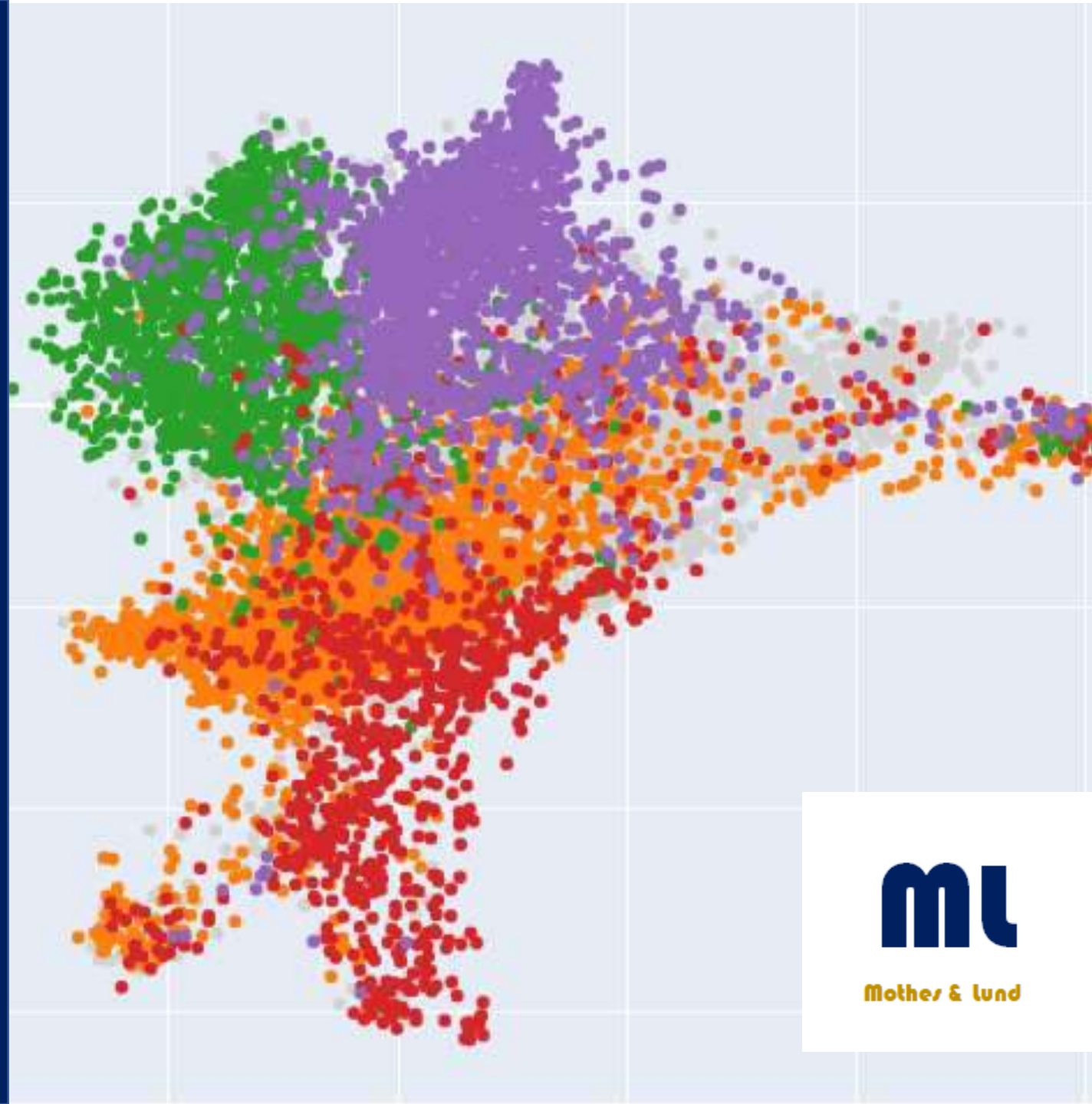
From Job Description to Salary Prediction

An application for understanding the data industry job market. Powered by AI.

Janina Mothes & Rachel Lund

January 2020

Motheslund.de



Mothes & Lund

Contents.

A bit about us.

