Statistical Modelling with R: Is Python an alternative?

Tobias Giesemann

July 26, 2019

1 Introduction

With the increasing availability of large data sets in social sciences, a need for advanced tools for statistical measurements has evolved in recent years. In this work I want to research, if the programming language Python could be used in a similarly convenient way as R. R is the predominant software for statistical analysis on the level of open-source and interpreted programming languages and it is constantly challenged by "newcomers" such as Python. So can Python, a more general programming language cope with this more specialized software?

Softwares for statistical modeling The available software packages for statistical modeling in social sciences have become increasingly diverse and can be categorized in three major groups:
1. high-level programs designed for an easy access to statistical modeling such as Stata or SPSS. These languages allow to analyze data by clicking through a user-friendly interface, but they also have a syntax option. These softwares are usually the ones taught to undergraduate students in the beginning of their studies and they are proprietary.
2. medium-level interpreted programming languages that offer all features necessary for not only statistical analysis but also for software development, web-development and larger projects. These languages contain structures like user-defined functions, classes, objects, functional programming options, individual type-structures and options for parallelization and vectorization. Prominent representatives of this kind of language are R, Python, Matlab or even JavaScript. These programs are mostly non-proprietary (excluding Matlab) and have community-driven extensions ("packages"). For statistical modeling R is probably the most widely used software package, whereas in the Data Science research and entrepreneurial environment, Python is often predominant.
3. low-level non-interpreted programming languages, such as C/C++, Java, Go. These languages are almost entirely used for large-scale software development. The code written in these languages has the great advantage of being compiled and having a fixed type system, which speeds up the execution time of programs

by several factors. For statistical modeling though, these programs are not as widely used, as the development cost is often too high.

As we can see, Python and R are both in the second group of languages and can thus be well compared. In the following I will point out key differences between the two languages and use some of the R code used in the lecture "Advanced Statistical Modeling" [2] and transfer it to Python. In the conclusion I want to compare not only the technical differences but also the touch and feel of both languages and frameworks.

2 Comparing R and Python

2.1 General

Here, I want to compare technical features of R and Python. The question is, if one is used to work in either environment, how would a transition to the other be most comfortable, what things are different, what are the same.

Coding environment Looking at the coding environment of R we have to assert that most researchers work with RStudio. RStudio is an IDE (Integrated Development Environment) that has some very handy features. First, it lets you run certain parts of code by marking and hitting `Ctrl+Enter`. This makes the work flow for data design very neat. Also, if there is a computationally intensive part in your code, you won't have to rerun it, if you just want to modify some minor (but possibly important) variable. It also comes with some standard features, such as a file manager, R-Console and Terminal, an overview of the current environment, code history and package manager. Another remarkable feature is the built-in Plots-Pane where you can zoom, save and browse through your recent plots. I will come back to visualization in R later. Software in R can easily be written in RProjects which allows you to transfer your working directory to other locations without having to re-build links or csv-loads that are in the same folder. Extension packages in R are loaded into the script via the `require(<packagename>)` command and installed via `install.packages(<packagename>)`. I will come back to the most important R packages later.

In Python, the coding environment is somewhat different. If you come from a software development point-of-view, you might be familiar with IDEs such as Atom or PyCharm, that provide a good language integration, but where you cannot select the chucks of code that you would like to be executed. To cope with this restriction, there is a locally hosted environment for web browsers called JyPyter Notebook, more recently JuPyter Lab. There, the workflow comes closer to the one in RStudio, as you can separate your code into different cells, each of which are individually runnable. I have considered this environment for my comparison with R and a recent make-over of the JuPyter-Lab has increased the usability further. The code is then saved in .ipynb-files. For the package management in Python, the user can choose between two dif-

Figure 1: Comparing R and Python

```
[1]: def easy_function(a,b,c):
         result=24
         for x in a:
             result += x
         if b != 0:
             return result/b
         else:
             if c == True:
                 return(result)
             else:
                 return 2*result

     result = easy_function([1,2,3,4], 6, True)
     print(result)

     5.666666666666667
```

(a) Easy function in Python

```
1  easy_function <- function(a,b,c){
2    result = 24
3    for(x in a){
4      result = result+x
5    }
6    if(b != 0){
7      return(result/b)
8    }
9    else{
10     if(c == TRUE){
11       return(result)
12     }
13     else{
14       return(result*2)
15       }
16   }
17 }
18
19 result = easy_function(c(1,2,3,4),6,TRUE)
20
```

(b) Easy function in R

ferent interpreters, PyPI (pip3 for Python3) and Anaconda (conda). Both of them offer nearly all available packages, the advantage of Anaconda is that it can create project-specific environments. This is needed if you want your code to run on a distinct environment, e.g. a Server with only certain available packages. The usage of Anaconda is thus mostly limited to the full software development in Python. Packages can be easily installed (Terminal on UNIX and Windows Pip Command Software) via `pip3 install <packagename>` or `conda install <packagename>`. This is not as straight-forward as in R, where everything can be done from the code itself. Loading packages into the code is then done via `import <packagename>` or `from <packagename>[.<class>] import [*|<class>|<method>|<variable>]` for a more refined import. You cannot use in your code anything that has not been explicitly imported, which keeps the imports well structured.

Code and Applications Comparing Python and R with respect to their code structure leads to a couple of interesting insights:
1. Control structures: As can be seen from Figure 1, we can see that the control structures maintain the same structure for Python and R alike. The main difference is that Python uses colons and requires fixed indentations to structure the code, whereas R uses C-style curly brackets. There are many voices who claim that the Python style is more human-readable ([1]) . One advantage over R and most other languages is undeniably that you will not have trouble figuring out which bracket belongs where. I don't want to go into detail here, there would be a lot to talk about, especially on the Python side, where developers have come up with great ideas to make the code more readable.
2. Parallelization and Scalability: From the point of computational science both R and Python are scalable and code can be written highly parallel. For

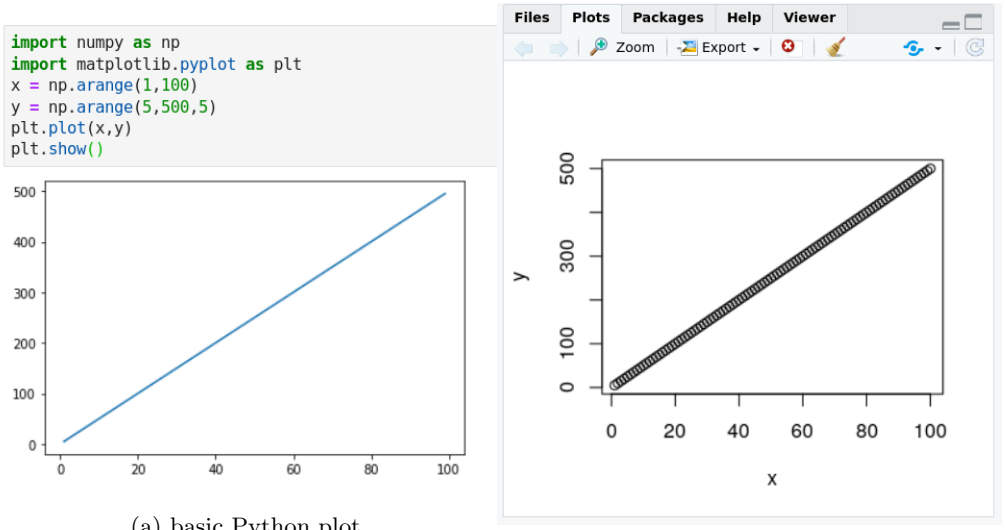
Python it is possible with the `multithreading` package, in R it is `parallel`. Both languages also offer support for MPI (Message Passing Interface) for large-scale parallelization over multiple machines, `mpi4py` on Python and `rmpi` for R. Finally, SIMD vectorization for computing array-based operations is native in numpy for Python and can easily be done via the `vectorize()` command in R. Finally, for R(`gpuR`) and for Python(i.e. `PyCUDA`) there exist possibilities to compute code on GPUs. As such, Python and R can both computationally exploit the surrounding hardware and neither of both languages has a big advantage over the other.
3. Data Structures: In R, the main data structures used, are the built-in data frames. Especially for statistical methods, the operations on these data frames are highly specialized. It can handle numerical as well as categorical and string data. I.e., the classic `lm()` method for a linear regression in R automatically ignores NaN-values in the computation. The basic data structures for arrays (data vectors) is also compatible with the data frames and a vector can be initialized via the `c(1,2,3,...)` command.
In Python this concept doesn't translate automatically, because Python starts from a list-based point of view. A list is thereby not automatically a vector, neither is a list of lists a data frame or matrix. Therefore the package `numpy`, one of the first packages for Python, gives many operations and a `numpy` array `numpy.array([1,2,3,...])` could be considered the equivalent to R vectors. Using `numpy` can be very efficient, as mentioned above, vectorization and sometimes even parallelization is already built in. the drawback of `numpy` is that it handles mainly numerical data. For the use of data frames as in R, there is another package available in Python, called `pandas`. A `pandas` data frame is the nearest equivalent to the R-native data frame, but some features, like automatically ignoring NaN-Values are not built-in. This is because, of course, Python did not start off as a language designed for handling large data sets like R.

