1. Definition of a problem statement and a short outline of the implementation

"*How accurately can ratings of Post Nords trustpilot reviews be predicted, and are there common words, topics or patterns in the reviews (and reviewers), hereof bad reviews specifically?*"

To answer the question, LDA modelling (UML) will be used to identify topics within the total amount reviews, and bad reviews (reviews with low rating) specifically.

Afterwards, different classifiers (and vectorizers) will be used to predict the ratings of reviews, based on the review text.

Additionally, a network of 1000 reviewers will be made, in order to uncover if Post Nords reviewers, reviews the same businesses (*eg. GLS or DAO*)

2. Description of data acquisition / how it was collected (by you or the publisher of the data)

The dataset has been created based on a webscraping of PostNords trustpilot page. Thus, 400.000 (of approx. 700.000) danish reviews have been collected. The data was collected using requests and scrapy, and contains; *name/username of the reviewer, the total amount of reviews made by the reviewer, the reviewers profile url, date, location, rating, review header & review text.*

Due to computation limitations (*eg. numpy core memory requirement*), it has been decided to only use 50.000 of the collected reviews, however, the remaining are available;

https://raw.githubusercontent.com/NicklasStiborg/M2Exam/main/trustpilot_reviews_200k_1.csv?token=AOUWZYJB2YO7CNOE2HVYEE3BRAFIQ (https://raw.githubusercontent.com/NicklasStiborg/M2Exam/main/trustpilot_reviews_200k_1.csv?token=AOUWZYJB2YO7CNOE2HVYEE3BRAFIQ) https://raw.githubusercontent.com/NicklasStiborg/M2Exam/main/trustpilot_reviews_200k_2.csv?token=AOUWZYKUAJ2TQQQ2WGENQALBRAGXA (https://raw.githubusercontent.com/NicklasStiborg/M2Exam/main/trustpilot_reviews_200k_2.csv?token=AOUWZYKUAJ2TQQQ2WGENQALBRAGXA)

In addition, the webscraping scripts can be accessed through;

https://raw.githubusercontent.com/NicklasStiborg/M2Exam/main/webscraping.py?token=AOUWZYJTFPVCEDAHQIWNRL3BRAI6K (https://raw.githubusercontent.com/NicklasStiborg/M2Exam/main/webscraping.py?token=AOUWZYJTFPVCEDAHQIWNRL3BRAI6K)

Furthermore, there have been collected 1000 profiles' list of reviews, containing; *name/username of the reviewer, the reviewers profile url & list of pages reviewed.* The reason for the smaller numbers of profiles collected, in comparison to reviews, is the due to Trustpilots scraping policy (https://www.trustpilot.com/robots.txt (https://www.trustpilot.com/robots.txt)). However, it is important to point out that even though only 1000 profiles have been collected, each profile can, in theory, have an unlimited amount of reviews, making the amount of edges in the network significant.

3. Data preparation (general)

Due to the data being collected, rather than obtain through third party, the data structure has been determined, and thus no major data preparation will be neeeded. However, a minor error in the values of the date column, have been found after the data collection, resulting in a small cleaning of the data. Furthermore, not all the reviews contains a review text, but some only a header text, resulting in a NaN values, which will be dropped from the dataframe. Alternatively, dummy values could be created and be subsetted, however the amount of data justifies the dropping of NaN values.

3.1 Library imports

In [ ]:
```
import pandas as pd
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import re
from itertools import *
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import LdaMulticore
from gensim.corpora.dictionary import Dictionary
from gensim.models import CoherenceModel
import pyLDAvis
import pyLDAvis.gensim_models as gensimvis
from nltk import word_tokenize
import lemmy
import seaborn as sns
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from gensim.models import Word2Vec
import networkx as nx
import community as community_louvain
import warnings
warnings.filterwarnings("ignore")
csfont = {'fontname':'Times New Roman'}
```

In [ ]:
```
#reading and creating df
df = pd.read_csv("https://raw.githubusercontent.com/NicklasStiborg/M2Exam/main/trustpilot_reviews_200k_1.csv?token=AOUWZYMK3KJPOAXYVNE6VTTBQJZYU")

#only use 50000 due to computation limits
df = df[:50000]

#dropping nan values
df = df.dropna()

#cleaning date column
df["date"] = [i[:10] for i in df["date"]]
```

In [ ]:
```
#wordcloud for all reviews
text = " ".join([i for i in df['text']])
wordcloud = WordCloud(max_font_size=40, max_words=100, background_color="white").generate(text)
fig = plt.figure(figsize = (20, 6))
plt.title("Frequent words in all reviews", fontsize=22, fontweight="bold")
plt.imshow(wordcloud, interpolation="bilinear", cmap="Blues")
plt.axis("off")
plt.show()
```

Frequent words in all reviews

In [ ]:
```
#wordcloud for reviews with 1 or 2 stars
text = " ".join([i for i in df['text'][df['rating'] <= 2]])
wordcloud = WordCloud(max_font_size=40, max_words=100, background_color="white").generate(text)
fig = plt.figure(figsize = (20, 6))
plt.title("Frequent words in reviews with 1 or 2 stars", fontsize=22, fontweight="bold")
plt.imshow(wordcloud, interpolation="bilinear", cmap="Blues")
plt.axis("off")
plt.show()
```

Frequent words in reviews with 1 or 2 stars

3.2 Text cleaning

In order to clean the text for NLP purposes, the review texts, will be processed in terms of *lowering, filtering (discarding digits), removal of punctuations and lemmatization*. Moreover, stopwords will be removed, in order to get a 'pure text'. Since this is danish text, the built-in lemmatizers and stopwords in nltk & gensim cannot be used, resulting in the use of lemmy (danish lemmatizer), and the following stop words list.

