Handelshøyskolen BI

MAN 51001 Analytics for Strategic Management

Term paper 60% - W

Predefinert informasjon

Startdato:	06-09-2022 09:00 CEST	**Termin:**	202310
Sluttdato:	04-05-2023 12:00 CEST	**Vurderingsform:**	Norsk 6-trinns skala (A-F)
Eksamensform:	P		
Flowkode:	202310||10055||IN08||W||P		
Intern sensor:	(Anonymisert)		

Navn:

Erlend Aas, Dominique Daniel Bye-Ribaut, Anita Røe

Informasjon fra deltaker

Tittel *:	The Predicted Usage Model (PUM)		
Navn på veileder *:	Chandler Johnson		
Inneholder besvarelsen konfidensielt materiale?:	Nei	**Kan besvarelsen offentliggjøres?:**	Ja

Gruppe

Gruppenavn:	(Anonymisert)
Gruppenummer:	10
Andre medlemmer i gruppen:	

Project paper

The Predicted Usage Model (PUM)

Hand-in date:
03.05.2023

Campus:
BI Oslo

Examination code and name:
MAN 51001 Analytics for Strategic Management

Programme:
Executive Master of Management

Content

Summary

This predictive analytics project explores how machine learning can contribute to increased customer usage of NTB’s foreign news articles. The result of this research is the Predicted Usage Model (PUM) – a regression model that forecasts the number of NTB customers that will run any given syndicated international news story. The idea is that PUM can help the foreign duty editors select the stories that have the greatest potential usage. The current baseline process does not involve the use of analytical tools but is mainly based on editors’ individual skills and experience. Thus, it is person-dependent and prone to variability. The proposed model outperformed a simple baseline model that uses the mean as a constant value for all the predictions. The paper describes how the model can be further improved in subsequent iterations, outlines how it can be operationalized within the existing system architecture, and lays out a way forward.

Introduction

At the core of NTB's business is the news service – a traditional wire service consisting of news articles divided into three main categories: Domestic, Foreign and Sports. While most of the domestic news is original content created by NTB's own journalists, the foreign news service consists mainly of syndicated stories from other agencies: The Associated Press (AP), Reuters, Agence France-Presse (AFP), Deutsche Presse-Agentur (DPA), TT Nyhetsbyrån and Ritzau.

The overarching business model of NTB is to sell the same product or service to many clients, or – as in the case of the news service – produce content that will get picked up and published by multiple customers. With a limited capacity – on average NTB produces 48 foreign news articles daily – and more than 200 disparate news outlets on the client list, it's imperative that NTB is able to select the syndicated stories with the greatest potential usage.

The baseline process: The role of the foreign duty editor

The foreign duty editor is responsible for choosing the international news articles that eventually will be offered to NTB customers. The syndicated stories – totaling approximately 2000 on a daily basis – are ingested directly into the NTB content management system (CMS). As the news comes in, the foreign duty editor is tasked with assessing the relative newsworthiness of each of these stories and their relevance to a diverse range of NTB clients. During the busiest periods, incoming stories may amount to 10-20 per minute. If a story gets selected, the foreign duty editor either assigns it to a reporter or processes it herself. Processing entails translating the story to Norwegian, adding information from other articles or sources if relevant, and choosing the appropriate tags with the assistance of a cloud-based auto-tagging tool (iMatrics). The article is then distributed to clients via the NTB news portal, or through the News API.

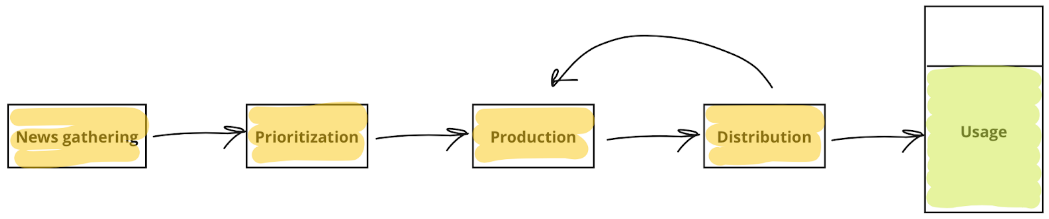


Figure 1. Sketch of the baseline process.

There is a tendency among news media managers to characterize every decision that doesn't involve the direct use of analytics tools as being based on "gut feeling" (Lilleby, 2019; Simonsen, 2023). Such characteristics don't acknowledge the value of experience, knowledge that might have been accumulated over decades of service, nor the great problem solving capabilities of the human brain. That being said, there are several potential drawbacks of the current process: it's susceptible to the duty editors' personal preferences, likes and dislikes, habits, lack of experience, disregard of – or unfamiliarity with – the business goals, and activism.

The solution: PUM

The Predicted Usage Model (PUM) forecasts the number of clients that will pick up each individual syndicated article. The model is trained on the semi-structured metadata that follows every news article, combined with historical data on article usage from Retriever. PUM is a regression model, where usage – the number of clients – is the target variable (also called the dependent variable), and the article metadata are the features (or the independent variables) whose values are used to predict the value of the target variable. By integrating a non-disruptive ML prediction into the current workflow, PUM can guide and assist the foreign duty editor in the decision making.

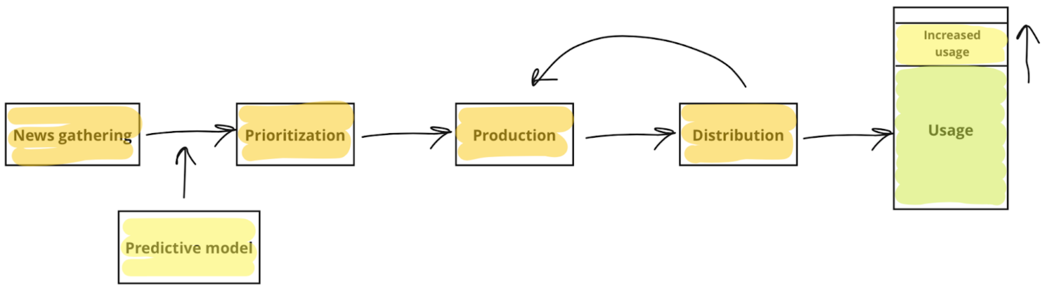


Figure 2. Sketch of process flow with integrated PUM.

Data preparation

Note: This paper is primarily aimed at stakeholders who don't necessarily have detailed knowledge about data science. Those technically inclined can dive into the full code listing in the appendix.

The data was sourced from two separate APIs: the NTB News API (production data) and the Retriever API (usage data). Python scripts were written to

1. extract the data by calling the APIs and uploading the articles to the Google Cloud
2. convert the date format
3. transform from single JSON object to a collection of JSONL (newline-delimited JSON), a format supported by Google BigQuery ML (BQML)

See appendix 2.1 for details. The BigQuery Python Client Library was then used to load the data into BigQuery tables (appendix 2.2). Since the two datasets lacked a common key, merging them required a solid mix of data science skills, domain expertise and creativity (appendix 2.3). In light of the challenges we had matching the two IDs, we suggest that the Retriever API should also include the original NTB article ID in the future. This improvement would be useful not only in this particular context, but also for other machine learning projects, as well as descriptive analytics tasks.

After preparing the data and converting it to tabular form, we ended up with 20 features and 10,815 unique instances, or articles (figure 3).

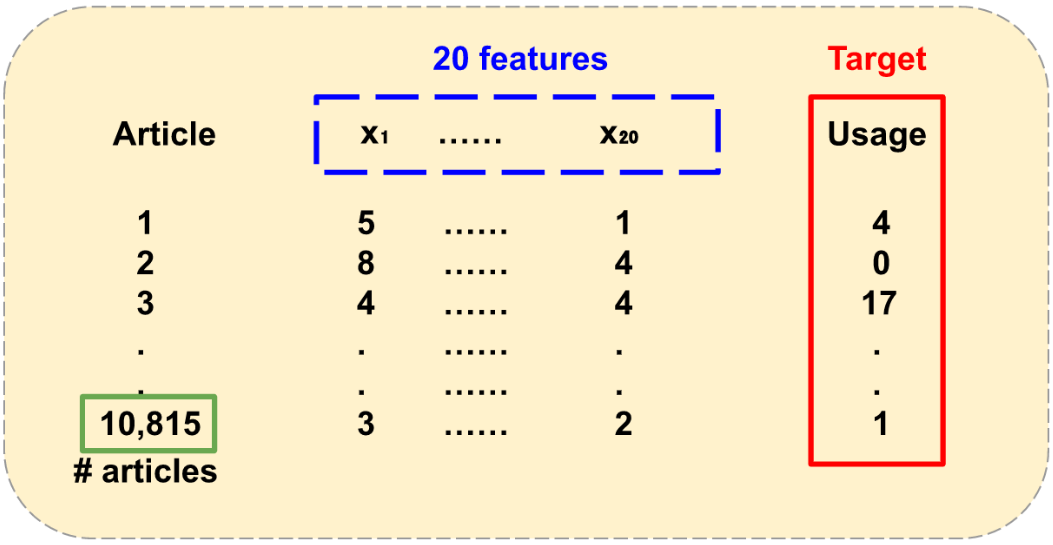
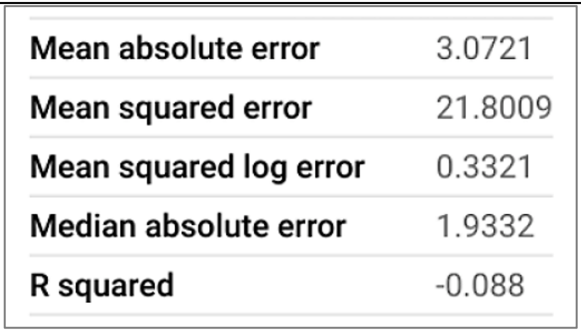


Figure 3. Illustration of model dataset.