Here we can conclude that both programming languages give us the commands, packages, scalability options and data structures to perform statistical modeling not only on a small scale but also on a large scale for possibly very large data sets.

Visualization Here I will not go into too much detail, as I have to check with the scope of this work. Anyways, there are several plotting environments available both for Python and for R. Starting with R, there is the native plotting (see Figure 2b), which seems banal at first glance but can be very powerful and offers all necessary plots to get a first glance at a dataset. In Python, this simple plotting framework is given by the `matplotlib` package, as can be seen in Figure 2a. It also offers all basic plots for data inspection. A framework for more advanced plots in Python is the `seaborn` package, that has a wider variety of plot and has a more scientific look to it. In particular it offers a simple method to plot the residuals to check for heteroscedasticity of a model.

Heading towards more sophisticated ways of visualization, I would consider

Figure 2: Comparing R and Python



(a) basic Python plot

(b) basic R plot

the `ggplot2` package in R as benchmark framework. `ggplot2` allows to make visualizations based on the Grammar of Graphics by Leland Wilkinson. The idea is to add visualization features to a basis plot. This package has an equivalent in Python called `plotnine`. The syntax of `plotnine` is almost the same as `ggplot2` in R, except for some minor differences. Unfortunately, not all features in R package have an equivalent in the python package. But for most visualizations, it still suffices. For further information on the `ggplot2` packages, please refer to online sources (i.e. https://ggplot2.tidyverse.org/).

To sum up the visualization part, we can state that both languages have a standard plotting library, where Python with `seaborn` might have a little more sophisticated package. But the intuitive handling of the standard plotting library in R together with the built-in plotting pane has a clear advantage over the external libraries in Python. Also scrolling up and down on previous visualizations becomes quite tedious over time in JuPyter Notebook. In addition, the R package `ggplot2` has more features and a better online documentation than the `plotnine` package in Python. Thus, R has a slight advantage over Python when it comes to plotting.

2.2 Hands-On

In this section I want to review on my experience trying to re-implement some parts of the lecture "Advanced Statistical Modeling"[2]. In particular I picked a range of research questions that appear often in statistical research and checked if there would be an easy way to implement the same models in Python. Please

find all the code in the appendix (pages 9-28).

Linear Regression One of the first methods coming to mind when hearing social science statistics is probably the normal linear regression model (ordinary least squares, OLS):

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n = \beta_0 + \sum_{i=1}^{n} \beta_i x_i \tag{1}$$

, where $\hat{y}$ is the dependent variable, $x \in X$ are the model's influence variables and β_i are the corresponding regression parameters. In R the implementation is as easy as can be:

```
lm1 = lm(dep_var ~ 1 + indep_var1 + indep_var2 + ...,
 data = data.frame)
summary(lm1)
```

For a quick analysis of the results, R offers the `summary()` function that shows key values like intercept, p-values and R^2 to the user.
For Python, a first research led to the models provided by the scientific package `scikit-learn` that has a function called linear regression. As can be seen in the appended Python Notebook number 1, this implementation comes from a machine-learning, numerical point of view. This becomes very clear when looking at the terminology, i.e. feature vectors, weighted and normalized input vectors. These methods are very useful in machine learning algorithms but are rarely applied to data sets in social sciences. In the implementation, each feature is a vector passed as a numpy array. So if your data are not numeric or saved in a pandas dataframe, they need to be extracted and converted to a numpy array. Basically this algorithm is only a wrapper function for the `scikit-learn` least-squares-optimization algorithm. It also doesn't work around missing values, such that they have to be excluded from the whole set. Also, the extraction of descriptives for the linear model is not given by a summary function but has to be computed from the result object. It is by far not as handy and user friendly as the R alternative.
Luckily, there is another package called `statsmodels` that looks like an imitation of the basic R methods. Still, NaN data must be excluded beforehand.

```
lm1 = smf.ols("dep_var ~ 1 + indep_var1 + indep_var2 + ...",
 data = pandas_df).fit()
lm1.summary()
```

Robust Linear Models and Quantile Linear Regression The computation robust linear models usually uses the Least-Abolute-Deviation (LAD) criterion as their optimization function, because it is less responsive towards outliers or whole groups of outliers. In R, this can be implemented via the

`quantreg` module and the function `rq(y∼x,tau=0.5,data=df)`. Here we can see that the robust linear regression is only a quantile regression at the 0.5-quantile.
In Python, one can again use the `statsmodels` package and there the function `rlm("dep_var ∼ indep_var",data = data).fit()`.
The quantile regression then needs to be done for not only the 50%-quantile but also for the other quantiles with respect to our research question. In R, this would happen as above: `rq(y∼x,tau=c(0.05,0.25,0.5,0.75,0.95),data=df)`, just note that the quantiles are now given as a vector. In Python, the quantile input parameter (`tau` in R) can only be one value and thus we need a simple iteration over tau. If tau is 0.5, this function yields the robust linear model from above

```
smf.quantreg('nm ~ wfl', data=miete_df).fit(q=tau)
```

Multi-Level Methods Multi-Level Methods are used if a researcher assumes a different distribution of correlations between groups in a data set. So you want to use a different regression for each group, but maybe still apply some general rules to the regression. You could i.e. either keep the intercept of the whole data set and vary only the slope for each group, or vice versa. In R, separate models would be applied via the package `lme4` and using the command

```
fit_separate <- lmList(math ~ homework|id_school, data = nels88)
```

In Python, this would again be done with the `statsmodels` package and the function:

```
lm_separate = smf.mixedlm("math ~ homework", data = nels_df, groups =
nels_df["id_school"], re_formula="~homework").fit()
```

The same can be done for keeping constant the intercept or the slope value for all groups, and can be seen in the JuPyter Notebook 4.

Generalized Linear Models(GLMs) For regressing on a dichotomous variable it is necessary to use GLMs, in particular the logistical regression to find correlations with other variables. Other important GLMs are the Probit Model and the models for Poisson and negative binomial distributions. All these models can be used in Python just as in R, so I will pick the Probit model as a representative here. In R GLMs are done via the built-in function `glm()`:

```
fit_probit <- glm(formula = any_contact~age+sex+eastwest,
family = binomial(link = "probit"), data = df_allbus)
```

In Python, again the package `statsmodels` is used:

```
probit_model = glm("any_contact ~ age + sex + eastwest",
data = allbus_df,
family = sm.families.Binomial(link=sm.families.links.probit)
).fit()
```

Summary Summing up, it becomes clear that Python comes from a machine-learning background rather than statistics. The use of `scikit-learn` for social scientific statistics is not recommended and to be fair, it's not what it is made for. This impression is amplified by the notion that the `statsmodels` package uses the base-R syntax for their methods, thus accepting the R-style for statistical modeling to be already ideal.

3 Conclusion

In this essay I compared Python with the current standard tool for statistical analysis, R. My findings are that, despite all technical similarities, R is the better option for statistics. There are a couple of reasons for that. Firstly, the visualizations in R using `ggplot2` offer a wider variety of visualizations and the equivalent `plotnine` package in Python is not as good yet. Also the built-in visualization pane is much more handy to use than having to scroll over the same visualizations over and over again in JuPyter Notebook. Secondly, the statistical modeling procedures are originally in R and the Python package `statsmethods` seems to emulate the basic R language style. Thus, even if you learn statistical modeling in Python, it would still be more intuitive to use the R environment, because it is basically the same syntax. Thirdly, the data structure of R with the built-in data frames is not as strict as in pandas dataframes and thus advantageous. Having to exclude NaN values before we can run basic statistic modeling in Python excludes a possibly large part of the data. It could be circumvented if you create a new pandas data frame containing only the data for the specific regression and then dropping the NaN values. But that only seems to add another possible cause of error. Last but not least, the intuitive execution of selected parts of the script in RStudio is superior to the possibilities given in JuPyter Notebook. Even though JuPyter works fine with the chunk execution policy,but sometimes you would just like to run a specific line of code.