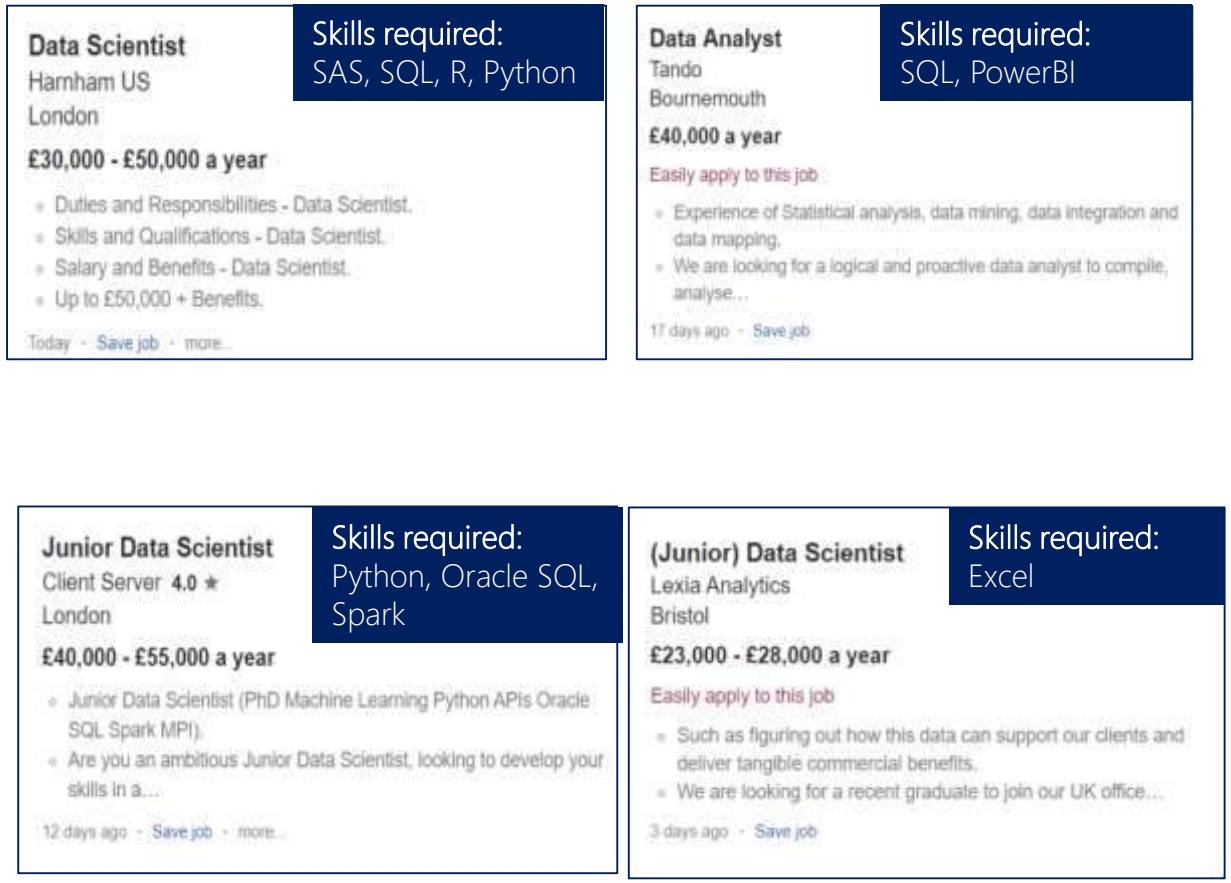
Janina Mothes

A native of Berlin, Janina completed her PhD in Theoretical Biophysics
at Humboldt University. After spending some time as a researcher she joined Bayer as an R&D IT Project Manager, before taking time out to undertake Data Science Retreat. She is also the owner of a very cute dog and is partial to the occasional glühwein.

Rachel Lund

Born and raised in London, Rachel trained as an economist, as an undergraduate at Oxford, later completing a masters at the Barcelona Graduate School of Economics. Having worked in economic consulting, Rachel spent three years as Head of Insight & Analytics at the British Retail Consortium; before taking some time out to visit South America and join Data Science retreat. Rachel also likes glühwein, especially when Janina is buying.

The problem we set out to solve.



