Python II Project

Torben Hinrichs – 12050173

Introduction

The goal of this analysis was to find a relationship between the job description of various jobs and the salary for that position. The data came from Google Jobs, using SerpAPI. Then the data gets cleaned and the salary extracted. Afterwards the words are tokenized using TFIDF and then tested on different models. The results were 'ok', which ist probably mostly due to the limited data and therefore the high discrepancy between the number of observations and the features.

Get Data

First the data needs to be collected. I used SerpAPI to collect the data directly from Google Jobs. Although the API worked very well, it was limited to a maximum of 100 requests per month. Given that the project expanded over a timeframe of up to 2 months, I was able to get a total of 562 observations (10 results per request, but then my code runs up to 100 per keyword, even if there are only 50 jobs). When improving this project in the future, I need to add data consistently to get the results more robust.

I concentrated on the Data Analysis Job market. This does not have any particular reason and the whole Analysis should work the same for any other field of jobs. To get a variety of jobs, and also because Google only delivers ~100 job postings when searching, I used the Api to get data for job listings with the keywords: 'Data Analyst', 'Data Scientist', 'Data Engineer', 'Data Consultant', 'Business Analyst', 'Business Intelligence Analyst', 'Machine Learning Engineer', 'Data Visualization Specialist' and 'AI Analyst'. As a location I chose Vienna, Austria. To get more data that might be expanded to other cities as well. But my request limit was reached anyway. All in all, I got 562 Observations. The data got stored in a csv file per keyword to load them later on. As the job descriptions were in German and English, my analysis also uses both languages.

Data preparation

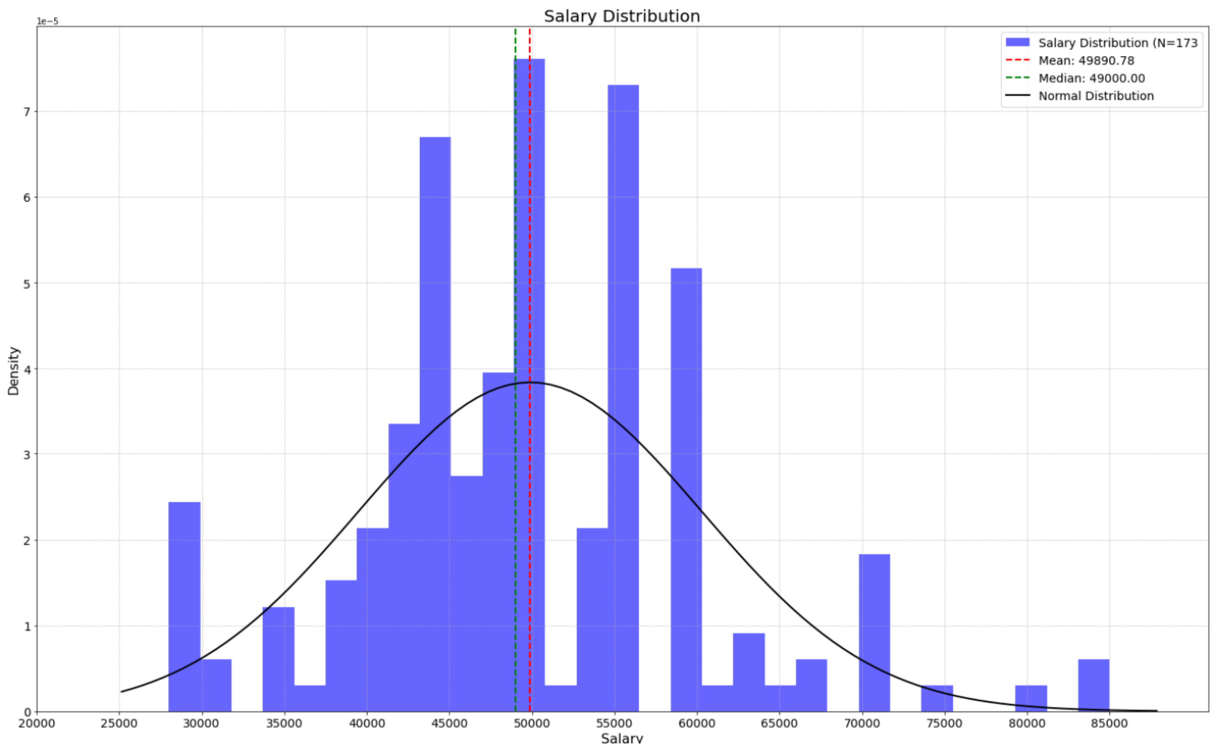
Clean Salary

First, I needed to extract the salary out of the job descriptions. I used regex to extract the salary using keywords like 'Gehalt', 'Salary', 'Income', 'Pay'. While that mostly worked fine, I did not get a 100% accuracy. But any other model I tried, using ML/AI tools which should extract the salary worked way worse, so I stuck to regex.

After extracting the salary, I cleaned it. Got rid of any 'EUR' and also deleted salaries with less than 3 digits, as well as 7 or more digits. Sometimes there were just 1s as a Salary which does not make sense, and any number with more than 6 digits is also highly unlikely and is probably a phone number or something similar.

Then I dropped any obseration, that did not have any salary or did not have 'full-time' in their keywords, as those were also part-time jobs which would skew the results. Also any Salary that way only 4 digits long got multiplied with 14, as it was most likely the monthly salary and therefore has to get transformed to yearly salary. I also deleted Outliers with a Z_score of < 3.

