Business Problem

News media outlets are always on the hunt for the world's emerging stories and developments. The New York Times data team has been tasked with solving the following problem:

Can a learning algorithm be used to accurately extract topics from news text documents?

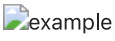
In this project, we will attempt to create a topic model to uncover the hidden word structure in a collection of newsgroup texts. The applications of this learning model would allow NYTimes another resource - beyond human intuition - at their disposal when dispatching their staff of journalists to investigate what is happening in the world. Another application of such a model would be to create a reader recommendation system for NYTimes' catalog.

In this project, we will focus on just the first step of topic modeling. Here is our proposed hypothesis:

'A model can be derived to input news related language material, so it can reliably organize the material into coherent topics.'

Data Understanding

Our hypothesis will be tested on this 20 Newsgroups data set 20 Newsgroups data set. It contains text data organized into 20 different newsgroups, each corresponding to a different topic. Here is a list of the 20 newsgroups partioned according to subject matter:



The data set contains 18,846 news document. Additionally each document includes a 'title,' or subject matter as seen in the table above. Topic modeling is considered unsupervised learning, meaning it can be compared to clustering. As we build clusters of words, we can check to see if these clusters resemble the subjects already given to these documents.

Data Preparation

Imports

In [1]:
```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px

from bs4 import BeautifulSoup
import re

import nltk
from nltk import pos_tag
from nltk.tokenize import word_tokenize, RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

from collections import Counter

from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF

import gensim
from gensim.utils import simple_preprocess
from gensim.models.coherencemodel import CoherenceModel
from gensim.corpora.dictionary import Dictionary
from gensim.models.nmf import Nmf

from operator import itemgetter

from pprint import pprint

import pyLDAvis
import pyLDAvis.gensim_models as gensimvis
```

Credit: davidlenz

In [2]:
```
# Fetch and create a csv file of the data set
def twenty_newsgroup_to_csv():
    newsgroups_train = fetch_20newsgroups(subset='all',
                                          remove=('headers', 'footers',
                                                  'quotes'))

    df = pd.DataFrame(
        [newsgroups_train.data,
         newsgroups_train.target.tolist()]).T
    df.columns = ['text', 'target']

    targets = pd.DataFrame(newsgroups_train.target_names)
    targets.columns = ['title']

    out = pd.merge(df, targets, left_on='target', right_index=True)
    out['date'] = pd.to_datetime('now')
    out.to_csv('20_newsgroup.csv')


twenty_newsgroup_to_csv()
```

In [3]:
```
df = pd.read_csv('data/20_newsgroup.csv')
```

In [4]:
```
df.head()
```

Out[4]:

	Unnamed: 0	text	target	title	date
0	0	\n\nI am sure some bashers of Pens fans are pr...	10	rec.sport.hockey	2021-12-18 14:41:06.564603
1	7	\n[stuff deleted]\n\nOk, here's the solution t...	10	rec.sport.hockey	2021-12-18 14:41:06.564603
2	8	\n\n\nYeah, it's the second one. And I believ...	10	rec.sport.hockey	2021-12-18 14:41:06.564603
3	24	I don't know the exact coverage in the states....	10	rec.sport.hockey	2021-12-18 14:41:06.564603
4	44	Here are the NHL's alltime leaders in goals an...	10	rec.sport.hockey	2021-12-18 14:41:06.564603

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18846 entries, 0 to 18845
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Unnamed: 0  18846 non-null  int64 
 1   text        18466 non-null  object
 2   target      18846 non-null  int64 
 3   title       18846 non-null  object
 4   date        18846 non-null  object
dtypes: int64(2), object(3)
memory usage: 736.3+ KB
```

Check for duplicates

In [6]:
```
df.duplicated(subset=['text']).sum()
```

Out[6]:
```
559
```

In [7]:
```
df = df.drop_duplicates(subset=['text'], keep="first")
df.duplicated(subset=['text']).sum()
```

Out[7]:
```
0
```

Check target distribution

In [8]:
```
df['target'].value_counts()
```

Out[8]:
```
10    980
15    975
8     967
5     966
3     965
11    962
13    959
9     957
1     954
12    954
```