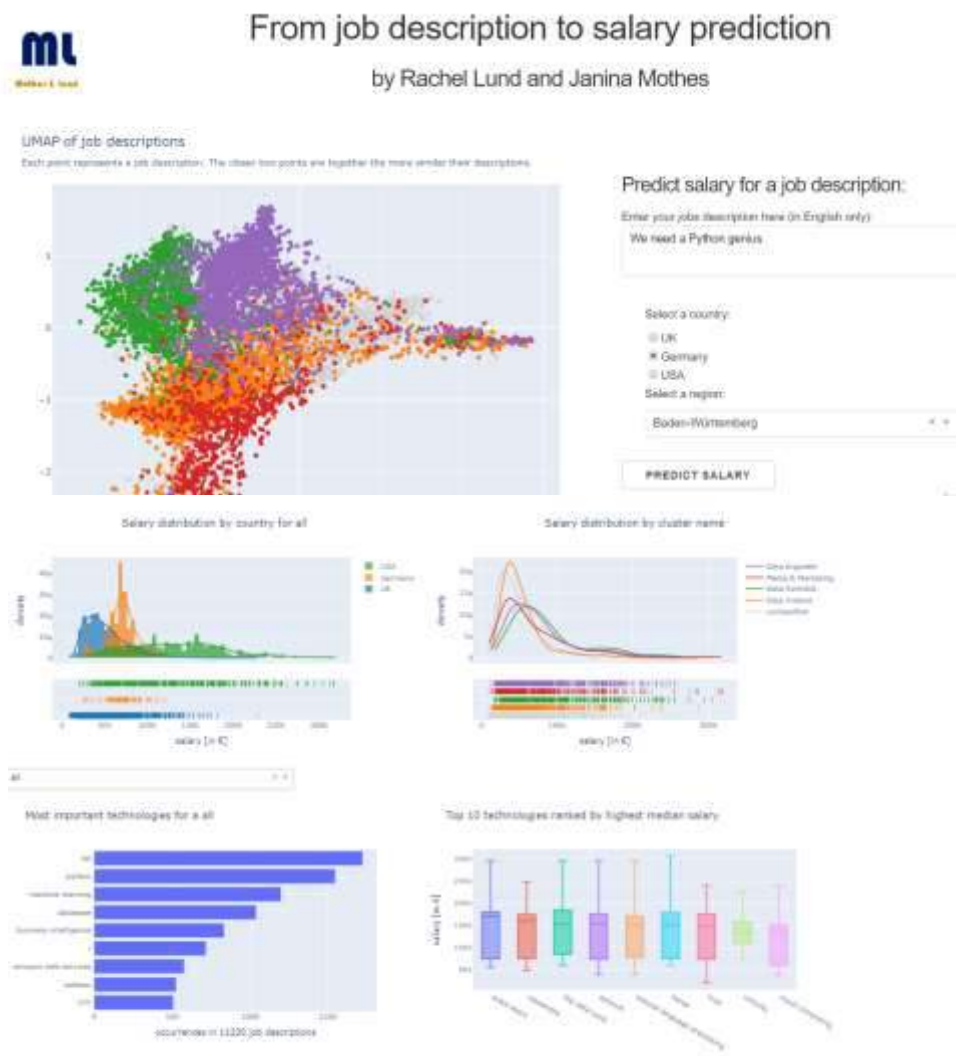
If you are a data professional , or want to hire people who work with data, navigating the job market can be both confusing and frustrating.

As the data industry grows at speed new specialisms are rapidly evolving. But without the established titles and job descriptions of more mature markets, two advertised roles may have the same title, but very different requirements or salaries. And roles with seemingly similar skill requirements may seemingly carry very different titles. This confusion makes job and candidate search inefficient. That helps no-one.

So we decided to build a tool that would:

- Tell you whether role is more suited to a data analyst, scientist or engineer
- Help understand what a reasonable salary expectation for a given role description is?
- Identify data technology skills are most in demand.

Our solution.



