Optimizing Social Media Customer Service for UK National Rail

Strategy and Performance Management Group Project

Abhik Modi | Eduardo Hernandez | Gaurav Sharma | Saurabh Chakravorty

Group 4

Section 1: Introduction

Section 1.1 Topic Introduction

In the Public Transportation sector, as well as any other business, customers are and should be one of the most vital pillars. The success of businesses based on providing a service is based on being customer-centric and putting customer's feedback and their satisfaction at the forefront of all business strategy.

Customer satisfaction and evaluation of service quality have a strong positive impact on customer's loyalty to the provider and how highly they speak about the provider. For a provider, the ability to retain current customers and in a better case scenario, increase the customer base leads to a higher profitability and revenue growth for the company.

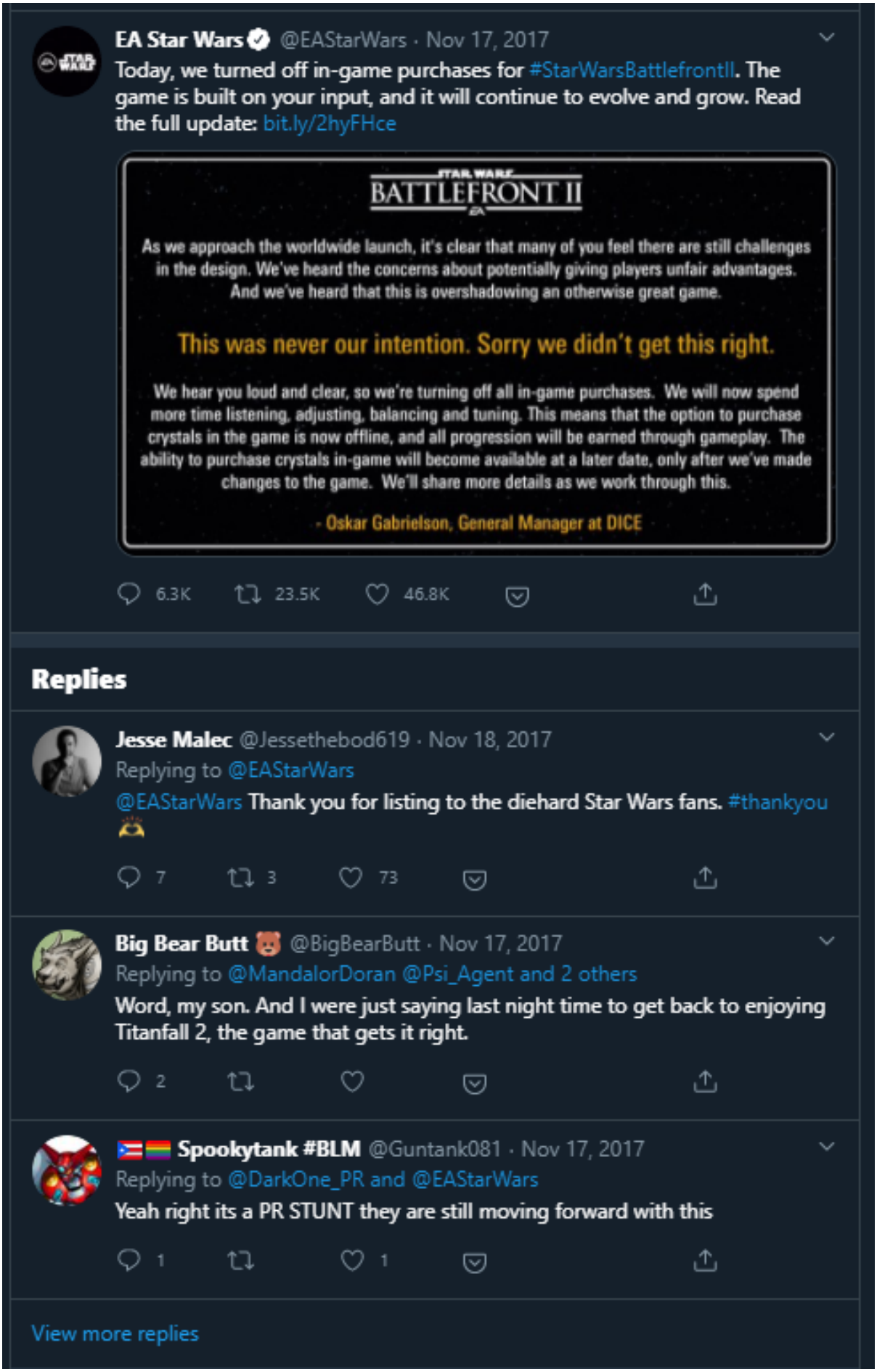
The question that arises now is how to measure and asses customer satisfaction or lack of it dissatisfaction with the company's service quality in real-time. Gone are the days when companies used to do surveys over through pen and paper to get a gist of company brand in the market. In this regard, social media has carved and played a pivotal role in the recent years. Through social media , vast portions of the population can easily praise or bring forth negative comments and feedback of any service, product or company. Since the dissemination of such opinions by the population is public, it may serve as a catalyst and have others come forward and provide their feedback as well. Therefore, social media acts as a platform by which many companies to collect and quantify the public's perception about their company.

With this as the base of our project, we decided to select a B2C organization namely **National Railway** to do an analysis of the customer's perception of the company. The goal of our project is to automate and digitalize the customer service pipeline with the help of Data Science tools and algorithms. Why social media response matters so much can be understood from the following case:

Example

In November of 2007, EA Sports's share prices dropped 8.5% month-to-date, wiping out $3.1billion in shareholder value in the proces. The reason for such a drop was due the backlash from the gaming community towards their then launched game *Star Wars: Battlefornt II*.

The backlash was about an in-game buying feature which gives the gamers who bought an added advantage over other gamers. The social media uproar was so big that not only their share prices dropped but they also had to roll back that feature and also post an apology through their twitter handle.



In the following sections we have selected one company involved in the Service Providing Business and present our case for in-housing an optimized Social Media support to its customers.

Section 1.2 National Rail - About the Company

National Rail is a British association licensed by Rail Delivery Group. This association hands out membership to Train Operating Companies (TOC) for passenger railway services across England, Scotland and Wales.

Some big members of the National Railway:

- Avanti West Coast (replacing Virgin Trains)
- CrossCountry
- Arriva Rail London etc.

There are roughly around 30 TOC members of the National Rail covering the British isles either entirely or in parts. Given this, companies use the rail network of the British Government, a majority of them make net franchise payment to the government and in return the government also provides subsidies to this companies to run their trains on the National Rail network. Also sometimes the government holds interest in including or excluding certain routes in the network between stations/cities and are passed to this companies to accommodate such changes.

We narrowed down to this specific organization because it is the umbrella organization under which we can get a hollistic view of the entire passenger railway service of the UK. Also, this organization undertakes ticket bookings, timetabling of trains, addressing customer queries and other such services which acts like a parent organization to all the private TOCs. Furthermore, given that there are 30 or more so TOC's, it would prove difficult to gather, accumulate and denormalize each of their data from multiple social media platforms into one and that too with multiple time steps.

Section 1.3 Problem Statement

We collected a few datasets related to revenue, number of passengers travelling, private investment, etc. While analyzing the data, we came up with the following insights:

1) The average revenue growth rate over the past three years (2016-2019) is 1.57% which is considerably low in comparison to the average revenue growth rate for the previous three years i.e. 2013-2016 is 4.77%.

National railways: passenger revenue[1], annual from 2000/01

£ Million

	2000/01	2001/02	2002/03	2003/04[3]	2004/05	2005/06	2006/07	2007/08[3]	2008/09	2009/10	2010/11	2011/12	2012/13	2013/14	2014/15	2015/16	2016/17	2017/18	2018/19
Ordinary fares	2,463	2,585	2,693	2,890	3,088	3,323	3,714	4,120	4,443	4,608	4,965	5,447	5,816	6,162	6,649	7,008	7,269	7,584	8,106
Season tickets	950	964	970	1,011	1,071	1,170	1,298	1,434	1,561	1,571	1,654	1,782	1,890	2,041	2,153	2,205	2,171	2,072	2,136
All tickets (current prices)	3,413	3,548	3,663	3,901	4,158	4,493	5,012	5,555	6,004	6,179	6,620	7,229	7,707	8,203	8,803	9,213	9,441	9,655	10,241
All tickets (2018/19 prices)[2]	4,852	4,994	5,031	5,249	5,449	5,737	6,215	6,721	7,073	7,177	7,549	8,136	8,503	8,887	9,417	9,777	9,796	9,827	10,241
% Growth in Revenue		2.92%	0.74%	4.34%	3.80%	5.28%	8.33%	8.15%	5.24%	1.47%	5.18%	7.78%	4.51%	4.52%	5.95%	3.83%	0.19%	0.32%	4.21%
AVG Growth Last Three Years				2.67%			5.81%			4.95%			5.82%			4.77%			1.57%

2) The average occupancy rate for departures from 4 biggest stations (London, Manchester, Liverpool, Birmingham) over a period of last three years has decreased. It is pretty evident from the table below.