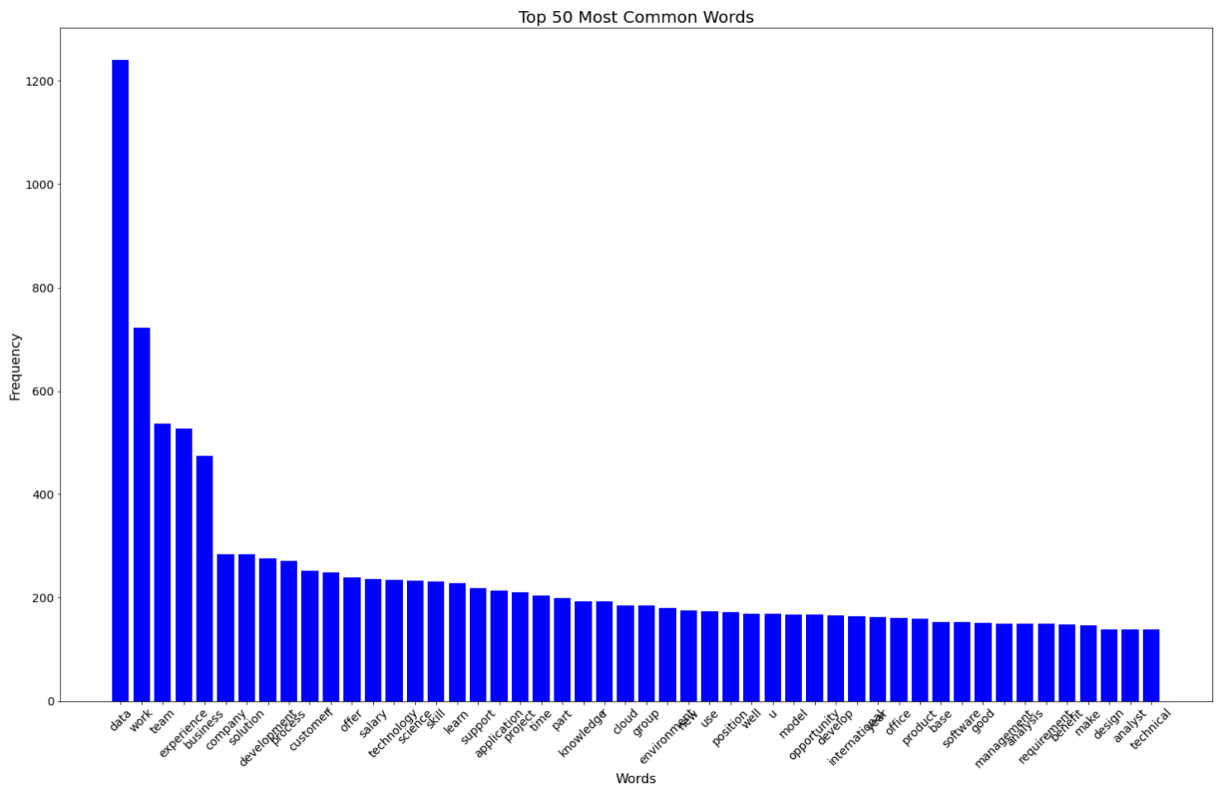
After all cleaning, I was left with the following salary distribution:

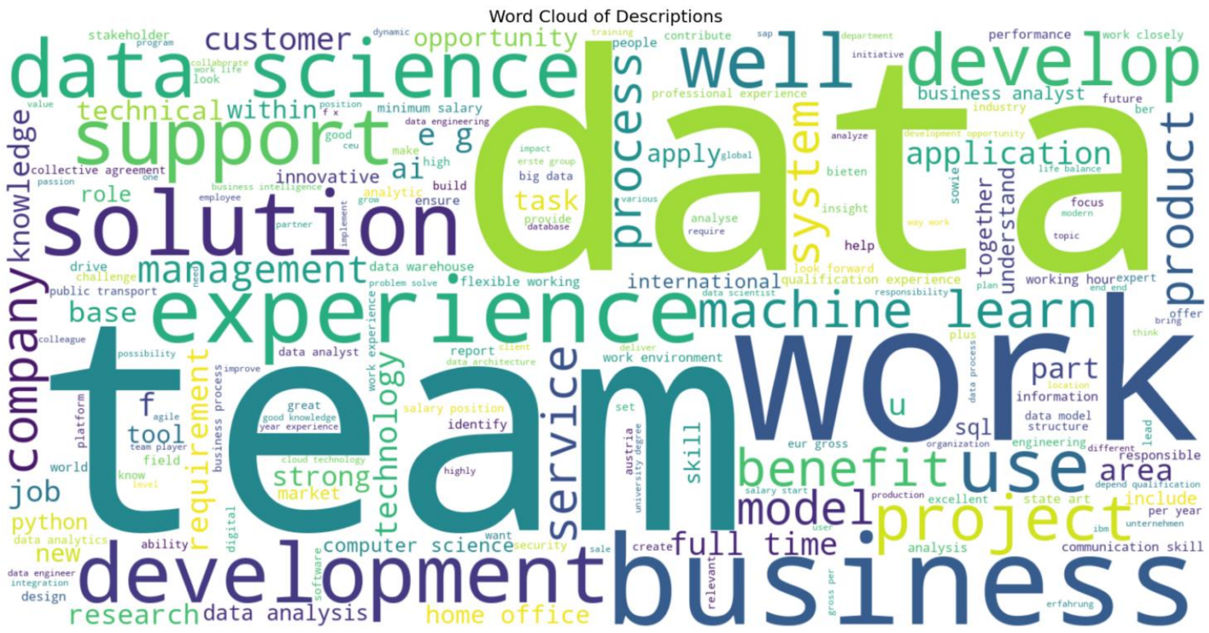


Clean Descriptions

For the descriptions, first I transformed every word to lowercase. I removed all \r, \n, \u200b, and afterwards any symbol that was not a letter. Lastly I reduced any more than 1 whitespace to 1 whitespace. For the stopwords, I needed to combine German and English stopwords, as the data is in both languages. So I removed any German and English stopword. Afterwards I use the WordNetLemmatizer to lemmatize the words to their roots.