Modeling

Google BigQuery ML (BQML) was used for the modeling as it supports training on dataset with nested columns that were challenging to flatten out. In addition, BQML does automatic feature preprocessing during training – missing value imputation and feature transformations. Numeric features are standardized automatically, while non-numerical columns are auto-encoded. The SQL syntax made working with the feature set, updating the view, and retraining the model, fairly straightforward. As with similar out-of-the-box machine learning platforms, BQML automatically splits the input data into training, validation and holdout sets to avoid overfitting.

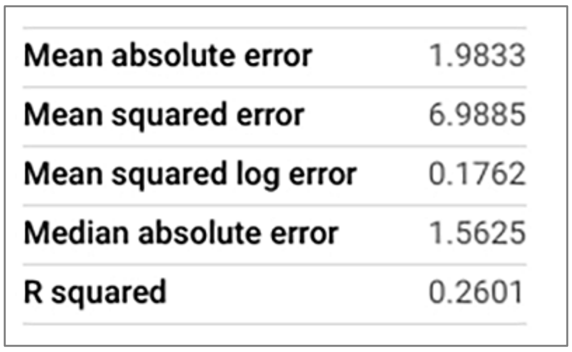
In line with the fail-fast philosophy, we started out with a very simple linear regression model (figure 4).



Mean absolute error	3.0721
Mean squared error	21.8009
Mean squared log error	0.3321
Median absolute error	1.9332
R squared	-0.088

Figure 4. Evaluation metrics for the first linear regression model.

This approach allowed us to quickly detect failures, iterate and improve. The first few runs produced poor-performing models, but by adding features and tuning the hyperparameters, we were able to improve the model performance substantially from iteration to iteration (figure 5).



Mean absolute error	1.9833
Mean squared error	6.9885
Mean squared log error	0.1762
Median absolute error	1.5625
R squared	0.2601

Figure 5. Evaluation metrics for boosted tree regressor.

To avoid time-dependent data leakage – in simple terms you don't want to use data from the present to predict past events – we chose to split the data sequentially. In addition, we made the decision to optimize the model for mean absolute error (MAE) instead of R-squared. Before we get into the rationale behind this choice, we will briefly introduce the two metrics. A basic understanding of MAE and R-squared is necessary, not only to understand the reasoning behind the hyperparameter tuning, but also the analysis of the model's performance in the evaluation chapter.

The coefficient of determination, also known as R-squared or R2, represents the proportion of the variation in the dependent variable that is predictable from the independent variables in the model. R-squared ranges from negative infinity to 1, but is normally between 0 and 1. For example, an R-squared of 0.70 indicates that approximately 70% of the observed variation can be explained by the model's features. A negative R-squared indicates that the model's performance is worse than a horizontal line which predicts the mean value every time. It's impossible, if not meaningless, to define an acceptable R-squared value. What can be judged as a good value depends on the use case, but overall, the higher the R-squared, the better the model fits the data. Though R-squared is the default hyperparameter objective in BQML, the metric has a few drawbacks. First of all, R-squared increases as the number of features in the model is increased, even when the features added to the model don't have predictive quality. Secondly, while a high R-squared is necessary for precise predictions, it is not adequate by itself. We should add that R-squared can be hard to grasp for non-technical stakeholders, which in turn may make them reluctant to support model deployment.

Mean absolute error (MAE) on the other hand, is easy to interpret and explain. MAE is simply the average absolute error between the actual and the predicted values. In figure 6, the blue line represents the predictions, while the data points represent the actual values. The error is the distance between the two.

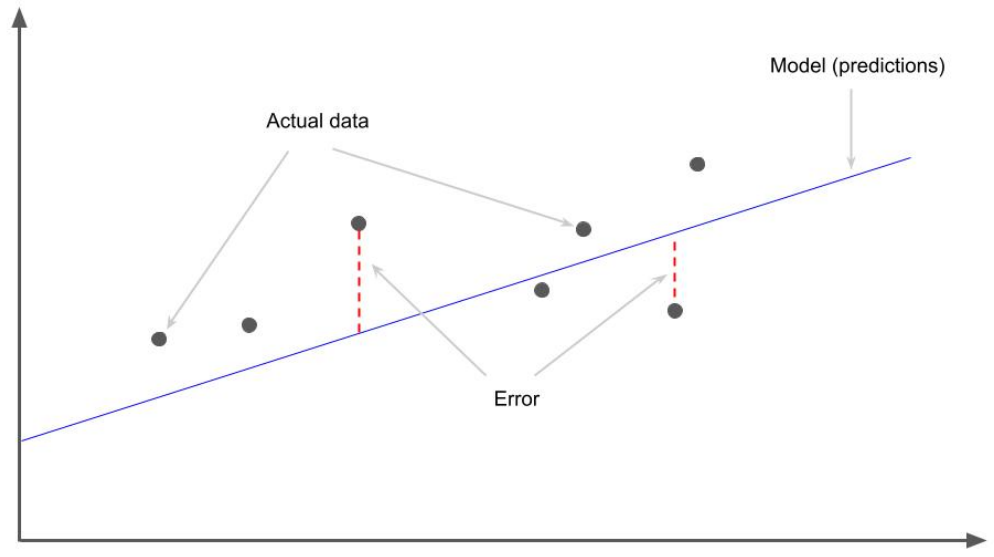
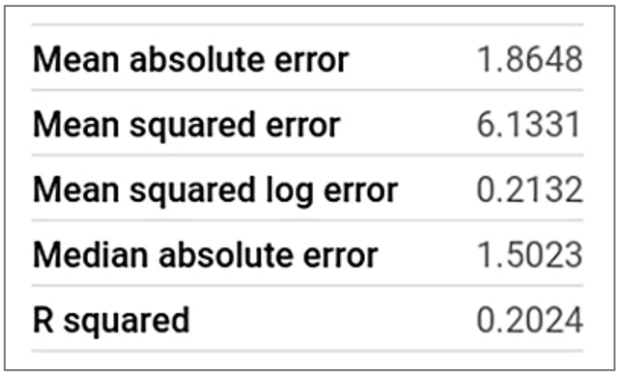


Figure 6. Illustration of prediction errors.

MAE is a satisfyingly intuitive metric because it returns the error in the units we care about – the units of the output variable – which in our case is "number of customers". The closer MAE is to zero, the better the model. By experimenting with more complex model types and tuning the hyperparameters, we were able to push the MAE lower and lower until the project deadline.

Evaluation

To the big question: How good is the final model? By themselves, the evaluation metrics give cause for optimism (figure 7).

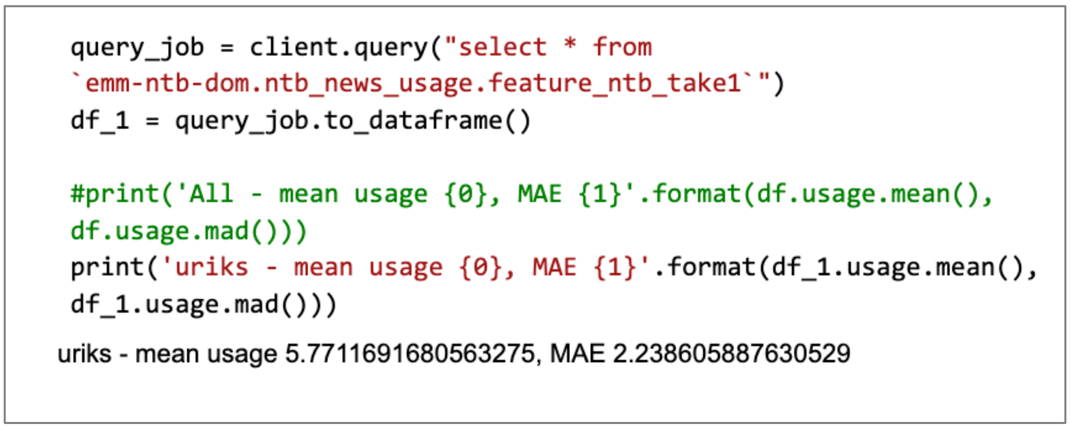


Mean absolute error	1.8648
Mean squared error	6.1331
Mean squared log error	0.2132
Median absolute error	1.5023
R squared	0.2024

Figure 7. Metrics for the final model. Boosted tree regressor.

A MAE of about 1.86 tells us that the model's predictions of usage are off by less than two customers on average. At first glance a relatively low number, but bear in mind that the actual average usage in the dataset is just 5.77. An R-squared of 0.2024 indicates that about 20% of the variance in usage is explained by the features. As stated above, there is no general rule for determining if an R-squared is adequate or too low. Some areas of study are simply more inherently unpredictable than others, and sometimes there is considerable value in explaining 20% of the variation.