Major Cities departures for an average week day by rail for 2019, 2018 and 2017

			2019		2018		2017	
Cities			**Numbers**	**% of Occupancy**	**Numbers**	**% of Occupancy**	**Numbers**	**% of Occupancy**
Birmingham[4]	Passengers		136,884	46%	132,808	48%	124,430	46%
	Total seats		299,138		274,807		273,104	
Liverpool[7]	Passengers		62,698	37%	63,570	37%	60,712	36%
	Total seats		169,415		169,606		167,258	
London[8]	Passengers		1,091,388	41%	1,079,778	42%	1,029,625	44%
	Total seats		2,669,878		2,549,217		2,364,274	
Manchester[9]	Passengers		105,118	40%	102,171	39%	95,100	43%
	Total seats		265,719		261,952		221,657	

3) One of the most important factor for the National Rail is the need for an increase in private investment as the passenger railway network in The Great Britain has been privatized since 1990. We observed that there is a decrease of investment of 20% from 2017/2018 to 2018/2019.

Private investment in the rail industry[1]: annual from 2006/07

£ Million

	2006/07	2007/08	2008/09	2009/10	2010/11	2011/12	2012/13	2013/14	2014/15	2015/16	2016/17	2017/18	2018/19
Track and Signalling	106	8	2	-4	0	1	3	0	1	2	1	6	9
Rolling Stock	326	400	345	423	274	369	352	323	715	622	767	1014	795
Stations	155	78	28	12	28	33	35	29	-128	58	52	72	112
Other Investment	156	79	79	29	74	99	80	72	61	119	105	189	137
Total Investment	743	566	455	460	377	503	470	423	648	801	925	1280	1053
Total investment (2018/19 prices[3])	923	685	536	535	430	567	520	459	694	851	961	1304	1053
% Growth in Revenue		-25.73%	-21.73%	-0.26%	-19.60%	31.72%	-8.29%	-11.64%	51.12%	22.66%	12.89%	35.73%	-19.26%

Finally, we deducted that the National Rail over the last 3 years is not performing well. Also, with Virgin Trains (the then biggest and the most established TOC) going out of service from March of 2020, the Passenger Railway of British is struggling to fill the gap left it.

Some Problems that we found while researching

The Big Read Department for Transport + Add to myFT

Rail: frustration grows with Britain's fragmented network

Privatisation was meant to deliver competition, innovation and improved service, but has it delivered?

Gill Plimmer and **Jonathan Ford** in London JANUARY 29 2018

274

MILLENNIAL RAILCARD: CUSTOMERS CHARGED MORE THAN PRICE OF DISCOUNT PASS WHILE ON HOLD TO NATIONAL RAIL

Calls cost more than the price of a 26-30 railcard

Helen Coffey | @LenniCoffey | Friday 16 March 2018 19:43 | 37 comments

UK railways need 'radical overhaul', campaigners say

28 December 2019

The entire British rail network should be radically overhauled to end the spectre of "nightmare journeys", campaigners have said.

Virgin Trains: Final service departs as UK's longest-running rail franchise ends

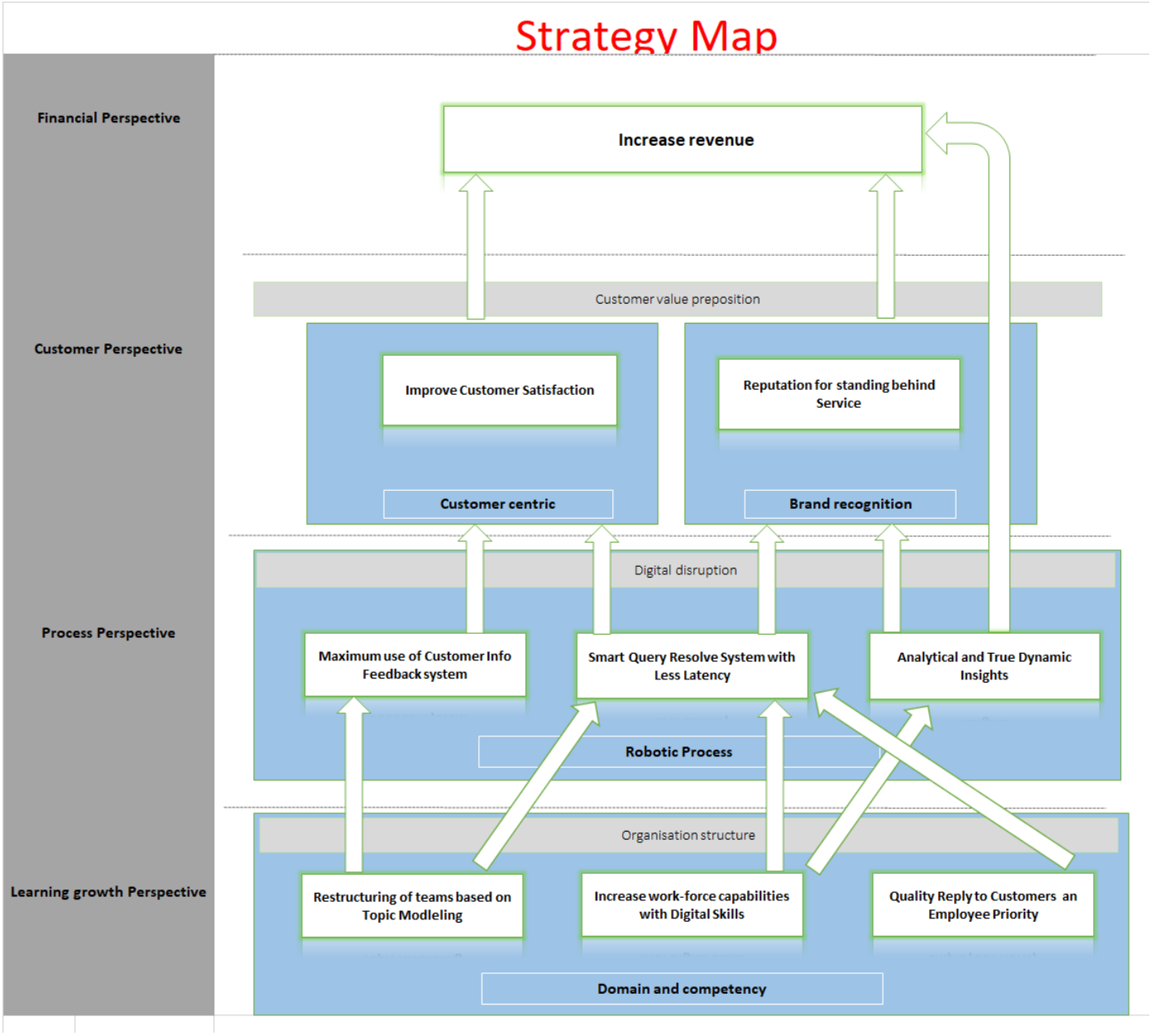
8 December 2019

Britain's longest-running rail franchise came to an end on Saturday after more than 22 years.

In a gist, we realized that the National Rail needs a faceover with a strategy which is more Customer-Oriented and to adapt themselves with more social media savvy customers.

Section 1.4 Strategy Map

We formulated the following strategy for the company which is as mentioned above is Customer Centric. The UK national railway network is one of the oldest railway network and is also well connected. Given this vast network and good connectivity, the customers being not happy with their Railways meant the company needs to understand the problems that the users of the service face and reflect the same in enhancing the company.

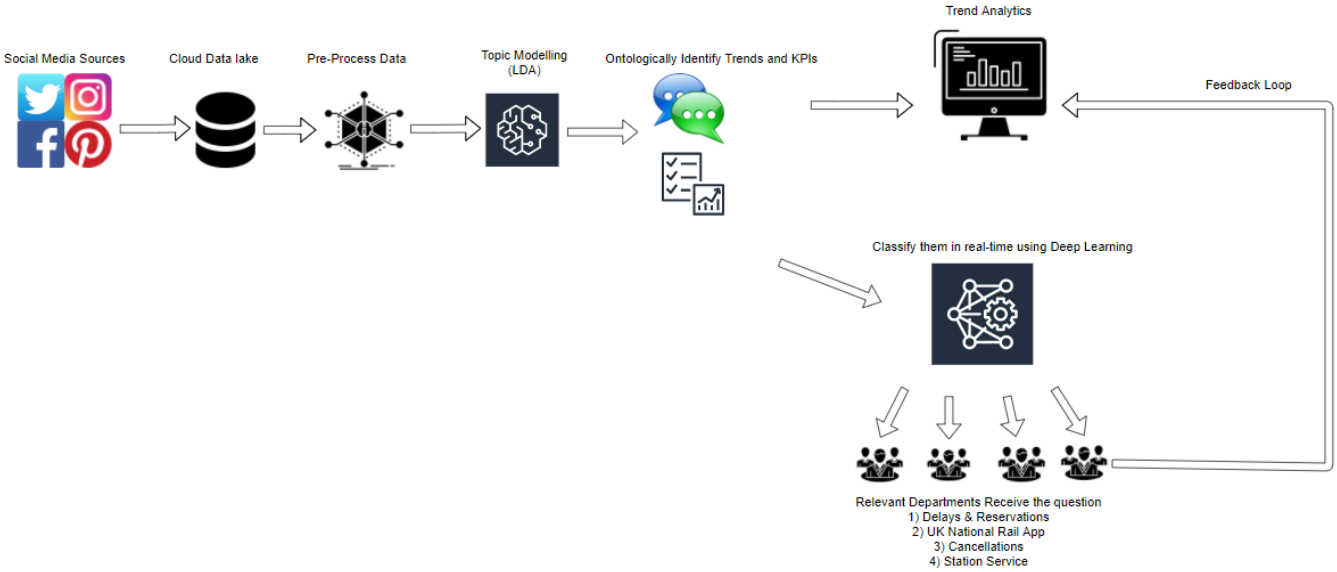


Section 1.5 Action Plan

With the help of Strategy Map, we were able to formulate two streams of improvement,
1) Real-Time quick and quality response to the customer.
2) Trend Analysis over a period of time.