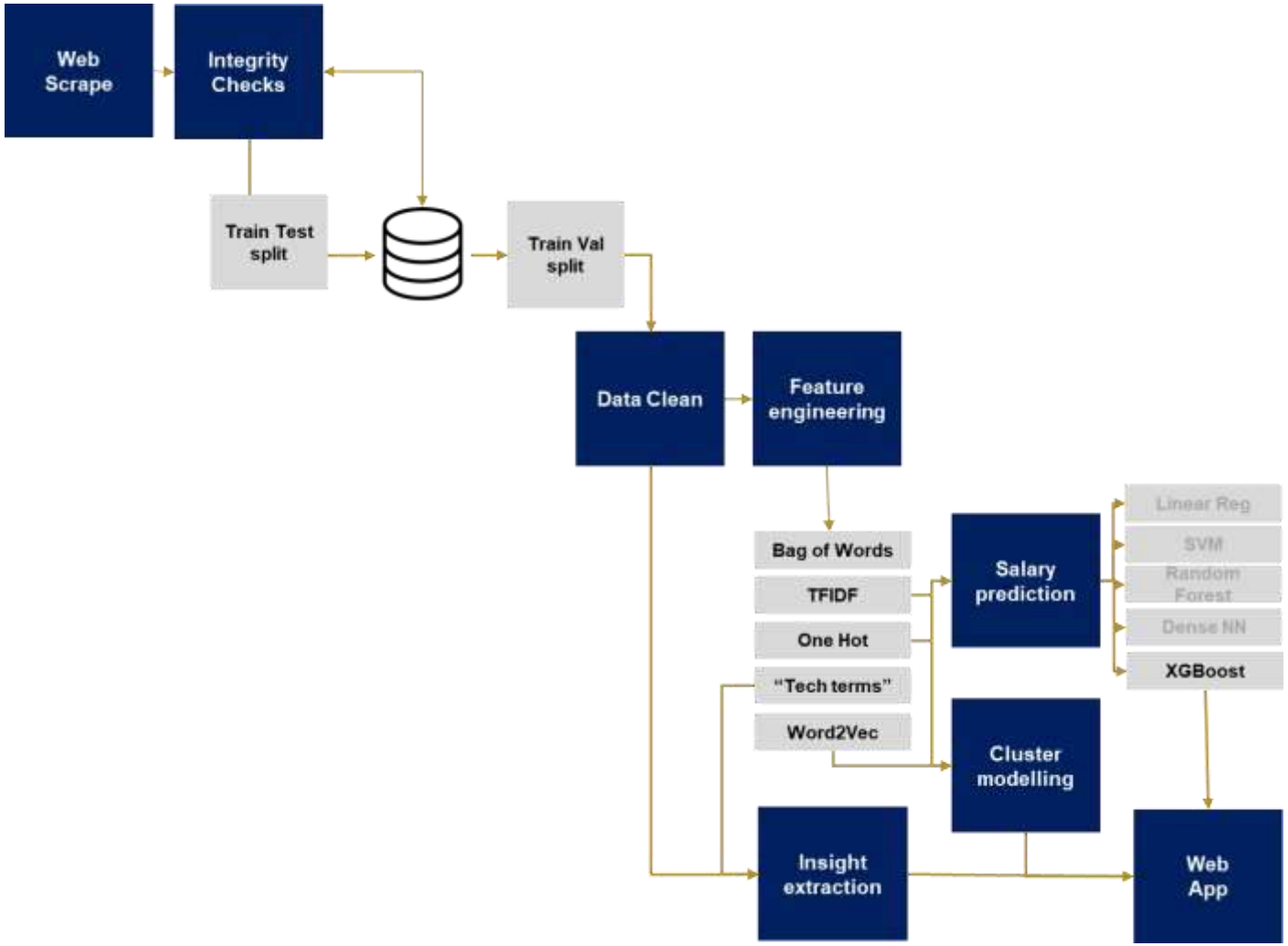
www.motheslund.de

An application that takes a job description and uses machine learning to give and estimate of appropriate salary and insights into salaries and skills by role type and location.

The application is a prototype. It is based on jobs information from October/November 2019, but it works in giving genuine salary estimates.

We continue to collect data, to allow us to update the application in future. For features we are planning to improve or develop (See slide “What’s next”).

How we built it.



Our solution was developed end to end from data collection to deployment in several phases:

Phase 1: Job descriptions and salary information was scraped from the internet and pushed to a postgreSQL database hosted in AWS

Phase 2: Data from the database was cleaned and encoded to allow it to be used for modelling

Phase 3: Summary statistics and visualizations were produced to help understand the data

Phase 4: A series of models were trained to find the best predictor of salaries. Cluster modelling was also implemented to look for groupings in job descriptions

Phase 5: The Application was built in a flask app and deployed on AWS Elastic Beanstalk

Phase 1: Scrape and Store

Amazon RDS

To train a model we needed to collect job descriptions, including salary data where available. We scraped a number of job websites using Python's Beautiful Soup and Urllib packages; searching on a number of data related search terms.

Each site had its own HTML layout and tags, so code had to differ by site. In addition to the full description and salary we captured other information such as whether it was full-time/part-time, the advertised job title, location and information from several other fields. Python's Spacy package was used to identify whether the job description was in English or not.

Scraped data was then pushed into the landing table in a PostgreSQL data base hosted in an AWS RDS instance, using Python's Sqlalchemy package.

Data were only moved to the working data table after basic cleaning and integrity checks. These integrity checks included verifying whether salary ranges were plausible – it was clear, for example, that in some cases when job adverts were uploaded an extra zero was sometimes accidentally added or omitted. Geographical data also needed to be consistent, to achieve this we used a combination of lookups and a bespoke function which called the OpenStreetMap API to check the region and country of a given place.

After these checks were passed data were moved to the working table, in the process duplicate jobs ads were deleted and each entry assigned a unique id.

Phase 2: Clean and encode

Data from the SQL database were called into Python and cleaned. Cleaning included putting data into consistent formats (i.e. string/list), removing punctuation and stop words (commonly used words such as 'a', 'the' or 'of') from descriptions, lemmatizing and tokenizing words in the descriptions using Python's Spacy package. Salary information was also stripped down to numeric values and separated into lower bound, upper bounds and average. All salaries were converted to euros based on October average exchange rates.

