziffer	E
0	1	01001	01001	010010000000	Flensburg	Kreisfreie Stadt	40	--	01	0	...	R	DEF01	2008-01-01	88519	56.73	1001	
1	2	01002	01002	010020000000	Kiel	Kreisfreie Stadt	40	--	01	0	...	R	DEF02	2006-01-01	247943	118.65	1002	
2	3	01003	01003	010030000000	Lübeck	Kreisfreie Stadt	40	--	01	0	...	R	DEF03	2006-02-01	216318	214.19	1003	
3	4	01004	01004	010040000000	Neumünster	Kreisfreie Stadt	40	--	01	0	...	R	DEF04	1970-04-26	79335	71.66	1004	
4	5	01051	01051	010510044044	Dithmarschen	Kreis	42	--	01	0	...	R	DEF05	2011-08-01	133447	1428.18	1051	

5 rows × 24 columns

In [ ]:
```
# County IDs not in geometric data
county_ids_rki = lk.IdLandkreis.unique()
county_ids_geo = germany_geo_df.Kennziffer.unique()

# Find IDs only in one of the two county ID sets
unmatched_ids = np.setxor1d(county_ids_rki, county_ids_geo)
print(f"County IDs with non-matching IDs: \n{unmatched_ids}")
print(
    f"Counties with non-matching IDS: \n{lk[lk.IdLandkreis.isin(unmatched_ids)].Landkreis.unique()}"
)
```

```
County IDs with non-matching IDs:
[11000 11001 11002 11003 11004 11005 11006 11007 11008 11009 11010 11011
 11012 16056]
Counties with non-matching IDS:
['SK Berlin Charlottenburg-Wilmersdorf'
 'SK Berlin Friedrichshain-Kreuzberg' 'SK Berlin Lichtenberg'
 'SK Berlin Marzahn-Hellersdorf' 'SK Berlin Mitte' 'SK Berlin Neukölln'
 'SK Berlin Pankow' 'SK Berlin Reinickendorf' 'SK Berlin Spandau'
 'SK Berlin Steglitz-Zehlendorf' 'SK Berlin Tempelhof-Schöneberg'
 'SK Berlin Treptow-Köpenick']
```

In [ ]:
```
# Aggregate data in Berlin in temporary data frame
lk_berlin = (
    lk[lk.IdLandkreis.isin(unmatched_ids)].groupby(["Meldedatum"]).sum()
).reset_index()
lk_berlin.loc[:, "IdLandkreis"] = 11000
lk_berlin.loc[:, "Landkreis"] = "Berlin"
```

In [ ]:
```
lk_berlin
```

Out[ ]:

	Meldedatum	IdLandkreis	AnzahlFall	cumsum	Landkreis
0	2020-02-29	11000	1	1	Berlin
1	2020-03-03	11000	6	7	Berlin
2	2020-03-04	11000	3	3	Berlin
3	2020-03-05	11000	10	16	Berlin
4	2020-03-06	11000	5	16	Berlin
...	...	...	...	...	...
605	2021-10-31	11000	117	157953	Berlin
606	2021-11-01	11000	1233	225359	Berlin
607	2021-11-02	11000	1379	226738	Berlin
608	2021-11-03	11000	1494	228232	Berlin
609	2021-11-04	11000	1118	229350	Berlin

610 rows × 5 columns

In [ ]:
```
# Drop Berlin rows from RKI data and append merged case numbers
lk.drop(
    lk.index[np.where(lk.IdLandkreis.isin(unmatched_ids))[0]],
    inplace=True,
)
lk = lk.append(lk_berlin)
```

In [ ]:
```
lk
```

Out[ ]:

	Landkreis	IdLandkreis	Meldedatum	AnzahlFall	cumsum
0	LK Ahrweiler	7131	2020-03-12	6	6
1	LK Ahrweiler	7131	2020-03-13	3	9
2	LK Ahrweiler	7131	2020-03-14	1	10
3	LK Ahrweiler	7131	2020-03-16	4	14