The first stream focuses on addressing any social media query posted by any customer/person as quick as possible by allocating to the relevant department that has the sources to do so.

The second stream focuses on accumulating details and insights over a period of time and developing Key Performance Indicators to get an overview of all the problems that the company faces and analyzing the trend.



In the following sections we create a *Proof of Concept* of a product that implements the Strategy according to the Action Plan.

Section 2: Code and understanding

Section 2.1 Installing and Importing Libraries

Prerequisite

- All code is written in Python 3.9 and shared in .ipynb format
- Please install dependencies in your machine if it does not supports or doesn't have required libraries
- For convenince please use **Jupyter Notebook** (https://jupyter.org/) in your local machine and save the files in same directory
- You can alternatively run in **Google Colab** (https://colab.research.google.com/notebooks/intro.ipynb) as well . If you use Colab then please upload the file "tweet_data.csv" shared with you, in local runtime preferably in folder : '/content/sample_data/'
- A .pdf version of the document is shared with all the cell markdown and the code showing output for convenience of understanding
- **"Patience is virtue"** : The model training takes time please hold on with us

In [1]:

```
!pip install pyLDAvis
!pip install spacy
!pip install wordcloud
!pip install nlp
!pip install tensorflow
!pip install https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.2
```

```
\site-packages (from spacy) (1.0.2)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in c:\users\ehern\onedr
ive\documents\frankfurt_school\winter_2020\strategy\project\vir\lib\site-p
ackages (from spacy) (4.49.0)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in c:\users\ehern\one
drive\documents\frankfurt_school\winter_2020\strategy\project\vir\lib\site
-packages (from spacy) (3.0.2)
Requirement already satisfied: importlib-metadata>=0.20; python_version <
"3.8" in c:\users\ehern\onedrive\documents\frankfurt_school\winter_2020\st
rategy\project\vir\lib\site-packages (from catalogue<1.1.0,>=0.0.7->spacy)
(2.0.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
c:\users\ehern\onedrive\documents\frankfurt_school\winter_2020\strategy\pr
oject\vir\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (1.25.1
0)
Requirement already satisfied: idna<3,>=2.5 in c:\users\ehern\onedrive\doc
uments\frankfurt_school\winter_2020\strategy\project\vir\lib\site-packages
(from requests<3.0.0,>=2.13.0->spacy) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in c:\users\ehern\onedriv
```

Importing Relevant Libraries

In [2]:

```
# Import all libraries
import numpy as np
import pandas as pd
import re
import sys
import csv
import en_core_web_sm
nlp = en_core_web_sm.load()
import pyLDAvis
import pyLDAvis.gensim
import gensim
from gensim import corpora, models
from datetime import datetime, date
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel
from gensim.models import ldamodel
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier,RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score, precision_score, confusion_matrix, recall_score, accu
from pprint import pprint
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import *
from tensorflow.keras.regularizers import L2
from tensorflow.keras.optimizers import *
from tensorflow.keras import Model, Input
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
import pyLDAvis
import pyLDAvis.sklearn
import pyLDAvis.gensim
import matplotlib.pyplot as plt
%matplotlib inline
pd.options.mode.chained_assignment = None
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
import os
```

```
c:\Users\ehern\OneDrive\Documents\frankfurt_school\Winter_2020\Strategy\proj
ect\vir\lib\site-packages\scipy\sparse\sparsetools.py:21: DeprecationWarnin
g: `scipy.sparse.sparsetools` is deprecated!
scipy.sparse.sparsetools is a private module for scipy.sparse, and should no
t be used.
  _deprecated()
```

Settting up working directory. In this case, it is on same system directory

In [3]:

```
###IF you are using Google Colab#####
#Setting working directory
#from google.colab import drive
#os.chdir('/content/sample_data/')
```

Section 2.2: Data Collection and Description

Data collection

The datasets are collected from following sources:

1. Kaggle (https://www.kaggle.com/thoughtvector/customer-support-on-twitter)
2. The data used in this project in the form of a balanced dataset collected from Kaggle named twcs.csv. This is the data which was used for our LDA topic modelling and real-time classifier (the 460 MB .csv file is shared with you)
3. The following piece of code (https://drive.google.com/file/d/1BDzADBb0LSGlP215zOHGHxazJTVPwo52/view?usp=sharing) is used to effectively clean the data and get the important features for our modelling and analysis.
4. Data generated from the code is a .csv file named "tweet_data.csv" which is shared with you
5. Weekly data from twitter (https://twitter.com/nationalraileng) for KPI's
6. In a real-case scenrio where our proposed automation would be implemented, weekly data would be collected from twitter adhering to the policy (https://developer.twitter.com/en/developer-terms/agreement-and-policy) of web scrapping in twitter for our KPI's.
7. The following snippet of code (https://drive.google.com/file/d/1n6sG5VxOPgXrsPmR2ullV711wwUIBCg2/view?usp=sharing) would be used for data gathering with access to twitter developer account and would return data almost identical to the dataset we have used which was collected in 2017. Some column names would be different and we would be missing one column which wasn't used due to the fact that Twitter changed the format of data which is given by its API.

3. All the code shared here for collection and cleaning are in .py format which are used to clean the data and is shared with you in following document with the raw data format(twcs.csv) and clean/pre-processed and labelled format(labelled_tweets.csv)

In [4]:

```
#####************************************######
# For the sake of convenience we use cleaned tweets file after filtering  for national rail
# The original file before filtering is already uploaded in Google drive named as twcs.csv
# For modelling purpose we use tweet_data.csv file
#####************************************######
# read the .csv file
df = pd.read_csv("../data/tweet_data.csv")
df.head()
```

Out[4]:

	Unnamed: 0	Unnamed: 0_x	tweet_id_x	author_id_x	inbound_x	created_at_x	text_x	re
0	621	1634	2344	116242	True	Tue Oct 31 22:37:26 +0000 2017	@nationalrailenq hi why are the trains from Ma...	
1	1272	3365	4563	116783	True	Tue Oct 31 22:46:49 +0000 2017	@nationalrailenq hi all, what's the score with...	
2	1658	4207	5683	117048	True	Tue Oct 31 23:16:51 +0000 2017	@nationalrailenq What's the reason all trains ...	
3	1659	4214	5687	117049	True	Tue Oct 31 23:07:53 +0000 2017	@nationalrailenq how much is a ticket from SOU...	
4	1660	4216	5692	117050	True	Tue Oct 31 23:02:01 +0000 2017	@nationalrailenq any idea whats going on with ...	

The brief decription about these columns are:

- tweet_id_x : Tweet identity number of the user
- author_id_x : The author ID of tweets, important information is the identity of user has been made anonymous
- inbound : It means tweet has come in as a request, it's "inbound" always
- created_at_x: Time stamp of query tweet created
- text_x: The tweet in the form of text message
- response_tweet_id_x: Tweet identity number made with respect to tweet user ID

- in_response_to_tweet_id_x: Number of response made with respect to tweet user ID in the following tweet
- time_x: Time of tweet done
- tweet_id_y: Tweet identity number of user who replied tweet
- author_id_y: Author of respond tweet in this case it's national railways
- inbound_y: "FALSE" as it's not inbound
- created_at_y: Time stamp of query tweet replied
- text_y: Response tweet made in the form of text
- response_tweet_id_y: Tweet identity number made with respect to response of tweet user ID
- in_response_to_tweet_id_y: Number of response made with respect to tweet user ID in the following tweet
- time_y: Time of the response tweet made
- duration : Time taken to reply each tweet

Section 2.3: Preprocessing

Defining all functions which are to be used in data cleaning in NLP task

We will perform the following steps:

- Tokenization: Split the text into sentences and the sentences into words. Lowercase the words and remove punctuation.
- Words that have fewer than 3 characters are removed.
- All stopwords are removed.
- Words are lemmatized — words in third person are changed to first person and verbs in past and future tenses are changed into present.
- Words are stemmed — words are reduced to their root form.