After all the description transformations I plotted the most common words in a bar chart, and also in a word cloud because it looks pretty:





Vectorizing, Train-Test-Splitting and K-Fold CV

I used the TFIDF vectorizer to transform the words into more and less important words. Afterwards I set X as the vectorized description words, and y as the salary. I split X and y into training and testing data, to train and test the models better. I used 0.2 as the percentage as that seems to be best practice. I also prepared the K-Fold Cross Validation with 5 splits to use later on in the models. I used the Cross Validation to make the results more robust.

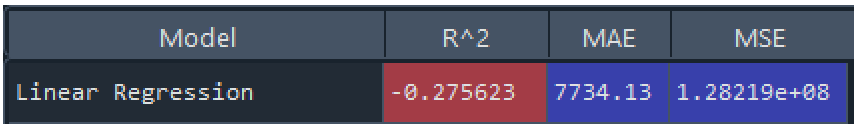
I also tried SelectKBest to reduce the dimensionality, but that did not really work out, as it gave weird results.

Regressions

Linear Regression on all data

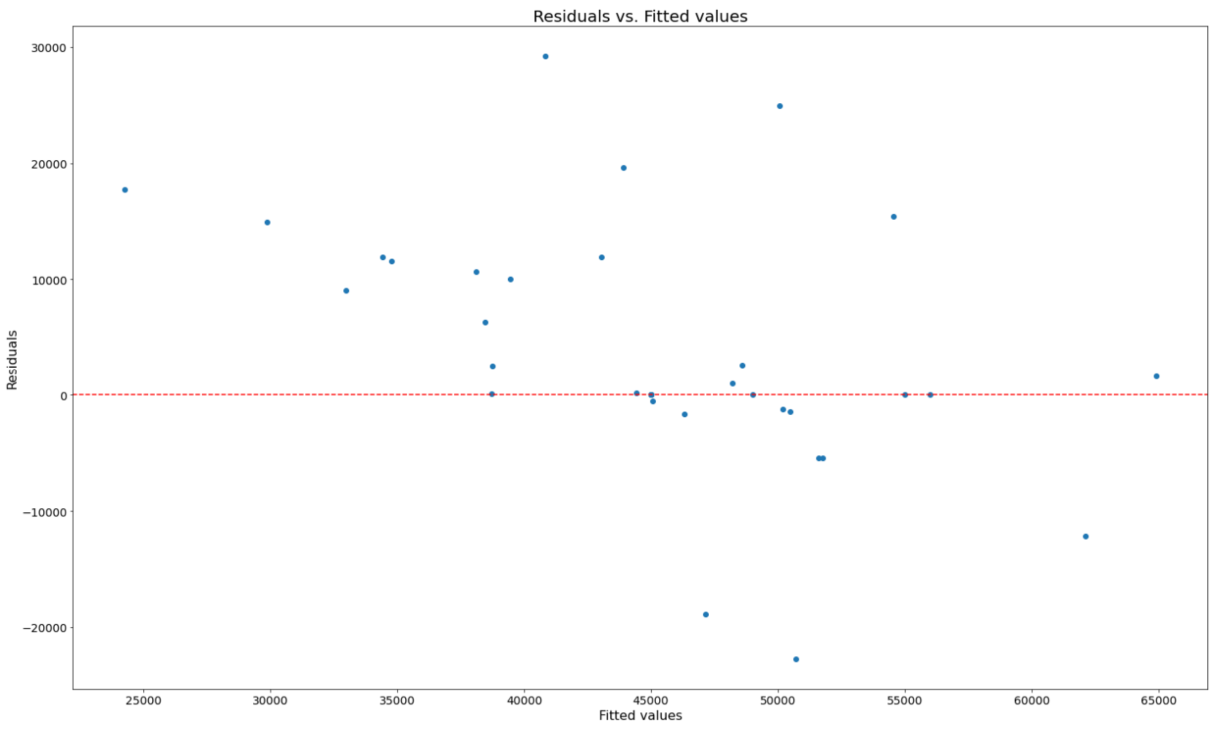
First, I simply fit a linear regression to check if that might already be a good model. I used the RandomizedSearchCV to find the best hyperparameter for every model. That was already slow enough for some models, so GridSearch might perform better, but not on my laptop.