In [ ]:
```
stopwordsDA = [
"ad","af","aldrig","alene","alle","allerede","alligevel","alt","altid","anden","andet",
"andre","at","bag","bare","begge","bl", "bl.a.","blandt","blev","blive","bliver","burde","bør",
"ca", "ca.","da","de","dem","den","denne","dens","der","derefter","deres","derfor","derfra",
"deri","dermed","derpå","derved","det","dette","dig","din","dine","disse","dit","dog","du",
"efter","egen","ej","eks", "eller","ellers","en","end","endnu","ene","eneste","enhver","ens",
"enten","er","et","f.eks.","far","fem","fik","fire","flere","flest","fleste","for",
"foran","fordi","forrige","fra","fx","få","får","før","først","gennem","gjorde","gjort",
"god","godt","gør","gøre","gørende","ham","han","hans","har","havde","have","hej","hel",
"heller","helt","hen","hende","hendes","henover","her","herefter","heri","hermed","herpå",
"hos","hun","hvad","hvem","hver","hvilke","hvilken","hvilkes","hvis","hvor","hvordan",
"hvorefter","hvorfor","hvorfra","hvorhen","hvori","hvorimod","hvornår","hvorved","i",
"igen","igennem","ikke","imellem","imens","imod","ind","indtil","ingen","intet","ja",
"jeg","jer","jeres","jo","kan","kom","komme","kommer","kun","kunne","lad","langs","lav",
"lave","lavet","lidt","lige","ligesom","lille","længere","man","mand","mange","med",
"meget","mellem","men","mens","mere","mest","mig","min","mindre","mindst","mine","mit",
"mod","må","måske","ned","nej","nemlig","ni","nogen","nogensinde","noget","nogle","nok",
"nu","ny","nyt","når","nær","næste","næsten","og","også","okay","om","omkring","op","os",
"otte","over","overalt","pga", "pga.","på","samme","sammen","se","seks","selv","selvom","senere",
"ser","ses","siden","sig","sige","sin","sine","sit","skal","skulle","som","stadig","stor",
"store","synes","syntes","syv","så","sådan","således","tag","tage","temmelig","thi","ti",
"tidligere","til","tilbage","tit","to","tre","ud","uden","udover","under","undtagen","var",
"ved","vi","via","vil","ville","vor","vore","vores","vær","være","været","øvrigt"]
```

3.2.1 Lowering and filtering

In [ ]:
```
def regText(text):
    return " ".join(re.sub("[^a-zA-ZÆØÅæøå]", " ",text.lower()).split())
df['text'] = df['text'].apply(regText)
```

3.2.2 Removing punctuation

In [ ]:
```
df['text'] = [i.strip(r'[" ,.!?:;"]') for i in df['text']]
```

3.2.3 Removing stopwords

In [ ]:
```
df['text'] = df['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in stopwordsDA]))
```

3.2.4 Lemmatizing

In [ ]:
```
lemmatizer = lemmy.load("da")
df['text'] = [lemmatizer.lemmatize("",i)[0] for i in df['text']]
```

In [ ]:
```
#wordcloud for all reviews
text = " ".join([i for i in df['text']])
wordcloud = WordCloud(max_font_size=40, max_words=100, background_color="white").generate(text)
fig = plt.figure(figsize = (20, 6))
plt.title("Frequent words in all reviews", fontsize=22, fontweight="bold")
plt.imshow(wordcloud, interpolation="bilinear", cmap="Blues")
plt.axis("off")
plt.show()
```

As shown above in the wordcloud, *pakke* (package) appears in multiple forms (*eg. pakken* (the package)). As also shown, the imported lemmatizer does not account for this, and since the word is so frequent in the dataset, further lemmitization will be used.

In [ ]:
```
def lemmatizerPckge(sen):
    if "pakkerne" in sen:
        sen1= sen.replace("pakkerne", "pakke")
        return sen1
    elif "pakker" in sen:
        sen1= sen.replace("pakker", "pakke")
        return sen1
    elif "pakken" in sen:
        sen1= sen.replace("pakken", "pakke")
        return sen1
    else:
        return sen

df['text'] = [lemmatizerPckge(i) for i in df['text']]
```

In [ ]:
```
#wordcloud for all reviews
text = " ".join([i for i in df['text']])
wordcloud = WordCloud(max_font_size=40, max_words=100, background_color="white").generate(text)
fig = plt.figure(figsize = (20, 6))
plt.title("Frequent words in all reviews", fontsize=22, fontweight="bold")
plt.imshow(wordcloud, interpolation="bilinear", cmap="Blues")
plt.axis("off")
plt.show()
```

4. Data summary

As previously mentioned, the dataframe consists of 7 features:

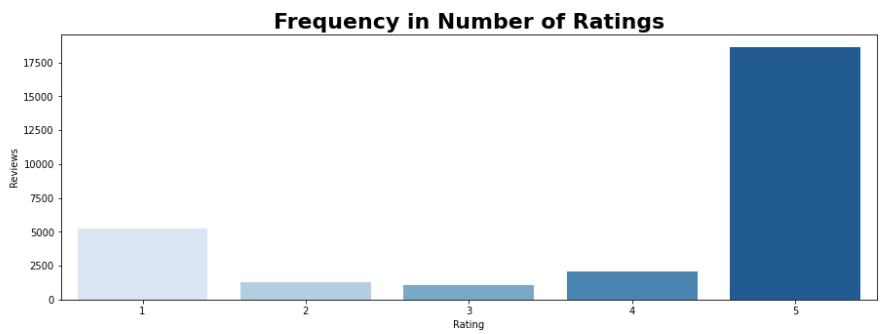
In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 28378 entries, 1 to 49999
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Unnamed: 0   28378 non-null  int64
 1   name         28378 non-null  object
 2   profileLink  28378 non-null  object
 3   date         28378 non-null  object
 4   rating       28378 non-null  int64
 5   reviewCount  28378 non-null  int64
 6   header       28378 non-null  object
 7   text         28378 non-null  object
dtypes: int64(3), object(5)
memory usage: 1.9+ MB
```

4.2.1 Distribution of ratings

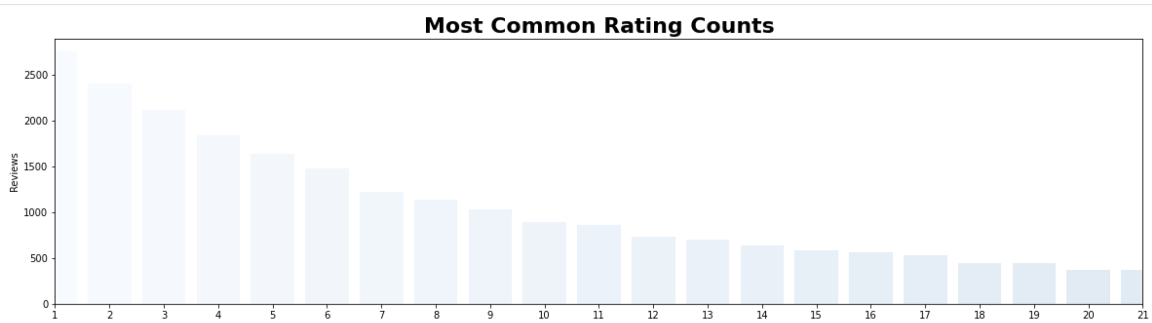
In [ ]:
```
ratingCounts = df.rating.value_counts()
plt.figure(figsize=(15,5))
plt.title("Frequency in Number of Ratings", fontsize=22, fontweight="bold")
sns.barplot(x=ratingCounts.index, y=ratingCounts.values, palette = 'Blues')
plt.xlabel('Rating')
plt.ylabel('Reviews')
```

Out[ ]:
```
Text(0, 0.5, 'Reviews')
```



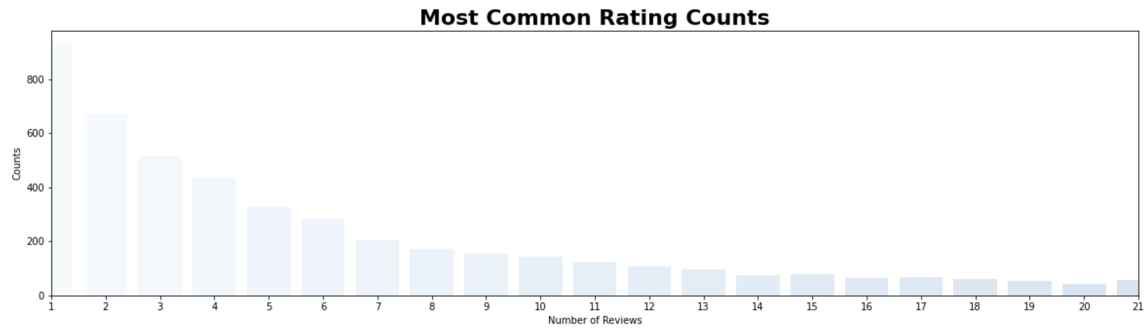
4.2.2 Amount of reviews per user

In [ ]:
```
reviewCounts = df.reviewCount.value_counts()
plt.figure(figsize=(20,5))
sns.barplot(x=reviewCounts.index, y=reviewCounts.values, palette="Blues", alpha = 1)
plt.xlim(0,20)
plt.xlabel("Number of Reviews")
plt.ylabel("Reviews")
plt.title("Most Common Rating Counts", fontsize=22, fontweight="bold",);
```



Furthermore, users with negative ratings (of 1), tends to have less reviews;

In [ ]:
```
dfRating1 =  df[df['rating']==1]
reviewCounts1 = dfRating1.reviewCount.value_counts()
plt.figure(figsize=(20,5))
sns.barplot(x=reviewCounts1.index, y=reviewCounts1.values, palette="Blues")
plt.xlim(0,20)
plt.xlabel("Number of Reviews")
plt.ylabel("Counts")
plt.title("Most Common Rating Counts", fontsize=22, fontweight="bold");
```



Thus, so far, it can be concluded that;

- The majority of ratings are positive
- Reviewers with bad rating have a tendency to review less