In [5]:

```
# Converting all the time stamp data to proper format
def get_time(string):
    time = datetime.strptime(string, "%a %b %d %H:%M:%S %z %Y").time()
    return time

# Removing stopwords in the corpus
def remove_stopwords(text):
    # We remove stopwords such as "the", "a" or "i" which convey no information
    tokens = []
    # Adding stopword http
    nlp.Defaults.stop_words.add("https")
    nlp.Defaults.stop_words.add("http")
    nlp.Defaults.stop_words.add("https://")
    for token in nlp(text):
        tokens.append(token.text)
    final_sentence = ''
    for word in tokens:
        lexicon = nlp.vocab[word]
        if lexicon.is_stop is False:
            final_sentence += ' '
            final_sentence += word
    return final_sentence



# For simplicity making the word into it's simlest form
def lemmatize(text):
    # We take the lemma of the word converting it to its base in present tense
    text = nlp(text)
    lemmatized_sentence = ','.join([word.lemma_ for word in text])
    return lemmatized_sentence

# Preprocessing for making it uniform
def preprocess(in_text):
    # Remove punctuations and numbers
    out_text = re.sub('[^a-zA-Z]', ' ', in_text)

    # Convert upper case to lower case
    out_text = "".join(list(map(lambda x:x.lower(), out_text)))

    # Remove single character
    out_text = re.sub(r"\s+[a-zA-Z]\s+", ' ', out_text)

    return out_text

# Generating corpus text
def gen(text):
    corpus = []
    for word in gensim.utils.simple_preprocess(text):
        corpus.append(word)
    return corpus
```

In [6]:

```
# Subsetting only for query tweets
data_text = df[['text_x']]
data_text.index = df.index
documents = data_text
```

In [7]:

```
# Processing for NLP
documents['text_x'] = documents['text_x'].apply(remove_stopwords)
documents['text_x'] = documents['text_x'].apply(lemmatize)
documents['text_x'] = documents['text_x'].apply(preprocess)
documents['text_x'] = documents['text_x'].apply(gen)
```

In [8]:

```
# Map the preprocess function to all comments
processed_docs = documents['text_x']
# Show the preprocessed result of the first 10 comments
processed_docs[:10]
```

Out[8]:

```
0    [nationalrailenq, hi, train, manchester, picca...
1    [nationalrailenq, hi, score, liverpool, lime, ...
2    [nationalrailenq, reason, train, lancaster, st...
3    [nationalrailenq, ticket, sou, london, zones, ...
4     [nationalrailenq, idea, go, preston, train, run]
5     [nationalrailenq, idea, go, preston, train, run]
6    [nationalrailenq, good, morning, train, bolton...
7     [nationalrailenq, leeds, south, milford, cancel]
8    [nationalrailenq, alert, set, app, stop, actua...
9    [nationalrailenq, illness, family, unable, pre...
Name: text_x, dtype: object
```

Create the Dictionary

In [9]:

```
# The top 5 words in the different datasets
dictionary = gensim.corpora.Dictionary(processed_docs)
#dictionary_a = gensim.corpora.Dictionary(processed_docs_a)
print('Top 5 words in the comments: ')
count = 0
for k, v in dictionary.iteritems():
    print(k, v)
    count += 1
    if count > 5:
        break
```

```
Top 5 words in the comments:
0 delay
1 hi
2 manchester
3 nationalrailenq
4 piccadilly
5 train
```

In [10]:

```
# Words with appearence less than 20 times or more than 20% of the word count
# Keep the most frequent 100000 words
dictionary.filter_extremes(no_below=20, no_above=0.2, keep_n=100000)
bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
```

In [11]:

```
# Human readable format of corpus (term-frequency)
# x denotes the first x document(s) in the documents
# arbitrary number can be used to get some insights from the documents
def term_frequency(dictionary, bow_corpus, x):
  return [[(dictionary[id], freq) for id, freq in cp] for cp in bow_corpus[:x]]

print('term frequency of the 3rd document in comments: ')
term_frequency(dictionary, bow_corpus,3)
```

```
term frequency of the 3rd document in comments:
```

Out[11]:

```
[[('delay', 1), ('hi', 1), ('manchester', 1), ('piccadilly', 1), ('york',
1)],
 [('hi', 1),
  ('bus', 1),
  ('lime', 1),
  ('liverpool', 1),
  ('open', 1),
  ('street', 1),
  ('sunday', 1),
  ('thank', 1)],
 [('reason', 1)]]
```

Section 2.4 Topic Modelling

Latent Dirichlet Allocation

What is Topic Modelling?

- Topic Modelling (https://link.springer.com/chapter/10.1007/978-3-642-29426-6_15) refers to identifying topics that best describes paragraph of text.
- It's an Unsupervised Machine Learning approach for extracting "**abstract**" set of topics that occur in collection of paragraphs.
- An extensively used model for topic extraction is **LDA(Latent Dirichlet Allocation)**.
- Each topic in an LDA refers to set of words and the aim of the model is to map it into set of relevant topics best suitable for distribution.

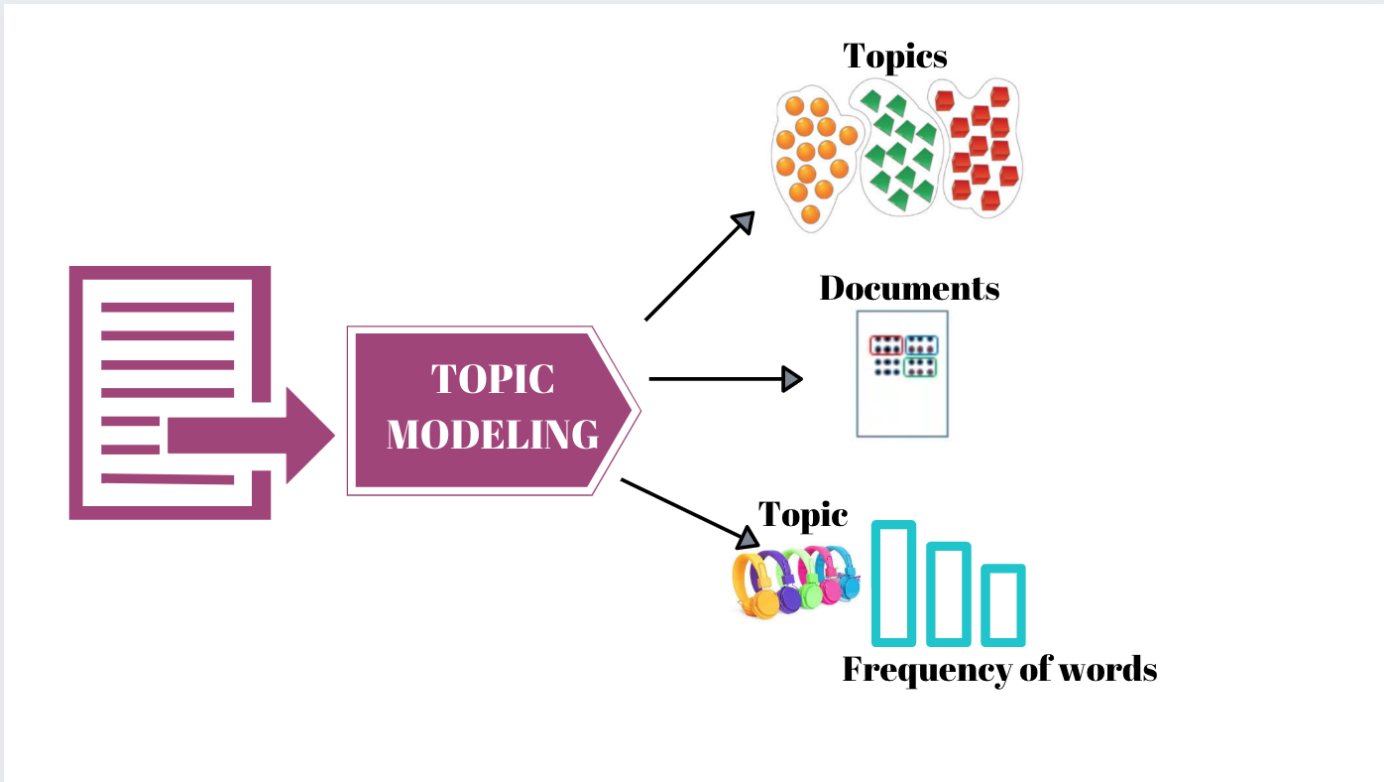


Image Ref:Micro-Medium

- The following cells dwells into the Mathematical Equation and concept of LDA and susbsequently Topic Modelling.

Mathematical understanding

- It is a probabilistic topic modeling method that gives us an approach to figure out possible topics from documents that we do not know of beforehand.
- The key assumptions behind LDA is that each given documents is a mix of multiple topics. Given a set of documents, one can use the LDA framework to learn not only the topic mixture (distribution) that represents each document but also word (distribution) that are associated with each topic to help understand what the topic might be referring to.
- The topic distribution (α) for each document is distributed as

$$\theta \sim Dirichlet(\alpha)$$

- The term (word) distribution on the other hand is also modeled by a Dirichlet distribution, just under a different parameter η.

$$\phi \sim Dirichlet(\eta)$$

- The utmost goal of LDA is to estimate the θ and ϕ which is equivalent to estimate which words are important for which topic and which topics are important for a particular document, respectively.
- The basic idea behind the parameters for the Dirichlet distribution is higher the value the more likely each document is to contain a mixture of most of the topics instead of any single topic. The same goes for η, where higher value denotes that each topic is likely to contain a mixture of most of the words and not any word specifically.

Why Topic Modelling?

- The purpose of doing Topic Modelloing is to allocate customer complaints to the respective department concerning it.
- This we wish to pursue inorder to automate the whole process and let the algorithm decide which customer complaint needs to be allocated what Topic for it to be addressed by the respectative department.
- As currently we are working with the Tweeter Data, our input is a tweet made by a person.