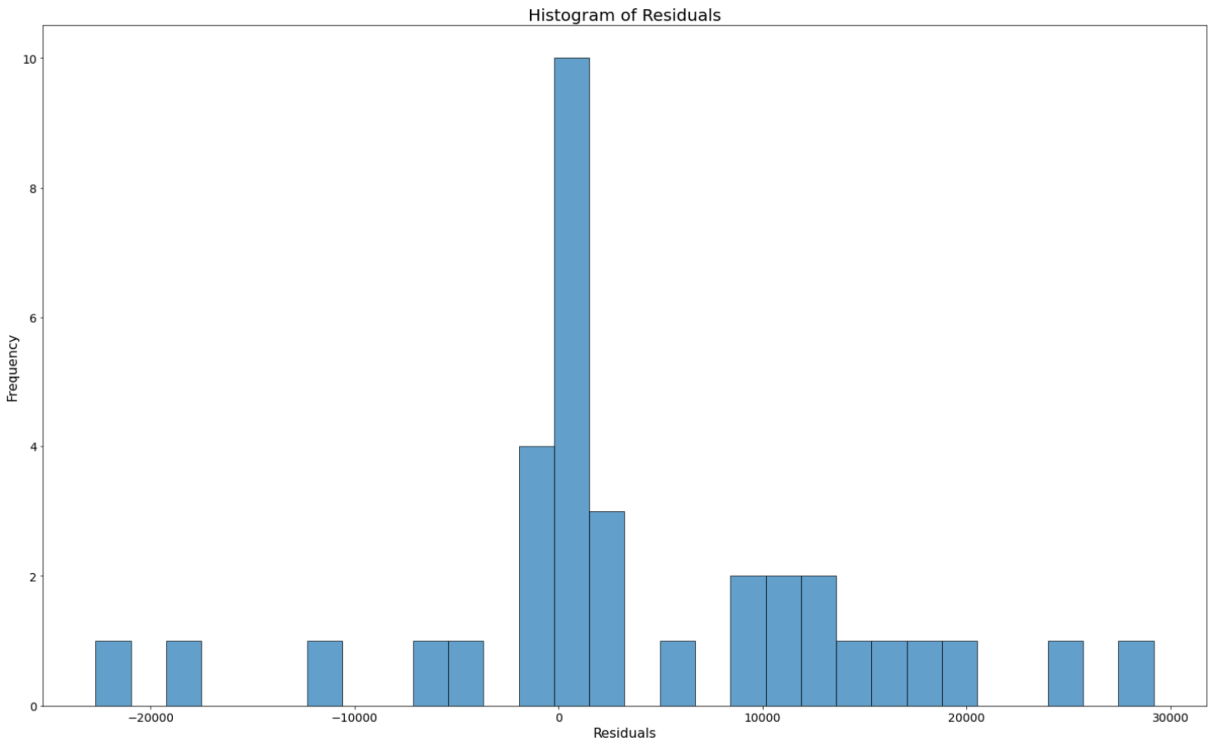
I then fit the model with the best parameter from RandomizedSearchCV and tested the fit on the test data. The results are:

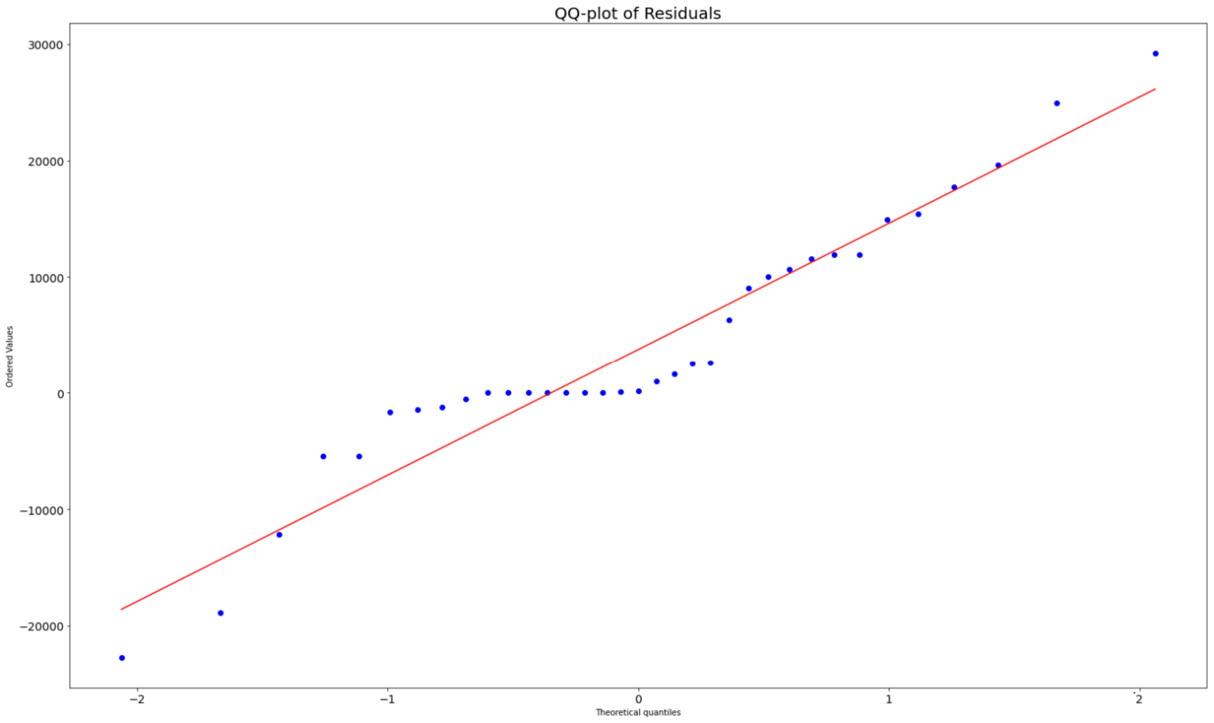


Model	R^2	MAE	MSE
Linear Regression	-0.275623	7734.13	1.28219e+08

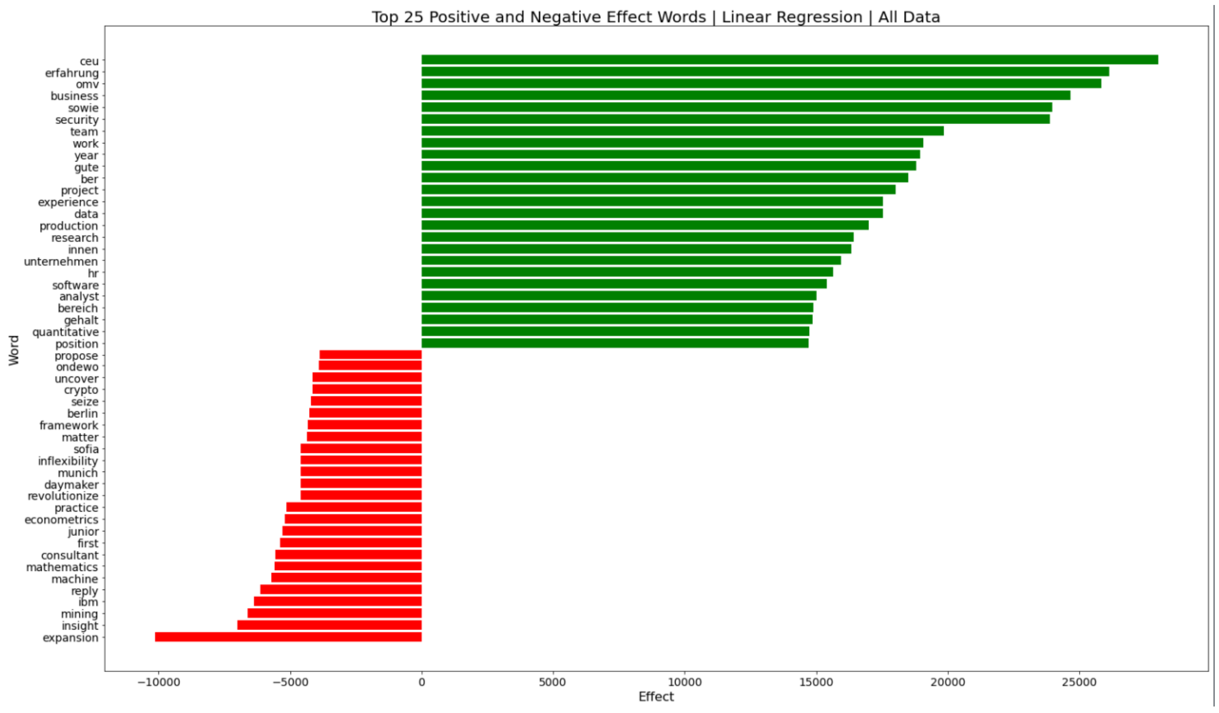
R^2 is negative, which means flipping a coin would give a better result, so a linear Regression might not be the best model. I then plotted the residuals to see how they behave:





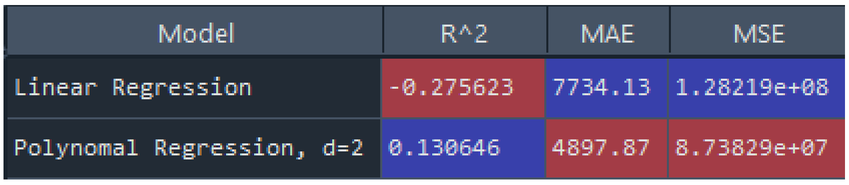


These plots indicate that a polynomial regression might be better. The word results of the Linear Regression look as follows:



Polynomial regression

I then fit a polynomial regression with degree 2 on the data. I used the same parameters as for regression before. While higher polynoms might make sense, as the residuals plotted before indicate that the number of polynoms is higher, but again, my laptop is not strong enough to use high polynoms.

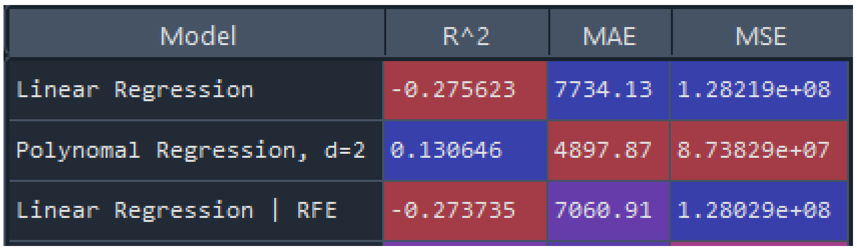


Model	R^2	MAE	MSE
Linear Regression	-0.275623	7734.13	1.28219e+08
Polynomal Regression, d=2	0.130646	4897.87	8.73829e+07

These results are much better, but the problem with the polynomial regression is, that it is not possible to easily read the words and their effects in the end. But that is my goal, so I need to find something else.