To establish whether the model can add value to NTB, we need a basis for comparison. For regression problems, the average can serve as a simple yet effective baseline model. We can calculate the baseline by predicting the mean for every observation in the dataset using Python, and we can use the MAE as a comparison metric (figure 8).



```
query_job = client.query("select * from
`emm-ntb-dom.ntb_news_usage.feature_ntb_take1`")
df_1 = query_job.to_dataframe()

#print('All - mean usage {0}, MAE {1}'.format(df.usage.mean(),
df.usage.mad()))
print('uriks - mean usage {0}, MAE {1}'.format(df_1.usage.mean(),
df_1.usage.mad()))
```

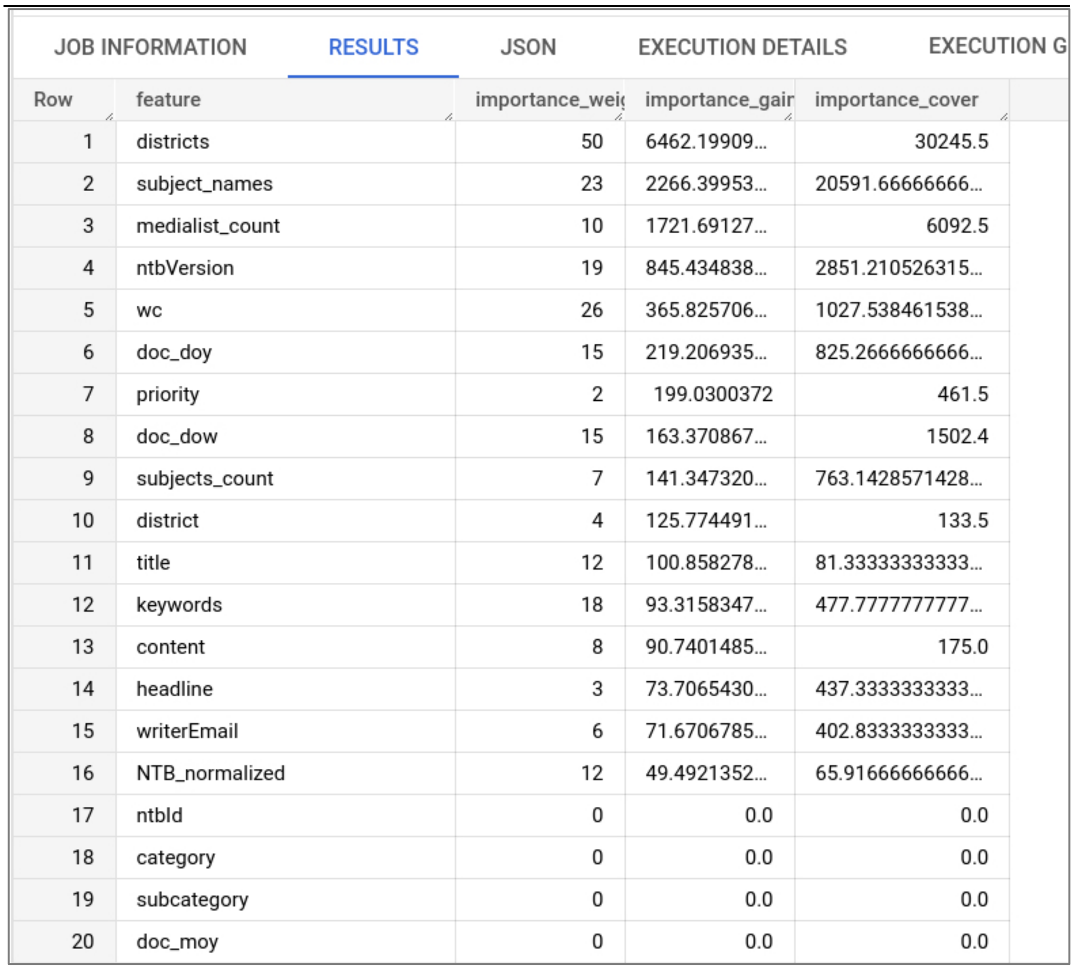
```
uriks - mean usage 5.7711691680563275, MAE 2.238605887630529
```

Figure 8.

As figures 7 and 8 show, our model with a MAE of 1.8648 was able to beat the baseline error rate of 2.2386. Thus, we can conclude that PUM has more predictive power than a model that only guesses the average for every article. That might not sound very impressive, but if the model can help increase article usage even by a few decimals, it could have substantial business impact.

Compared to the earlier models trained on the same data set, the final version also showed a considerably lower mean squared error (MSE). MSE is widely regarded as one of the most useful evaluation metrics for regression models. As with other loss functions that squares the errors, it gives more weight to large errors and is sensitive to outliers.

The feature importance list, when sorted by importance gain, shows that "districts" and "subject_names" are the features with the most predictive quality (figure 9). "Districts" is a geographical tag (city, country or continent), while "subject_names" are the main topics of the individual articles. The number of topics per article in our dataset ranges from zero to 17.



Row	feature	importance_wei	importance_gair	importance_cover
1	districts	50	6462.19909...	30245.5
2	subject_names	23	2266.39953...	20591.66666666...
3	medialist_count	10	1721.69127...	6092.5
4	ntbVersion	19	845.434838...	2851.210526315...
5	wc	26	365.825706...	1027.538461538...
6	doc_doy	15	219.206935...	825.2666666666...
7	priority	2	199.0300372	461.5
8	doc_dow	15	163.370867...	1502.4
9	subjects_count	7	141.347320...	763.1428571428...
10	district	4	125.774491...	133.5
11	title	12	100.858278...	81.33333333333...
12	keywords	18	93.3158347...	477.7777777777...
13	content	8	90.7401485...	175.0
14	headline	3	73.7065430...	437.3333333333...
15	writerEmail	6	71.6706785...	402.8333333333...
16	NTB_normalized	12	49.4921352...	65.91666666666...
17	ntbId	0	0.0	0.0
18	category	0	0.0	0.0
19	subcategory	0	0.0	0.0
20	doc_moy	0	0.0	0.0

Figure 9.

From a domain expert’s perspective, it makes sense that geography and topics are the most useful variables for predicting usage. In the media business, geographical and cultural proximity has always been considered important criteria when determining a story’s newsworthiness. Now we may have the data to support it.

Improving the model

Although the model outperformed the simple baseline model, it is by no means the finished product. As Provost and Fawcett states in *Data Science For Business*, "iteration is the rule rather than the exception" (2013, p. 27). Judging from our experience, the model's performance can be further improved through feature engineering, hyperparameter tuning and experimentation with different model types. Platforms like BQML may enable people with limited machine learning experience to train high-quality models, but they can't fully substitute the nous and experience of professionals. Experts in the fields of data science and statistics will no doubt be able to find imperfections in the current model. This should not be taken as proof of model inadequacy, but rather as a testament to its potential. Most of the available input data was ingested in its purest form; only the most necessary transformation of the data was performed. Therefore, it's quite likely that model performance can be improved substantially through feature engineering, like creating new features and removing irrelevant ones. Manual encoding of categorical variables should also be considered, as well as standardization of numerical features. BQML does automatic one-hot encoding and standardization for all available model types except random forest and boosted tree models, the latter of which was the type of our final and best performing model.

Another issue, though not directly related, is that of selection bias. The articles in our dataset are only the articles that – for whatever reason – were picked for translation. In other words, they are just a subset of the population of possible articles. There are all these other instances – stories that never got picked up and translated in the first place – that the model never gets to see. Thus, the data we're basing our predictions on might already be biased by the baseline selection routine. There are a couple of ways we can correct for this. One is through the use of importance weighting techniques, the other is to start investing in unbiased data by picking a share of the syndicated stories at random for processing.

The observant reader will notice that there is no mention of the article text. In this project we sacrificed the text data for manageability. Mining the document text would perhaps make the model perform better, but it would also make it far more complicated. At a later stage, a more sophisticated model can make use of the unstructured text data through text mining techniques like bag-of-words and sentiment analysis.

Operationalization: fitting the model in the existing pipeline

For a predictive model to be adopted by the editorial staff, it should be integrated into the technical architecture of the business' existing production software, and blend seamlessly into the newsroom workflow. A tool that would slow down the current process, or create friction in any other way, no matter how ingenious or valuable, will be rejected. By focusing on these preconditions, we were able to narrow down and find the most viable solution. As shown in figure 10, today the auto-tagging is only implemented *after* selection:

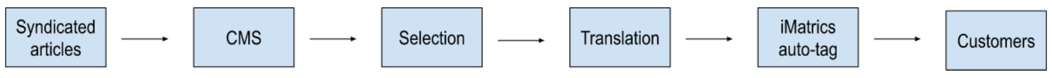


Figure 10. Existing pipeline.

By introducing auto-tagging further upstream, all the syndicated articles can be automatically run through PUM pre-selection, thus providing the foreign duty editor with a powerful new tool (figure 11.)