The library used here is Gensim in Python for simplicity whose documentation can be found here (https://radimrehurek.com/gensim/sklearn_api/w2vmodel.html)

We restrict the topic modelling topics to 4 only. Let's check the results using coherence score

In [12]:

```
# LDA model
# build LDA model for comments
lda_model = gensim.models.LdaModel(bow_corpus,
                                   num_topics=4,
                                   id2word=dictionary)

lda_model.print_topics(num_topics=4, num_words=10)
```

Out[12]:

```
[(0,
  '0.041*"delay" + 0.035*"cancel" + 0.030*"time" + 0.029*"travel" + 0.026*"w
ork" + 0.025*"https" + 0.025*"co" + 0.024*"london" + 0.024*"go" + 0.023*"ap
p"'),
 (1,
  '0.039*"ticket" + 0.031*"manchester" + 0.025*"today" + 0.024*"help" + 0.02
3*"return" + 0.021*"run" + 0.020*"know" + 0.020*"thank" + 0.020*"day" + 0.01
9*"euston"'),
 (2,
  '0.073*"ticket" + 0.051*"run" + 0.040*"hi" + 0.029*"london" + 0.022*"cance
l" + 0.022*"bus" + 0.022*"thank" + 0.020*"service" + 0.018*"today" + 0.018
*"day"'),
 (3,
  '0.051*"delay" + 0.043*"https" + 0.043*"co" + 0.034*"station" + 0.032*"ser
vice" + 0.027*"ticket" + 0.023*"website" + 0.023*"time" + 0.020*"euston" +
0.019*"need"')]
```

Let's see the cloud word of these topics visually

In [13]:

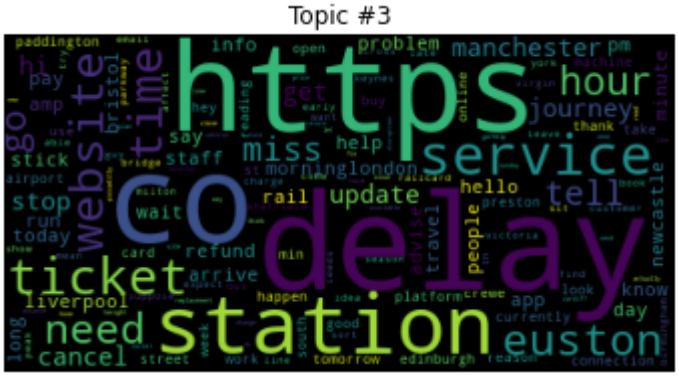
```
# lda is assumed to be the variable holding the LdaModel object
import matplotlib.pyplot as plt
for t in range(lda_model.num_topics):
    plt.figure()
    plt.imshow(WordCloud().fit_words(dict(lda_model.show_topic(t, 200))))
    plt.axis("off")
    plt.title("Topic #" + str(t))
    plt.show()
```

Topic #0

Topic #1

Topic #2

Topic #3



Section 2.5: Validation

In [14]:

```
# Compute Perplexity - Measures the non-cohesiveness of the topics
print(f'Perplexity for comments: {lda_model.log_perplexity(bow_corpus)}')
```

```
Perplexity for comments: -5.156702217629449
```

Section 2.5.1: Gridsearch for the number of topics with the highest Coherence score

In [15]:

```
def compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=2):

    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model = gensim.models.LdaModel(corpus=corpus, num_topics=num_topics, id2word=dictio
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, co
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values
```

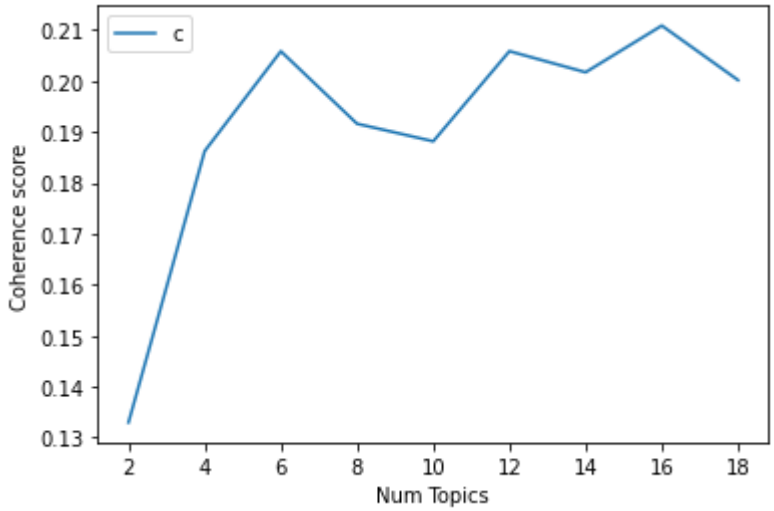
In [16]:

```
model_list, coherence_values = compute_coherence_values(dictionary=dictionary,
                                                        corpus=bow_corpus,
                                                        texts=processed_docs,
                                                        start=2,
                                                        limit=20,
                                                        step=2)
```

In [17]:

```
# Show graph
limit=20; start=2; step=2;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()

# Print the coherence scores
for m, cv in zip(x, coherence_values):
    print("Num Topics =", m, " has Coherence Value of", round(cv, 3))
```




```
Num Topics = 2  has Coherence Value of 0.133
Num Topics = 4  has Coherence Value of 0.186
Num Topics = 6  has Coherence Value of 0.206
Num Topics = 8  has Coherence Value of 0.192
Num Topics = 10  has Coherence Value of 0.188
Num Topics = 12  has Coherence Value of 0.206
Num Topics = 14  has Coherence Value of 0.202
Num Topics = 16  has Coherence Value of 0.211
Num Topics = 18  has Coherence Value of 0.2
```

Observation: We see the coherence score optimum on topic 4 . In terms of making clusters we restrict it to 4 keeping in mind the organisation side as more number of departments would mean more people getting assigned in different groups

In [18]:

```
# Select the model and print the topics
optimal_model = model_list[2]
model_topics = optimal_model.show_topics(formatted=False)
pprint(optimal_model.print_topics(num_words=10))
```

```
[(0,
  '0.072*"ticket" + 0.053*"go" + 0.035*"time" + 0.034*"cancel" + '
  '0.030*"station" + 0.022*"delay" + 0.021*"co" + 0.021*"https" + 0.019*"ru
n" '
  '+ 0.018*"manchester"'),
 (1,
  '0.060*"ticket" + 0.042*"travel" + 0.038*"hi" + 0.034*"thank" + 0.030*"tim
e" '
  '+ 0.026*"day" + 0.023*"say" + 0.022*"buy" + 0.021*"use" + 0.020*"peak"'),
 (2,
  '0.055*"delay" + 0.038*"co" + 0.038*"https" + 0.031*"hi" + 0.026*"service"
+ '
  '0.025*"cancel" + 0.019*"london" + 0.018*"today" + 0.017*"bus" + '
  '0.016*"hour"'),
 (3,
  '0.058*"run" + 0.058*"ticket" + 0.045*"london" + 0.037*"euston" + '
  '0.028*"need" + 0.025*"delay" + 0.024*"https" + 0.024*"co" + 0.021*"statio
n" '
  '+ 0.020*"day"'),
 (4,
  '0.061*"service" + 0.047*"run" + 0.045*"today" + 0.036*"app" + '
  '0.027*"street" + 0.023*"hi" + 0.023*"tomorrow" + 0.020*"return" + '
  '0.019*"liverpool" + 0.016*"replacement"'),
 (5,
  '0.046*"station" + 0.033*"help" + 0.033*"liverpool" + 0.025*"railcard" + '
  '0.023*"thank" + 0.022*"card" + 0.021*"happen" + 0.021*"hi" + 0.021*"work"
+ '
  '0.020*"stop"')]
```

Section 2.5.2: Visualization of LDA Results

In [19]:

```
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, bow_corpus, dictionary)
vis
```

Out[19]:

Following observation can be made from the topic modelling LDA:

- The number of clusters formed are 4 without overlapping so the nature of complaints are orhtogonal to each other.
- We can assign the customer complaints to different departments with less overlapping of conflicts. This leads to assigning each tweet to one and only one department, in turn reducing the redundancy.
- The coherence score is low so the query relates high to the topic modelling cluster created.

Section 2.6: Supervised Learning for Classification

- We utilize the Supervised Learning approach to classify each tweet to one of the Topic that were extracted after conducting LDA. This is done to assign the tweet to that Department which addresses that Topic of query
- In the following code training of a Supervised Alogorithm based model is done.