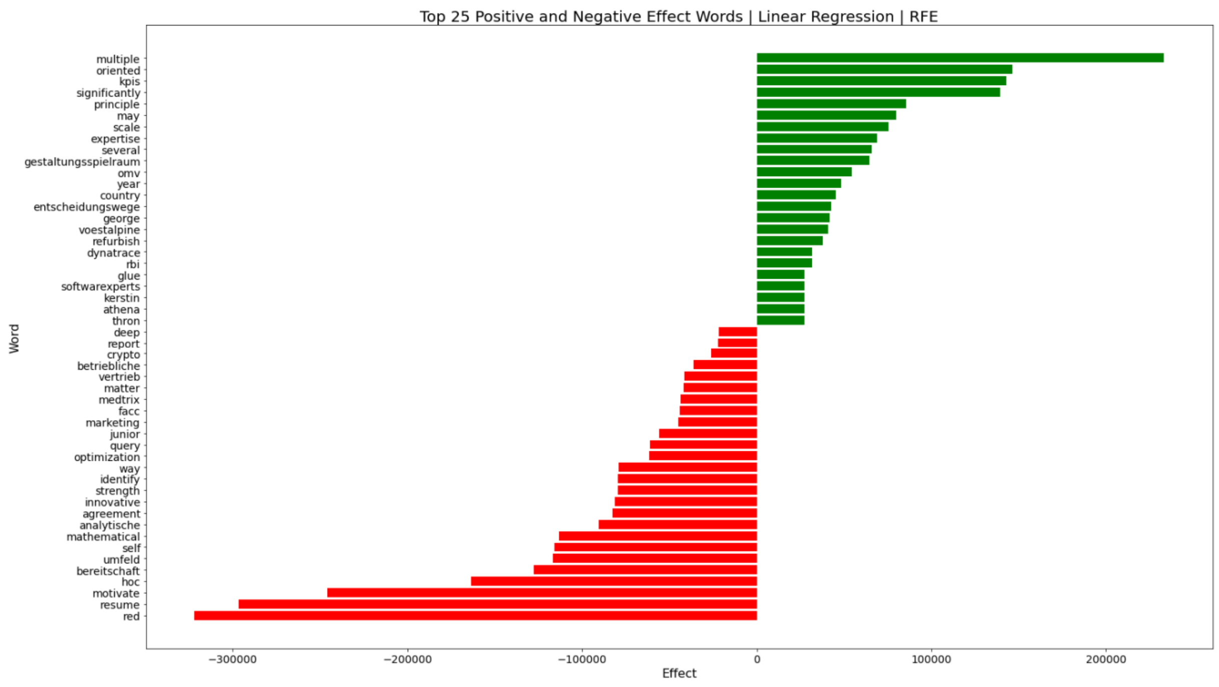
Linear Regression with reduced dimensionality

As my data has many words and only a few observations, I tried to reduce the words by using RFE. I selected 50 features. I then trained a Linear Regression on these reduced numbers of words.



Model	R^2	MAE	MSE
Linear Regression	-0.275623	7734.13	1.28219e+08
Polynomal Regression, d=2	0.130646	4897.87	8.73829e+07
Linear Regression | RFE	-0.273735	7060.91	1.28029e+08

The Mean Absolute Error got smaller, but not by a lot. And R^2 is still negative. These would be the resulting words:

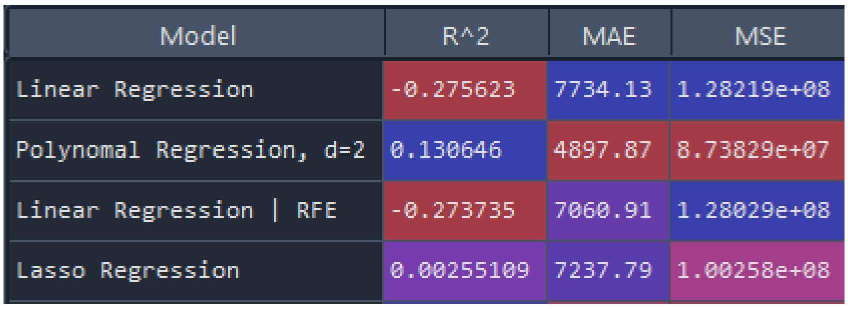


On top of the model performing not a lot better than the original linear regression, it also has unrealistic effects due to the reduced number of features. I also plotted the residuals, and they show a similar result as above. So I did not include them here.