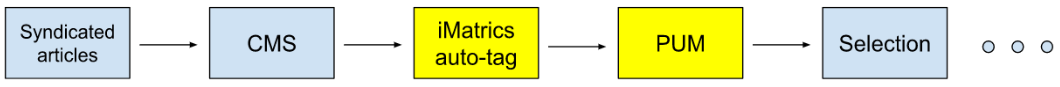
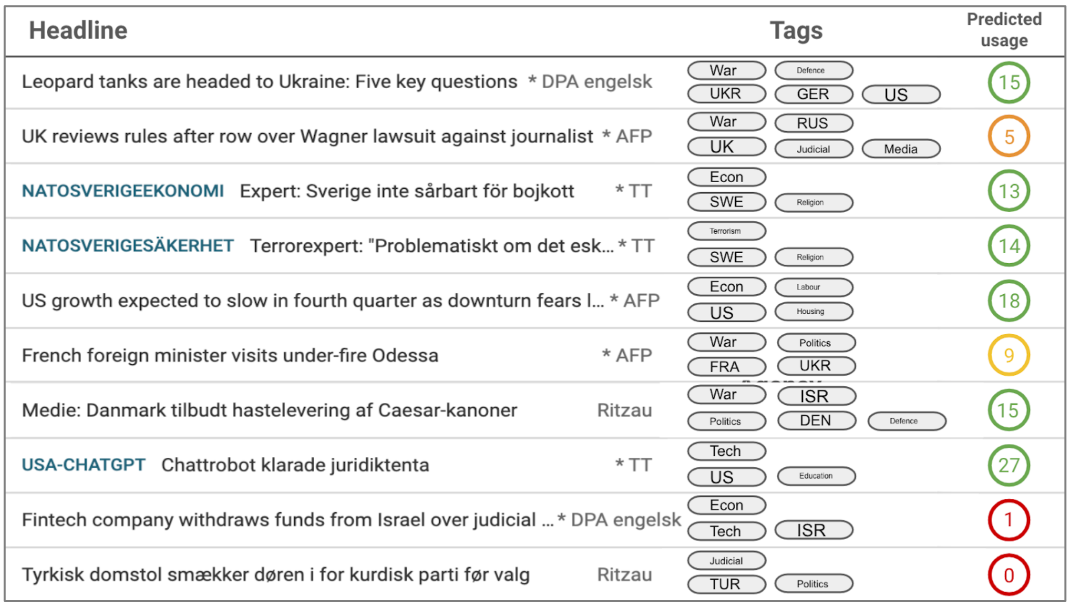


Figure 11.

The iMatrics auto-tagging feature is language independent (*Solutions | IMatrics*, n.d.), and it is already integrated in the CMS, increasing the possibility of a viable implementation. In operation, PUM assigns a usage number to each syndicated article. Figure 12 shows what the ML integration might look like when deployed in production.



Headline	Tags	Predicted usage
Leopard tanks are headed to Ukraine: Five key questions * DPA engelsk	War, Defence, UKR, GER, US	15
UK reviews rules after row over Wagner lawsuit against journalist * AFP	War, RUS, UK, Judicial, Media	5
NATOSVERIGEEKONOMI Expert: Sverige inte sårbart för bojkott * TT	Econ, SWE, Religion	13
NATOSVERIGESÄKERHET Terrorexpert: "Problematiskt om det esk... * TT	Terrorism, SWE, Religion	14
US growth expected to slow in fourth quarter as downturn fears l... * AFP	Econ, Labour, US, Housing	18
French foreign minister visits under-fire Odessa * AFP	War, Politics, FRA, UKR	9
Medie: Danmark tilbudt hastelevering af Caesar-kanoner Ritzau	War, ISR, Politics, DEN, Defence	15
USA-CHATGPT Chattrobot klarade juridiktenta * TT	Tech, US, Education	27
Fintech company withdraws funds from Israel over judicial ... * DPA engelsk	Econ, Tech, ISR	1
Tyrkisk domstol smækker døren i for kurdisk parti før valg Ritzau	Judicial, TUR, Politics	0

Figure 12. Illustration of NTB's CMS with ML integration.

One of the strengths of this approach, is that it requires no end user training or action. PUM operates independently of such factors.

Deployment considerations

Obviously, the model must go through a more rigorous evaluation process before NTB can take the risk of deployment. The code needs to be cleaned, and the model must be tested in a real-life environment. The latter can probably be done semi-manually, cost-efficient and in parallel with the existing pipeline. Both Sourcefabric, the CMS supplier, and iMatrics, the developer of the auto-tagging integration, must be consulted about the viability of the proposed solution. Does the system scale, or will auto-tagging cause latency when applied to a greater volume of articles?

Model deployment is the process of integrating a model into an existing production environment to make practical business decisions based on data. If and when NTB decides to deploy, a comprehensive framework for model governance should be in place. Complete and thorough documentation of all systems and processes is crucial. On the technical side, the predictions must be made available for the production system through an API-layer. In addition, we need to monitor the data quality and the model for skew and drifts as well as build pipelines for deployment and versioning. Both AWS and Google Cloud have managed services to handle ML ops (Sage Maker and Vertex AI), including managing ML Ops in one place and deploying the model for inferences in another. The decision concerning what platform to use should be taken by top management together with the data science and developer teams who will manage the operations when in production.

The world around us is constantly changing, and so is the news. The ideal model would therefore be dynamic, meaning continuously updated, or at least automatically retrained with fresh data in fixed periodic intervals. Data that goes far back in time will probably not be very useful for predicting future usage. If the model is not fed fresh data, its performance will most probably quickly degrade.

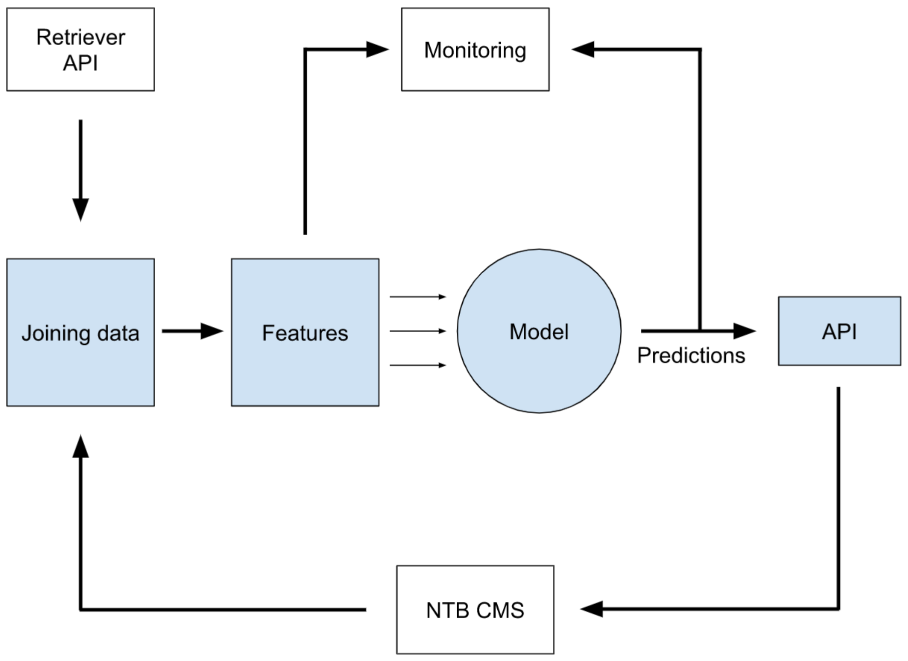


Figure 13. ML process flow.

Conclusion

This research has demonstrated how a machine learning model can use existing data to predict the number of NTB customers that will publish any given foreign news story, thus equipping the duty editor with a powerful new tool. Much of the raw data is already available to the duty editors, but extracting insight from it is difficult and time-consuming, if not impossible. In a streamlined, non-intrusive way, the Predicted Usage Model manifests the business' strategy of becoming more data-driven. Processing data yields information, which in turn can give relevant insight that should lead to meaningful actions with measurable impact. PUM encompasses it all.

The paper outlines how the model can be operationalized within the existing system architecture. The implementation of PUM into the workflow would entail a significant shift from a purely people-dependent process to a more robust and consistent, system-dependent process.

Furthermore, we have suggested how the model can be improved through future iterations. Since our project team of non-experts were able to beat the benchmark model using predominantly the automatic features of BQML, it's fair to assume that a proper data scientist will find ways to squeeze more predictive juice out of the data.

Due to the intrinsically unpredictable nature of the news, the media ecosystem and the people in it, no model predicting article usage will ever be 100% correct. Or even 80 %. But if our model can pick up only small signals in the data, and the duty editors can act upon them, the benefits to NTB can be substantial: greater service consistency, increased article usage, higher customer satisfaction and greater customer retention.

As noted earlier, the model is vulnerable to biases that might already exist in the data. This is an issue that needs to be addressed. On the upside, machine-based systems can be tested and improved, whereas “it is a lot harder to get humans to acknowledge their biases (…) let alone do the hard work required to overcome them” (McAfee, 2017, p. 53). The ability to learn, to capture dynamic changes over time, is one of the model’s core strengths. It takes all the available data as input, and its outputs – the predictions – are pure products of math.

Based on the model’s performance, its potential business impact, and the seemingly natural path towards production, we recommend that NTB as a first step assemble a cross-functional team to

1. Evaluate the model
2. Define metrics for success
3. Estimate the financial benefits
4. Assess the cost of integrating PUM in the existing workflow

The endgame is not to replace the foreign duty editors, but to augment their capabilities.

References

Brynjolfsson, E., & Mcafee, A. (2017). *Machine, Platform, Crowd*. WW Norton.

Fawcett, T., & Provost, F. (2013). *Data Science for Business*. O'Reilly.