4	LK Ahrweiler	7131	2020-03-17	6	20
...	...	...	...	...	...
605	Berlin	11000	2021-10-31	117	157953
606	Berlin	11000	2021-11-01	1233	225359
607	Berlin	11000	2021-11-02	1379	226738
608	Berlin	11000	2021-11-03	1494	228232
609	Berlin	11000	2021-11-04	1118	229350

187717 rows × 5 columns

Task: Create a joint dataframe with an additional column that contains the relative incidences (new cases of COVID-19 divided by county population). What are the five top and bottom counties in terms of relative incidence for the current day?

In [ ]:
```
# Merge into single data frame
germany_geo_df.rename(columns={'Kennziffer':'IdLandkreis'}, inplace=True)
merged_lk = lk.merge(germany_geo_df, how='outer', on='IdLandkreis')
```

In [ ]:
```
# Compute relative incidence
merged_lk['relative_incidence'] = merged_lk['AnzahlFall']/merged_lk['EWZ']

# Compute relative cumulative case numbers
merged_lk['relative_cumulative'] = merged_lk['cumsum']/merged_lk['EWZ']
```

In [ ]:
```
merged_lk.head()
```

Out[ ]:

	Landkreis	IdLandkreis	Meldedatum	AnzahlFall	cumsum	FID	RS	AGS	SDV_RS	GEN	...	NUTS	WSK	EWZ	KFL	EV
0	LK Ahrweiler	7131	2020-03-12	6.0	6.0	144	07131	07131	071310007007	Ahrweiler	...	DEB12	2009-01-01	128914	787.02	
1	LK Ahrweiler	7131	2020-03-13	3.0	9.0	144	07131	07131	071310007007	Ahrweiler	...	DEB12	2009-01-01	128914	787.02	
2	LK Ahrweiler	7131	2020-03-14	1.0	10.0	144	07131	07131	071310007007	Ahrweiler	...	DEB12	2009-01-01	128914	787.02	
3	LK Ahrweiler	7131	2020-03-16	4.0	14.0	144	07131	07131	071310007007	Ahrweiler	...	DEB12	2009-01-01	128914	787.02	
4	LK Ahrweiler	7131	2020-03-17	6.0	20.0	144	07131	07131	071310007007	Ahrweiler	...	DEB12	2009-01-01	128914	787.02	

5 rows × 30 columns

In [ ]:
```
# Case numbers for most recent date with >0 new cases
print(merged_lk[(merged_lk['Meldedatum']=='2021-11-04') & (merged_lk['AnzahlFall']>0)][['Landkreis', 'AnzahlFall'

# Top and bottom 5 counties in terms of relative cumulative incidence for today
lk_today = merged_lk[merged_lk['Meldedatum']=='2021-11-04']
print('Bottom LK: ', lk_today.sort_values('relative_cumulative').loc[:, 'Landkreis'].iloc[:5].values)
print('Top LK: ', lk_today.sort_values('relative_cumulative').loc[:, 'Landkreis'].iloc[-5:].values)
```

```
                Landkreis  AnzahlFall
497           LK Ahrweiler        43.0
967     LK Aichach-Friedberg       103.0
1512      LK Alb-Donau-Kreis       122.0
1937     LK Altenburger Land        41.0
2396         LK Altenkirchen        32.0
```

```
...                      ...         ...
185718          SK Wuppertal       104.0
186225           SK Würzburg        24.0
186532        SK Zweibrücken         7.0
187106   Städteregion Aachen        80.0
187716                Berlin      1118.0

[395 rows x 2 columns]
Bottom LK:  ['LK Plön' 'LK Schleswig-Flensburg' 'LK Rendsburg-Eckernförde'
 'LK Friesland' 'LK Ostholstein']
Top LK:  ['LK Hildburghausen' 'LK Wartburgkreis'
 'LK Sächsische Schweiz-Osterzgebirge' 'LK Bautzen' 'LK Erzgebirgskreis']
```

Task: Using `geopandas` and the created dataframe plot Germany's counties and their current relative incidence color-coded. Where is the relative incidence currently highest? What might be the causes for this result? What type of colormap is appropriate for this visualization and why?