```
14    953
6     942
2     941
7     937
4     928
17    918
16    887
0     780
18    756
19    606
Name: target, dtype: int64
```

Check null values

In [9]:
```
df.isna().sum()
```

Out[9]:
```
Unnamed: 0    0
text          1
target        0
title         0
date          0
dtype: int64
```

In [10]:
```
df[df['text'].isna()]
```

Out[10]:

	Unnamed: 0	text	target	title	date
10	127	NaN	10	rec.sport.hockey	2021-12-18 14:41:06.564603

In [11]:
```
df = df.dropna(subset = ['text'])
```

Rename identity tag column

In [12]:
```
df['Unnamed: 0'].head(20)
```

Out[12]:
```
0       0
1       7
2       8
3      24
4      44
5      66
6      76
7      79
8      97
9     105
11    133
12    179
13    220
14    229
15    237
16    238
17    308
18    356
19    363
20    365
Name: Unnamed: 0, dtype: int64
```

In [13]:
```
df['Unnamed: 0'].nunique()
```

Out[13]:
```
18286
```

In [14]:
```
df = df.rename(columns={'Unnamed: 0': 'identifier'})
```

Explore a single document

In [15]:
```
df['text'][1000]
```

Out[15]:
```
"\nThink!\n\nIt's the SCSI card doing the DMA transfers NOT the disks...\n\nThe SCSI card can do DMA transf
ers containing data from any of the SCSI devices\nit is attached when it wants to.\n\nAn important feature
of SCSI is the ability to detach a device. This frees the\nSCSI bus for other devices. This is typically us
ed in a multi-tasking OS to\nstart transfers on several devices. While each device is seeking the data the
\nbus is free for other commands and data transfers. When the devices are\nready to transfer the data they
can aquire the bus and send the data.\n\nOn an IDE bus when you start a transfer the bus is busy until the
disk has seeked\nthe data and transfered it. This is typically a 10-20ms second lock out for other\nprocess
es wanting the bus irrespective of transfer time.\n"
```

Explore target titles

In [16]:
```
titles = df['title'].unique()
print(titles)
```

```
['rec.sport.hockey' 'comp.sys.ibm.pc.hardware' 'talk.politics.mideast'
 'comp.sys.mac.hardware' 'sci.electronics' 'talk.religion.misc'
 'sci.crypt' 'sci.med' 'alt.atheism' 'rec.motorcycles' 'rec.autos'
 'comp.windows.x' 'comp.graphics' 'sci.space' 'talk.politics.guns'
 'misc.forsale' 'rec.sport.baseball' 'talk.politics.misc'
 'comp.os.ms-windows.misc' 'soc.religion.christian']
```

There seem to be six subjects:

1. Computers
2. Recreation/Sports
3. Misc.
4. Science
5. Politics
6. Religion

Text cleaning

In [17]:
```
def text_preprocessor(text):
    # Strip html text
    soup = BeautifulSoup(text, "lxml")
    data = soup.get_text()
    # Lower case all text
    data = data.lower()
    # Remove edge cases with multiple periods
    pattern = re.compile(r'\s+')
    data = re.sub(pattern, ' ', data.replace('.', ' '))
    # Remove punctuation and other special characters
    pattern = r'[^a-zA-Z\s]'
    data = re.sub(pattern, '', data)
    # Tokenize
    data = word_tokenize(data)
    # Get list of nltk's stopwords
    stopwords_list = stopwords.words('english')
    add_stopwords = ['one', 'like', 'time', 'would', 'will']
    stopwords_list.extend(add_stopwords)
    # Remove stop words
    data = [word for word in data if word not in stopwords_list]
    # Initialize a WordNetLemmatizer object
    wordnet_lemmatizer = WordNetLemmatizer()
    # Convert the tokens into their lemma
    data = [wordnet_lemmatizer.lemmatize(token) for token in data]
    # Convert the list of words back into
    # a string by joining each word with a space
    data = ' '.join(data)
    # Remove double spaces
    data = data.replace('  ', ' ')
    # Remove opening and trailing spaces
    data = data.strip()
    # Return the cleaned text data
    return data
```

In [18]:
```
df.text = df.text.apply(text_preprocessor)
```

```
/Users/dan/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/bs4/__init__.py:332: MarkupResemblesLocatorWarning: "." looks like a filename, not markup. You should probably open this file and pass the filehandle into Beautiful Soup.
  warnings.warn(
```

In [19]:
```
print(df.text[:5])
```