Lilleby, A. M. (2019, June 24). *Disse skal være «supersommervikarer» i landets Amedia-aviser: – Vi gir dem et krasjkurs*. Medier24. Retrieved April 4, 2023, from https://m24.no/amedia-kurs-sommervikarer/disse-skal-vaere-supersommervikarer-i-landets-amedia-aviser--vi-gir-dem-et-krasjkurs/221961

Simonsen, E. (2023, March 15). *BT gjør endringer for publisering: – En revolusjon*. Medier24. Retrieved April 4, 2023, from https://m24.no/bergen-bergens-tidende-bt/bt-gjor-endringer-for-publisering-en-revolusjon-1/592938

Solutions. (n.d.). iMatrics. Retrieved April 12, 2023, from https://imatrics.com/solutions/

Appendix

1. *Data sources*

The dataset has been built by joining raw data from two data sources. Metadata from NTB as API in xml and json format and usage data from Retriever API available as json.

NTB:

```
/ntbWeb/api/x1/search/full?search.service=news&search.subcategory=
Nyheter
Limitations: can fetch 10.000 rows max per call, 30-50.000 total
```

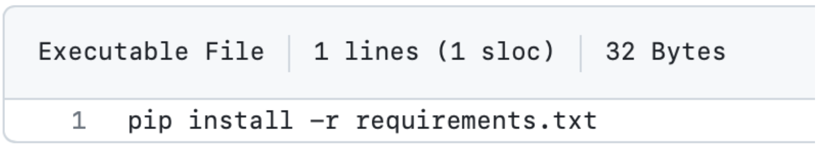
Retriever:

```
/retriever/api/?from=2022-09-01&to=2022-09-25&summary=true
```

2. *Analytics steps/Data analytics minutes*

2.1 Data_extraction

2.1.1 Install_requirements.sh



```
Executable File | 1 lines (1 sloc) | 32 Bytes
pip install -r requirements.txt
```

2.1.2 Ntb_extracter.py

```python
import os
import requests
import json

import datetime
from datetime import timedelta
from google.cloud import storage

BUCKET_NAME = 'ntb-usage'
NTB_ENCODING = 'utf8'
SHOWNUMRESULTS = 5000 # server will crash at around 10.000


def fetch_to_gcs(url, payload, filename):
    filename, content_str = getContentFromURL(url, payload,
filename)
    upload_blob_from_memory(BUCKET_NAME,
                            content_str,
                            destination_blob_name=f"ntb/{filena
me}")


def getContentFromURL(url, payload, filename,
url_get=requests.get):
    # TODO for NTB: better authentisering than Basic htpp
    response = url_get(url,
                       params=payload,
                       auth=(os.getenv('NTB_USR'),
os.getenv('NTB_PWD')))

    # want to stop here, expect 401 if env not setup
    response.raise_for_status()
    print(
        f"http status code: {response.status_code}, encoding:
{response.encoding}"
    )

    decoded_content = response.content.decode(NTB_ENCODING)
    content_str = fromJsonToJsonl(decoded_content)
    filename += 'l'
    return filename, content_str


def fromJsonToJsonl(data):
    docs = extractJsonDocs(data)

    return '\n'.join([json_dump(doc) for doc in docs])

def extractJsonDocs(data):
    records = json.loads(data)
    numResults = records['result']['numResults']
    docs = records['result']['documents']

    # Checking if number of results is not larger than the
requested number
    # default appears to be quite low
    if numResults > len(docs):
        raise Exception('More results than fetched, you are
missing data',
        ' Max: '+ len(docs), ' actual: '+numResults)

    print(f'Number of news: {numResults}')

    # fix date format and nested arrays
    for doc in docs:
        # format in = "2022-02-28T23:55:59.000+0100"
        # weak management of date not adjusting to UTC, will be
UTC in BQ
        time_s = doc["time"]
        doc["time"] = time_s.replace('T', ' ')[:-9]

        keys = ['regions', 'districts', 'subjects',
'mediaList']
        for key in keys:
            if key in doc: doc[key] = doc[key][0]

    return docs


def json_dump(record):
    return json.dumps(
        record,
ensure_ascii=False).encode(NTB_ENCODING).decode(NTB_ENCODING)


# Right out from the doc, should check
# reasumable upload:
https://cloud.google.com/storage/docs/performing-resumable-
uploads
# and streaming:
https://cloud.google.com/storage/docs/streaming#storage-stream-
upload-object-python
def upload_blob_from_memory(bucket_name, contents,
destination_blob_name):

    storage_client = storage.Client(
    )  #Alt: storage.Client(project="my-project-id")
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(destination_blob_name)

    try:
        blob.content_encoding = NTB_ENCODING
        blob.upload_from_string(contents)
    except Exception as e:
        print(f"Failed to upload to GCS, exception is {e}")

    print(f"{destination_blob_name} uploaded to
{bucket_name}.")
```

```
fetch = fetch_to_gcs


# Split from_date to to_date into reasonable number of days
defined by batch size
def extract_ntb(from_d, to_d, days_per_batch=10):
    # More than Batch size
    if (to_d - from_d > timedelta(days=days_per_batch)):
        do_fetch(from_d, from_d +
timedelta(days=days_per_batch))
        return extract_ntb(from_d +
timedelta(days=(days_per_batch + 1)), to_d,
                           days_per_batch)
    # Less than or Equal Batch size
    return do_fetch(from_d, to_d)


def do_fetch(from_date, to_date):
    url, payload = build_url(from_date.replace(hour=0,
minute=0, second=0),
                             to_date.replace(hour=23,
minute=59, second=59))
    filename =
f"ntb_{from_date.date().isoformat()}_{to_date.date().isoformat(
)}.json"
    fetch(url, payload, filename)


def build_url(f_datetime, t_datetime):
    payload = {
        'search.service': 'news',
        'search.subcategory': 'Nyheter',
        'search.startDate':
f_datetime.replace(microsecond=0).isoformat(),
        'search.endDate':
t_datetime.replace(microsecond=0).isoformat(),
        'search.showNumResults': SHOWNUMRESULTS
    }

    return "https://nyheter.ntb.no/ntbWeb/api/j1/search/full",
payload


if __name__ == "__main__":
    f_d = datetime.datetime.fromisoformat("2022-12-01")
    t_d = datetime.datetime.fromisoformat("2022-12-02")
    #fetch = fetch_printer
    extract_ntb(f_d, t_d, days_per_batch=3)
```

2.1.3 Retriever_extract.py

```
import urllib.parse
from urllib.parse import urlunparse
from urllib.parse import urlencode
import datetime
from datetime import timedelta

from google.cloud import storage
import requests
import json

# Google Cloud config: check
https://cloud.google.com/python/docs/reference/google-cloud-
core/latest/config
# short version for now: run from a terminal where
# you are logged in, typically  `gcloud auth login`
# Must run (maybe enough) gcloud beta auth application-default
login
# set the right project `gcloud config set project PROJECT_ID`
BUCKET_NAME = 'ntb-usage'
RETRIEVER_ENCODING = 'iso-8859-1'


# just print the URL it would use (for testing)
def fetch_printer(url, filename):
    print("url: ", url)
    print("filename: ", filename)


def fetch_to_gcs(url, filename):
    filename, content_str = getContentFromURL(url, filename)
    upload_blob_from_memory(BUCKET_NAME,
                            content_str,
                            destination_blob_name=f"retriever/{
filename}")


def getContentFromURL(url, filename, url_get=requests.get):
    # should have send the param as a dictionary, from the doc
    #payload = {'key1': 'value1', 'key2': 'value2'}
    #r = requests.get('https://httpbin.org/get',
params=payload)
    response = url_get(url)
    decoded_content =
response.content.decode(RETRIEVER_ENCODING)
    content_str = make_json_newLineDelimited(decoded_content)
    filename += 'l'
    return filename, content_str
```

```
def make_json_newLineDelimited(data):
    records = json.loads(data)
    return '\n'.join([json_dump(record) for record in records])


def json_dump(record):
    return json.dumps(record, ensure_ascii=False).encode(
        RETRIEVER_ENCODING).decode(RETRIEVER_ENCODING)


# Right out from the doc, should check
# reasumable upload:
https://cloud.google.com/storage/docs/performing-resumable-
uploads
# and streaming:
https://cloud.google.com/storage/docs/streaming#storage-stream-
upload-object-python
def upload_blob_from_memory(bucket_name, contents,
destination_blob_name):

    storage_client = storage.Client(
    )  #Alt: storage.Client(project="my-project-id")
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(destination_blob_name)

    try:
        blob.content_encoding = RETRIEVER_ENCODING
        blob.upload_from_string(contents)
    except Exception as e:
        print('Failed to upload to GCS, exception is {e}')

    print(f"{destination_blob_name} uploaded to
{bucket_name}.")


fetch = fetch_to_gcs


# Split from_date to to_date into reasonable number of days
defined by batch size
def extract_retriever(from_d, to_d, days_per_batch=10):
    # More than Batch size
    if (to_d - from_d > timedelta(days=days_per_batch)):
        do_fetch(from_d, from_d +
timedelta(days=days_per_batch))
        return extract_retriever(from_d +
timedelta(days=(days_per_batch + 1)),
                                 to_d, days_per_batch)
    # Less than or Equal Batch size
    return do_fetch(from_d, to_d)
```

```
def do_fetch(from_date, to_date):
    url = build_url(from_date.replace(hour=0, minute=0,
second=0),
                    to_date.replace(hour=23, minute=59,
second=59))
    filename =
f"retriever_{from_date.date().isoformat()}_{to_date.date().isof
ormat()}.json"
    fetch(url, filename)


def build_url(f_datetime, t_datetime):
    q = urlencode({
        'source': 'ntb',
        'from': f_datetime.replace(microsecond=0).isoformat(),
        'to': t_datetime.replace(microsecond=0).isoformat()
    })

    return urlunparse(
        ('https', 'retriever-info.com', 'external/ntb/api', '',
q, ''))


if __name__ == "__main__":
    f_d = datetime.datetime.fromisoformat("2022-08-15")
    t_d = datetime.datetime.fromisoformat("2022-08-31")

    #f_d = datetime.datetime.fromisoformat("2022-08-01")
    #t_d = datetime.datetime.fromisoformat("2022-08-14")
    #fetch = fetch_printer
    extract_retriever(f_d, t_d, days_per_batch=10)
```