In [20]:

```
# Getting label from LDA
documents = documents.reset_index(drop=True)
df = df.reset_index(drop=True)
documents['label'] = 0
for i in range(len(documents)):
    doc = lda_model.id2word.doc2bow(documents['text_x'][i])
    t   = lda_model[doc]
    documents['label'][i] = max(t,key=lambda item:item[1])[0]

# Assigning labels to the data
df['label'] = documents['label']
```

Section 2.6.1 Classification based on CART methods

Using vectorization process here to generate features in vector space

In [21]:

```
vectorizer = TfidfVectorizer(max_features=2500,min_df=7,max_df=0.8)
# Train and test set created
X = vectorizer.fit_transform(df['text_x'].values).toarray()
y = df['label']
```

In [22]:

```
# Lets build a classification model with random forest and gradient boosting
class classification:

  def __init__(self, X, y):

    self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X, y, test_size
    self.labels = np.unique(y).tolist()

  def calc_metrics_class(self):

    precision = precision_score(self.pred, self.y_test, average='weighted')
    recall = recall_score(self.pred,self.y_test, average='weighted')
    f1 = f1_score(self.pred,self.y_test, average='weighted')
    accuracy = accuracy_score(self.pred,self.y_test)
    print("The precision for the model is :", precision, '\n', "The recall for the model is
          "The accuracy of the model is : ", accuracy, '\n')
    #self.confusion_matrix()

  def gradient_boost(self):
      print("Performing modelling for Gradient boost")
      GradBoostClasCV = GradientBoostingClassifier(random_state=42)
      model_params = {
                        "max_depth": [10],
                        "subsample": [ 0.6],
                        "n_estimators":[10],
                        "learning_rate": [0.01],
                        "criterion": ['mae']
                      }
      grid_model = GridSearchCV(estimator=GradBoostClasCV, param_grid=model_params, cv=5, n
      grid_model.fit(self.X_train,self.y_train)
      print("Best parameters =", grid_model.best_params_)
      model_clf = GradBoostClasCV.set_params(**grid_model.best_params_)
      model_clf.fit(self.X_train, self.y_train)
      self.pred = model_clf.predict(self.X_test)
      self.calc_metrics_class()

  def random_forest(self):
      print("Performing modelling for Random forest")
      rf_model = RandomForestClassifier(random_state=42)
      param_grid = {
                    'n_estimators': [30],
                    'max_features': [0.9],
                    'min_samples_split': [3]
                    }
      grid_model = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, n_jobs=-1)
      grid_model.fit(self.X_train,self.y_train)
      print("Best parameters =", grid_model.best_params_)
      model_clf = rf_model.set_params(**grid_model.best_params_)
      model_clf.fit(self.X_train, self.y_train)
      self.pred = model_clf.predict(self.X_test)
      self.calc_metrics_class()

p1 = classification(X,y)
```

In [23]:

```
# Let's check modelling with R.F
p1.random_forest()
```

```
Performing modelling for Random forest
Best parameters = {'max_features': 0.9, 'min_samples_split': 3, 'n_estimator
s': 30}
The precision for the model is : 0.6929305428346255
 The recall for the model is :  0.654639175257732
 The f1 score of the model is : 0.6638427805041778
 The accuracy of the model is :  0.654639175257732
```

In [24]:

```
# Let's check modelling with G.B it takes long time to run so please have some patience whi
p1.gradient_boost()
```

```
Performing modelling for Gradient boost
Best parameters = {'criterion': 'mae', 'learning_rate': 0.01, 'max_depth': 1
0, 'n_estimators': 10, 'subsample': 0.6}
The precision for the model is : 0.8755357809950125
 The recall for the model is :  0.42783505154639173
 The f1 score of the model is : 0.5390884968862942
 The accuracy of the model is :  0.42783505154639173
```

Section 2.6.2 Classification based on deep learning

What we see in this is the accuracy is quite low so we now move to world of deep learning use a neural network approach

We shall be using TensorFlow, a library designed for Deep Learning by Google and has extremelly efficient matrix operations which deep learning is based on. It works extremelly well with Python and suits the purpose of designing a real-time classifier for our extracted topics.

In [25]:

```
# We import the necessary libraries.
```

The following cell block will tokenize our questions data, meaning that it will created a list for every sentence which contains each word in a separated format. Additionally, we also create the word_index which is a Python dictionary that maps each word in our text to its respective number so that it can be identified numerically.

In [26]:

```
tokenizer = Tokenizer(oov_token='<OOV>')
tokenizer.fit_on_texts(df['text_x'])
word_index = tokenizer.word_index # We will need the index to create the embedding matrix
corpus_size = len(word_index) + 1 # We need the size of the corpus for training parameters
```

As computers do not understand natural text, we must first convert each and every word within our data into its numerical form. The tokenizer does this by merely creating a sequence for every sentence with its respective number in its place. This number comes from having fit it on the text in our previous cell block and can be found

next to its respective word in the word index. We also convert our labels to a one-hot encoded matrix which is merely a matrix that conerts each label to a vector the size of how many labels there are. For example, if there are 10 labels, then you will get a vector of size 10. There will be 9 zeros and 1 one. This must be done because TensorFlow needs to have the labels in matrix and numerical form.

In [27]:

```
sentences = tokenizer.texts_to_sequences(df['text_x']) # Tokenizes text
# We now pad the text to make them the same size as TF requires it.
padded_sentences = pad_sequences(sentences, maxlen=100, padding='post')
labels = tf.keras.utils.to_categorical(df['label']) # We one-hot encode our y
```

Recently, Deep Learning has made progress by using what is called word embeddings. The idea is that words can be converted to a vector of certain dimensions (200 in our case) which will hold numerical values which represent its projection onto a latent space. This projection would see words such as dog and puppy close to one another and allow deep learning models to have a better representation and understanding of words. This better understanding allows the model to better classify the text based on its syntactical structure. The following code will extract word embeddings from the GloVe repository for us and create an embedding matrix for the words within our texts. However, you must make sure that the glove embeddings are in the appropriate folder to use it.

Note : We will use 'glove.twitter.27B.200d.txt' for embedddings so please upload the file in the working directory. The size of file is quite large around 2 G.B as it contains embedding information for classification

In [29]:

```
embeddings_index = {}
with open('../data/glove.twitter.27B.200d.txt', encoding='utf-8') as file:
    for line in file:
        embedding = line.split()
        word = ''.join(embedding[:-200])
        try:
            coefficients = np.asarray(embedding[-200:], dtype='float32')
            embeddings_index[word] = coefficients
        except:
            embeddings_index[word] = 0

embedding_matrix = np.zeros((len(word_index) + 1, 200))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

After an iterative scheme of working with multiple different parameters and model architectures, we found the best model to consist of, in order:

1) The embeddings which were extracted.
2) A special case of a Recurrent Neural Network that works really well with long text sequences due to its inner memory state, which is called a bidirectional Long-Short Term Memory (LSTM) layer.
3) A fully connected neural layer.
4) The final layer is a simple softmax layer which allows the model to generate probability distributions for each class and then performing an argmax function so as to choose the most likely class the input sequence belongs to.

In [36]:

```
input_ = Input(shape=(100,))
embed = Embedding(input_dim=corpus_size,
                  output_dim=200,
                  weights=[embedding_matrix],
                  trainable=True,
                  mask_zero=True,
                  name='Embeddings')(input_)
logits_1 = Bidirectional(LSTM(units=194,
                              kernel_regularizer=L2(0.001),
                              recurrent_regularizer=L2(0.001),
                              recurrent_dropout=0.6,
                              dropout=0.6,
                              return_sequences=False))(embed)
logits_3 = (Dense(units=124,
                  activation='relu',
                  kernel_initializer='he_uniform',
                  kernel_regularizer=L2(0.01)))(logits_1)
dropout = Dropout(.4)(logits_3)
output = Dense(units=4, activation='softmax')(dropout)

model = Model(inputs=input_, outputs=output)
optimizer = Adam(learning_rate=0.002, beta_1=0.9, beta_2=0.99, decay=(0.002/3))
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
Embeddings (Embedding)       (None, 100, 200)          840000
_________________________________________________________________
bidirectional_1 (Bidirection (None, 388)               613040
_________________________________________________________________
dense_2 (Dense)              (None, 124)               48236
_________________________________________________________________
dropout_1 (Dropout)          (None, 124)               0
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 500
=================================================================
Total params: 1,501,776
Trainable params: 1,501,776
Non-trainable params: 0
_________________________________________________________________
```

We now split the data into training and validation sets so that we can train the model and see its accuracy on data that it is not trained on.

In [37]:

```
X_train, X_val, y_train, y_val = train_test_split(padded_sentences, labels,
                                                  test_size=0.20, shuffle=True,
                                                  random_state=3)
```

We now fit the model training data and tell it and run iteratively for 10 epochs. It shall output accuracy on both the training data at the end of each run as well as on the data it hasn't been trained on (validation data) so that

we can see the model's genralizability

Please use a Colab GPU for training

It is faster

.from keras.utils import multi_gpu_model from tensorflow.python.client import device_lib import tensorflow as tf print(device_lib.list_local_devices()) with tf.device('/gpu:0'): model.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))"

In [38]:

```
# Alternatively if not using colab can use this too
model.fit(X_train, y_train, epochs=3, validation_data=(X_val, y_val))
```

```
Epoch 1/3
49/49 [==============================] - 46s 947ms/step - loss: 3.3432 - acc
uracy: 0.3426 - val_loss: 2.1435 - val_accuracy: 0.4459
Epoch 2/3
49/49 [==============================] - 49s 992ms/step - loss: 1.7677 - acc
uracy: 0.4923 - val_loss: 1.5828 - val_accuracy: 0.5052
Epoch 3/3
49/49 [==============================] - 49s 1s/step - loss: 1.3362 - accura
cy: 0.5935 - val_loss: 1.5689 - val_accuracy: 0.4459
```

Out[38]:

```
<tensorflow.python.keras.callbacks.History at 0x2940b107c08>
```

This line of code merely outputs the final accuracy and loss metric the model acheives on the validation data.

In [39]:

```
loss, accuracy = model.evaluate(X_val, y_val)
```

```
13/13 [==============================] - 2s 131ms/step - loss: 1.5689 - accu
racy: 0.4459
```

Observation We acheive a good accuracy by doing deep learning methods.

Now we can predict constantly textual context from the tweets and assign into different customer compalaint departments for getting the quey resolved

For the purpose of latest analysis we use the twitter data which has been recently scrapped named under file "labelled_tweets.csv"

Section 3: Key Performance Indicators

The second stream which dealt with Trend Analysis, for that we came up certain Key Performance Indicators which will form the KRA's for our departments. The company might also have to monitor over a period of time these KPI's and accordingly change and modify the metrics and initiatives of Strategy Map.

A Tableau File "Tableau_File.twbx" is made for creating visualisation graphs. Please refer to the shared file Here we have inserted snapshots from the Tableau file and explained each graph in detail.

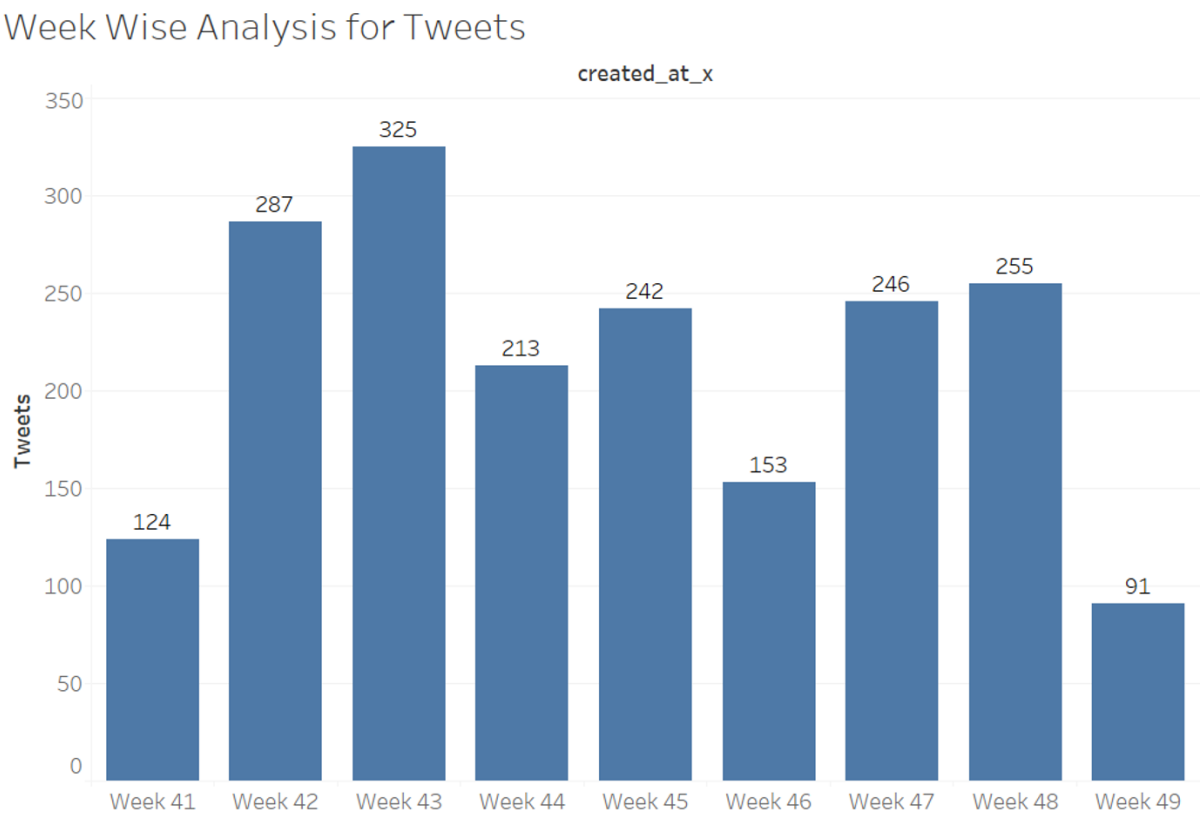
We name the four topics formed after clustering are:

1. UK National Application
2. Reservations
3. Delay in services
4. Stations

- On all these clusters formed we perform an extensive analysis on the results. This forms a base of our customer centric strategy which could help the organisation like UK National Railway resolve customer complaints in much more effective way.We also gain strategic insights after these analysis which could help them figure out key metrics for analytical decison making process in their process chain.
- For measuring complaints we do analysis by measuring number of tweet queries i.e number of tweets per day.

The key performance indicators are:

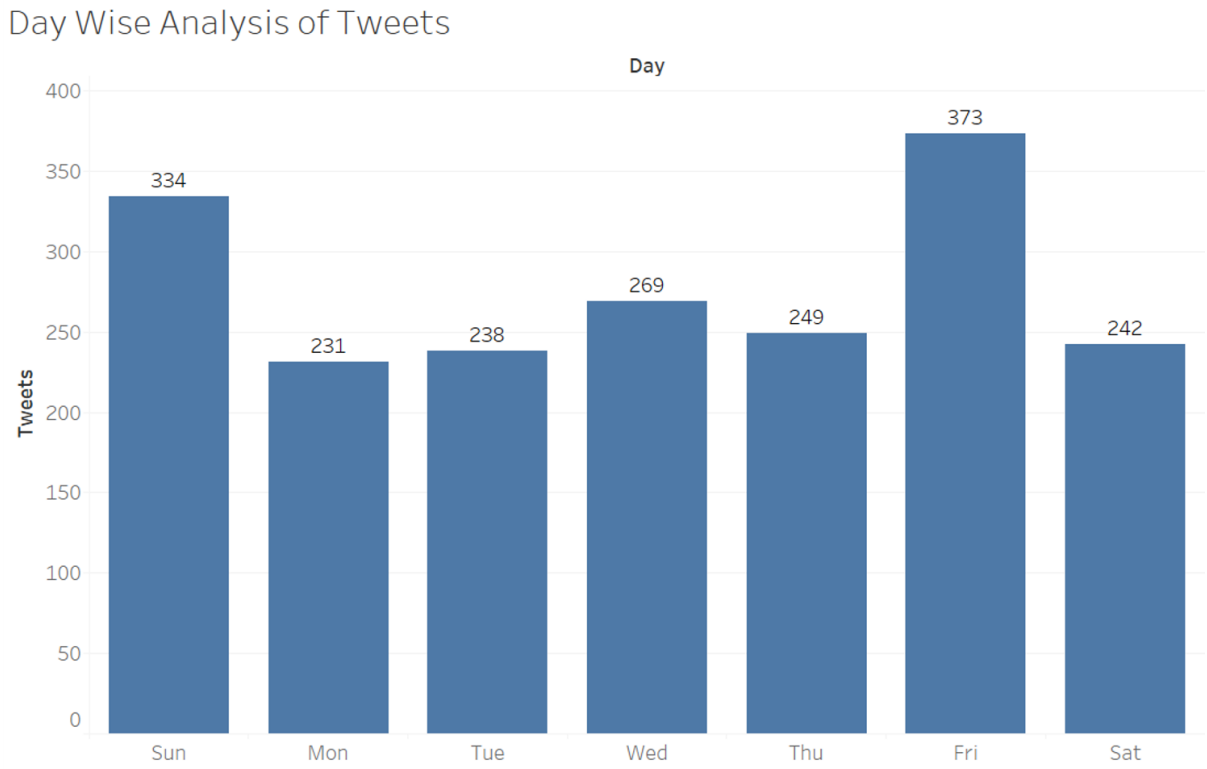
1. **Week on week analysis** : Week on week analysis on the tweets shows seasonality of customer complaints which could be useful in knowing the average rate of enquiries and complaints.



Sum of Tweets for each created_at_x Week. The data is filtered on Label Department, which keeps Delays & Service, Reservation- Confirm/Change/Cancellations, Stations and UK National Rail App. The view is filtered on created_at_x Week, which excludes Null, Week 18 and Week 35.

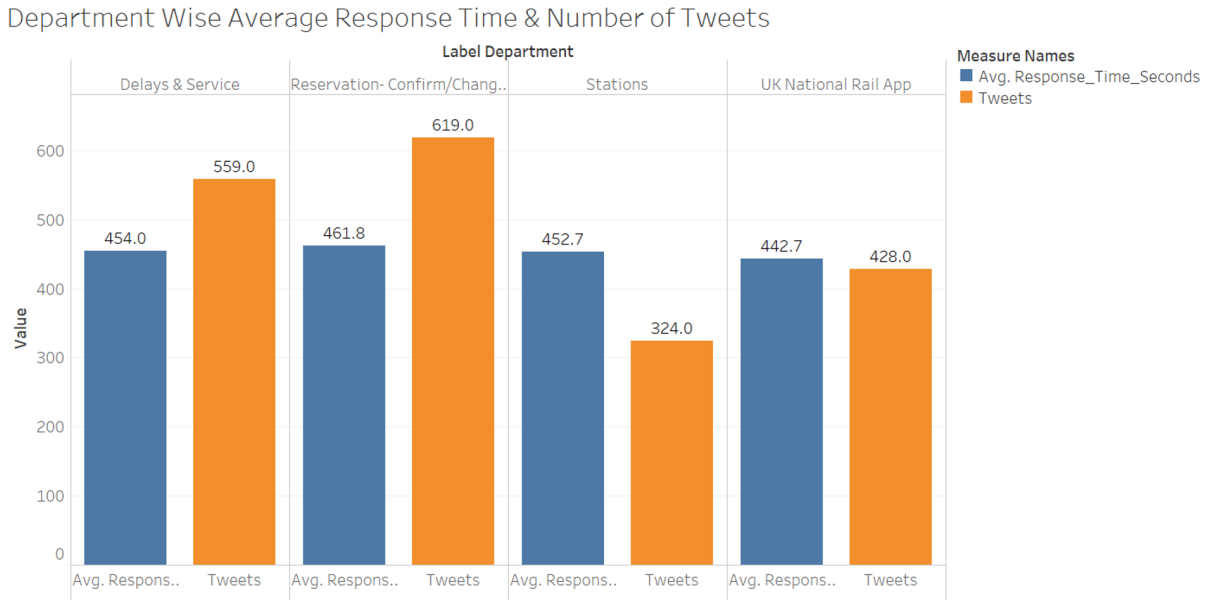
2. **Day on day analysis** : Day on day analysis of customer tweets helps in gaining following insights:

   - We observe that maximum number of queries comes on Friday and Sundays which makes sense as people travel on weekends
   - Employee should more be allocated on weekends so that more number of people's query can be resolved in short time



Sum of Tweets for each Day. The data is filtered on created_at_x Week and Label Department. The created_at_x Week filter excludes Null, Week 18 and Week 35. The Label Department filter keeps Delays & Service, Reservation-Confirm/Change/Cancellations, Stations and UK National Rail App.

3. **Average response time**: The average response time is defined as time taken by customer services to respond to query tweets. Here, we observe that the average time taken to repsond each tweet is alomost same. The average response time is not giving us the correct picture as it is not taking account the number of tweets received by each department. Hence we need to check the performance by weighting this average time with number of tweets received by individual departments.

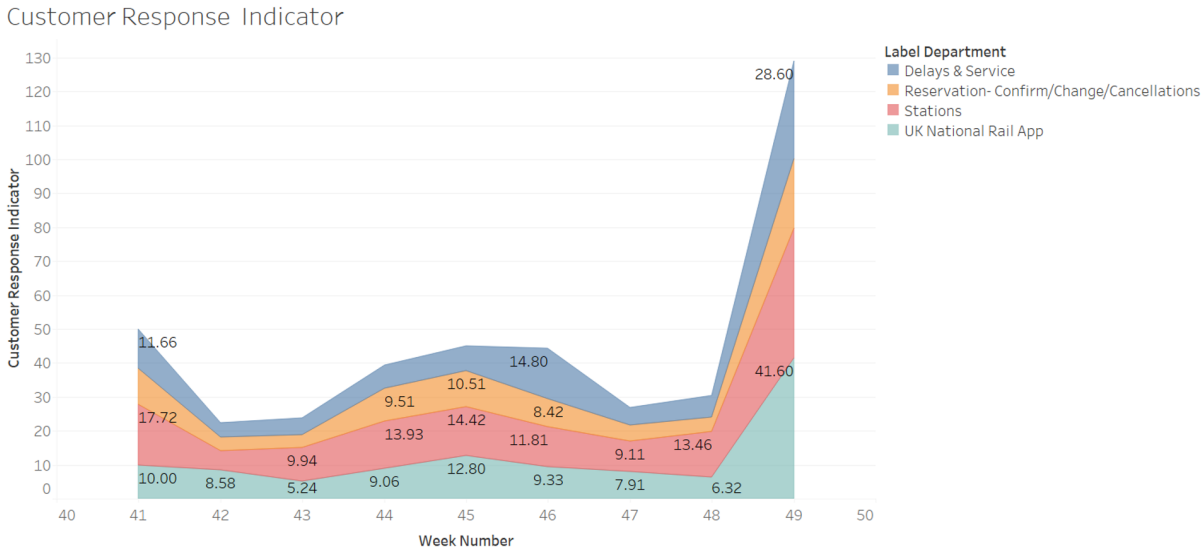