```
0    sure bashers pen fan pretty confused lack kind...
1    stuff deleted ok here solution problem move ca...
2    yeah second believe price ive trying get good ...
3    dont know exact coverage state canada covered ...
4    nhls alltime leader goal point end season much...
Name: text, dtype: object
```

In [20]:
```
# Preserving this version of clean_text for evaluation
df['clean_text'] = df['text']
```

In [21]:
```
df.head()
```

Out[21]:

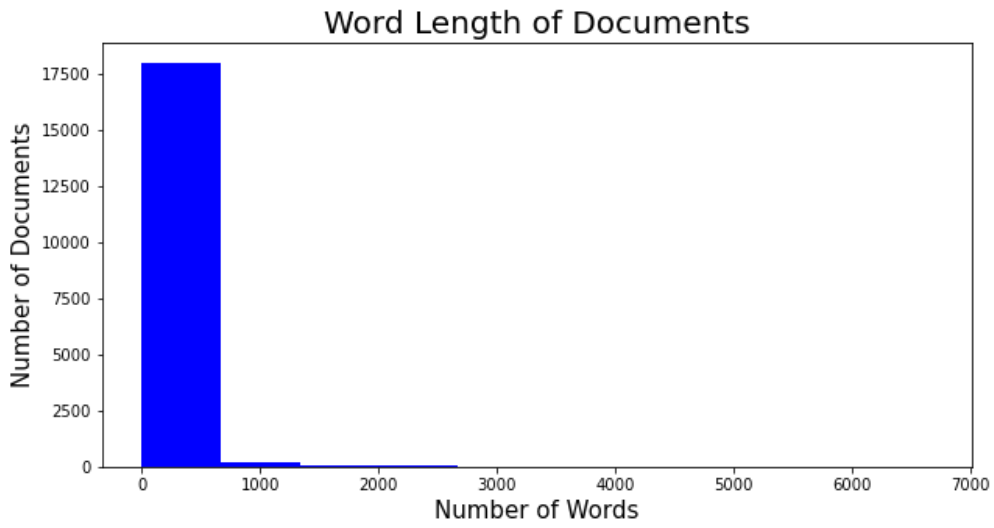
	identifier	text	target	title	date	clean_text
0	0	sure bashers pen fan pretty confused lack kind...	10	rec.sport.hockey	2021-12-18 14:41:06.564603	sure bashers pen fan pretty confused lack kind...
1	7	stuff deleted ok here solution problem move ca...	10	rec.sport.hockey	2021-12-18 14:41:06.564603	stuff deleted ok here solution problem move ca...
2	8	yeah second believe price ive trying get good ...	10	rec.sport.hockey	2021-12-18 14:41:06.564603	yeah second believe price ive trying get good ...
3	24	dont know exact coverage state canada covered ...	10	rec.sport.hockey	2021-12-18 14:41:06.564603	dont know exact coverage state canada covered ...
4	44	nhls alltime leader goal point end season much...	10	rec.sport.hockey	2021-12-18 14:41:06.564603	nhls alltime leader goal point end season much...

Data Exploration

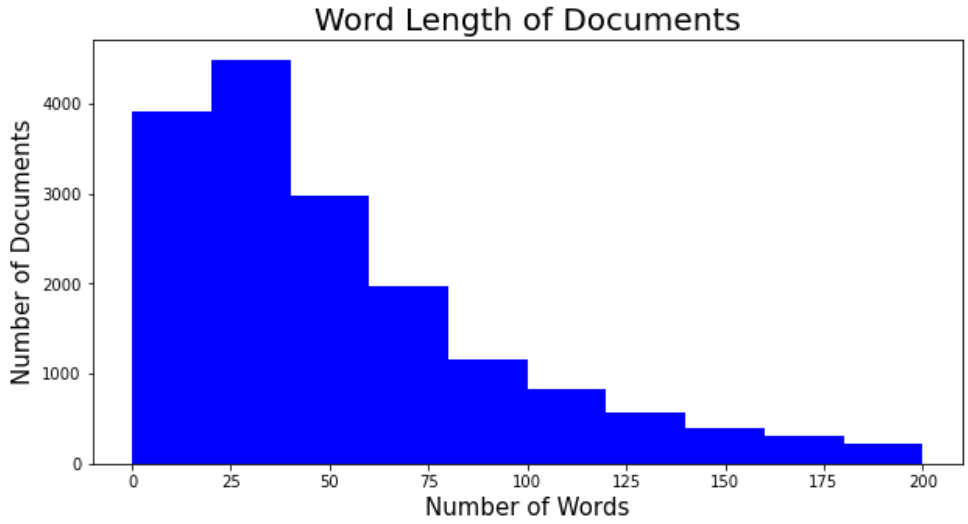
Now that we have cleaned our text, we will analyze some of our data's characteristics:

Number of words per document

In [22]:
```
fig, ax = plt.subplots(figsize=(10, 5))
length_docs = df['text'].str.split().map(lambda x: len(x))
ax.hist(length_docs, color='blue')
ax.set_title('Word Length of Documents', fontsize=20)
ax.set_ylabel('Number of Documents', fontsize=15)
ax.set_xlabel('Number of Words', fontsize=15)
plt.show()
```



In [23]:
```
fig, ax = plt.subplots(figsize=(10, 5))
length_docs = df['text'].str.split().map(lambda x: len(x))
ax.hist(length_docs, color='blue', range=(0, 200))
ax.set_title('Word Length of Documents', fontsize=20)
ax.set_ylabel('Number of Documents', fontsize=15)
ax.set_xlabel('Number of Words', fontsize=15)
plt.show()
```



The majority of the documents have 50 words or less.

Grab all words in documents

In [24]:
```
def get_all_words(text):
    words = []
    for x in text:
        for y in x.split():
            words.append(y.strip())
    return words


all_words = get_all_words(df.text)
all_words[:10]
```

Out[24]:
```
['sure',
 'bashers',
 'pen',
 'fan',
 'pretty',
 'confused',
 'lack',
 'kind',
 'post',
 'recent']
```

Most common words across all documents

In [25]:
```
counter = Counter(all_words)
most_common = counter.most_common(10)
most_common = dict(most_common)
most_common
```

Out[25]:
```
{'x': 7568,
 'people': 6443,
 'dont': 6388,
 'get': 6297,
 'know': 6183,
 'also': 5538,
 'think': 5108,
 'use': 4933,
 'u': 4497,
 'make': 4205}
```

Total unique words across all documents

In [26]:
```
len(list(counter))
```

Out[26]:
```
105177
```

Part of speech tagging

In [27]:
```
# POS-tag the lemmatized words
df.text = [nltk.pos_tag(word_tokenize(token)) for token in df.text]