Lasso Regression

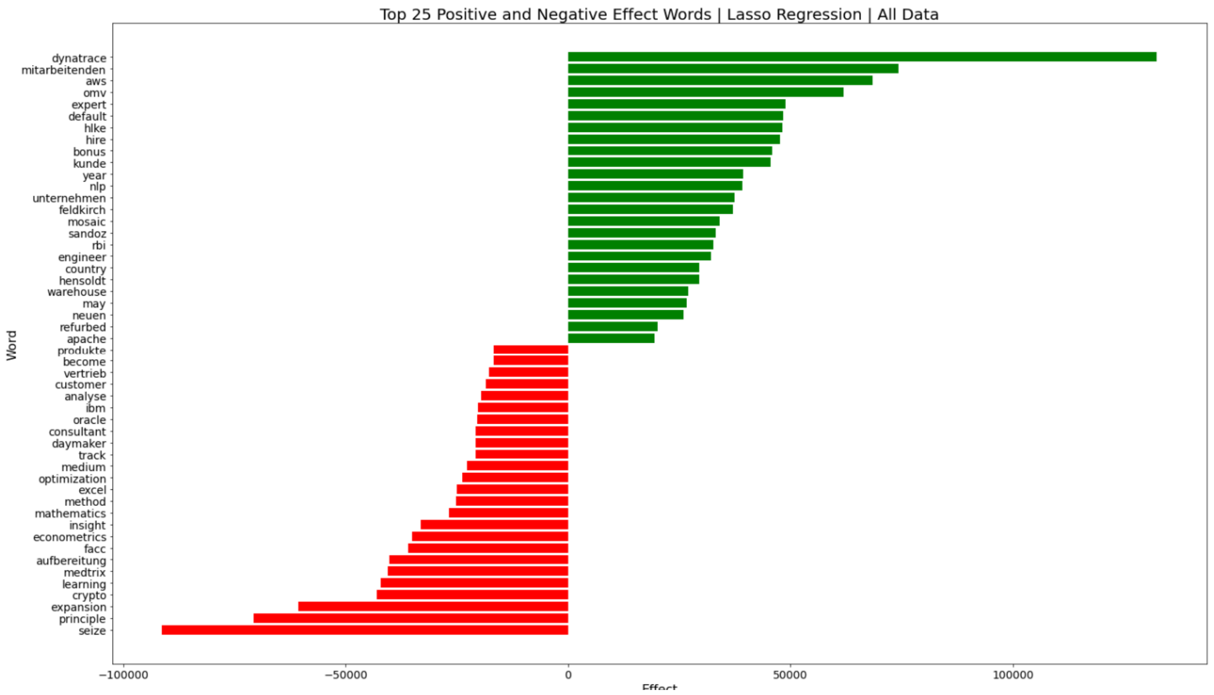
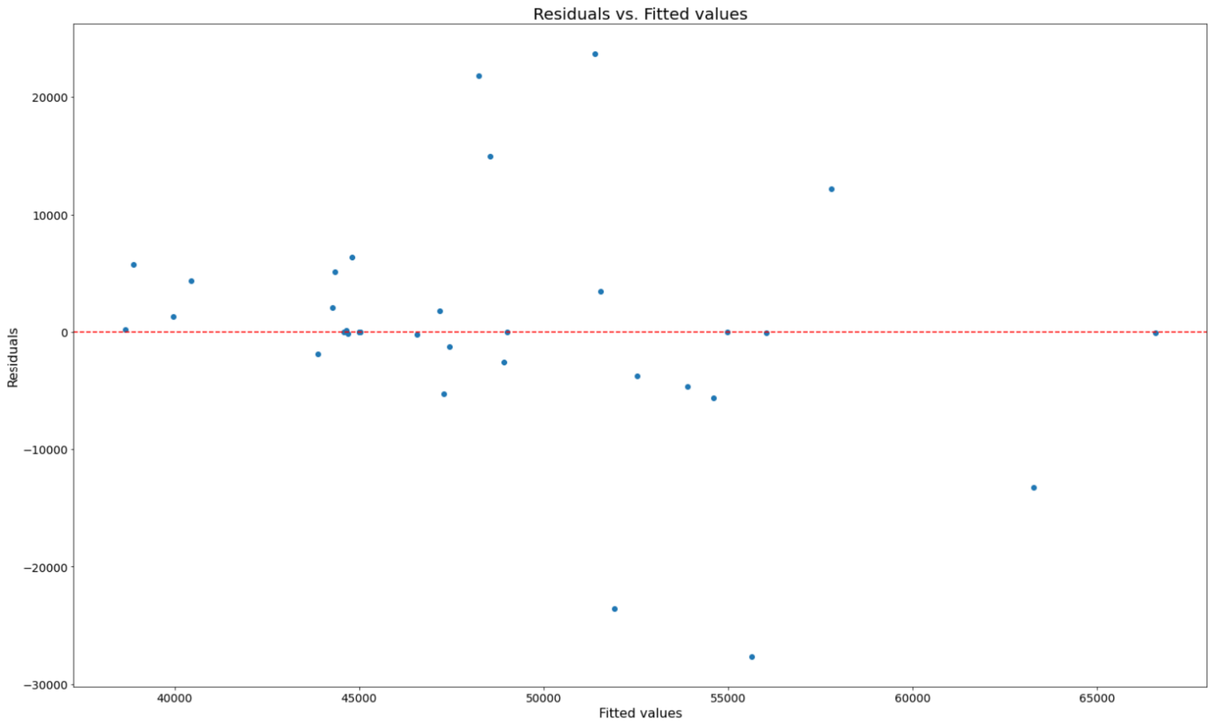
Lastly, I tried a Lasso Regression. As the Lasso Regression already reduces the number of features by itself, I did not use RFE before. (I tried that as well, it is in the python file but not part of 'official' solution)

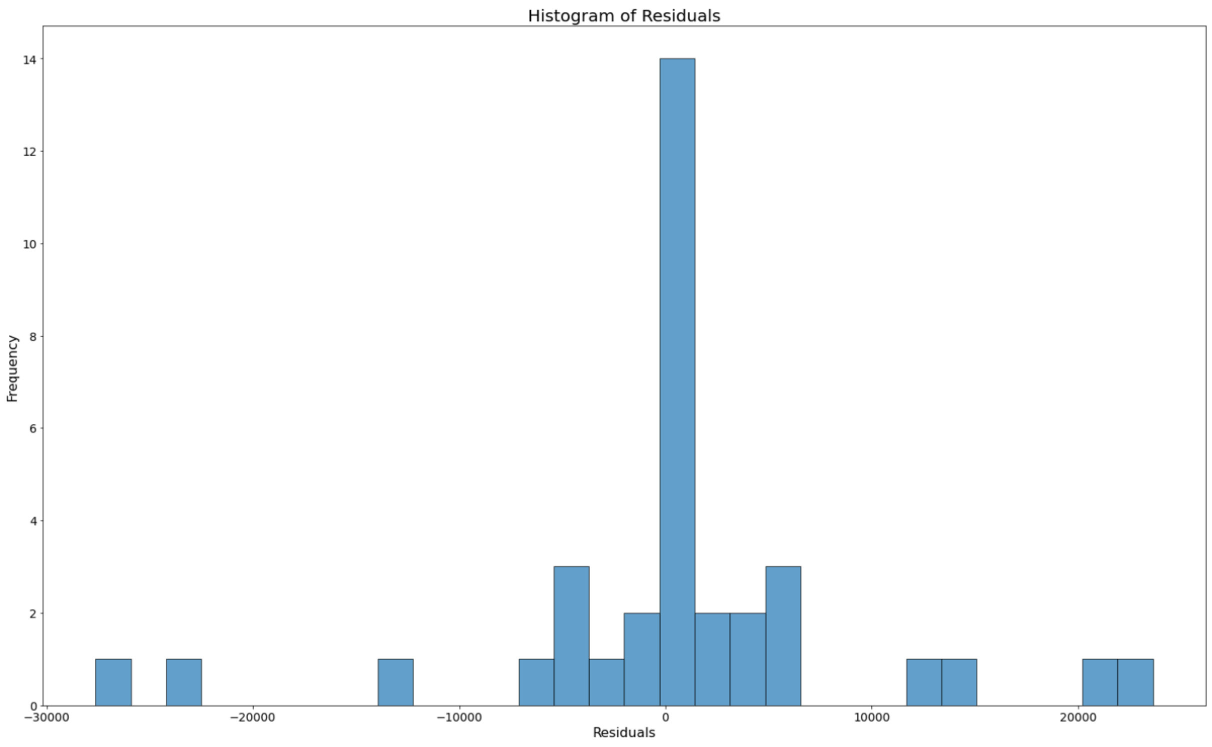
Again, I used RandomizedSearchCV to find the best parameters and trained the model.