This Resum is not to condemn Python's statistical packages or the language as a whole. If you are used to Python and you don't know R, I would even recommend to learn about `statsmethods` and `plotnine` first. You would still get the desired results and great visualizations. But if you don't know either language or both equally well, and your task is to run some statistical analysis, I would recommend learning or using R for the task.

References

[1] Grace Rogers Zachariah Hughes Elyse Myer-Tyson Ceyhun Ozgur, Taylor Colliau. Matlab vs. python vs. r. *Journal of Data Science*, pages 355–372, 2017.

[2] Stephan Poppe. *Advanced Statistical modeling-lecture*. University of Leipzig, 2019.

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here.This paper was not previously presented to another examination board and has not been published.

Tobias Giesemann

Appendix

...see below
Pages 10-13 Linear Models
Pages 14-18 Robust Models & Quantile Regression
Pages 19-24 Mixed Models
Pages 25-28 GLMs

Statistical modelling with Python

Simple linear Regression with Python-native SkiKit-Learn Package

Imports necessary for Regression models with SciKitLearn

In [2]:

```
# imports necessary for statistical modelling
# convention is to define imports at the top of scripts

import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

Using the ALLBUS data set for standard linear regressions

In [3]:

```
# import data set, set directory
home_dir = "/home/tobias_giesemann/Dropbox/Uni_Master/02SS19/01Advanced_Statistical_Modelling/Essay/"

# using pandas dataframe as a similar data structure to R dataframes
allbus_df = pd.read_csv(home_dir+"data/allbus_small.csv", index_col=0)

#show head
allbus_df.head()
```

Out[3]:

	income	sex	age	eduyears	eastwest	socialclass_self	leftright
1	1800.0	FRAU	47.0	13.0	NEUE BUNDESLAENDER	MITTELSCHICHT	2.0
2	2000.0	MANN	52.0	13.0	NEUE BUNDESLAENDER	MITTELSCHICHT	4.0
3	2500.0	MANN	61.0	9.0	ALTE BUNDESLAENDER	MITTELSCHICHT	8.0
4	860.0	FRAU	54.0	12.0	ALTE BUNDESLAENDER	NaN	3.0
5	NaN	MANN	71.0	NaN	ALTE BUNDESLAENDER	OBERSCHICHT	7.0

In [4]:

```
#description of numeric variables
allbus_df.describe()
```

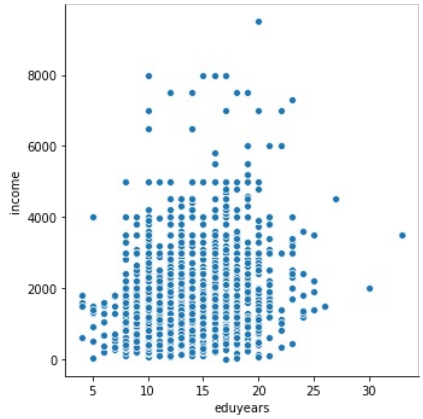
Out[4]:

	income	age	eduyears	leftright
count	2654.000000	3486.000000	3303.000000	3335.000000
mean	1609.666164	51.143144	12.632758	5.078561
std	1100.712568	17.567575	3.694805	1.700014
min	1.000000	18.000000	4.000000	1.000000
25%	850.000000	37.000000	10.000000	4.000000
50%	1400.000000	52.000000	12.000000	5.000000
75%	2000.000000	65.000000	15.000000	6.000000
max	9500.000000	97.000000	33.000000	10.000000

Plotting in Python with package "Seaborn"

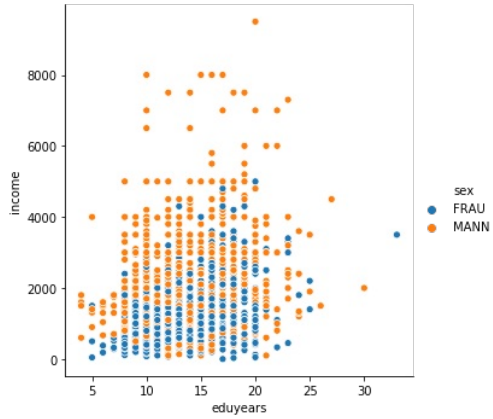
In [13]:

```
# basic plotting with seaborn plotting package
basic_plot = sns.relplot(x="eduyears",y="income", data=allbus_df)
```



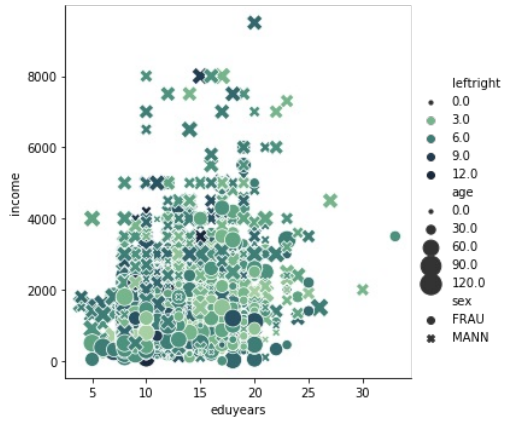
In [12]:

```
# little more advanced plotting
fair_plot = sns.relplot(x="eduyears",y="income", hue="sex", data=allbus_df)
```



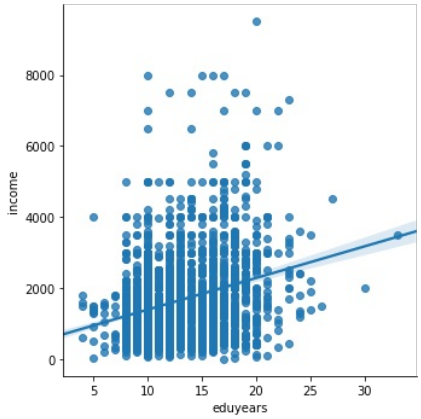
In [11]:

```
# too much for one plot, but great features
overly_advanced_plot = sns.relplot(x="eduyears",
            y="income",
            hue="leftright",
            style="sex",
            size="age",
            palette="ch:r=-.5,l=.75",
            sizes=(10,300),
            data=allbus_df)
overly_advanced_plot.savefig(home_dir+'figures/python/overly_advanced_plot.pdf')
```



In [10]:

```
# simple and fast plot for linear models -> very powerful
lm_plot = sns.lmplot('eduyears', 'income', data=allbus_df, fit_reg=True)
```



Linear Models with SciKit Learn

In [4]:

```
# reshape input variables
regression_df1 = allbus_df[['income','age']]
regression_df1.dropna(inplace=True)
X = np.array([regression_df1.age]).reshape(-1, 1)
y = np.array([regression_df1.income]).reshape(-1, 1)

# define model
regression_model = LinearRegression()
regression_model.fit(X, y)

y_predicted = regression_model.predict(X)

# model evaluation
mse = mean_squared_error(y, y_predicted)
r2 = r2_score(y, y_predicted)

# printing values
print('Estimate Std.:' ,regression_model.coef_)
print('Intercept:', regression_model.intercept_)
print('Mean squared error: ', mse)
print('R2 score: ', r2)
```

```
Estimate Std.: [[2.79715344]]
Intercept: [1465.57232066]
Mean squared error:  1208951.0226202777
R2 score:  0.002015611062190281
```

```
/home/tobias_giesemann/.local/lib/python3.6/site-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

Linear Regression with Package StatsModels

In [23]:

```
import statsmodels.api as sm

# import data set, set directory
home_dir = "/home/tobias_giesemann/Dropbox/Uni_Master/02SS19/01Advanced_Statistical_Modelling/Essay/"

# using pandas dataframe as a similar data structure to R dataframes
allbus_df = pd.read_csv(home_dir+"data/allbus_small.csv", index_col=0)

#show head
allbus_df.head()

# drop nan values
allbus_df. dropna(inplace = True)
```