Avg. Response_Time_Seconds and Tweets for each Label Department. Color shows details about Avg. Response_Time_Seconds and Tweets. The data is filtered on created_at_x Week and Response_Time_Seconds. The created_at_x Week filter excludes Null, Week 18 and Week 35. The Response_Time_Seconds filter ranges from 0 to 13318.

4. **Customer reponse indicator**: This can be defined as average time take to reply tweets normalised by number of tweets.

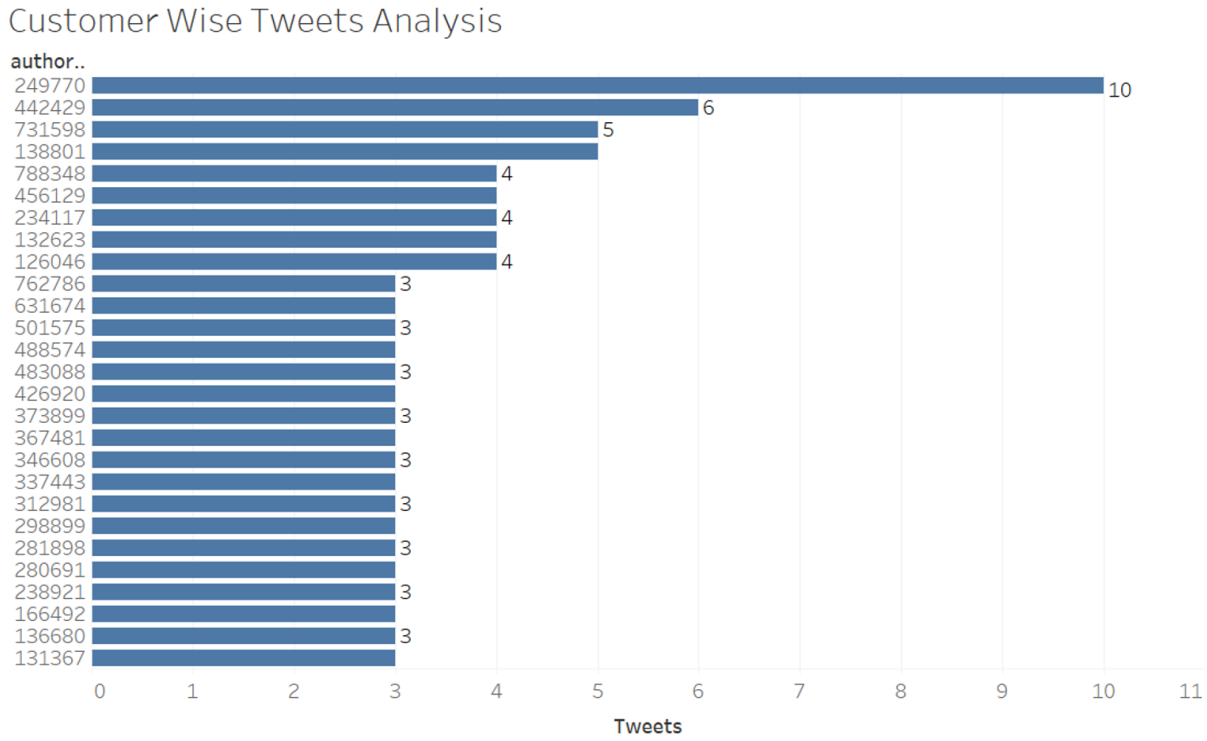
$$Customer\:response \:indicator = \frac{Average\:response\:time}{Total \:number \:of \:tweets}$$

Lower the customer response indicator the better that department is performing.



The plot of CSAT for created_at_x Week. Color shows details about Label Department. The data is filtered on created_at_x Week and Response_Time_Seconds. The created_at_x Week filter excludes Null, Week 18 and Week 35. The Response_Time_Seconds filter ranges from 0 to 13318.

5. **Tweet sources**: This KPI will help us understand that if any of our tweets is coming from an influencer. As these influencers hold power to manipulate our brand value, we should take swift actions if this scenario arises.



Sum of Tweets for each author_id_x. The data is filtered on Label Department and created_at_x Week. The Label Department filter keeps Delays & Service, Reservation- Confirm/Change/Cancellations, Stations and UK National Rail App. The created_at_x Week filter excludes Null, Week 18 and Week 35. The view is filtered on sum of Tweets, which ranges from 3 to 10.

Section 4 : Conclusion

The quote goes as following, “The first step to solving a problem is recognizing there is one.”

As mentioned in the introductory section, the UK National Railway is currently in a state of turmoil and with the product that we developed and presented above, the UK National Railway will be in a better position to grasp the problems that it faces.

Furthermore, improvement does not stop with just introspecting the problem but also helps by providing prescriptive analytics. The process that we have created will not only help UK National Railway to streamline their Social Media monitoring but also help to incentivize good performers through achievements their KRA's based on the KPIs. In the short-term, UK National Railways should see their revenues increase via more attentive customer satisfaction by addressing urgent queries. In the long term, identifying trends among public opinion would lead to structural improvement by fixing issues that may have gone unnoticed.

UK National Railways shouldn't just wonder and wait what future holds for them but rather create it themselves by utilizing customer focused analytics.

References

The following sources were referred for the project:

1. https://www.mordorintelligence.com/industry-reports/digital-process-automation-market (https://www.mordorintelligence.com/industry-reports/digital-process-automation-market)
2. https://link.springer.com/article/10.1007/s11573-018-0915-7 (https://link.springer.com/article/10.1007/s11573-018-0915-7)
3. https://towardsdatascience.com/end-to-end-topic-modeling-in-python-latent-dirichlet-allocation-lda-35ce4ed6b3e0 (https://towardsdatascience.com/end-to-end-topic-modeling-in-python-latent-dirichlet-allocation-lda-35ce4ed6b3e0)
4. https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/ (https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/)
5. https://nlp.stanford.edu/projects/glove/ (https://nlp.stanford.edu/projects/glove/)

In [ ]:

```
print("Thank you for holding on with us!")
```