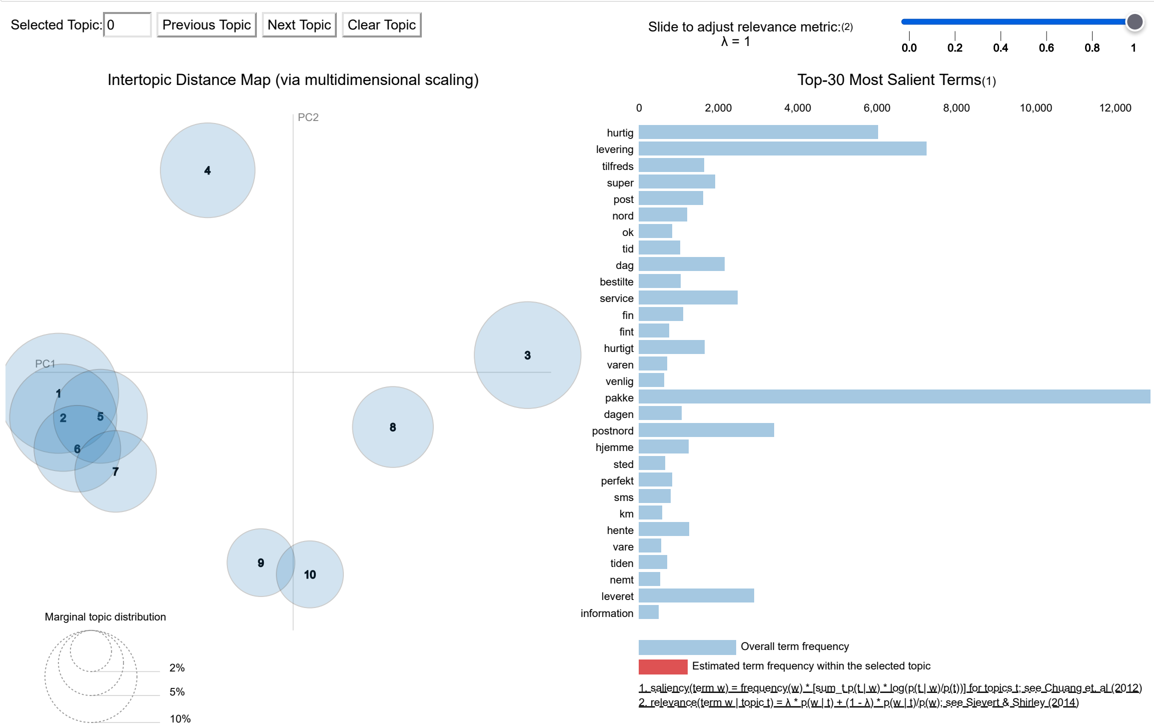
5. Topics of reviews (LDA)

Using LDA (UML), the following topics are created;

In [ ]:
```
#LDA/topic modelling
#tokenizing
tokenized = [word_tokenize(i) for i in df.text]
#creating dictionary
dict = Dictionary(tokenized)
print(dict)
#creating corpus
corpus = [dict.doc2bow(i) for i in tokenized]
ldaModel = LdaMulticore(corpus, id2word=dict, num_topics=10, workers=4, passes=10)
print(ldaModel.print_topics(-1))
ldaDisplay = gensimvis.prepare(ldaModel, corpus, dict)
coherenceModelLda = CoherenceModel(model=ldaModel, texts=tokenized, dictionary=dict, coherence='c_v')
```

In [ ]:
```
pyLDAvis.display(ldaDisplay)
```

Out[ ]:



As shown, the LDA makes a couple of interesting topics hereof;

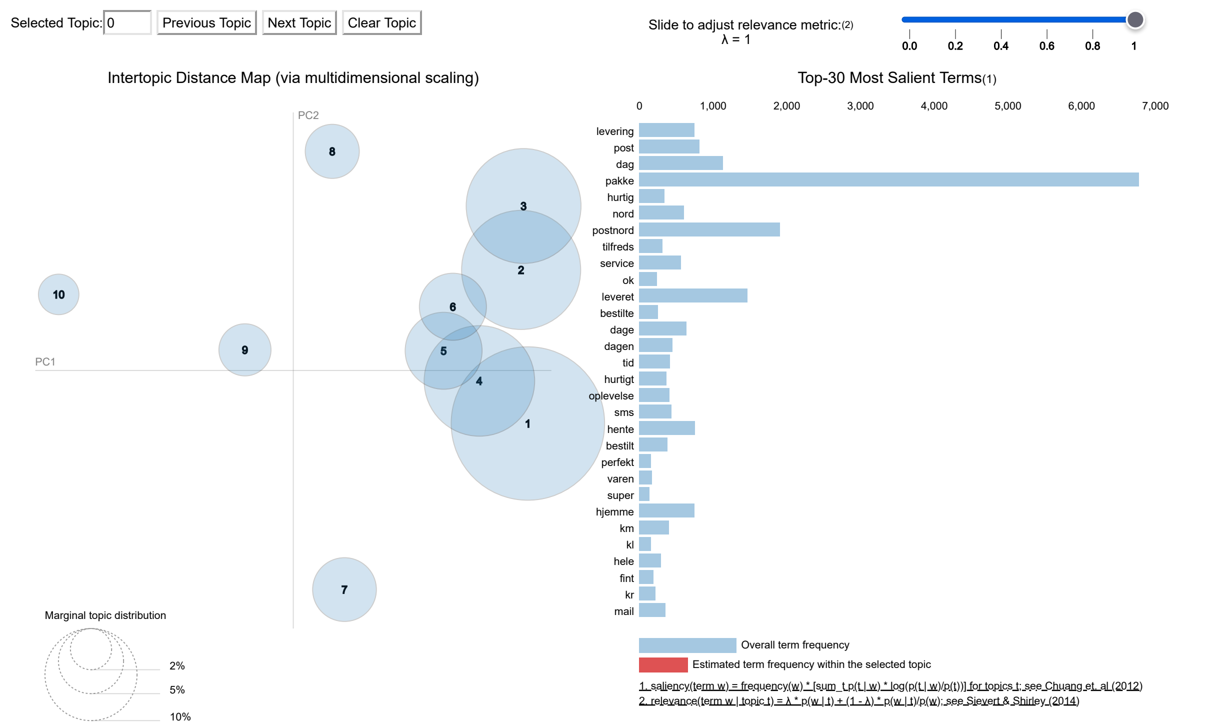
- Topic 7; which majorly revolves around payment *(betale (to pay), kr (danish crowns), told (import fees), moms (VAT), gebyr (fee), betalt (paid))*
- Topic 1; which majorly revolves around delivery *(pakke (package), dage (days), hente (pickup), leveret (delivered), sendt (sent), modtaget (received))*
- Topic 10; which majorly revolves around wrong deliveries or trouble with deliveries *(ødelagt (destroyed), sted (place), afhentningssted (pickup place), dårligt (bad), forkert (wrong), levere (delivery), afleveret (delivered), posthus (post office))*

In [ ]:
```
#LDA/topic modelling with bad ratings
#tokenizing
tokenizedWithBadRating = [word_tokenize(i) for i in df['text'][df['rating'] <= 2]]
#creating dictionary
dictBR = Dictionary(tokenizedWithBadRating)
#creating corpus
corpusBR = [dict.doc2bow(i) for i in tokenizedWithBadRating]
ldaModelBR = LdaMulticore(corpus, id2word=dict, num_topics=10, workers=4, passes=10)
print(ldaModelBR.print_topics(-1))
ldaDisplayBR = gensimvis.prepare(ldaModelBR, corpusBR, dict)
coherenceModelLdaBR = CoherenceModel(model=ldaModel, texts=tokenizedWithBadRating, dictionary=dictBR, coherence='c_v')
```