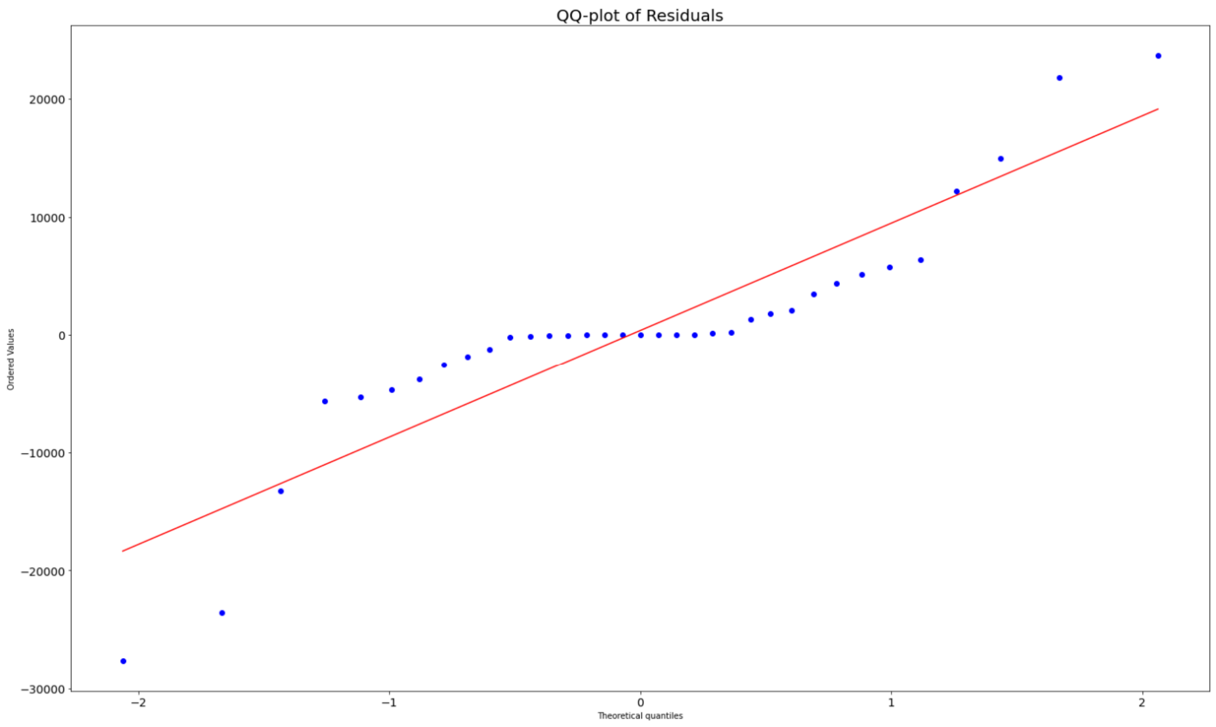


Model	R^2	MAE	MSE
Linear Regression	-0.275623	7734.13	1.28219e+08
Polynomal Regression, d=2	0.130646	4897.87	8.73829e+07
Linear Regression | RFE	-0.273735	7060.91	1.28029e+08
Lasso Regression	0.00255109	7237.79	1.00258e+08

Finally the R^2 is positive again! Although not very high (0.002) and the Mean Absolute Error, as well as the Mean Squared Error did not improve a lot. But, it still is the best model I could come up with in the limited time frame. I am planning on improving the models further, but for now these are my final results:


Top 25 Positive and Negative Effect Words | Lasso Regression | All Data
Word
Effect
dynatrace
mitarbeitenden
aws
omv
expert
default
hlke
hire
bonus
kunde
year
nlp
unternehmen
feldkirch
mosaic
sandoz
rbi
engineer
country
hensoldt
warehouse
may
neuen
refurbed
apache
produkte
become
vertrieb
customer
analyse
ibm
oracle
consultant
daymaker
track
medium
optimization
excel
method
mathematics
insight
econometrics
facc
aufbereitung
medtrix
learning
crypto
expansion
principle
seize
−100000
−50000
0
50000
100000

Residuals vs. Fitted values
Residuals
Fitted values
20000
10000
0
−10000
−20000
−30000
40000
45000
50000
55000
60000
65000





Improvements for the future

I think the most potential lies in better data. I want to keep collecting data to get more and more observations, as 173 is not a lot. Also, some other models might make sense to try. I also tried RandoForest models (one is in the code as well), but they also left me with the problem of interpretation. Unfortunately, I have not yet found a solution, which delivers interpretable results, and also fits the realationship well. RandomForest and Polynomial Regressions are a better fit, but are not that directly interpretable as I would it like to be. Maybe I can get some insights in the 15 minutes after my presentation.