Linear Regression

As we can see, this emulation of the R syntax is much easier to implement

In [25]:

```
import statsmodels.formula.api as smf

lm1 = smf.ols('income ~ 1+age+sex+eduyears+eastwest+leftright', data=allbus_df).fit()
print(lm1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 income   R-squared:                       0.218
Model:                            OLS   Adj. R-squared:                  0.216
Method:                 Least Squares   F-statistic:                     134.5
Date:                Thu, 25 Jul 2019   Prob (F-statistic):          4.95e-126
Time:                        16:40:57   Log-Likelihood:                -20109.
No. Observations:                2418   AIC:                         4.023e+04
Df Residuals:                    2412   BIC:                         4.026e+04
Df Model:                           5
Covariance Type:            nonrobust
=================================================================================================
                                    coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------------
Intercept                      -412.2895    128.587     -3.206      0.001    -664.442    -160.137
sex[T.MANN]                     647.4251     40.427     16.015      0.000     568.149     726.701
eastwest[T.NEUE BUNDESLAENDER] -371.3275     42.863     -8.663      0.000    -455.380    -287.275
age                               8.8806      1.188      7.477      0.000       6.552      11.210
eduyears                         99.1138      5.674     17.468      0.000      87.987     110.240
leftright                        27.9986     11.870      2.359      0.018       4.722      51.275
==============================================================================
Omnibus:                      730.914   Durbin-Watson:                   1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3151.537
Skew:                           1.402   Prob(JB):                         0.00
Kurtosis:                       7.839   Cond. No.                         356.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Residual Plot with Python

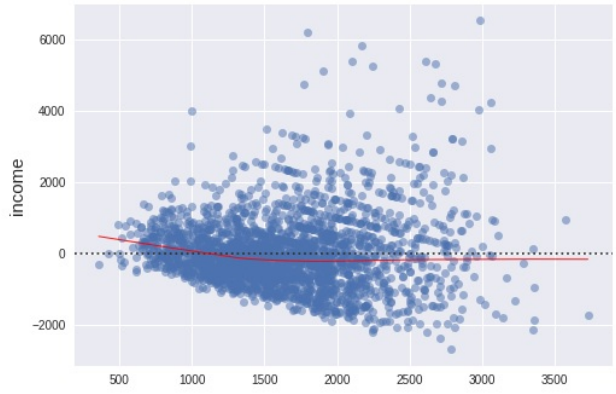
As this plot is often needed to check for homoscedascity, I will check if there is an easy implementation in python as well. As we will see, this is not the case, and we rather need to build our own plot.

In [41]:

```
# fitted values (need a constant term for intercept)
model_fitted_y = lm1.fittedvalues

# residual plot
sns.residplot(x=model_fitted_y,
              y="income",
              data=allbus_df,
              lowess = True,
              scatter_kws={'alpha':0.5},
              line_kws={"color": "red", "lw": 1, "alpha": 0.9})
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0e6a0f6438>
```



Here, we would rather have to check for heteroscedascity and maybe consider a quantile-regression model

Advanced linear Regression Models with Python

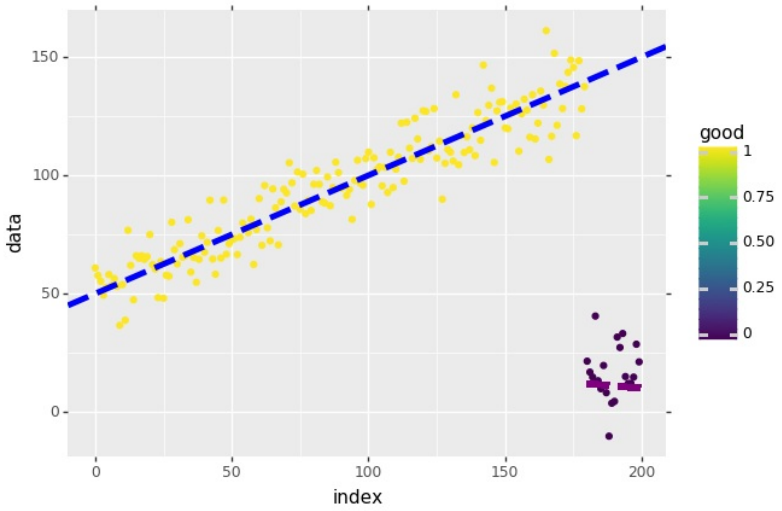
Robust linear Modelling

Here, we want to build on the example from session 6 to emulate the behaviour of R

In [59]:

```
import statsmodels.api as sm
import numpy as np
import pandas as pd
import random
import seaborn as sns
from matplotlib import pyplot as plt
from plotnine import *

random.seed(12354)

# make good data
good_data = np.array([50+0.5*x+ np.random.normal(0,10,1) for x in range(0,180)])

bad_data = np.array([20-0.1*x + np.random.normal(0,10,1) for x in range(0,20)])

data = pd.DataFrame(np.concatenate((good_data,bad_data)))
data = data.reset_index()
data.columns = ["index", "data"]
data.good = None
data.loc[data.index >=180, "good"] = 0
data.loc[data.index <180, "good"] = 1

#sns.lmplot("index", "data", data=data,fit_reg=True)

plot1 = (ggplot(data=data,
        mapping = aes(x="index", y = "data"))
 + geom_point(mapping=aes(color="good"))
 + geom_abline(intercept=50,
             slope=0.5,
               linetype = "dashed",
              color="blue",
              size=2)
 + geom_segment(x = 180,
                y = 20 -0.1*80,
                xend=200,
                yend=20-0.1*100,
                color="purple",
                 linetype="dashed",
                size=2)
)
plot1
```

```
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:54: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cbook.iterable(val) and not is_string(val):
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/scales/scale.py:93: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cbook.iterable(self.breaks) and cbook.iterable(self.labels):
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:553: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  return cbook.iterable(var) and not is_string(var)
```



Out[59]:

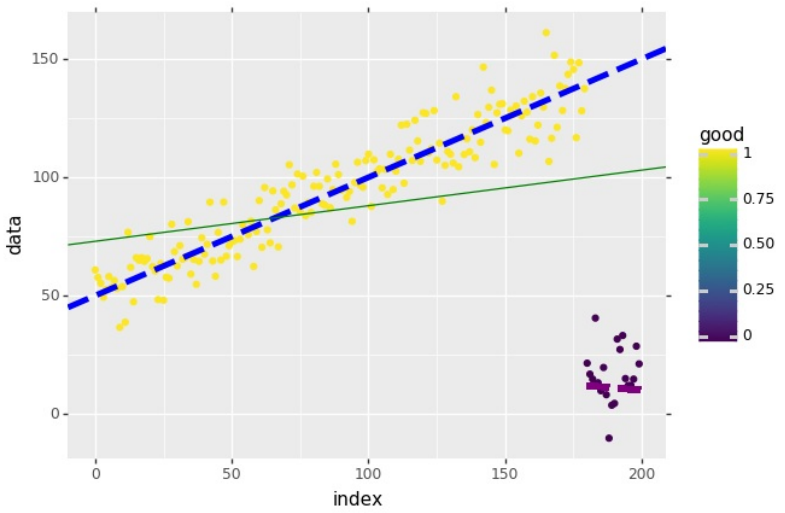
```
<ggplot: (8761243691065)>
```

In [60]:

```
import statsmodels.formula.api as smf

lm1 = smf.ols('data ~ 1+index', data=data).fit()
print(lm1.summary())

plot2 = (plot1
         + geom_abline(intercept = 72.955,
                       slope=0.1504,
                       color = "green")
        )
plot2
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   data   R-squared:                       0.056
Model:                            OLS   Adj. R-squared:                  0.052
Method:                 Least Squares   F-statistic:                     11.83
Date:                Thu, 25 Jul 2019   Prob (F-statistic):           0.000712
Time:                        18:59:53   Log-Likelihood:                -989.18
No. Observations:                 200   AIC:                             1982.
Df Residuals:                     198   BIC:                             1989.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     73.0660      4.817     15.167      0.000      63.566      82.566
index          0.1440      0.042      3.439      0.001       0.061       0.227
==============================================================================
Omnibus:                       45.576   Durbin-Watson:                   0.237
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               71.211
Skew:                          -1.262   Prob(JB):                     3.44e-16
Kurtosis:                       4.475   Cond. No.                         229.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:54: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cbook.iterable(val) and not is_string(val):
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/scales/scale.py:93: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cbook.iterable(self.breaks) and cbook.iterable(self.labels):
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:553: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  return cbook.iterable(var) and not is_string(var)
```



Out[60]:

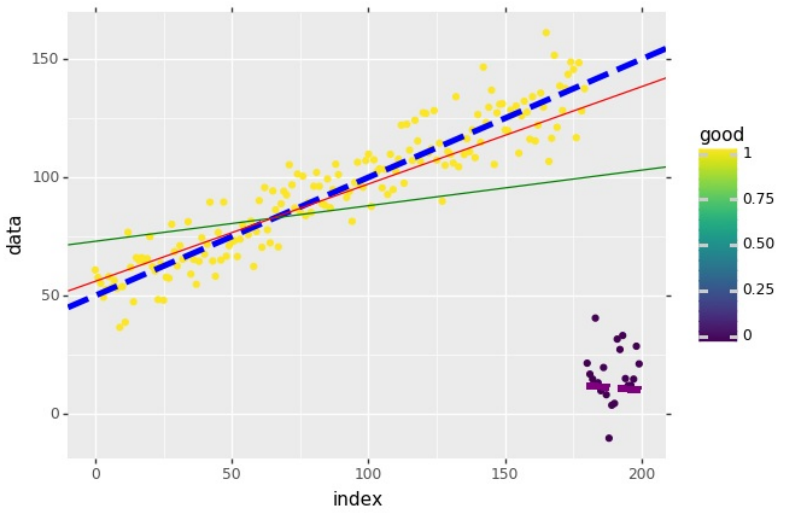
```
<ggplot: (-9223363275611110782)>
```

In [61]:

```
# import easy API
from statsmodels.formula.api import ols, rlm

# fit rlm model
rlm1 = rlm("data ~ index",data = data).fit()

# check summary
print(rlm1.summary())

# plot result
plot3 = (plot2
         + geom_abline(intercept = 56.0387,
                       slope=0.4115,
                       color = "red")
        )
plot3
```

```
                    Robust linear Model Regression Results
==============================================================================
Dep. Variable:                   data   No. Observations:                  200
Model:                            RLM   Df Residuals:                      198
Method:                          IRLS   Df Model:                            1
Norm:                          HuberT
Scale Est.:                       mad
Cov Type:                          H1
Date:                Thu, 25 Jul 2019
Time:                        18:59:57
No. Iterations:                    33
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     56.5439      1.856     30.468      0.000      52.907      60.181
index          0.4075      0.016     25.261      0.000       0.376       0.439
==============================================================================

If the model instance has been used for another fit with different fit
parameters, then the fit options might not be the correct ones anymore .
```

```
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:54: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cbook.iterable(val) and not is_string(val):
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/scales/scale.py:93: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cbook.iterable(self.breaks) and cbook.iterable(self.labels):
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:553: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  return cbook.iterable(var) and not is_string(var)
```



Out[61]:

```
<ggplot: (-9223363275611159443)>
```

In [3]:

```
import pandas as pd
from statsmodels.formula.api import ols, rlm
from statsmodels.regression.quantile_regression import QuantReg
import statsmodels.formula.api as smf
import seaborn as sns
from plotnine import *

# import data set, set directory
home_dir = "/home/tobias_giesemann/Dropbox/Uni_Master/02SS19/01Advanced_Statistical_Modelling/Essay/"

# using pandas dataframe as a similar data structure to R dataframes
miete_df = pd.read_csv(home_dir+"data/mietspiegel.csv", sep="\t")

miete_df = miete_df[miete_df.wfl <=200]

#show head
miete_df.head()
```

Out[3]:

	nm	nmqm	wfl	rooms	bj	bez	wohngut	wohnbest	ww0	zh0	badkach0	badextra	kueche
0	608.4	12.67	48	2	1957.5	Untergiesing	0	0	0	0	1	0	0
1	780.0	13.00	60	2	1983.0	Bogenhausen	1	0	0	0	1	0	1
2	822.6	7.48	110	5	1957.5	Obergiesing	0	0	0	1	1	1	0
3	500.0	8.62	58	2	1957.5	SchwanthalerhÃ¶he	0	0	0	0	1	0	1
4	595.0	8.50	70	3	1972.0	Aubing...	0	0	0	0	0	0	0

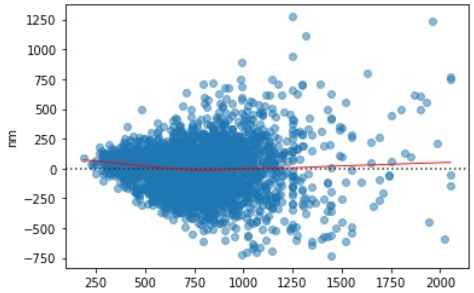
In [27]:

```
lm1 = ols("nm~wfl", data = miete_df).fit()
print(lm1.summary())

# let's look at the residual plot:
# fitted values (need a constant term for intercept)
model_fitted_y = lm1.fittedvalues

# residual plot
sns.residplot(x=model_fitted_y,
              y="nm",
              data=miete_df,
              lowess = True,
              scatter_kws={'alpha':0.5},
              line_kws={"color": "red", "lw": 1, "alpha": 0.9})
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     nm   R-squared:                       0.607
Model:                            OLS   Adj. R-squared:                  0.607
Method:                 Least Squares   F-statistic:                     4722.
Date:                Thu, 25 Jul 2019   Prob (F-statistic):               0.00
Time:                        19:30:37   Log-Likelihood:                -20598.
No. Observations:                3061   AIC:                          4.120e+04
Df Residuals:                    3059   BIC:                          4.121e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     37.0605     11.143      3.326      0.001      15.212      58.909
wfl           10.0804      0.147     68.719      0.000       9.793      10.368
==============================================================================
Omnibus:                      222.535   Durbin-Watson:                   1.728
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              853.362
Skew:                           0.270   Prob(JB):                    4.95e-186
Kurtosis:                       5.530   Cond. No.                         231.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa578989a20>
```



In [4]:

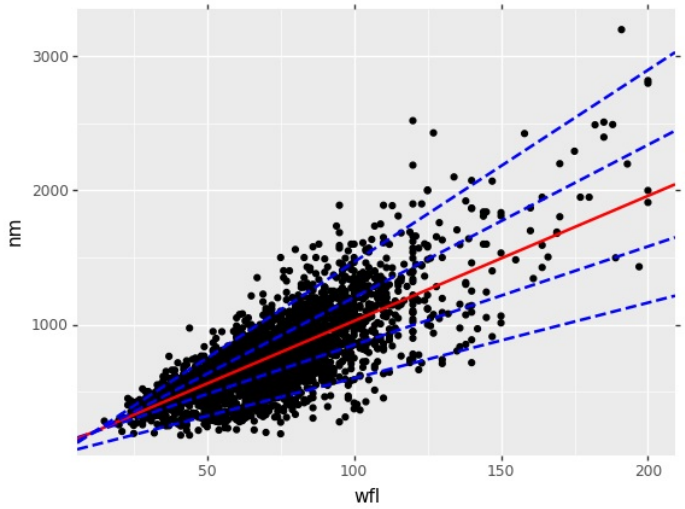
```
quantile_model = smf.quantreg('nm ~ wfl', data=miete_df)
for quantile in [0.05,0.25,0.5,0.75,0.95]:
    res = quantile_model.fit(q=[0.05,0.25,0.5,0.75,0.95])
    print(f"Results for {quantile}- quantile")
    print(res.summary())
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-04e0b61df238> in <module>
      1 quantile_model = smf.quantreg('nm ~ wfl', data=miete_df)
----> 2 res = quantile_model.fit(q=[0.05,0.25,0.5,0.75,0.95])(
      3 print(res.summary()))
      4
      5 quantile_model = smf.quantreg('nm ~ wfl', data=miete_df)

~/.local/lib/python3.6/site-packages/statsmodels/regression/quantile_regression.py in fit(self, q, vcov, kernel, bandwidth,
max_iter, p_tol, **kwargs)
    118         '''
    119
--> 120         if q < 0 or q > 1:
    121             raise Exception('p must be between 0 and 1')
    122

TypeError: '<' not supported between instances of 'list' and 'int'
```

In [31]:

```
plot1 = (ggplot(data=miete_df,
        mapping = aes(x="wfl", y = "nm"))
 + geom_point(mapping=aes())
 + geom_abline(intercept=36.6788,
             slope=5.6303,
               linetype = "dashed",
              color="blue",
              size=1)
         + geom_abline(intercept=114.0541 ,
             slope=7.3425 ,
               linetype = "dashed",
              color="blue",
              size=1)
         + geom_abline(intercept=93.3333 ,
             slope=9.3333 ,
              color="red",
              size=1)
         + geom_abline(intercept=63.8298 ,
             slope=11.3830  ,
               linetype = "dashed",
              color="blue",
              size=1)
         + geom_abline(intercept=35.0000 ,
             slope=14.3182,
               linetype = "dashed",
              color="blue",
              size=1)
)
plot1
```

```
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:54: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cbook.iterable(val) and not is_string(val):
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/scales/scale.py:93: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cbook.iterable(self.breaks) and cbook.iterable(self.labels):
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:553: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  return cbook.iterable(var) and not is_string(var)
```



Out[31]:

```
<ggplot: (-9223363265064663566)>
```

In [53]:

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
import pandas as pd
from plotnine import *
import numpy as np

# import data set, set directory
home_dir = "/home/tobias_giesemann/Dropbox/Uni_Master/02SS19/01Advanced_Statistical_Modelling/Essay/"

# using pandas dataframe as a similar data structure to R dataframes
nels_df = pd.read_csv(home_dir+"data/nels88.csv", sep=",")

#show head
nels_df.head()
```

Out[53]:

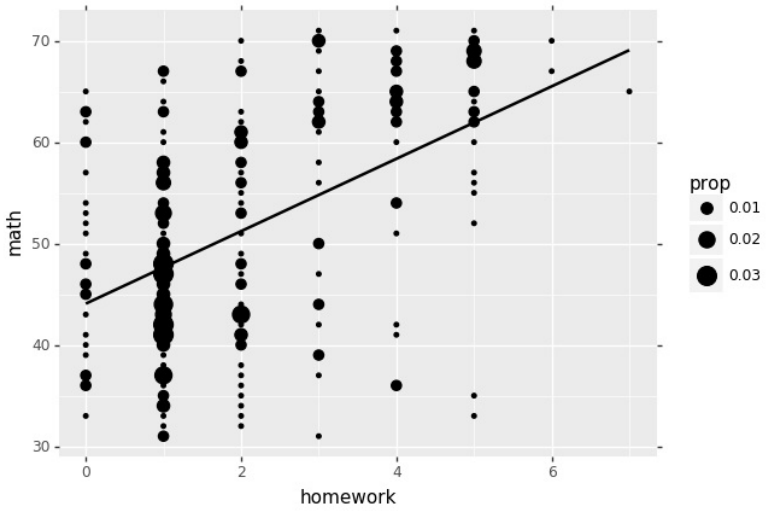
	id_region	id_school	public	ratio	percmin	cstr	scsize	urban	id_student	math	homework	white	sex	ses	parented
0	2	1	public	19	0	2	3	2	3	48	1	white	female	-0.13	2
1	2	1	public	19	0	2	3	2	8	48	0	white	male	-0.39	2
2	2	1	public	19	0	2	3	2	13	53	0	white	male	-0.80	2
3	2	1	public	19	0	2	3	2	17	42	1	white	male	-0.72	2
4	2	1	public	19	0	2	3	2	27	43	2	white	female	-0.74	2

In [47]:

```
# start with linear model for mean estimation

lm_const = smf.ols("math~1", data=nels_df).fit()
# print(lm_const.resid)



# check for homework-effort
lm_homework = smf.ols("math ~ homework", data = nels_df).fit()
print(lm_homework.summary())
lm_homework_intercept = 44.0739
(ggplot(data = nels_df,
       mapping = aes(x = "homework",
                     y = "math"
                     )
       ) +
  geom_count() +
  geom_smooth(method = "lm",
              se = False
              ))
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   math   R-squared:                       0.247
Model:                            OLS   Adj. R-squared:                  0.244
Method:                 Least Squares   F-statistic:                     84.64
Date:                Thu, 25 Jul 2019   Prob (F-statistic):           1.25e-17
Time:                        20:45:49   Log-Likelihood:                -958.18
No. Observations:                 260   AIC:                             1920.
Df Residuals:                     258   BIC:                             1927.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     44.0739      0.989     44.580      0.000      42.127      46.021
homework       3.5719      0.388      9.200      0.000       2.807       4.336
==============================================================================
Omnibus:                        2.792   Durbin-Watson:                   1.502
Prob(Omnibus):                  0.248   Jarque-Bera (JB):                2.728
Skew:                          -0.198   Prob(JB):                        0.256
Kurtosis:                       2.691   Cond. No.                         4.62
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/scales/scale.py:93: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cbook.iterable(self.breaks) and cbook.iterable(self.labels):
/home/tobias_giesemann/.local/lib/python3.6/site-packages/numpy/core/fromnumeric.py:2389: FutureWarning: Method .ptp is depr
ecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:553: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  return cbook.iterable(var) and not is_string(var)
```



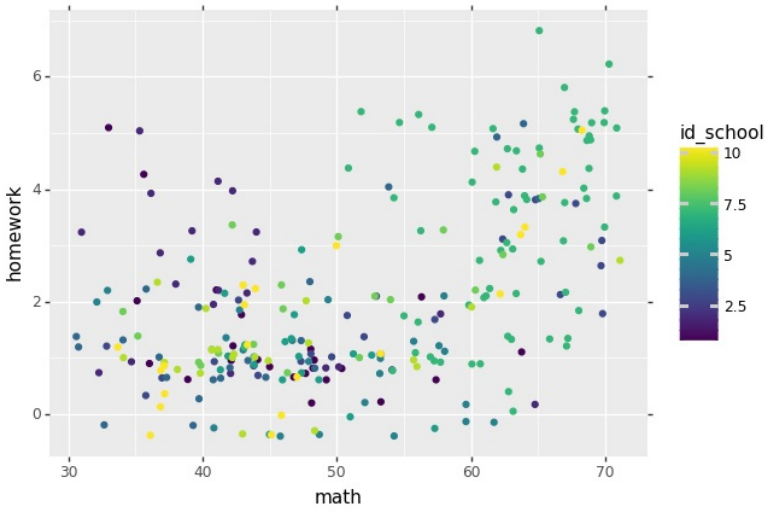
Out[47]:

```
<ggplot: (8731428660018)>
```

In [27]:

```
# This one doesn't work unfortunately
(ggplot(data = nels_df,
       mapping = aes(x = "math",
                     y = "homework", #should be the residuals...
                     color = "id_school"
                     )
       ) +
  geom_jitter( ))
```

```
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/scales/scale.py:93: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cbook.iterable(self.breaks) and cbook.iterable(self.labels):
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:553: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  return cbook.iterable(var) and not is_string(var)
```



Out[27]:

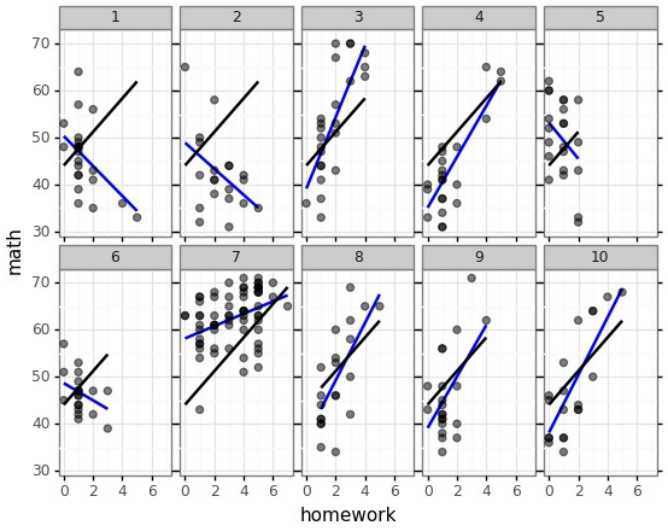
```
<ggplot: (-9223363305407798128)>
```

Separate Regression

In [59]:

```
# check for homework-effort

lm_separate = smf.mixedlm("math ~ homework", data = nels_df, groups = nels_df["id_school"], re_formula="~homework").fit()
(ggplot(data = nels_df,
       mapping = aes(x = "homework",
                     y = "math",
                     group = "id_school"
                    )
       ) +
  geom_line(mapping=aes(y=lm_separate.fittedvalues
                        ),
            color="blue",
            size=1
            )+
  geom_line(mapping=aes(y=lm_homework.fittedvalues
                        ),
            color="black",
            size=1
            )+
  geom_point(alpha=0.5,size=2) +
  facet_wrap("~id_school",nrow=2)+
  theme_bw())
```

```
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/scales/scale.py:93: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cbook.iterable(self.breaks) and cbook.iterable(self.labels):
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:553: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  return cbook.iterable(var) and not is_string(var)
```



Out[59]:

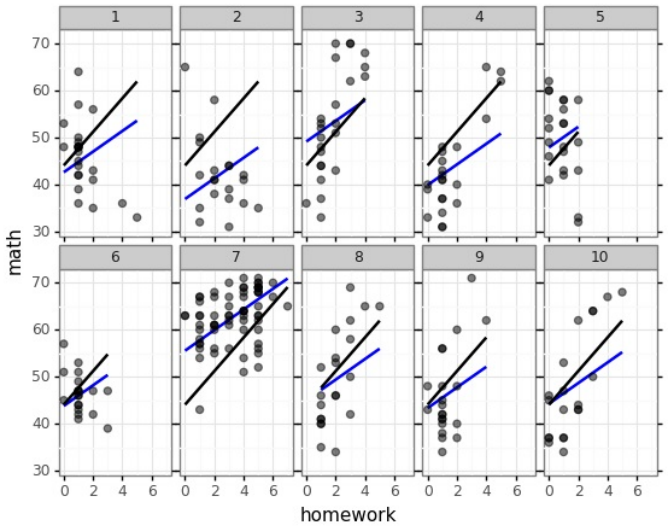
```
<ggplot: (8731433745819)>
```

Fixed Slope, random intercept

In [51]:

```
# check for homework-effort
# fixed slope and random intercept
lm_const_slope = smf.mixedlm("math ~ 0 +homework", data = nels_df, groups = nels_df["id_school"]).fit()
(ggplot(data = nels_df,
       mapping = aes(x = "homework",
                     y = "math",
                     group = "id_school"
                    )
       ) +
  geom_line(mapping=aes(y=lm_const_slope.fittedvalues
                        ),
            color="blue",
            size=1
            )+
  geom_line(mapping=aes(y=lm_homework.fittedvalues
                        ),
            color="black",
            size=1
            )+
  geom_point(alpha=0.5,size=2) +
  facet_wrap("~id_school",nrow=2)+
  theme_bw())
```

```
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/scales/scale.py:93: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cbook.iterable(self.breaks) and cbook.iterable(self.labels):
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:553: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  return cbook.iterable(var) and not is_string(var)
```



Out[51]:

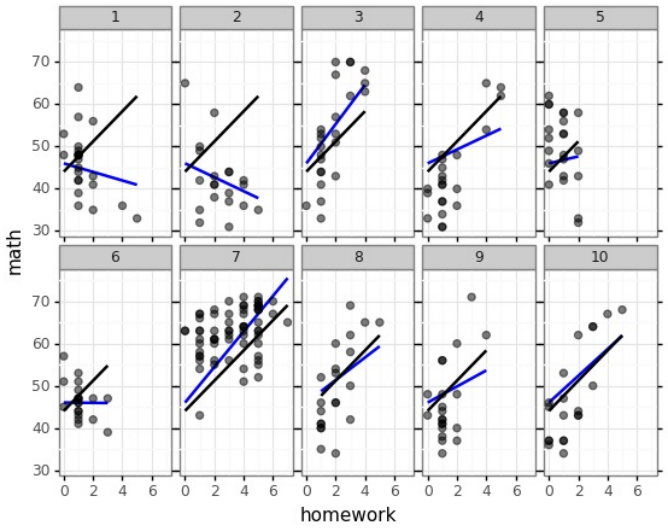
```
<ggplot: (8731433729505)>
```

Random Slope, Fixed Intercept

In [60]:

```
# check for homework-effort

lm_fix_intercept = smf.mixedlm("math ~ homework", data = nels_df, groups = nels_df["id_school"], re_formula="~0+homework").fit()
(ggplot(data = nels_df,
       mapping = aes(x = "homework",
                     y = "math",
                     group = "id_school"
                    )
       ) +
  geom_line(mapping=aes(y=lm_fix_intercept.fittedvalues
                        ),
            color="blue",
            size=1
            )+
  geom_line(mapping=aes(y=lm_homework.fittedvalues
                        ),
            color="black",
            size=1
            )+
  geom_point(alpha=0.5,size=2) +
  facet_wrap("~id_school",nrow=2)+
  theme_bw())
```

```
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/scales/scale.py:93: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cbook.iterable(self.breaks) and cbook.iterable(self.labels):
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:553: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  return cbook.iterable(var) and not is_string(var)
```



Out[60]:

```
<ggplot: (8731433655395)>
```

Generalized Linear Models

Logit Models

In [61]:

```
import pandas as pd
import seaborn as sns
from statsmodels.formula.api import logit, glm
import statsmodels.api as sm

from plotnine import *

# import data set, set directory
home_dir = "/home/tobias_giesemann/Dropbox/Uni_Master/02SS19/01Advanced_Statistical_Modelling/Essay/"

# using pandas dataframe as a similar data structure to R dataframes
allbus_df = pd.read_csv(home_dir+"data/allbus_full.csv")
print("full data set: ", allbus_df.shape)
allbus_df = allbus_df[["eastwest","sex","age","lt15"]]
allbus_df.contacts = allbus_df.lt15
allbus_df["any_contact"] = 0
allbus_df.loc[allbus_df.contacts >0, "any_contact"] = 1

print(allbus_df.contacts.unique())
print(allbus_df.any_contact.unique())


#show head
allbus_df.head()
```

```
full data set:  (3490, 793)
[nan  1.  2.  4.  3.  6.  5.  9.  8. 12.  7. 18. 11. 10.]
[0 1]
```

```
/home/tobias_giesemann/.local/lib/python3.6/site-packages/ipykernel_launcher.py:16: UserWarning: Pandas doesn't allow column
s to be created via a new attribute name - see https://pandas.pydata.org/pandas-docs/stable/indexing.html#attribute-access
  app.launch_new_instance()
```

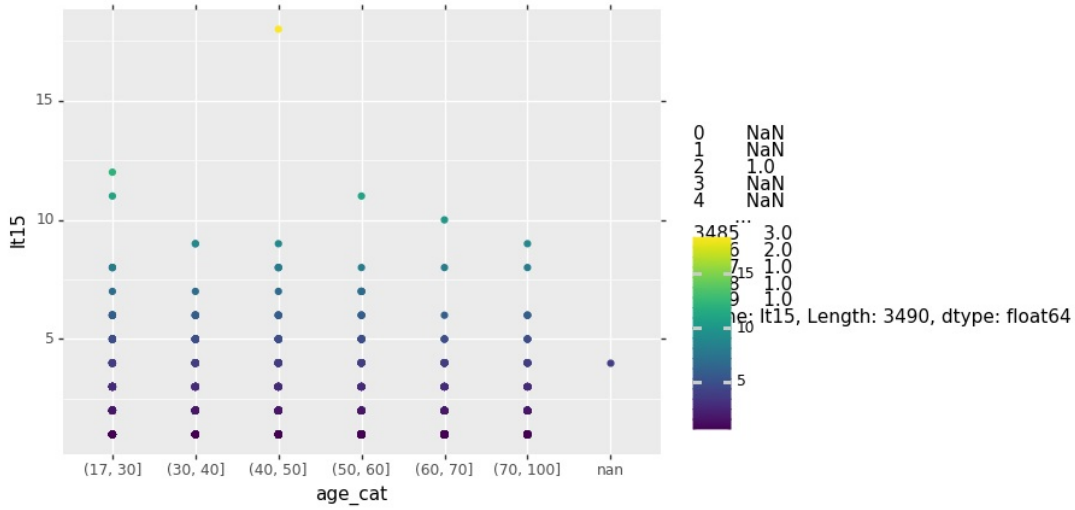
Out[61]:

	eastwest	sex	age	lt15	any_contact
0	NEUE BUNDESLAENDER	FRAU	47.0	NaN	0
1	NEUE BUNDESLAENDER	MANN	52.0	NaN	0
2	ALTE BUNDESLAENDER	MANN	61.0	1.0	1
3	ALTE BUNDESLAENDER	FRAU	54.0	NaN	0
4	ALTE BUNDESLAENDER	MANN	71.0	NaN	0

In [58]:

```
allbus_df["age_cat"] = pd.cut(allbus_df.age, [17,30,40,50,60,70,100])

(ggplot(data = allbus_df,
       mapping = aes(x = "age_cat",
                     y = "lt15"))
 +geom_point(mapping=aes(color=allbus_df["lt15"]))
)
```

```
/home/tobias_giesemann/.local/lib/python3.6/site-packages/mizani/bounds.py:352: RuntimeWarning: invalid value encountered in
less
  outside = (x < range[0]) | (x > range[1])
/home/tobias_giesemann/.local/lib/python3.6/site-packages/mizani/bounds.py:352: RuntimeWarning: invalid value encountered in
greater
  outside = (x < range[0]) | (x > range[1])
/home/tobias_giesemann/.local/lib/python3.6/site-packages/matplotlib/colors.py:527: RuntimeWarning: invalid value encountere
d in less
  xa[xa < 0] = -1
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/layer.py:449: UserWarning: geom_point : Removed 1756 rows
containing missing values.
  self.data = self.geom.handle_na(self.data)
/home/tobias_giesemann/.local/lib/python3.6/site-packages/plotnine/utils.py:553: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  return cbook.iterable(var) and not is_string(var)
```



