# 6. Advanced plotting

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import seaborn as sns
```

We have seen already two options to plot data: we can use the "raw" Matplotlib which in principle allows one to create any possible plot, however with lots of code, and we saw the simpler internal Pandas solution. While the latter solution is very practical to quickly look through data, it is rather cumbersome to realise more complex plots.

Here we look at another type of plotting resting on the concepts of the grammar of graphics. This approach allows to create complex plots where data can be simply split in a plot into color, shapes etc. without having to do a grouping operation in beforehand. We will mainly look at Seaborn, and finish with an example with Plotnine, the port to Python of ggplot.

## Importing data

We come back here to the dataset of swiss towns. To make the dataset more interestig we add to it some categorical data. First we attempt to add the main language for each town. It is a good example of the type of data wranglig one ofen has to do by combining information from different sources.

In [2]:
```
#load table indicating to which canton each town belongs
cantons = pd.read_excel('Datasets/be-b-00.04-osv-01.xls',sheet_name=
1)[['KTKZ','ORTNAME']]
```

In [3]:
```
#load general table with infos on towns
towns = pd.read_excel('Datasets/2018.xls', skiprows=list(range(5))+list
(range(6,9)),
                      skipfooter=34, index_col='Commune',na_values=['*
','X'])
towns = towns.reset_index()
```

In [4]:
```
#merge tables using the town name. This adds the canton abbreviation to
the main table
towns_canton = pd.merge(towns, cantons, left_on='Commune', right_on='ORT
NAME',how = 'inner')
```

In [5]:
```
#load data indicating languages of each canton
language = pd.read_excel('Datasets/je-f-01.08.01.02.xlsx',skiprows=[0,2,
3,4],skipfooter=11)
languages = language[['Allemand (ou suisse allemand)','Français (ou pato
is romand)',
         'Italien (ou dialecte tessinois/italien des grisons)']]
languages = languages.apply(pd.to_numeric, errors='coerce')
#check which language has majority in each canton
languages['language'] = np.argmax(languages.values.astype(float),axis=1)
code={0:'German', 1:'French', 2:'Italian'}
languages['Language'] = languages.language.apply(lambda x: code[x])
languages['canton'] = language['Unnamed: 0']
languages = languages[['canton','Language']]

#load table matching canton name to abbreviation
cantons_abbrev = pd.read_excel('Datasets/cantons_abbrev.xlsx')
#add full canton name to table by merging on abbreviation
canton_language = pd.merge(languages, cantons_abbrev,on='canton')
```

In [6]:
```
#add language by merging on canton abbreviation
towns_language = pd.merge(towns_canton, canton_language, left_on='KTKZ',
right_on='abbrev')
```

In [7]:
```
towns_language['town_type'] = towns_language['Surface agricole en %'].ap
ply(lambda x: 'Land' if x<50 else 'City')
```

In [8]:
```
#Create a new party column and a new party score column
parties = pd.melt(towns_language,id_vars=['Commune'], value_vars=['UDC
','PS','PDC'],
                  var_name= 'Party', value_name='Party score')
towns_language = pd.merge(parties, towns_language, on='Commune')