2.2 Data_load

2.2.1 Ntb_loader.py

```
from google.cloud import bigquery

# Construct a BigQuery client object.
client = bigquery.Client()

table_id = 'emm-ntb-dom.ntb_news_usage.import_ntb'
schema_path = "../infra/schema_import_ntb.json"
schema = client.schema_from_json(schema_path)
```

```
# beware of inserting duplicate - should be explicit about
write strategy
job_config = bigquery.LoadJobConfig(
    schema=schema,
    source_format=bigquery.SourceFormat.NEWLINE_DELIMITED_JSON
)

uri = "gs://ntb-usage/ntb/ntb_2023-*.jsonl"

load_job = client.load_table_from_uri(
    uri,
    table_id,
    location="europe-north1",  # Must match the destination
dataset location.
    job_config=job_config,

)  # Make an API request.

load_job.result()  # Waits for the job to complete.

destination_table = client.get_table(table_id)

print("Loaded {} rows.".format(destination_table.num_rows))
```

2.2.2 Retriever_loader.py

```
from google.cloud import bigquery

# Construct a BigQuery client object.
client = bigquery.Client()

table_id = 'emm-ntb-dom.ntb_news_usage.import_retriever'
schema_path = "../infra/schema_import_retriever.json"
schema = client.schema_from_json(schema_path)
job_config = bigquery.LoadJobConfig(
    schema=schema,
    source_format=bigquery.SourceFormat.NEWLINE_DELIMITED_JSON
)

uri = "gs://ntb-usage/retriever/retriever_2023-*.jsonl"

load_job = client.load_table_from_uri(
    uri,
    table_id,
    location="europe-north1",  # Must match the destination
dataset location.
    job_config=job_config,
)  # Make an API request.


load_job.result()  # Waits for the job to complete.


destination_table = client.get_table(table_id)

print("Loaded {} rows.".format(destination_table.num_rows))
```

2.3 Data_transformation

2.3.1 Ntb_matchers.py

```
def match_ids(retriever_id: str, ntb_id: str) -> bool:
    """Return True if the shortened retriever ID matches the
shortened NTB ID.
    Arguments:
    retriever_id -- the retriever ID to match
    ntb_id -- the NTB ID to match
    """
    if not retriever_id or not ntb_id:
        raise ValueError("Both inputs must be non-empty
strings.")
    shorten_retriever_id =
remove_trailing_zeroes(keep_after_TB(retriever_id))
    shorten_NTB_id = remove_hyphen(keep_after_NTB(ntb_id))
    return shorten_retriever_id == shorten_NTB_id


def remove_hyphen(a_string: str) -> str:
    """Return the input string with all hyphens removed."""
    return a_string.replace(HYPHEN, "")


def keep_after_NTB(a_ntb_id: str) -> str:
    """Return the substring of the input string after the first
occurrence of 'NTB'."""
    return a_ntb_id.split(NTB_PREFIX, 1)[1]


def remove_trailing_zeroes(retriever_id: str) -> str:
    """Return the input string with the last two characters
removed."""
    return retriever_id[:-RETRIEVER_ID_SUFFIX_LENGTH]


def keep_after_TB(a_retriever_id: str) -> str:
    """Return the substring of the input string after the first
```

```
occurrence of 'TB'."""
    return a_retriever_id.split(TB_PREFIX, 1)[1]
```

2.4 Data_xplore

2.4.1 NTB_notebook.ipynb

Reference SQL syntax from the original job

Use the `jobs.query` method to return the SQL syntax from the job. This can be copied from the output cell below to edit the query now or in the future. Alternatively, you can use this link back to BigQuery to edit the query within the BigQuery user interface.

```
# Running this code will display the query used to generate
your previous job

job = client.get_job('bquxjob_7be5430b_185016f356c') # Job ID
inserted based on the query results selected to explore
print(job.query)

select distinct(district) from `emm-ntb-
dom.ntb_news_usage.import_ntb`
```

Result set loaded from BigQuery job as a DataFrame

Query results are referenced from the Job ID ran from BigQuery and the query does not need to be re-run to explore results. The `to_dataframe` method downloads the results to a Pandas DataFrame by using the BigQuery Storage API.
To edit query syntax, you can do so from the BigQuery SQL editor or in the `Optional:` sections below.

```
# Running this code will read results from your previous job

query_job = client.query("select * from `emm-ntb-
dom.ntb_news_usage.feature_ntb`")
df = query_job.to_dataframe()
df.head()
```

Show descriptive statistics using describe()

Use the `pandas DataFrame.describe()` method to generate descriptive statistics. Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding `NaN` values. You may also use other Python methods to interact with your data.

```
query_job = client.query("select * from `emm-ntb-
dom.ntb_news_usage.feature_ntb_take1`")
df_1 = query_job.to_dataframe()

#print('All - mean usage {0}, MAE {1}'.format(df.usage.mean(),
df.usage.mad()))
print('uriks - mean usage {0}, MAE
{1}'.format(df_1.usage.mean(), df_1.usage.mad()))
```

uriks - mean usage 5.7711691680563275, MAE 2.238605887630529

```
<ipython-input-9-a87b3ee79db7>:5: FutureWarning: The 'mad' method
is deprecated and will be removed in a future version. To compute
the same result, you may do `(df - df.mean()).abs().mean()`.
  print('uriks - mean usage {0}, MAE {1}'.format(df_1.usage.mean(),
df_1.usage.mad()))
```

2.5 Infra

2.5.1 Gitignore at main

```
# from
https://github.com/github/gitignore/blob/main/Terraform.gitignore
# Local .terraform directories
**/.terraform/*

# .tfstate files
*.tfstate
*.tfstate.*

# Crash log files
crash.log
crash.*.log

# Exclude all .tfvars files, which are likely to contain
sensitive data, such as
# password, private keys, and other secrets. These should not be
part of version
# control as they are data points which are potentially sensitive
and subject
# to change depending on the environment.
*.tfvars
*.tfvars.json

# Ignore override files as they are usually used to override
resources locally and so
# are not checked in
```

```
override.tf
override.tf.json
*_override.tf
*_override.tf.json

# Include override files you do wish to add to version control
using negated pattern
# !example_override.tf

# Include tfplan files to ignore the plan output of command:
terraform plan -out=tfplan
# example: *tfplan*

# Ignore CLI configuration files
.terraformrc
terraform.rc

# Ignore .terraform.lock.hcl
.terraform.*
```

2.5.2 Bq.tf

```
resource "google_bigquery_dataset" "ntb_news_usage" {
  project       = local.project_id
  dataset_id    = "ntb_news_usage"
  friendly_name = "Dataset for the NTB news usage"
  description   = "Todo: add description"
  location      = local.project_default_region
  #location          = "EU" # for multi region in EU
  default_table_expiration_ms = 75 * 24 * 3600 * 1000 #
testing: everything is wipped after one month
}

resource "google_bigquery_table" "import_ntb" {
  project    = local.project_id
  dataset_id =
google_bigquery_dataset.ntb_news_usage.dataset_id
  table_id   = "import_ntb"
  deletion_protection=false

  time_partitioning {
    type = "DAY"
  }

  labels = {
    env = "default"
  }
  schema = file("./schema_import_ntb.json")
}

resource "google_bigquery_table" "import_retriever" {
  project    = local.project_id
  dataset_id =
google_bigquery_dataset.ntb_news_usage.dataset_id
  table_id   = "import_retriever"
  deletion_protection=false

  time_partitioning {
    type = "DAY"
  }

  labels = {
    env = "default"
  }

  schema = file("./schema_import_retriever.json")
}

resource "google_bigquery_table" "imported_ntb" {
  project             = local.project_id
  dataset_id          =
google_bigquery_dataset.ntb_news_usage.dataset_id
  table_id            = "imported_ntb"
  description         = "View aggregating NTB-id with details
from the latest version and with normalized NTBid"
  deletion_protection = false
  view {
    query          = templatefile("./imported_ntb_view.sql", {
pj = local.project_id, ds =
google_bigquery_dataset.ntb_news_usage.dataset_id, tbl =
google_bigquery_table.import_ntb.table_id })
    use_legacy_sql = false
  }
}

resource "google_bigquery_table" "imported_retriever" {
  project             = local.project_id
  dataset_id          =
google_bigquery_dataset.ntb_news_usage.dataset_id
  table_id            = "imported_retriever"
  description         = "View exposing feature usable for usage
prediction and with normalized NTBid"
  deletion_protection = false
  view {
    query          =
templatefile("./imported_retriever_view.sql", { pj =
local.project_id, ds =
google_bigquery_dataset.ntb_news_usage.dataset_id, tbl =
google_bigquery_table.import_retriever.table_id })
    use_legacy_sql = false
  }
}

resource "google_bigquery_table" "feature_usage" {
  project             = local.project_id
  dataset_id          =
google_bigquery_dataset.ntb_news_usage.dataset_id
  table_id            = "feature_ntb"
  description         = "View exposing features and usage
target"
  deletion_protection = false

  view {
    query          = templatefile("./feature_usage_view.sql", {
pj = local.project_id, ds =
google_bigquery_dataset.ntb_news_usage.dataset_id, tbl =
google_bigquery_table.import_retriever.table_id })
    use_legacy_sql = false
  }
}

resource "google_bigquery_table" "feature_usage_take1" {
  project             = local.project_id
  dataset_id          =
google_bigquery_dataset.ntb_news_usage.dataset_id
  table_id            = "feature_ntb_take1"
  description         = "trying ML"
  deletion_protection = false

  view {
    query          =
templatefile("./feature_usage_take1_view.sql", { pj =
local.project_id, ds =
google_bigquery_dataset.ntb_news_usage.dataset_id, tbl =
google_bigquery_table.import_retriever.table_id })
    use_legacy_sql = false
  }
}
```