In order to undertake any modelling the cleaned and tokenized text had to be turned into numbers. We experimented with a number of options for encoding the data:

- Bag of Words: if a word appears in a description it gets a 1 against that word, otherwise it gets a 0 (One-hot encoding). Using Countvectorizer from the scikit-learn package.
- One hot encoding of only 'tech terms' : only appearances of variables in the 'tech dictionary' (scraped from glossarytech.com)). Using Multilabelbinarizer from the scikit-learn package.
- Term Frequency – Inverse Document Frequency (TFIDF): Each word in each job ad (document) gets a score. If it appears a lot in one job ad it gets a higher score (in that ad), however if its a really common word among all of the job ads its score is lowered. Using TFIDFvectorizer from the scikit-learn package.
- Word2Vec Encoding: Each word in a job description is represented by a vector, the more often two words appear together, the closer their normalized vectors will be. This encoding is trained on the corpus of job descriptions scraped. Using the gensim package.

Phase 3: Visualize and summarize

We scraped 34,116 job adverts…

Annual Salary (average €)	Count	Median	Min	Max
Total	11,324	51,300	17,033	317,500
UK	9,538	45,600	17,033	22,800
USA	1,431	130,000	20,374	317,500
Germany	308	72,500	26,000	135,000
Other	19	115,000	51,300	213,369

Having prepared the data we could look at some summary and distributional statistics. There are several things to note:

- Only around a third of job ads contained salary information
- The data are not geographically balanced. It was far easier to get salary information for job descriptions in the UK and extremely difficult for Germany, with the US seeing salaries advertised slightly more often.
- The fact that the UK salaries appear quite a bit lower on average is somewhat misleading. The US and German salaries include a higher proportion of higher skilled jobs, where as UK figures are capturing a lot of entry level jobs and those requiring lower skill levels such as data administrators. This is something we would like to address in future.

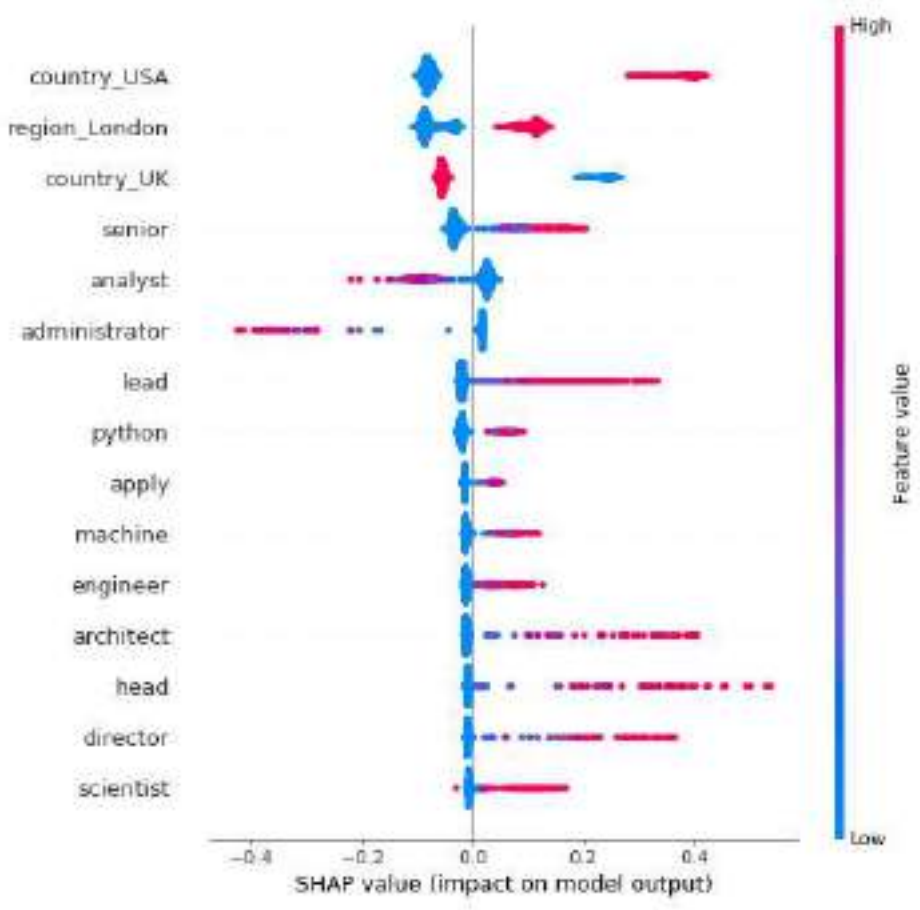
Phase 4: Model (Salary Prediction)

	RMSE % error (Train)	RMSE % error (Validation)	RMSE % error (Test)
All predictions as the mean of the train set	58.1	55.9	58.7
Linear Regression	18.4	23.9	25.1
SVM	28.6	28.8	28.5
Random Forest	17.4	24.5	24.8
XGBoost	13.7	20.3	21.0
Neural Net	17.1	20.7	22.1
Multimodel Ensemble	14.7	22.1	22.3
XGBoost with full train set	**10.6**		**19.8**

We trained a number of models to find the best predictor of salaries, using the different types of encoding. The table shows the Root Mean Squared percentage error (RMSE %) for out best implementation of each model.