towns_language
```

Out[8]:

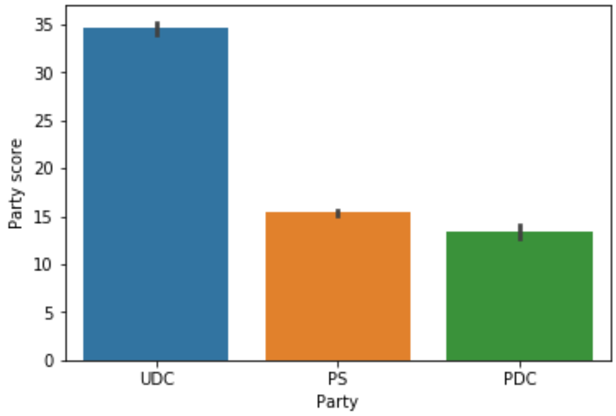
| | Commune | Party | Party score | Code commune | Habitants | Variation en % | Densité de la population par km² | Etrangers en % |
|---|---|---|---|---|---|---|---|---|
| **0** | Aeugst am Albis | UDC | 30.929249 | 1 | 1977 | 8.388158 | 249.936789 | 13.100658 |
| **1** | Aeugst am Albis | PS | 18.645940 | 1 | 1977 | 8.388158 | 249.936789 | 13.100658 |
| **2** | Aeugst am Albis | PDC | 2.076428 | 1 | 1977 | 8.388158 | 249.936789 | 13.100658 |
| **3** | Affoltern am Albis | UDC | 33.785785 | 2 | 11900 | 7.294203 | 1123.701605 | 27.848740 |
| **4** | Affoltern am Albis | PS | 19.080314 | 2 | 11900 | 7.294203 | 1123.701605 | 27.848740 |
| **5** | Affoltern am Albis | PDC | 4.585387 | 2 | 11900 | 7.294203 | 1123.701605 | 27.848740 |
| **6** | Bonstetten | UDC | 29.100156 | 3 | 5435 | 5.349874 | 731.493943 | 14.149034 |
| **7** | Bonstetten | PS | 20.403265 | 3 | 5435 | 5.349874 | 731.493943 | 14.149034 |
| **8** | Bonstetten | PDC | 3.378541 | 3 | 5435 | 5.349874 | 731.493943 | 14.149034 |
| **9** | Hausen am Albis | UDC | 34.937369 | 4 | 3571 | 6.279762 | 262.573529 | 14.533744 |
| **10** | Hausen am Albis | PS | 19.393305 | 4 | 3571 | 6.279762 | 262.573529 | 14.533744 |
| **11** | Hausen am Albis | PDC | 2.881915 | 4 | 3571 | 6.279762 | 262.573529 | 14.533744 |
| **12** | Hedingen | UDC | 30.114599 | 5 | 3687 | 8.123167 | 564.624809 | 14.971522 |
| **13** | Hedingen | PS | 22.478008 | 5 | 3687 | 8.123167 | 564.624809 | 14.971522 |
| **14** | Hedingen | PDC | 3.918166 | 5 | 3687 | 8.123167 | 564.624809 | 14.971522 |
| **15** | Kappel am Albis | UDC | 48.615099 | 6 | 1110 | 20.915033 | 140.151515 | 18.018018 |
| **16** | Kappel am Albis | PS | 10.285425 | 6 | 1110 | 20.915033 | 140.151515 | 18.018018 |
| **17** | Kappel am Albis | PDC | 2.744469 | 6 | 1110 | 20.915033 | 140.151515 | 18.018018 |
| **18** | Knonau | UDC | 32.876136 | 7 | 2168 | 20.444444 | 335.085008 | 17.158672 |
| **19** | Knonau | PS | 18.436553 | 7 | 2168 | 20.444444 | 335.085008 | 17.158672 |
| **20** | Knonau | PDC | 3.126052 | 7 | 2168 | 20.444444 | 335.085008 | 17.158672 |
| **21** | Maschwanden | UDC | 43.383446 | 8 | 626 | 1.623377 | 133.475480 | 12.140575 |
| **22** | Maschwanden | PS | 22.732529 | 8 | 626 | 1.623377 | 133.475480 | 12.140575 |
| **23** | Maschwanden | PDC | 3.502396 | 8 | 626 | 1.623377 | 133.475480 | 12.140575 |
| **24** | Mettmenstetten | UDC | 35.671015 | 9 | 4861 | 14.565166 | 373.062164 | 14.873483 |
| **25** | Mettmenstetten | PS | 18.800282 | 9 | 4861 | 14.565166 | 373.062164 | 14.873483 |
| **26** | Mettmenstetten | PDC | 3.649155 | 9 | 4861 | 14.565166 | 373.062164 | 14.873483 |
| **27** | Obfelden | UDC | 36.174029 | 10 | 5131 | 9.496372 | 680.503979 | 20.015591 |

## Basic plotting

We finally have a table with mostly numerical information but also two categorical data: language and town type (land or city). With Seaborn we can now easily make all sorts of plots. For example what are the average scores of the different parties:

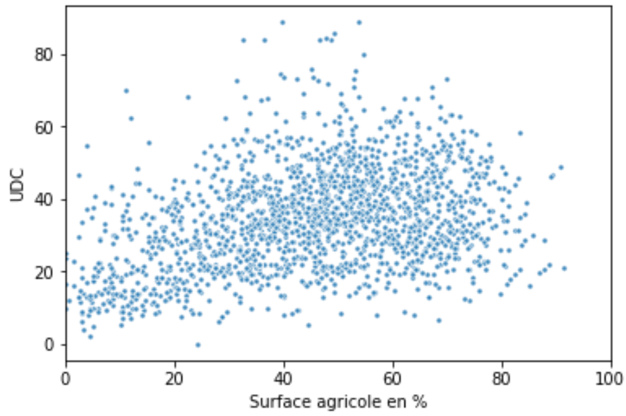
In [9]:
```
sns.barplot(data = towns_language, y='Party score', x = 'Party');
```