# Makes each tuple its own string
```

```
def tuple_to_str(text):
    data = [str(item) for item in text]
    return data


df.text = df.text.apply(tuple_to_str)

# Joins the tuples into one continuous string


def join(text):
    data = [' '.join(text)]
    return data


df.text = df.text.apply(join)

# Remove list format to prepare for vectorizer
df.text = [','.join(map(str, l)) for l in df.text]

# Remove whitespace


def join_tuple(text):
    data = text.replace(', ', ',')
    return data


df.text = df.text.apply(join_tuple)
df.text
```

Out[27]:
```
0        ('sure','JJ') ('bashers','NNS') ('pen','VBP') ...
1        ('stuff','NN') ('deleted','VBD') ('ok','RB') (...
2        ('yeah','RB') ('second','JJ') ('believe','JJ')...
3        ('dont','NN') ('know','VBP') ('exact','JJ') ('...
4        ('nhls','JJ') ('alltime','JJ') ('leader','NN')...
                              ...
18841    ('believeing','VBG') ('blindly','RB') ('im','J...
18842    ('holocaust','JJ') ('spanish','JJ') ('inquisit...
18843    ('rest','NN') ('deleted','VBD') ('father','JJR...
18844    ('asking','VBG') ('believe','JJ') ('thing','NN...
18845    ('bible','JJ') ('verse','NN') ('ag','NN') ('us...
Name: text, Length: 18286, dtype: object
```

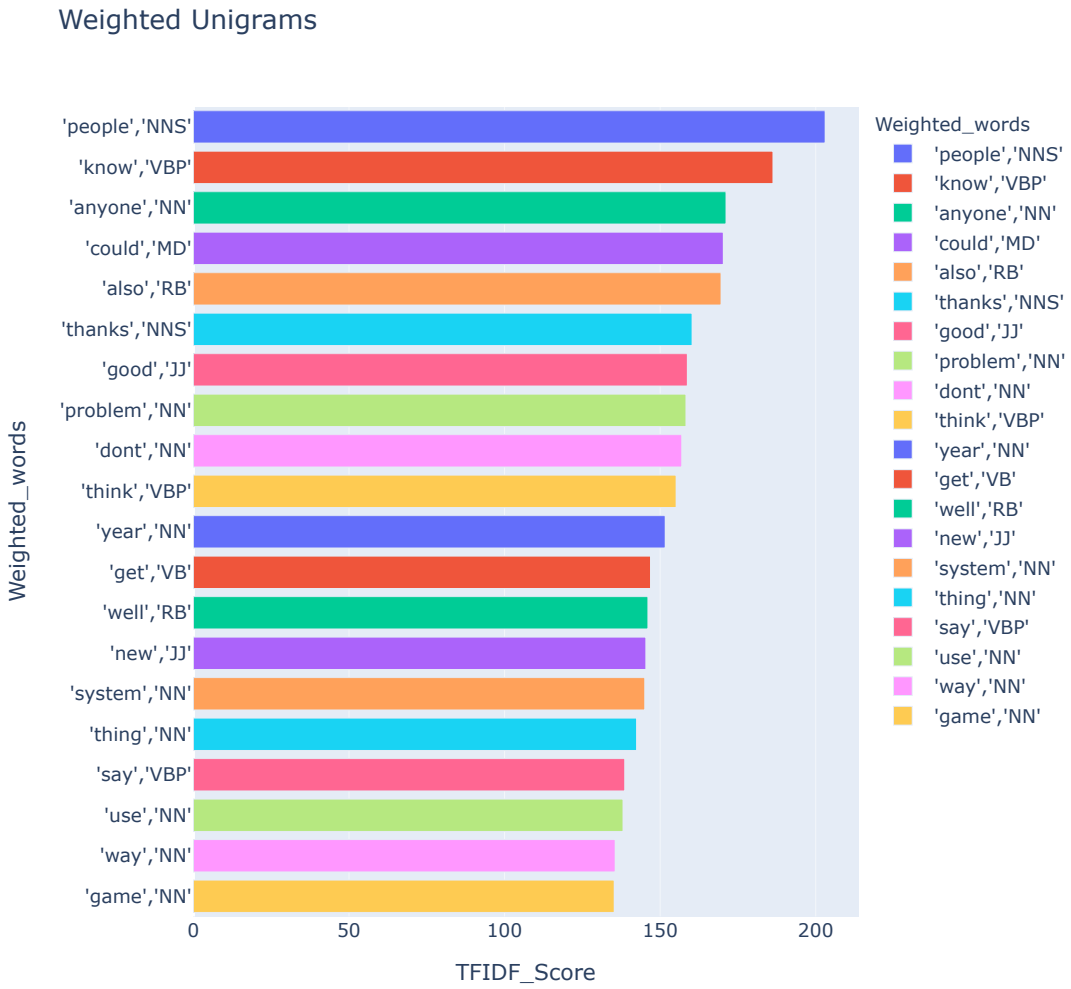
Create a function to find most common n-grams

Credit: Madhav Mathur

In [28]:
```
def get_weighted_ngrams(text, num_words, ngram):
    vec = TfidfVectorizer(ngram_range=(ngram, ngram),
                          lowercase=False,
                          token_pattern=r'\((.*?,.*?)\)').fit(text)
    bag_of_words = vec.transform(text)