- The TFIDF encoding gave the best results for all models.
- All models performed better than just predicting the mean value of the test set.
- XGBoost performed best on the validation data set and still performed well on the test data.
- XGBoost outperformed model ensembles. This is not so surprising given that the models all saw similar error patterns (see the following slide)
- We didn't really have enough data for the neural net to perform well; it quickly over trained even with a simple structure. However, with more data it could outperform the XGBoost.

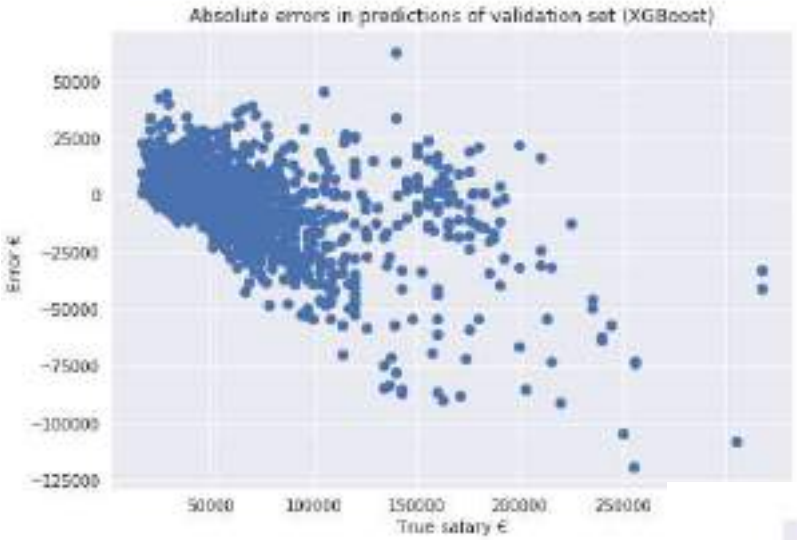
The results of the XGBoost make sense

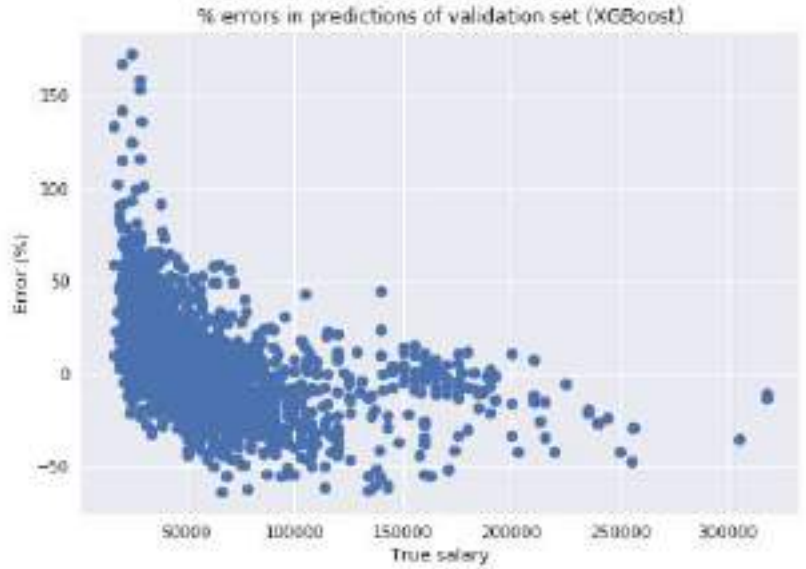


The chart opposite is a SHAP diagram. It indicates which features are driving the results of the modelling. Where there is a line of pink dots to the right of the vertical axis, it indicates that high values of that feature (or the presence of that feature) positively impact salary, whereas if they are to the left of the vertical axis high values negatively impact salary.

- Location has a strong influence on salary: Being in the USA or London means the salary for the job is likely to be higher, while if its elsewhere in the UK its likely to have a lower salary.
- Having words such as 'senior', 'head', 'director' and 'lead' in the job description are likely to mean the salary is higher as you would expect, while 'administrator' and 'analyst' are likely to mean the salary is lower.
- Skills such as 'Python' and 'Machine (Learning)' also positively impact salary.