Hint: To use the native plotting functionality of `geopandas` convert the data frame you just created into a `GeoDataFrame`.

In [ ]:
```
# Plot map
geo_lk = geopandas.GeoDataFrame(lk_today)
geo_lk.plot(column='relative_incidence')
print('Sadly, this cell kill out kernel :(')
```

Designing a Testing Strategy

Suppose you are in charge of estimating the relative incidence in Germany on a national level. Let's say you have a certain varying budget of tests to distribute each day. However, you do *not* know the total number of tests available at the start of the day. Instead as the day progresses you are informed about new test capacities in batches of tests. You have to distribute this testing capacity immediately as it becomes available. To do so, after receiving a new batch of tests you can ask a designated contact in any county to test a certain number of randomly selected people in that county.

How would you distribute the tests arriving in batches to estimate the relative incidence in Germany without introducing (sampling) bias?

Task: Implement an algorithm to sample from a categorical distribution over arbitrary categories given a vector of probability weights and a function returning uniform random samples on the unit interval. That is, an algorithm which draws with replacement from a fixed number of categories according to a set of weights.

Note: Any other sampling functionality from `numpy` or `scipy` beyond `np.random.uniform` should not be used!

In [ ]:
```
def sample_categorical(categ, p, size=()):
    """
    Sample from a categorical distribution.

    Parameters
    ----------
    categ : array-like, shape=(n,)
        Categories to sample from.
    p : array-like, shape=(n,)
        Probability weights of drawing from the different categories.
    size : tuple
        Size of the sample.
    """
    samples = np.empty(size, dtype=object)
    prob = p / np.sum(p)
    cumsum = np.cumsum(prob)
    for i in range(samples.shape[0]):
        for j in range(samples.shape[1]):

            rand = np.random.uniform()
            arr = np.where(rand <= cumsum)
            idx = arr[0][0]
            samples[i][j] = categ[idx]

    return samples
```

In [ ]:
```
t = sample_categorical(categ=["a", "b", "c"], p = [1, 4, 6], size=(4, 5))
t
```

Out[ ]:
```
array([['b', 'a', 'c', 'c', 'b'],
       ['b', 'c', 'c', 'b', 'b'],
       ['c', 'c', 'c', 'b', 'a'],
       ['a', 'b', 'b', 'c', 'a']], dtype=object)
```

Task: Using the above sampling algorithm design a testing strategy which allocates a newly received batch of tests across the different counties at any time of the day.

In [ ]:
```
categ = germany_geo_df['GEN'].values
p = germany_geo_df['EWZ'].values
categ.shape, p.shape
```

Out[ ]:
```
((401,), (401,))
```

In [ ]:
```
def testing_strategy(n_tests, counties, population):
    """
    Testing strategy for COVID-19 on a county level.

    Parameters
    ----------
    n_tests : int
        Number of available tests.
    counties : array-like
        Counties where tests can be distributed.
    population : array-like
        Population of each county.
    """
    sample_test = sample_categorical(counties, population, (1,n_tests))
    unique, counts = np.unique(sample_test, return_counts=True)
    return dict(zip(unique, counts))
```

In [ ]:
```
testing_strategy(10, categ, p)
```

Out[ ]:
```
{'Augsburg': 1,
 'Donau-Ries': 1,
 'Frankfurt am Main': 1,
 'Heidekreis': 1,
 'Karlsruhe': 1,
 'Main-Kinzig-Kreis': 1,
 'Osnabrück': 1,
 'Rems-Murr-Kreis': 1,
 'Westerwaldkreis': 2}
```

Task: How would you argue that your sampling strategy is *unbiased*, meaning that it constitutes a representative sample of the German population?