```
/usr/local/lib/python3.5/dist-packages/scipy/stats/stats.py:1713: FutureW
arning: Using a non-tuple sequence for multidimensional indexing is depre
cated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this wi
ll be interpreted as an array index, `arr[np.array(seq)]`, which will res
ult either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```




Do land towns vote more for the right-wing party ?

In [10]:
```
g = sns.scatterplot(data = towns_language, y='UDC', x = 'Surface agricol
e en %', s = 10, alpha = 0.5);
g.set_xlim([0,100]);
```

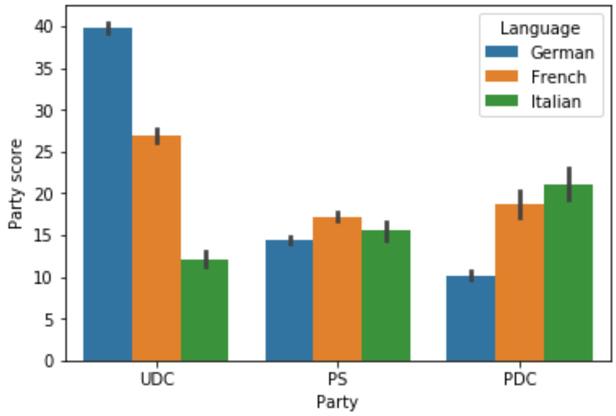



## Using categories as "aesthetics"

The greate advantage of using these packages is that they allow to include categories as "aesthetics" of the plot. For example we looked before at average party scores. But are they different between language regions ? We can just specify that the hue (color) should be mapped to the town language:

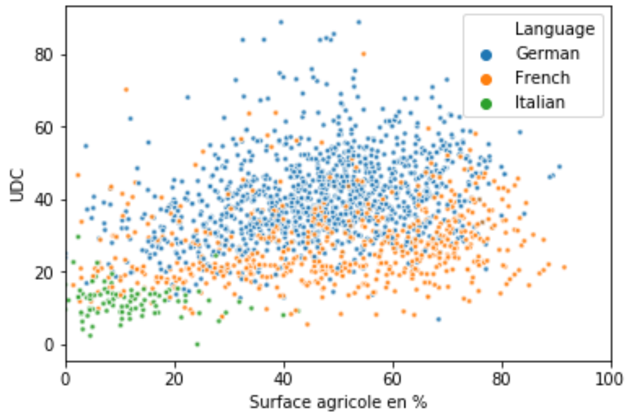
In [11]:
```
sns.barplot(data = towns_language, y='Party score', x = 'Party', hue = '
Language');
```

```
/usr/local/lib/python3.5/dist-packages/scipy/stats/stats.py:1713: FutureW
arning: Using a non-tuple sequence for multidimensional indexing is depre
cated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this wi
ll be interpreted as an array index, `arr[np.array(seq)]`, which will res
ult either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```




Similarly with scatter plots. Is the relation between land and voting on the right language dependent ?

In [12]:
```
g = sns.scatterplot(data = towns_language, y='UDC', x = 'Surface agricol
e en %', hue = 'Language',
                     s = 10, alpha = 0.5);
g.set_xlim([0,100]);
```

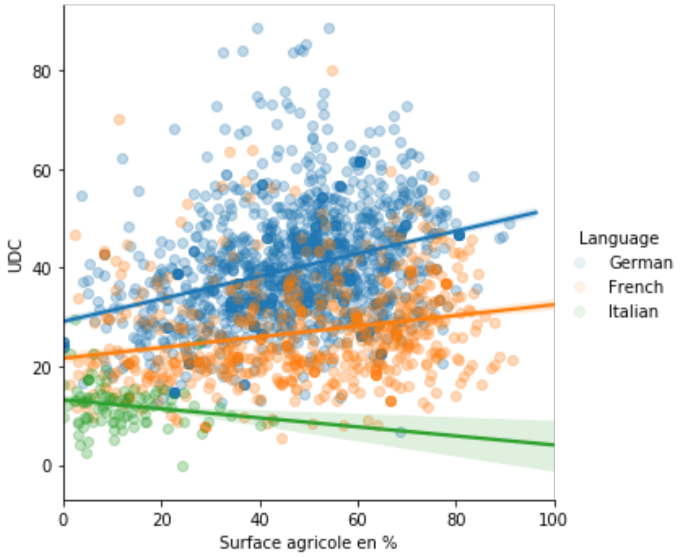



## Statistics

We see difference in the last plot, but it is still to clearly see the relation. Luckiliy these packages allow us to either create summary statistics or to fit the data:

In [13]:
```
g = sns.lmplot(data = towns_language, x = 'Surface agricole en %', y='UD
C', hue = 'Language', scatter=True,
              scatter_kws={'alpha': 0.1});
g.ax.set_xlim([0,100]);
```

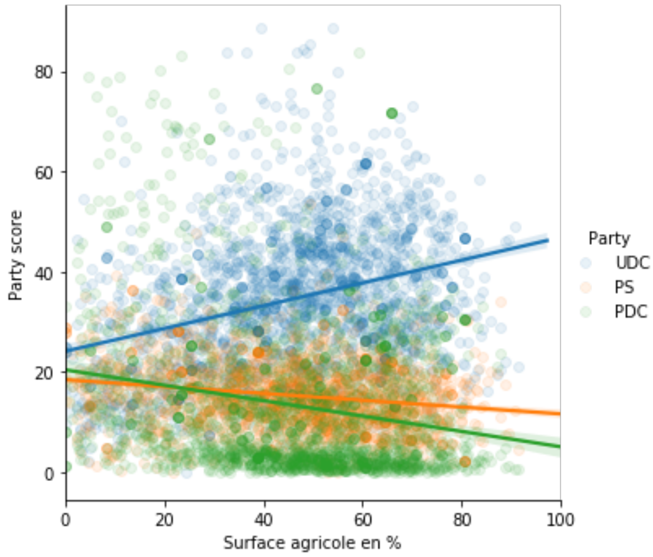
```
/usr/local/lib/python3.5/dist-packages/scipy/stats/stats.py:1713: FutureW
arning: Using a non-tuple sequence for multidimensional indexing is depre
cated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this wi
ll be interpreted as an array index, `arr[np.array(seq)]`, which will res
ult either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```




Now we can also do the same exercise for all parties. Does the relation hold?