There is room for improvement



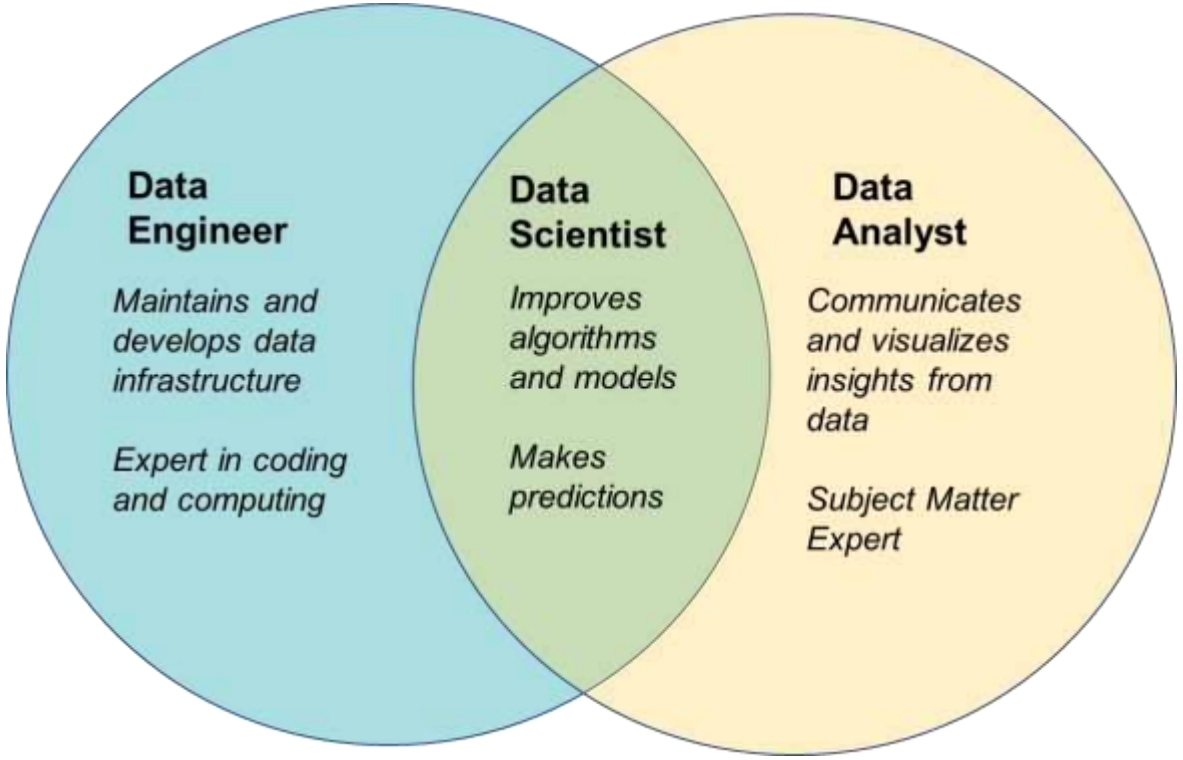


When we look at the results of the XGBoost we can see that errors are large in absolute terms at higher salaries, while in percentage terms there are some very large errors at larger lower salaries.

To improve model performance we tried splitting the data and modelling higher and lower salaries separately. We also tried modelling the UK and USA separately. Neither approach improved the results.

This is an area of further investigation for us.