It is unbiased since every citizen of germany at any time of the day has an equal probability of receiving a test (assuming the are distributed without any bias within a county). This is due to taking the proportion of citizens per county into account and distributing the test independently of previous samples.

EXAMple

$$\begin{aligned}\mathbb{E}_p[(x-\mathbb{E}_p[x])^2] &= \mathbb{E}_p[x^2 - 2x\cdot\mathbb{E}_p[x] + (\mathbb{E}_p[x])^2] \\ &= \mathbb{E}_p[x^2] - 2\mathbb{E}_p[x\cdot\mathbb{E}_p[x]] + \mathbb{E}_p[(\mathbb{E}_p[x])^2] \\ &= \mathbb{E}_p[x^2] - 2\mathbb{E}_p[x]\cdot\mathbb{E}_p[x] + (\mathbb{E}_p[x])^2 \\ &= \mathbb{E}_p[x^2] - (\mathbb{E}_p[x])^2\end{aligned}$$

Theroy Question

Fortune after n trials with m wins: $c_n = c_0 \cdot 2^m \cdot \frac{1}{2}^{(n-m)}$

Expected fortune:

$$\begin{aligned}\mathbb{E}_{p(m|n)}[c_n] &= \sum_{m=0}^{m=n} c_0\cdot 2^m\cdot(\frac{1}{2})^{(n-m)}\cdot p(m|n) \\ &= c_0\cdot\sum_{m=0}^{m=n}\binom{n}{m}(\frac{1}{2})^m\cdot 2^m\cdot(\frac{1}{2})^{(n-m)}\cdot(\frac{1}{2})^{(n-m)} \\ &= c_0\cdot\sum_{m=0}^{m=n}\binom{n}{m}1^m\cdot(\frac{1}{4})^{(n-m)} \\ &= c_0\cdot(\frac{5}{4})^n\end{aligned}$$

Variance of fortune:

$$\begin{aligned}&\mathbb{E}_{p(m|n)}[c_n^2] - [\mathbb{E}_{p(m|n)}[c_n]]^2 \\ &= \mathbb{E}_{p(m|n)}[(c_0\cdot 2^m\cdot(\frac{1}{2})^{(n-m)})^2] - [c_0\cdot(\frac{5}{4})^n]^2 \\ &= \mathbb{E}_{p(m|n)}[c_0^2\cdot 4^m\cdot(\frac{1}{4})^{(n-m)}] - [c_0^2\cdot(\frac{25}{16})^n] \\ &= c_0^2\cdot\sum_{m=0}^{m=n}[\binom{n}{m}\cdot 4^m\cdot(\frac{1}{2})^m\cdot(\frac{1}{4})^{(n-m)}\cdot(\frac{1}{2})^{(n-m)}] - [c_0^2\cdot(\frac{25}{16})^n] \\ &= c_0^2\cdot\sum_{m=0}^{m=n}[\binom{n}{m}\cdot 2^m\cdot(\frac{1}{8})^{(n-m)}] - [c_0^2\cdot(\frac{25}{16})^n] \\ &= c_0^2\cdot[(\frac{17}{8})^n - (\frac{25}{16})^n]\end{aligned}$$

p-th non-central moment:

$$
\begin{aligned}
\mathbb{E}_{p(m|n)}[c_n^p] &= \sum_{m=0}^{m=n} [(c_0 \cdot 2^m \cdot (\frac{1}{2})^{(n-m)})^p \cdot p(m|n)] \\
&= c_0^p \cdot \sum_{m=0}^{m=n} [(2^p)^m \cdot ((\frac{1}{2})^p)^{(n-m)} \cdot p(m|n)] \\
&= c_0^p \cdot \sum_{m=0}^{m=n} [\binom{n}{m} (2^{(p-1)})^m \cdot ((\frac{1}{2})^{(p+1)})^{(n-m)}] \\
&= c_0^p \cdot [2^{(p-1)} + (\frac{1}{2})^{(p+1)}]^n
\end{aligned}
$$