Out[58]:

```
<ggplot: (8771815829859)>
```

Probit Modell

In [79]:

```
# Probit Modell als GLM


probit_model = glm("any_contact ~ age + sex + eastwest", data = allbus_df, family = sm.families.Binomial(link=sm.families.links.probit)).
fit()
print(probit_model.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:            any_contact   No. Observations:                 3486
Model:                            GLM   Df Residuals:                     3482
Model Family:                Binomial   Df Model:                            3
Link Function:                 probit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -2288.8
Date:                Thu, 25 Jul 2019   Deviance:                       4577.6
Time:                        22:18:47   Pearson chi2:                 3.49e+03
No. Iterations:                     5
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                        0.7664      0.071     10.745      0.000       0.627       0.906
sex[T.MANN]                     -0.0074      0.043     -0.170      0.865      -0.092       0.078
eastwest[T.NEUE BUNDESLAENDER]  -0.5642      0.046    -12.145      0.000      -0.655      -0.473
age                             -0.0114      0.001     -9.150      0.000      -0.014      -0.009
================================================================================================

/home/tobias_giesemann/.local/lib/python3.6/site-packages/ipykernel_launcher.py:4: DeprecationWarning: Calling Family(..) wi
th a link class as argument is deprecated.
Use an instance of a link class instead.
  after removing the cwd from sys.path.
```

Logit Modell

In [88]:

```
#selbiges Modell wie oben
logit_model = glm("any_contact ~ age + sex + eastwest", data = allbus_df,
                  family = sm.families.Binomial(link=sm.families.links.logit # or None, da logit default für Binomialfamilie
                                                )).fit()
print(logit_model.summary())

print("Maximal beobachtbare Effekte der Einflussgrößen: \n",0.25*logit_model.params)
print("Multiplikative Effektinterpretation: \n", (np.exp(logit_model.params)-1)*100)

# Margins konnten leider nicht herausgearbeitet werden, da ist die Dokumentation unvollständig,
# allerdings funktioniert es für das logit modell über einen Umweg:

logit_model2 = logit("any_contact ~ age + sex + eastwest", allbus_df).fit()
print(logit_model2.summary())

margins = logit_model2.get_margeff().summary_frame()
print(margins)
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:            any_contact   No. Observations:                 3486
Model:                            GLM   Df Residuals:                     3482
Model Family:                Binomial   Df Model:                            3
Link Function:                  logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -2288.7
Date:                Thu, 25 Jul 2019   Deviance:                       4577.5
Time:                        22:33:19   Pearson chi2:                 3.49e+03
No. Iterations:                     4
Covariance Type:            nonrobust
===================================================================================================
                                      coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------------------
Intercept                           1.2419      0.117     10.651      0.000       1.013       1.470
sex[T.MANN]                        -0.0115      0.070     -0.163      0.871      -0.149       0.126
eastwest[T.NEUE BUNDESLAENDER]     -0.9089      0.076    -12.021      0.000      -1.057      -0.761
age                                -0.0186      0.002     -9.120      0.000      -0.023      -0.015
===================================================================================================
Maximal beobachtbare Effekte der Einflussgrößen:
 Intercept                          0.310475
sex[T.MANN]                        -0.002864
eastwest[T.NEUE BUNDESLAENDER]     -0.227232
age                                -0.004643
dtype: float64
Multiplikative Effektinterpretation:
 Intercept                        246.218916
sex[T.MANN]                       -1.139177
eastwest[T.NEUE BUNDESLAENDER]   -59.704452
age                               -1.840169
dtype: float64
Optimization terminated successfully.
         Current function value: 0.656554
         Iterations 5
                          Logit Regression Results
==============================================================================
Dep. Variable:            any_contact   No. Observations:                 3486
Model:                          Logit   Df Residuals:                     3482
Method:                          MLE   Df Model:                            3
Date:                Thu, 25 Jul 2019   Pseudo R-squ.:                 0.05277
Time:                        22:33:20   Log-Likelihood:                -2288.7
converged:                       True   LL-Null:                       -2416.3
Covariance Type:            nonrobust   LLR p-value:                 5.390e-55
===================================================================================================
                                      coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------------------
Intercept                           1.2419      0.117     10.651      0.000       1.013       1.470
sex[T.MANN]                        -0.0115      0.070     -0.163      0.871      -0.149       0.126
eastwest[T.NEUE BUNDESLAENDER]     -0.9089      0.076    -12.021      0.000      -1.057      -0.761
age                                -0.0186      0.002     -9.120      0.000      -0.023      -0.015
===================================================================================================
                                    dy/dx  Std. Err.          z      Pr(>|z|)  \
sex[T.MANN]                    -0.002660   0.016330  -0.162902  8.705953e-01
eastwest[T.NEUE BUNDESLAENDER] -0.211037   0.016105 -13.104195  3.115508e-39
age                            -0.004312   0.000451  -9.559622  1.181875e-21

                                Conf. Int. Low  Cont. Int. Hi.
sex[T.MANN]                          -0.034666        0.029345
eastwest[T.NEUE BUNDESLAENDER]       -0.242602       -0.179473
age                                  -0.005196       -0.003428

/home/tobias_giesemann/.local/lib/python3.6/site-packages/ipykernel_launcher.py:1: DeprecationWarning: Calling Family(..) wi
th a link class as argument is deprecated.
Use an instance of a link class instead.
  """Entry point for launching an IPython kernel.
```

Poisson -Regression

In [99]:

```
#selbiges Modell wie oben
logit_model = glm("lt15 ~ age + sex + eastwest", data = allbus_df,
                  family = sm.families.Poisson()).fit()
print(logit_model.summary())

print("Multiplikative Effektinterpretation: \n", (np.exp(logit_model.params)-1)*100)
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                   lt15   No. Observations:                 1733
Model:                            GLM   Df Residuals:                     1729
Model Family:                 Poisson   Df Model:                            3
Link Function:                    log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -2858.5
Date:                Thu, 25 Jul 2019   Deviance:                       1447.2
Time:                        23:28:51   Pearson chi2:                 1.85e+03
No. Iterations:                     4
Covariance Type:            nonrobust
=================================================================================================
                                    coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------------------
Intercept                         0.9280      0.052     17.816      0.000       0.826       1.030
sex[T.MANN]                       0.0219      0.034      0.650      0.516      -0.044       0.088
eastwest[T.NEUE BUNDESLAENDER]   -0.1311      0.042     -3.150      0.002      -0.213      -0.050
age                              -0.0043      0.001     -4.381      0.000      -0.006      -0.002
=================================================================================================
Multiplikative Effektinterpretation:
 Intercept                        152.942160
sex[T.MANN]                        2.218618
eastwest[T.NEUE BUNDESLAENDER]   -12.290349
age                               -0.428598
dtype: float64
```

Hier funktioniert der Trick nicht, ein einfacheres Modell zur Margin Schätzung zu verwenden. Die müsste man dann wohl zu Fuß berechnen.

Negativ-Binomial- Regression

In [101]:

```
#selbiges Modell wie oben
logit_model = glm("lt15 ~ age + sex + eastwest", data = allbus_df,
                  family = sm.families.NegativeBinomial()).fit()
print(logit_model.summary())

print("Multiplikative Effektinterpretation: \n", (np.exp(logit_model.params)-1)*100)
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                   lt15   No. Observations:                 1733
Model:                            GLM   Df Residuals:                     1729
Model Family:        NegativeBinomial   Df Model:                            3
Link Function:                    log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -3323.1
Date:                Thu, 25 Jul 2019   Deviance:                       438.23
Time:                        23:29:41   Pearson chi2:                     610.
No. Iterations:                     5
Covariance Type:            nonrobust
=================================================================================================
                                    coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------------------
Intercept                         0.9326      0.092     10.179      0.000       0.753       1.112
sex[T.MANN]                       0.0227      0.059      0.386      0.699      -0.093       0.138
eastwest[T.NEUE BUNDESLAENDER]   -0.1338      0.071     -1.892      0.059      -0.272       0.005
age                              -0.0044      0.002     -2.584      0.010      -0.008      -0.001
=================================================================================================
Multiplikative Effektinterpretation:
 Intercept                        154.116026
sex[T.MANN]                        2.297112
eastwest[T.NEUE BUNDESLAENDER]   -12.522152
age                               -0.437916
dtype: float64
```