2.5.3 Bucket.tf

```
resource "google_storage_bucket" "ntb_usage" {
     provider       = google-beta
     project        = local.project_id
     name           = "ntb-usage"
     location       = local.project_default_region
     # Remove with prod data if you don't want to loose data when
destroy
     force_destroy = true

     public_access_prevention = "enforced"
}
```

2.5.4 Config.tf

```
locals {
  project_id              = "emm-ntb-dom"
  project_number          = 1055482349569
  project_default_region = "europe-north1"
  project_default_zone = "europe-north1-a"
  gcp_service_list = [
 #  todo: list "XXX.googleapis.com",
   ]
}

resource "google_project_service" "gcp_services" {
  count              = length(local.gcp_service_list)
  project            = local.project_id
  service            = local.gcp_service_list[count.index]
  disable_on_destroy = false
}

# Backend config:
#
https://developer.hashicorp.com/terraform/language/settings/bac
kends/gcs
resource "google_storage_bucket" "tf_state" {
     provider      = google-beta
     project       = local.project_id
     name          = "ntb-usage-tf-state"
     location      = local.project_default_region

     # prevent accidental destroy of terraform state
     lifecycle {
       prevent_destroy = true
     }
     public_access_prevention = "enforced"

     ## as recommanded in the doc to ease rollback
     #versioning {
     #  enabled = true
     #}

     #encryption is enabled by default in Google Cloud
```

```
      # must be configured in AWS
      # Check doc for CMEK, EKM etc
}

terraform {
  required_version = ">= 1.3.2"

  backend "gcs" {
    bucket  = "ntb-usage-tf-state"
    # maybe use per env prefix ntb-usage/dev osv
    prefix  = "ntb-usage"
  }

  required_providers {
    archive = {
      source  = "hashicorp/archive"
      version = "= 2.0.0"
    }

    google = {
      source  = "hashicorp/google"
      version = "= 4.41.0"
    }

    google-beta = {
      source  = "hashicorp/google-beta"
      version = "= 4.41.0"
    }
  }
}
```

2.5.5 Feature_usage_take1_view.sql

```sql
select n.*, r.docdate, r.wc, r.headline, r.usage
from `emm-ntb-dom.ntb_news_usage.imported_ntb` n
left outer join `emm-ntb-dom.ntb_news_usage.imported_retriever` r
on n.NTB_normalized = r.NTB_normalized
where n.category in ("Utenriks")
```

2.5.6 Feature_usage_view.sql

```sql
select n.*, r.docdate, r.wc, r.headline, r.usage
from `emm-ntb-dom.ntb_news_usage.imported_ntb` n
left outer join `emm-ntb-dom.ntb_news_usage.imported_retriever` r
on n.NTB_normalized = r.NTB_normalized
where n.category in ("Utenriks", "Innenriks", "Sport")
```

2.5.7 Imported_ntb_view.sql

```sql
select agg.*, details.ntbVersion, details.keywords,
details.title, details.district, details.content,
details.writerEmail, ARRAY_LENGTH(details.subjects) as
subjects_count, details.category, details.subcategory,
details.districts,
details.priority,  ARRAY_LENGTH(details.mediaList) as
medialist_count, Array(select name from
unnest(details.subjects)) as subject_names

from (
        SELECT max(id) as latest_id, substring(replace(ntbId,
'-', ''), 4) as NTB_normalized, ntbId
        FROM `emm-ntb-dom.ntb_news_usage.import_ntb`
        group by NTB_normalized, ntbId) as agg
     inner join `emm-ntb-dom.ntb_news_usage.import_ntb` as
details
     on agg.latest_id = details.id

order by latest_id
```

2.5.8 Imported_retriever_view.sql

```sql
SELECT  doc_id, substr(doc_id, strpos(doc_id, 'TB')+2, 32) as
NTB_normalized, docdate, wc, (lookalike_print+lookalike_web) as
usage, headline
FROM `emm-ntb-dom.ntb_news_usage.import_retriever`
```

2.5.9 Schema_import_ntb.json

```json
[
  {
    "mode": "required",
    "name": "id",
    "type": "STRING"
  },
  {
    "mode": "required",
    "name": "ntbId",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "ntbVersion",
    "type": "INTEGER"
  },
  {
    "mode": "NULLABLE",
    "name": "time",
    "type": "TIMESTAMP"
  },
  {
    "mode": "NULLABLE",
    "name": "messageType",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "title",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "keywords",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "category",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "byline",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "service",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "priority",
    "type": "INTEGER"
  },
  {
    "mode": "NULLABLE",
    "name": "district",
    "type": "STRING"
  },
  {
    "mode": "REPEATED",
    "name": "districts",
    "type": "STRING"
```

```
},
{
  "mode": "NULLABLE",
  "name": "editorialInfo",
  "type": "STRING"
},
{
  "mode": "NULLABLE",
  "name": "writerEmail",
  "type": "STRING"
},
{
  "mode": "NULLABLE",
  "name": "subcategory",
  "type": "STRING"
},
{
  "mode": "REPEATED",
  "name": "regions",
  "type": "STRING"
},
{
  "mode": "NULLABLE",
  "name": "content",
  "type": "STRING"
},
{
  "mode": "REPEATED",
  "name": "subjects",
  "type": "RECORD",
  "fields": [
    {
      "mode": "NULLABLE",
      "name": "name",
      "type": "STRING"
    },
    {
      "mode": "NULLABLE",
      "name": "reference",
      "type": "STRING"
    }
  ]
},
{
  "mode": "REPEATED",
  "name": "mediaList",
  "type": "RECORD",
  "fields": [
    {
      "mode": "NULLABLE",
      "name": "url",
      "type": "STRING"
      },
      {
        "mode": "NULLABLE",
        "name": "mediaType",
        "type": "STRING"
      },
      {
        "mode": "NULLABLE",
        "name": "scanpixPicture",
        "type": "BOOLEAN"
      },
      {
        "mode": "NULLABLE",
        "name": "mimeType",
        "type": "STRING"
      },
      {
        "mode": "NULLABLE",
        "name": "caption",
        "type": "STRING"
      },
      {
        "mode": "NULLABLE",
        "name": "id",
        "type": "STRING"
      }
    ]
  }
]
```

2.5.10 Schema_import_teriever.json

```
[
  {
    "mode": "required",
    "name": "doc_id",
    "type": "STRING",
    "description": "Match an article from NTB, id includes part
of NTB-id"
  },
  {
    "mode": "NULLABLE",
    "name": "docdate",
    "type": "TIMESTAMP",
    "description": "Unclear witch date it is (retrieval or
NTB)"
  },
  {
    "mode": "NULLABLE",
    "name": "lookalike_print",
    "type": "INTEGER"
  },
  {
    "mode": "NULLABLE",
    "name": "lookalike_web",
    "type": "INTEGER"
  },
  {
    "mode": "NULLABLE",
    "name": "wc",
    "type": "INTEGER"
  },
  {
    "mode": "NULLABLE",
    "name": "imgtext",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "headline",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "intro",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "category",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "story",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "preintro",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "author",
    "type": "STRING"
  },
  {
    "mode": "NULLABLE",
    "name": "subheadline",
    "type": "STRING"
  },
  {
    "fields": [
      {
        "mode": "NULLABLE",
        "name": "url",
        "type": "STRING"
      },
      {
        "mode": "NULLABLE",
        "name": "story",
        "type": "STRING"
      },
      {
        "mode": "NULLABLE",
        "name": "preintro",
        "type": "STRING"
      },
      {
        "mode": "NULLABLE",
        "name": "imgtext",
        "type": "STRING"
      },
      {
        "mode": "NULLABLE",
        "name": "mediatype",
        "type": "STRING"
      },
      {
        "mode": "NULLABLE",
        "name": "headline",
        "type": "STRING"
      },
      {
        "mode": "NULLABLE",
        "name": "intro",
        "type": "STRING"
      },
      {
        "mode": "NULLABLE",
        "name": "docdate",
        "type": "TIMESTAMP"
      },
      {
        "mode": "NULLABLE",
        "name": "source_name",
        "type": "STRING"
      },
      {
        "mode": "NULLABLE",
        "name": "subheadline",
          "type": "STRING"
        },
        {
          "mode": "NULLABLE",
          "name": "doc_id",
          "type": "STRING"
        },
        {
          "mode": "NULLABLE",
          "name": "ntb_story",
          "type": "STRING"
        }
      ],
      "mode": "REPEATED",
      "name": "lookalikes",
      "type": "RECORD"
    }
 ]
```