```
[(0, '0.054*"pakke" + 0.051*"bestilte" + 0.036*"hurtigt" + 0.033*"varen" + 0.029*"dagen" + 0.029*"leveret" + 0.026*"dag" + 0.023*"kl" + 0.023*"vare" + 0.022*
roblem"'), (1, '0.087*"post" + 0.074*"nord" + 0.063*"pakke" + 0.017*"hele" + 0.013*"gået" + 0.012*"indholdet" + 0.011*"skade" + 0.007*"kassen" + 0.007*"vej" 
0.006*"korrekt"'), (2, '0.086*"pakke" + 0.062*"dag" + 0.035*"dage" + 0.025*"postnord" + 0.018*"leveret" + 0.012*"d" + 0.010*"levering" + 0.008*"levere" + 0.0
*"inden" + 0.007*"tog"'), (3, '0.024*"oplevelse" + 0.024*"pakke" + 0.019*"postnord" + 0.016*"gang" + 0.014*"kr" + 0.014*"post" + 0.013*"service" + 0.012*"bet
e" + 0.011*"dårlig" + 0.010*"brev"'), (4, '0.094*"pakke" + 0.038*"tid" + 0.027*"leveret" + 0.012*"døren" + 0.012*"dør" + 0.012*"hjemme" + 0.011*"ringede" + 0
09*"aftale" + 0.008*"service" + 0.008*"afleveret"'), (5, '0.056*"postnord" + 0.055*"pakke" + 0.020*"hjemme" + 0.015*"gang" + 0.012*"gange" + 0.011*"hente" + 
009*"døren" + 0.008*"oplevet" + 0.007*"haft" + 0.006*"leverer"'), (6, '0.073*"ok" + 0.042*"perfekt" + 0.027*"postnord" + 0.020*"pakke" + 0.020*"service" + 0.
7*"fungerede" + 0.016*"tak" + 0.015*"forvente" + 0.015*"stjerne" + 0.014*"god"'), (7, '0.081*"tilfreds" + 0.041*"pakke" + 0.028*"fint" + 0.021*"gik" + 0.020*
inte" + 0.018*"tiden" + 0.017*"rigtig" + 0.017*"postnord" + 0.015*"sms" + 0.014*"hurtigt"'), (8, '0.200*"levering" + 0.184*"hurtig" + 0.053*"super" + 0.046*"
rvice" + 0.032*"fin" + 0.020*"pakke" + 0.018*"tak" + 0.016*"hurtigt" + 0.016*"venlig" + 0.014*"information"'), (9, '0.113*"pakke" + 0.028*"leveret" + 0.027*"
sked" + 0.020*"hente" + 0.015*"sted" + 0.015*"afleveret" + 0.015*"km" + 0.013*"sms" + 0.012*"posthus" + 0.012*"hjemme"')]
```

In [ ]:
```
pyLDAvis.display(ldaDisplayBR)
```

Out[ ]:



As shown above, with bad ratings, topics are similarly about;

- Topic 1; deliveries *(pakke (package), leveret (delivered), hente(pickup), sendt (sent), afleveret (delivered), væk (gone), sted (place))*
- Topic 5; payment *(kr (danish crowns), betale (payment), told (import fee), moms (VAT), betalt (paid), gebyr (fee))*
- Topic 4; bad experiences with deliveries and shipments *(service (service), dårlig (bad), leveret (delivered), afleveret (dropped off), adresse (address), posthus (post office))*

Thus, bad reviews' topics seems to, for the most part, deflect the overall reviews' topic structure.

5. Prediction of ratings (SML)

Using vectorizers (*BoW (countVectorizer) & TD-IDF (tdidfVectorizer)*) and classifiers (*SVC & RandomForest*) a prediction will be made based on a train and test split of the dataset. The test size will be 25% (best practice), and the random_state will be 21 (in order to avoid random seeds at every run).

Due to several combination opportunies, a class with pipelines will be made.

In [ ]:
```
class pipelineConstructor():
    def __init__(self):
        self.data = []