    sum_words = bag_of_words.sum(axis=0)
    words_weighted = [(word, sum_words[0, idx])
                      for word, idx in vec.vocabulary_.items()]
    words_weighted = sorted(words_weighted, key=lambda x: x[1], reverse=True)
    return words_weighted[:num_words]
```

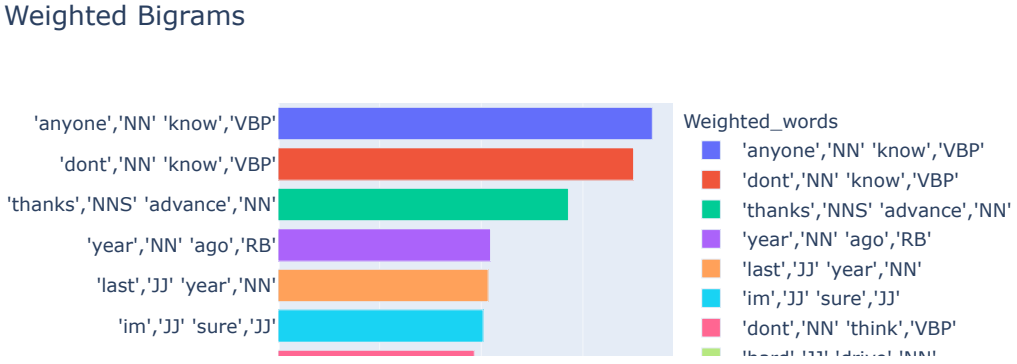
Unigram analysis

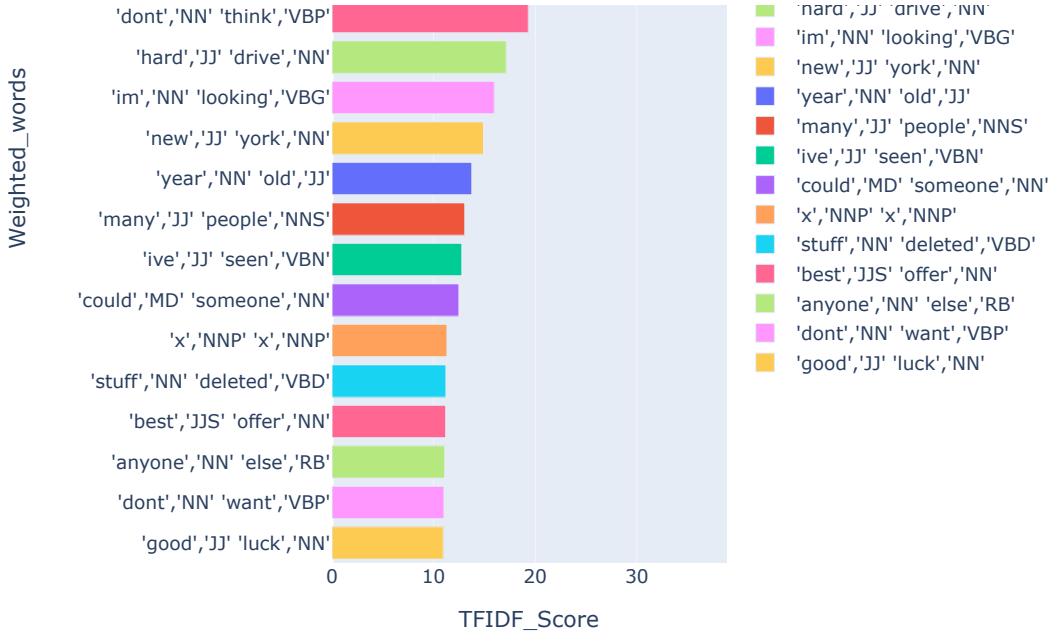
In [29]:
```
weighted_uni = get_weighted_ngrams(df.text, 20, 1)
weighted_uni = dict(weighted_uni)
temp = pd.DataFrame(columns=["Weighted_words", 'TFIDF_Score'])
temp["Weighted_words"] = list(weighted_uni.keys())
temp["TFIDF_Score"] = list(weighted_uni.values())
fig = px.bar(temp,
             x="TFIDF_Score",
             y="Weighted_words",
             title='Weighted Unigrams',
             orientation='h',
             width=700,
             height=700,
             color='Weighted_words')
fig.show()
```



Bigram analysis

In [30]:
```
weighted_bi = get_weighted_ngrams(df.text, 20, 2)
weighted_bi = dict(weighted_bi)
temp = pd.DataFrame(columns=["Weighted_words", 'TFIDF_Score'])
temp["Weighted_words"] = list(weighted_bi.keys())
temp["TFIDF_Score"] = list(weighted_bi.values())
fig = px.bar(temp,
             x="TFIDF_Score",
             y="Weighted_words",
             title='Weighted Bigrams',
             orientation='h',
             width=700,
             height=700,
             color='Weighted_words')
fig.show()
```





The tfidf-weighted unigram and bigram results dont give us much more insight into our business problem of clustering. The majority of highest weighted words don't seem to strongly connect to any of the 6 target subjects. After we see how our model does, we can consider filtering all POS-tags but nouns to see if interprability and accuracy improve.