2.6 Tests

2.6.1 Test_ntb_extract.py

```
import unittest
import json
import datetime
import data_extraction.ntb_extracter

class TestNtbExtract(unittest.TestCase):
    f = datetime.datetime(2022, 12, 5, 21, 10, 36, 1234)
    t = datetime.datetime(2022, 12, 5, 21, 15, 36, 1234)

    def test_payload_for_url(self):
      url, payload =
data_extraction.ntb_extracter.build_url(self.f, self.t)

      self.assertDictEqual(payload, {'search.service': 'news',
'search.subcategory': 'Nyheter', 'search.startDate': '2022-12-
05T21:10:36', 'search.endDate': '2022-12-05T21:15:36'})

    def test_nested_array_to_jsonl(self):
        input = '''
          [
               {"foo":"bar", "answer":42, "subs":[[{"süb":1},
{"sub":2}]], "sabs":[[{"süb":1}, {"sub":2}]] },
               {"foo":"ball", "answer":66,"subs":[[{"sub":1},
{"sub":2}]]}
```

```
          ]'''
        data = json.loads(input)
        for item in data:
            if 'subs' in item: item['subs'] = item['subs'][0]
            if 'sabs' in item: item['sabs'] = item['sabs'][0]
       # print(json.dumps(data, indent=2))


    input_1_doc_json = '''{
    "result": {
        "numResults": 15033,
        "documents": [ {
                "id": 17859418,
                "ntbId": "NTB12ca9da6-0020-4dea-b10d-
84dd2dc0a610",
                "service": "news",
                "title": "Zelenskyj vil ha globalt forbud mot
russiske fly og skip",
                "priority": 4,
                "time": "2022-02-28T23:55:59.000+0100",
                "messageType": null,
                "byline": "NTB-AFP",
                "ntbVersion": "00",
                "content": "what ever",
                "keywords": "ukr zelenskyj forbud",
                "regions": [
                    [
                        "Europa"
                    ]
                ],
                "district": "Europa",
                "districts": [
                    [
                        "Ukraina",
                        "Russland",
                        "Europa"
                    ]
                ],
                "category": "Utenriks",
                "subcategory": "Nyheter",
                "writerEmail": "uvakt@ntb.no",
                "subjects": [
                    [
                        {
                            "reference": "16000000",
                            "name": "Krig/Konflikter"
                        },
                        {
                            "reference": "11000000",
                            "name": "Politikk"
                        }
                    ]
                ]
            }]
          }
      }
'''
    def test_extract_json_fixes_time(self):
      output =
data_extraction.ntb_extracter.extractJsonDocs(self.input_1_doc_js
on)
      # time_s = "2022-02-28T23:55:59.000+0100"
      time_s = output[0]["time"]

      self.assertEqual("2022-02-28 23:55:59", time_s)


    def test_time_iso_jsonl(self):

      output =
data_extraction.ntb_extracter.fromJsonToJsonl(self.input_1_doc_js
on)
      expected = '''{"id": 17859418, "ntbId": "NTB12ca9da6-0020-
4dea-b10d-84dd2dc0a610", "service": "news", "title": "Zelenskyj
vil ha globalt forbud mot russiske fly og skip", "priority": 4,
"time": "2022-02-28 23:55:59", "messageType": null, "byline":
"NTB-AFP", "ntbVersion": "00", "content": "what ever",
"keywords": "ukr zelenskyj forbud", "regions": ["Europa"],
"district": "Europa", "districts": ["Ukraina", "Russland",
"Europa"], "category": "Utenriks", "subcategory": "Nyheter",
"writerEmail": "uvakt@ntb.no", "subjects": [{"reference":
"16000000", "name": "Krig/Konflikter"}, {"reference": "11000000",
"name": "Politikk"}]}'''

      self.assertEqual(expected, output)
```

2.6.2 Test_ntb_matchers.py

```
import unittest

import data_tranformation.ntb_matchers as ntb_matchers

NTB_ID = "NTB-match-123"
RETRIEVER_MATCH = "startTBmatch12300"
RETRIEVER_NO_MATCH = "startTBnot00"


class TestNTB_Matchers(unittest.TestCase):
    def test_match(self):
        isMatch = ntb_matchers.match_ids(RETRIEVER_MATCH,
NTB_ID)
        self.assertTrue(isMatch)

    def test_not_match(self):
        isMatch = ntb_matchers.match_ids(RETRIEVER_NO_MATCH,
NTB_ID)
        self.assertFalse(isMatch)


if __name__ == "__main__":
    print("imported ntb matchers. Test ID is ", NTB_ID)
    unittest.main()
```

2.6.3 Test_retriever_extract.py

```
import unittest
import urllib.parse
from urllib.parse import urlparse
import datetime
import data_extraction.retriever_extract as rf


class TestRetriever_fetcher(unittest.TestCase):
    f = datetime.datetime(2022, 12, 5, 21, 10, 36, 1234)
    t = datetime.datetime(2022, 12, 5, 21, 15, 36, 1234)

    def test_usage_of_fixed_base_URL(self):
        url = rf.build_url(self.f, self.t)
        components = urlparse(url)
        # Willingly hard coded since only one URL as far as I
know
        self.assertEqual('https', components.scheme)
        self.assertEqual('retriever-info.com',
components.hostname)
        self.assertEqual("/external/ntb/api", components.path)

    def test_query_source_is_ntb(self):
        qs = self.extract_query_from_build_url()
        self.assertEqual('ntb', qs['source'][0])

    def test_query_has_from_date_isoformat(self):
        qs = self.extract_query_from_build_url()
        f = qs['from'][0]
        self.assertIsNotNone(f)
        self.assertIsInstance(datetime.datetime.fromisoformat(f
),
                                      datetime.datetime)

    def test_query_has_to_date_isoformat(self):
        qs = self.extract_query_from_build_url()
        t = qs['to'][0]
        self.assertIsNotNone(t)
        self.assertIsInstance(datetime.datetime.fromisoformat(t
),
                                      datetime.datetime)

    # must rewrite to have  f < t and f > t
    def test_query_from_lt_to(self):
        qs = self.extract_query_from_build_url()
        f = qs['from'][0]
        t = qs['to'][0]
        self.assertLess(f, t)

    def extract_query_from_build_url(self):
        url = rf.build_url(self.f, self.t)
        qs = urllib.parse.parse_qs(urlparse(url).query)
        return qs


class TestFromURLtoString(unittest.TestCase):

    class FakeHttpRespnse:

        def __init__(self):
            self.content = bytes('[{"the content": "åøæ"}]',
"iso-8859-1")
            self.encoding = "iso-18859-1"

    def test_iso8859_1(self):
        #a
        fake = self.FakeHttpRespnse()
        #a
        filename, output = rf.getContentFromURL("url",
                                                "file.name",
                                                url_get=lambda
enc: fake)
        #a
        self.assertEqual('{"the content": "åøæ"}', output)


# Testing with some Norwegian letters... but I actually trust
Python string to work :-)
class TestJsonToJsonl(unittest.TestCase):

    def test_empty_array(self):
        input = "[]"
        output = rf.make_json_newLineDelimited(input)
        self.assertEqual("", output)

    def test_one_simple_object(self):
        input = '[{"foo":"bår", "answer":42}]'
        output = rf.make_json_newLineDelimited(input)
        self.assertEqual('{"foo": "bår", "answer": 42}',
output)

    def test_two_simple_objects(self):
        input = '[{"foo":"bar","answer":42},{"foo":"tball",
"answer":66}]'
        output = rf.make_json_newLineDelimited(input)
        self.assertEqual(
            '{"foo": "bar", "answer": 42}\n{"foo": "tball",
"answer": 66}',
            output)

    def test_two_simple_objects_withNewLine(self):
        input = '[\n{"foo":"bar",\n "answer":42},\n\t
{"foo":"tball", "answer":66}]'
        output = rf.make_json_newLineDelimited(input)
        self.assertEqual(
            '{"foo": "bar", "answer": 42}\n{"foo": "tball",
"answer": 66}',
            output)

    def test_one_object_nested(self):
        input = '[{"foo":"bar", "answer":42, "subs":[{"sub":1},
{"sub":2}]}]'
        output = rf.make_json_newLineDelimited(input)
        self.assertEqual(
            '{"foo": "bar", "answer": 42, "subs": [{"sub": 1},
{"sub": 2}]}',
            output)

    def test_two_object_nested(self):
        input = '''
          [
               {"foo":"bar", "answer":42, "subs":[{"süb":1},
{"sub":2}]},
               {"foo":"tball", "answer":66,"subs":[{"sub":1},
{"sub":2}]},
     {"foo":"fighter", "answer":82}
          ]'''

        output = rf.make_json_newLineDelimited(input)

        expected = '''{"foo": "bar", "answer": 42, "subs":
[{"süb": 1}, {"sub": 2}]}
{"foo": "tball", "answer": 66, "subs": [{"sub": 1}, {"sub":
2}]}
```

```
{"foo": "fighter", "answer": 82}'''
        self.assertEqual(expected, output)
```