    #vectorizer name
    def vectorizerName(self, vect):
        if vect == "countVect":
            vectName = 'countVect'
            return vectName
        elif vect == "tfVect":
            vectName = 'tfVect'
            return vectName
        else:
            return "Error: invalid vectorizer"

    #vectorizer
    def vectorizer(self, vect):
        stop = stopwordsDA
        if vect == "countVect":
            vectorizer = CountVectorizer(stop_words=stop)
            return vectorizer
        elif vect == "tfVect":
            vectorizer = TfidfVectorizer(stop_words=stop)
            return vectorizer
        else:
            return "Error: invalid vectorizer"

    #pipeline assembly function
    def pipelineCreate(self, vect, classifier):
        vect = self.vectorizer(vect)
        vectName = self.vectorizerName(vect)

        if classifier == "svc":
            classifyName = 'svc'
            classify = SVC()
        elif classifier == "rf":
            classifyName = 'rf'
            classify = RandomForestClassifier()
        else:
            return "Error: invalid classifier"

        pipeline = Pipeline([(vectName, vect), (classifyName, classify)])
        return pipeline

pipelineCon = pipelineConstructor()
```

The target/label value (y) is set as the rating, and the feature data as the reviews (X).

It is also important to note that the classification is multi classification, since the reviewers gives a 1-5 rating.

In [ ]:
```
y = df['rating']
X_train, X_test, y_train, y_test = train_test_split(
                                                    df['text'], y,
                                                    test_size = 0.25,
                                                    random_state=21
)
```

5.1 SVC classifier

In [ ]:
```
pipelineSvcCv = pipelineCon.pipelineCreate("countVect", "svc")
pipelineSvcCv.fit(X_train, y_train)
print(pipelineSvcCv.score(X_test, y_test))
```

```
0.7950669485553207
```

In [ ]:
```
pipelineSvcTv = pipelineCon.pipelineCreate("tfVect", "svc")
pipelineSvcTv.fit(X_train, y_train)
print(pipelineSvcTv.score(X_test, y_test))
```

```
0.8008456659619451
```

5.2 Random forest classifier

In [ ]:
```
pipelineRfCv = pipelineCon.pipelineCreate("countVect", "rf")
pipelineRfCv.fit(X_train, y_train)
print(pipelineRfCv.score(X_test, y_test))
```

```
0.7740662438336857
```

In [ ]:
```
pipelineRfTv = pipelineCon.pipelineCreate("tfVect", "rf")
pipelineRfTv.fit(X_train, y_train)
print(pipelineRfTv.score(X_test, y_test))
```

```
0.7815362931642001
```

Thus, the highest accuracy prediction was achieved using tf-idf vectorizer and svc, with a precision of 80,26%.

6. Network of reviewers

In [ ]:
```
#reading and creating df
network = pd.read_csv('https://raw.githubusercontent.com/NicklasStiborg/M2Exam/main/profiles_reviews_1k.csv?token=AOUWZYJSYQPXVHJYT7BQPUTBRCCNK')
#removing characters from the reviewList column
network['reviewList'] = network['reviewList'].str.replace(r"\'", ' ')
network['reviewList'] = network['reviewList'].str.replace(r"[", ' ')
network['reviewList'] = network['reviewList'].str.replace(r"]", ' ')
# Split this data on comma in reviewList
network['reviewList'] = network['reviewList'].str.split(',')
network = network.explode(column = 'reviewList')
print(network)
```

```
     Unnamed: 0                            name  \
0             0                           Kim A
1             1  Johanna I.U.Aamand Al Tebarani
1             1  Johanna I.U.Aamand Al Tebarani
1             1  Johanna I.U.Aamand Al Tebarani
1             1  Johanna I.U.Aamand Al Tebarani
..          ...                             ...
999         999                             DWH
999         999                             DWH
999         999                             DWH
999         999                             DWH
999         999                             DWH