Modeling

Model-less Baseline

In [31]:
```
df.title.value_counts(normalize=True)
```

Out[31]:
```
rec.sport.hockey            0.053538
soc.religion.christian      0.053319
rec.motorcycles             0.052882
comp.windows.x              0.052827
comp.sys.ibm.pc.hardware    0.052773
sci.crypt                   0.052609
sci.med                     0.052444
rec.sport.baseball          0.052335
sci.electronics             0.052171
comp.graphics               0.052171
sci.space                   0.052116
misc.forsale                0.051515
comp.os.ms-windows.misc     0.051460
rec.autos                   0.051241
comp.sys.mac.hardware       0.050749
talk.politics.mideast       0.050202
talk.politics.guns          0.048507
alt.atheism                 0.042656
talk.politics.misc          0.041343
talk.religion.misc          0.033140
Name: title, dtype: float64
```

Our model-less baseline says that if we randomnly assign a target title to a document, we would only have a 5% chance of being correct.

Use coherence score to choose the number of topics to run our model on

Credit: Rob Salgado

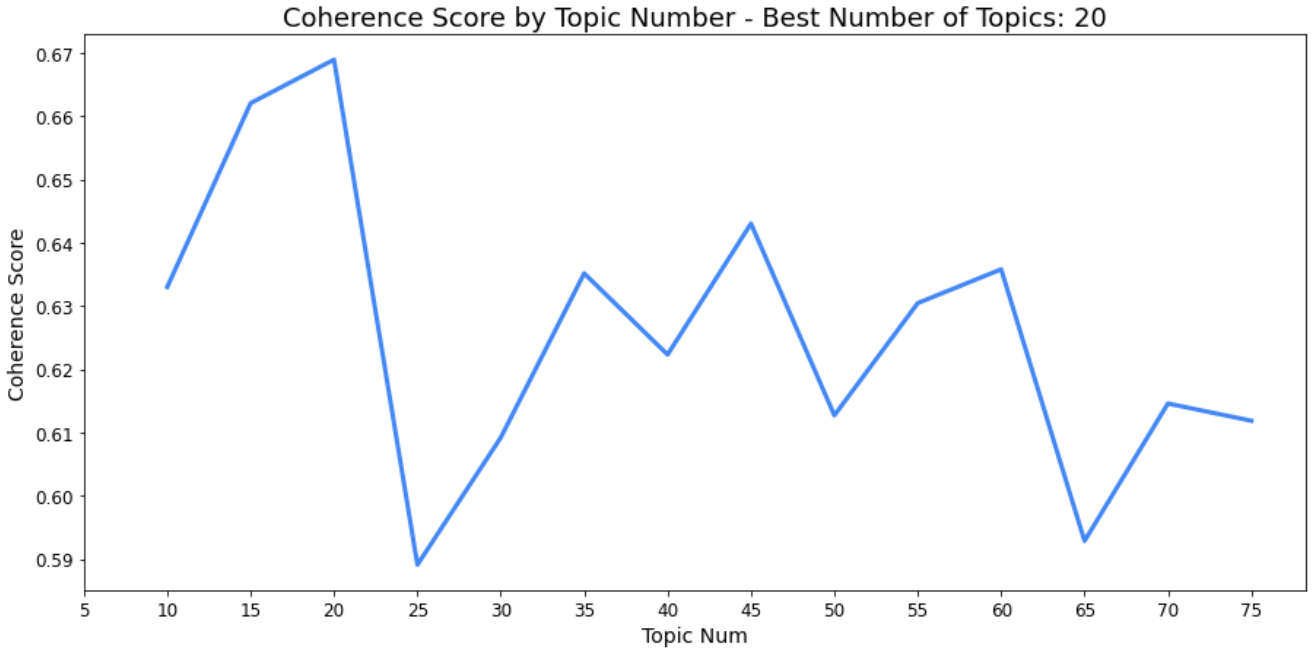
It's important to note that we eliminated extreme cases by:

1. Filtering words used in less than 3 documents
2. Filtering words used in more than 85% of documents
3. Capping total words to 45,000 (total corpus of words from above is 105,177)

In [32]:
```
# Use NMF to get the best number of topics via coherence score
texts = df['text']
```