In [14]:
```
g = sns.lmplot(data = towns_language, x = 'Surface agricole en %', y='Pa
rty score',
               hue = 'Party', scatter=True,
              scatter_kws={'alpha': 0.1});
g.ax.set_xlim([0,100]);
```

```
/usr/local/lib/python3.5/dist-packages/scipy/stats/stats.py:1713: FutureW
arning: Using a non-tuple sequence for multidimensional indexing is depre
cated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this wi
ll be interpreted as an array index, `arr[np.array(seq)]`, which will res
ult either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```




## Adding eve more information

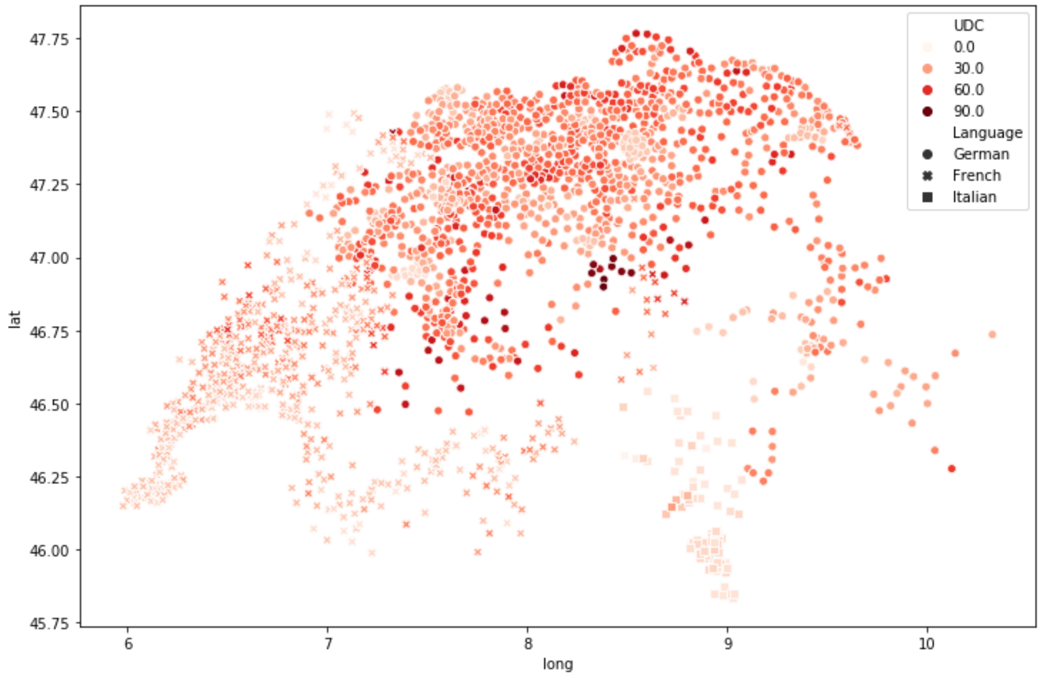
We can recover from some other place (Poste) the coordinates of each town. Again by merging we can add that information to our main table:

In [15]:
```
coords = pd.read_csv('Datasets/plz_verzeichnis_v2.csv', sep=';')[['ORTBE
Z18','Geokoordinaten']]
coords['lat'] = coords.Geokoordinaten.apply(lambda x: float(x.split(',
')[0]) if type(x)==str else np.nan)
coords['long'] = coords.Geokoordinaten.apply(lambda x: float(x.split(',
')[1]) if type(x)==str else np.nan)
```

In [16]:
```
towns_language = pd.merge(towns_language,coords, left_on='Commune', righ
t_on='ORTBEZ18')
```

So now we can in addition look at the geography of these parameters. For example, who votes for the right-wing party ?

In [17]:
```
fix, ax = plt.subplots(figsize = (12,8))
sns.scatterplot(data = towns_language, x= 'long', y = 'lat', hue='UDC',
style = 'Language', palette='Reds');
```




In [18]:
```
# MZ: if used to ggplot -> use 'plotnine' package
# same grammar as ggplot
```