                          profileLink              reviewList
0    /users/6177b3531f562700128d8ed2    PostNord i Danmark
1    /users/52c66bc700006400015cac69     PostNord i Danmark
1    /users/52c66bc700006400015cac69     PostNord i Danmark
1    /users/52c66bc700006400015cac69     PostNord i Danmark
1    /users/52c66bc700006400015cac69           Jollyroom.dk
..                                ...                    ...
999  /users/4cdbc2d6000064000105069c            smukkere.dk
999  /users/4cdbc2d6000064000105069c                Bodylab
999  /users/4cdbc2d6000064000105069c           "BON A PARTE"
999  /users/4cdbc2d6000064000105069c               STYLEPIT
999  /users/4cdbc2d6000064000105069c              NiceHair

[15971 rows x 4 columns]
```

In [ ]:
```
#removing all of the PostNord values om fra reviewList column, because everyone is connected to everyone already, so there's no need for these conne
ctions in the visualisations of the network
network = network[~network.reviewList.str.contains("PostNord i Danmark")]
#keeping only the columns of the ID, Name and Firms
name_review = network[['Unnamed: 0', 'name', 'reviewList']]
#renaming the Unnamed: 0 column to personID
name_review = name_review.rename(columns={"Unnamed: 0": "personID"})
print(name_review.head())
```

```
   personID                            name         reviewList
1         1  Johanna I.U.Aamand Al Tebarani      Jollyroom.dk
1         1  Johanna I.U.Aamand Al Tebarani              Saxo
1         1  Johanna I.U.Aamand Al Tebarani              Saxo
1         1  Johanna I.U.Aamand Al Tebarani    Faraos Cigarer
1         1  Johanna I.U.Aamand Al Tebarani       Petworld.dk
```

In [ ]:
```
#merging the same dataframe on the reviewList column to get connection between people
edges = pd.merge(name_review, name_review, on='reviewList')
#removing all selfloops
edges = edges[edges.name_x != edges.name_y]
#asserting combination only appers one time and creating a weight column that indicates how many of the same firms these two people has reviewed
edges = edges.groupby(['name_x', 'name_y']).size().reset_index()
#renaming the columns
edgelist = edges.rename(columns={"name_x": "source", "name_y": "target", 0:'weight'})
print(edgelist)
```

```
                    source                  target  weight
0       -Birgitte Petersen  -Hedvig Betty Dahlquist       1
1       -Birgitte Petersen       -Joan Kuur Nielsen       1
2       -Birgitte Petersen              ALLAN STARE      34
3       -Birgitte Petersen                      AYE       1
4       -Birgitte Petersen               Aase Bille       3
...                    ...                      ...     ...
763859                   😀             tove  frimor      15
763860                   😀                  trine h       4
763861                   😀              ulla  zelmer      39
763862                   😀                   verner      36
763863                   😀                       xx      76

[763864 rows x 3 columns]
```

In [ ]:
```
#counting the weights column, some people have high weights values because of the number of times each of these person has reviewed the same firm
edgelist['weight'].value_counts()
```

Out[ ]:
```
1       82428
2       69800
4       52772
3       52060
6       47624
        ...
923         2
1078        2
1112        2
654         2
721         2
Name: weight, Length: 788, dtype: int64
```

Now that the edgelist is created the three different centrality degrees are being computed for each node.

In [ ]:
```
#creating a graph object from the edgelist generated
G = nx.from_pandas_edgelist(edgelist, source='source', target='target', edge_attr='weight', create_using=nx.Graph())
```

In [ ]:
```
#calculating centralities
centrality_dgr = nx.degree_centrality(G)
centrality_eigen = nx.eigenvector_centrality_numpy(G, weight='weight')
centrality_between = nx.betweenness_centrality(G, weight='weight')
#partition = community_louvain.best_partition(G, weight='weight')
```

In [ ]:
```
#adding calculations as attributes to the nodes of the graph
nx.set_node_attributes(G, centrality_dgr, 'centrality_dgr')
nx.set_node_attributes(G, centrality_eigen, 'centrality_eigen')
nx.set_node_attributes(G, centrality_between, 'centrality_between')
#nx.set_node_attributes(G, partition, 'partition')
```

In [ ]:
```
#nodesDf = pd.DataFrame.from_dict(dict(G.nodes(data=True)),orient='index')
#nodesDf.groupby('partition')['centrality_eigen'].nlargest(5)
```

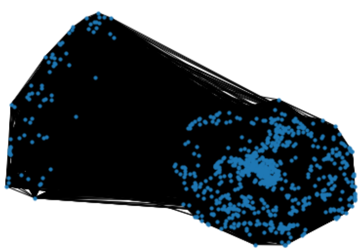
In [ ]:
```
#heading the nodes of the graph
print(G.nodes(data=True)['Jan Hansen'])
```

```
{'centrality_dgr': 0.9161290322580645, 'centrality_eigen': 0.016182952780932697, 'centrality_between': 0.0}
```

In [ ]:
```
betCent = nx.betweenness_centrality(G)
maxBc = max(list(betCent.values()))
nodes = set()
for k,v in betCent.items():
    if v == maxBc:
        nodes.add(k)
print(nodes)
```