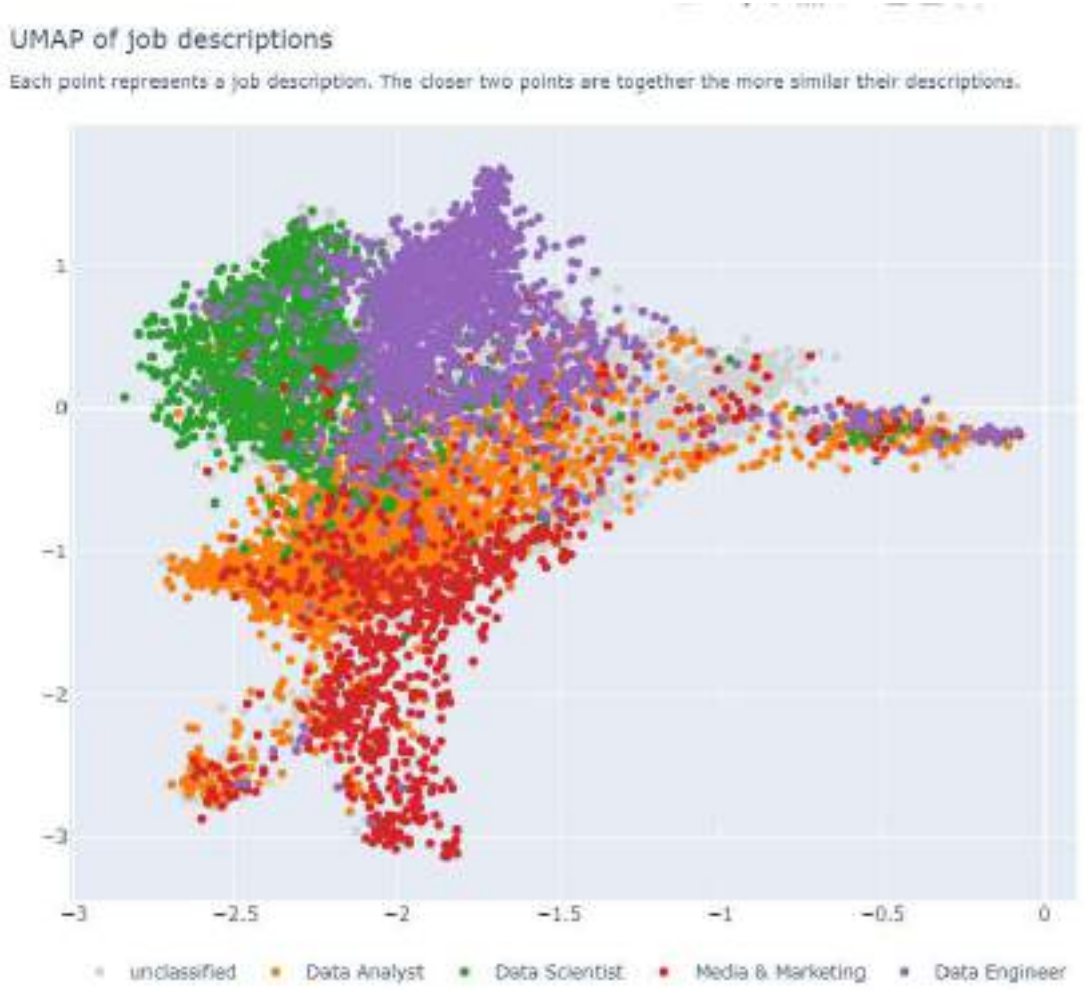
Phase 4: Model (Cluster analysis)



In addition to predicting salaries we wanted to look at whether we could cluster job descriptions, to give a better indication of what specialism within the data industry a job was closest too.

The diagram opposite shows a version of a dichotomy of jobs that is frequently used today. However, it is common to find elements of all three job types under all three job titles. Specialization also goes deeper than these broad areas. Within Data Analyst you may be a BI Analyst or you may be a Computer Vision or NLP Data Scientist.

Phase 4: Model (Cluster analysis)



To undertake the cluster analysis we first had to reduce the dimensionality of the data. This also allowed us to plot each job description on a two-dimensional chart as shown opposite.

To do this we applied the UMAP algorithm.

To search for clusters we initially applied the HBDSCAN algorithm. This identified some interesting patterns in the data, but its use to the broader goals of the project was less clear. Instead we decided to keep it simple. We allocated job descriptions to groupings based on the frequency of key words in the job description. This gives the colorings shown opposite. The 'Media and Marketing' grouping originally fell under 'Data Analyst', but with a view to breaking down the group 'Data Analyst' (which was large) we found a specialization around analysis of marketing and media.

Further work would almost certainly turn up further sub-specialties in the groupings.

If you go onto the app and roll over the chart you can see the variation in job titles for jobs that are similar. If you put in your own job description it will appear on the chart and you can see to where it most closely aligns.

Phase 5: Build & deploy

To put the analysis into a tool that people could actually use, a number of interactive charts were designed, it was then wrapped in a flask app and deployed it to elastic beanstalk on AWS.

Elastic beanstalk allows new instances to be spun up when multiple people are using the site; preventing it from crashing when multiple people are using it at once.

Each time 'predict salary' is clicked on the app, text entered is cleaned and encoded, the results of the XGBoost model are applied to make a prediction of salary and the UMAP algorithm reduces the text to a point on the chart.

What comes next.

Data, data, data
The performance and relevance of the modelling would be improved by gathering more data. To keep it relevant we will need to update it over time; while broadening our data collection beyond the terms used or to additional sources will enrich the content. We continue to gather data.

Design and UX
We're data scientists, not web-designers. We think we the user experience can be improved. We'd also like to reduce the load time for the page itself.

A job recommender system
Develop a recommender system for similar jobs based on the cluster analysis. So when you input a role description or your CV similar jobs are shown to you

Configure the app to update data over time.
At the moment the app shows a static snapshot of data collected at the end of 2019. We'd like to streamline the data collection and feed into the app so the information changes based on the most up to data information. This would also mean re-estimating the model on a regular basis.

Trend analysis

Add job trend data on the dashboard. As we collect data over time we would like to analyze and display trend information

Please get in touch.

We'd be happy to discuss the project, data and analysis further.

Rachel : https://uk.linkedin.com/in/rachel-lund-3500972b

Janina: https://de.linkedin.com/in/janinamothes/en