```
{'Kirsten Ganer', 'Ann Aagaard', 'Pasca Catalina', 'Jes Bakkensen', 'Asger Larsen', 'Dimka Steckel', 'Bente Kjær Jensen', 'PRM', 'Anne-Lise', 'Knud Kruse', '
gstørvejens Bageri, Maria kirkega', 'Tina Jørgensen', 'Alina Pavel', 'Vibeke Therkildsen', 'Ib Stjernekilde', 'Anne Margrethe Nielsen', 'Tina Poulsen', 'Lone
jer', 'Lisa Christensen', 'Grete Dreier', 'Lars Bjerre Jensen', 'Poul Bjerre', 'Tove Muhlig', 'Lars Horskjaer', 'Marianne  Leere', 'channe simonsen', 'Trine
inkløv', 'Helga Neilsen', 'Vivi', 'Maria Andersen', 'Lone Knoblauch', 'Peter  Vestergaard', 'Simon', 'Anne Holst Pedersen', 'HEIDI CHRISTENSEN', 'Allan Bach
ursen', 'Rita Nielsen', 'Kenneth M', 'Susanne Høst', 'Lene Iversen', 'bente clod', 'Hanne Christiansen', 'Berit Andersen', 'Gry', 'Anders Knuhtsen', 'Birthe
nsen', 'Christian Kjer', 'Henrik', 'Jonas', 'ALLAN STARE', 'Arne Ipsen', 'Hans Erik Lind Madsen', 'Erling Rasmussen', 'GAFFASHOP.DK', 'Dyhrberg.Lotte', 'Jett
Toft', 'Lone', 'Birger', 'Mette Jensen', 'anna sophie', 'Mette', 'Gaja', 'xx', 'Søren B', 'Joergen Andersen', 'Connie Johansen', 'Lisbet\xa0Christiansen', 'M
ael Kleiter', 'Elizabeth Hansen', 'Ellen Skytte', 'Philip Nielsen', 'Per  Sponholtz', 'Rikke Danvold', 'Morten L', 'Fru Randi Westermann', 'Marianne Petersen
'Lone baunkjær', 'Kirsten Larsen', 'Camilla', 'Ruth Simonsen', 'N Madsen', 'ERIK ASTRUP', 'Benedikte Brandt', 'HANS J. WELLEJUS', 'Henning', 'Henning Nymann'
'Anja H.', 'Merethe Reifling', 'bjarne jørgensen', 'Erik D Nielsen', 'Hella Johannsen', 'Inger Westphall', 'Neel', 'Anna Adamczyk', 'Fødderne Op', 'HANS CHRI
IAN PEDERSEN', 'Per Feldbech Rasmussen', 'rita jacobsen', 'Helle Samson', 'PH', 'NICOLAJ HAUPT-LARSEN', 'Koroma.Sama', 'ROSEMARIE MAGNO', 'Ostenfeld,Helena',
John thomsen', 'Fr Britta Ebbesen', 'Bo', 'Peter Jespersen', 'Jensen.Judith', 'Irina Popescu', 'Alice Salomon', '😀', 'Hans Ulrik Bruhn', 'Morten Lilholt', '
rian Beck Kaiser', 'Erik Lund Lauridsen', 'Hanne Hammer', 'Maria', 'Jytte Ravnborg Pederssen', 'Bent Jensen', 'Henrik Eriksen', 'Tove Thomsen', 'ida', 'jette
ndersen', 'Sarwat A.', 'Gitte Hansen Erlandsen', 'Søren Nielsen', 'Antje Clausen', 'Helle  Hyldahl', 'Aise Akbulut', 'Lone Clausen', 'Jan Vejling', 'Kunde',
va Christensen', 'NGG(NIS)', '-Birgitte Petersen', 'Andreas Fabricius Weideli Hansen', 'Bodil Christensen', 'Lone Jørgensen', 'Knud-Erik Jensen', 'Hjørdis Le
sen', 'Eva', 'Hans Ove Nielsen', 'Dennis N', 'Evy Olsen', 'Louise', 'Søren Vennevold', 'Mia Andersen', 'Birgit Lemvig Nielsen', 'Jan Winther', 'Claus Simonse
', 'Jane christiansen', 'Kirsten Bay Nielsen', 'Schierning', 'Karin Dam', 'Lena', 'Ejvind Andersen', 'Sanne Vallebo', 'annmarie Olsen', 'Fru Nina Bysted', 'J
te Walther Birk', 'Bettina Tuelund', 'Ellen Buus', 'Dorte Aalund', 'Fru Margit Bertelsen', 'Harald', 'Cengizhan Eroglu', 'Jonna Uldahl', 'Karin A. Jensen', '
s Waterstradt', 'Marianne Poulsen', 'Monjoin Sébastien', 'mogens jørgensen', 'Hvass Henrik', 'Margit Olsen', 'Bjarne Jensen', 'Elsemette Cassøe', 'Johansen,J
na', 'Ove Bille', 'Helle T. Rømer', 'Pernille Ørum  Svendsen', 'Per Sørensen', 'Elisabeth Feldam', 'Lone Bang', 'Morten Lund', 'Torben Sørensen', 'Martin Nie
en', 'Glenn', 'Preben', 'Else Egsgaard', 'RONNY JOHANSSON', 'Sonja Dahlstrøm.', 'janni sørensen', 'Lilly Jensen', 'N. Hermansen', 'karina olsen', 'John Søren
n'}
```

In [ ]:
```
nx.draw(G, node_size = 10)
plt.show()
```



As shown, there are some high density areas, indicating that reviewers have indeed reviewed the same businesses.