```
texts = [str(d).split() for d in texts]

# Create a dictionary,
dictionary = Dictionary(texts)

# Limit the number of features
dictionary.filter_extremes(no_below=3, no_above=0.85, keep_n=45000)

# Create the bag-of-words format
corpus = [dictionary.doc2bow(text) for text in texts]

# Create a list of the topic numbers within a range of 10-75, step 5
topic_nums = list(np.arange(10, 75 + 1, 5))

# Run the nmf model and calculate the coherence score
# for each number of topics
coherence_scores = []

for num in topic_nums:
    nmf = Nmf(corpus=corpus,
              num_topics=num,
              id2word=dictionary,
              chunksize=2000,
              passes=5,
              kappa=.1,
              minimum_probability=0.01,
              w_max_iter=300,
              w_stop_condition=0.0001,
              h_max_iter=100,
              h_stop_condition=0.001,
              eval_every=10,
              normalize=True,
              random_state=42)

    # Run the coherence model to get the score
    cm = CoherenceModel(model=nmf,
                        texts=texts,
                        dictionary=dictionary,
                        coherence='c_v')

    coherence_scores.append(round(cm.get_coherence(), 5))

# Get the number of topics with the highest coherence score
scores = list(zip(topic_nums, coherence_scores))
best_num_topics = sorted(scores, key=itemgetter(1), reverse=True)[0][0]

# Plot the results
fig = plt.figure(figsize=(15, 7))

plt.plot(topic_nums, coherence_scores, linewidth=3, color='#4287f5')

plt.xlabel("Topic Num", fontsize=14)
plt.ylabel("Coherence Score", fontsize=14)
plt.title('Coherence Score by Topic Number - Best Number of Topics: {}'.format(
    best_num_topics),
    fontsize=18)
plt.xticks(np.arange(5, max(topic_nums) + 1, 5), fontsize=12)
plt.yticks(fontsize=12)

plt.show()
```

```
/Users/dan/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/gensim/models/nmf.py:588: DeprecationWa
rning:

Using or importing the ABCs from 'collections' instead of from 'collections.abc' is deprecated since Python
3.3, and in 3.9 it will stop working
```

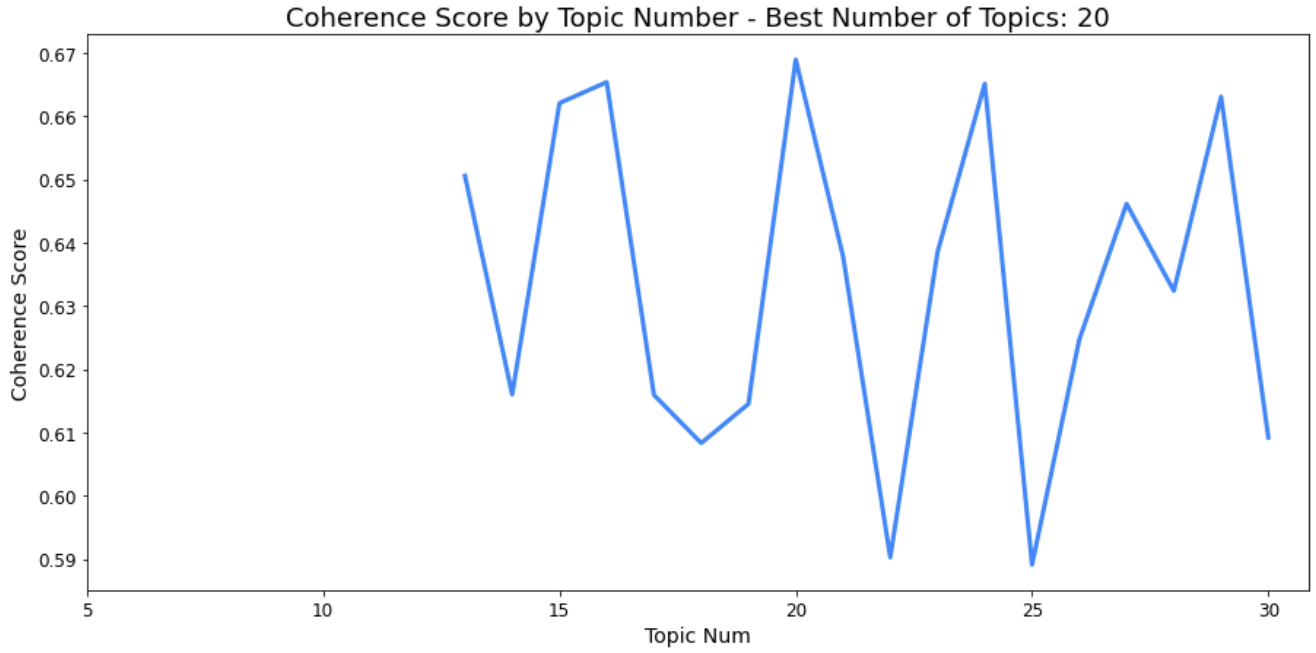


Check coherence score of each topic number range 13-30

In [33]:

```
# Use NMF to get the best number of topics via coherence score
texts = df['text']
texts = [str(d).split() for d in texts]

# Create a dictionary
dictionary = Dictionary(texts)

# Limit the number of features
dictionary.filter_extremes(no_below=3, no_above=0.85, keep_n=45000)

# Create the bag-of-words format
corpus = [dictionary.doc2bow(text) for text in texts]

# Create a list of the topic numbers within a range of 13-30, step 1
topic_nums = list(np.arange(13, 30 + 1, 1))

# Run the nmf model and calculate the coherence score
# for each number of topics
coherence_scores = []

for num in topic_nums:
    nmf = Nmf(corpus=corpus,
              num_topics=num,
              id2word=dictionary,
              chunksize=2000,
              passes=5,
              kappa=.1,
              minimum_probability=0.01,
              w_max_iter=300,
              w_stop_condition=0.0001,
              h_max_iter=100,
              h_stop_condition=0.001,
              eval_every=10,
              normalize=True,
              random_state=42)

    # Run the coherence model to get the score
    cm = CoherenceModel(model=nmf,
                        texts=texts,
                        dictionary=dictionary,
                        coherence='c_v')

    coherence_scores.append(round(cm.get_coherence(), 5))

# Get the number of topics with the highest coherence score
scores = list(zip(topic_nums, coherence_scores))
best_num_topics = sorted(scores, key=itemgetter(1), reverse=True)[0][0]
```

```
# Plot the results
fig = plt.figure(figsize=(15, 7))

plt.plot(topic_nums, coherence_scores, linewidth=3, color='#4287f5')

plt.xlabel("Topic Num", fontsize=14)
plt.ylabel("Coherence Score", fontsize=14)
plt.title('Coherence Score by Topic Number - Best Number of Topics: {}'.format(
    best_num_topics),
    fontsize=18)
plt.xticks(np.arange(5, max(topic_nums) + 1, 5), fontsize=12)
plt.yticks(fontsize=12)

plt.show()
```



Although the value with the highest coherence score was 20, I am deciding to run the model to find 24 topics as this produced the best results over many iterations.

In [34]:
```
best_num_topics = 24
```

NMF vs. LDA

For modeling we will be using NMF. Though LDA performs well, it uses Gibbs sampling (random selection of documents). Within a particular corpus of words, NMF will give us repeatable results.

NMF model

In [35]:
```
# Use the number of topics with the highest coherence score to run the nmf model

texts = df['text']

# Create the tfidf weights
tfidf_vectorizer = TfidfVectorizer(min_df=3,
                                   max_df=0.85,
                                   max_features=45000,
                                   ngram_range=(1, 2),
                                   lowercase=False,
                                   token_pattern=r'\((.*?,.*?)\)')

tfidf = tfidf_vectorizer.fit_transform(texts)

# Save the feature names for later to create topic summaries
tfidf_fn = tfidf_vectorizer.get_feature_names()

# Run the nmf model
nmf = NMF(n_components=best_num_topics,
          init='nndsvd',
```

```
            max_iter=1000,
            l1_ratio=0.0,
            solver='cd',
            alpha=0.0,
            tol=1e-4,
            random_state=42).fit(tfidf)
```

Define utility functions

In [36]:
```
# Function to sort top words for topics
def top_words(topic, n_top_words):
    return topic.argsort()[:-n_top_words - 1:-1]

# Function to populate topic df
def topic_table(model, feature_names, n_top_words):
    topics = {}
    for topic_idx, topic in enumerate(model.components_):
        t = (topic_idx)
        topics[t] = [feature_names[i] for i in top_words(topic, n_top_words)]
    return pd.DataFrame(topics)

# Function to tokenize whitespace
def whitespace_tokenizer(text):
    pattern = r"(?u)\b\w\w+\b"
    tokenizer_regex = RegexpTokenizer(pattern)
    tokens = tokenizer_regex.tokenize(text)
    return tokens

# Function to remove duplicate words
def unique_words(text):
    ulist = []
    [ulist.append(x) for x in text if x not in ulist]
    return ulist
```

Use the top words for each cluster by tfidf weight to create 'topics'

In [37]:
```
# Getting a df with each topic by document
docweights = nmf.transform(tfidf_vectorizer.transform(texts))

n_top_words = 8

topic_df = topic_table(nmf, tfidf_fn, n_top_words).T

# Cleaning up the top words to create topic summaries
topic_df['topics'] = topic_df.apply(lambda x: [' '.join(x)],
                                    axis=1)  # Joining each word into a list
topic_df['topics'] = topic_df['topics'].str[0]  # Removing the list brackets
topic_df['topics'] = topic_df['topics'].apply(
    lambda x: whitespace_tokenizer(x))  # tokenize
topic_df['topics'] = topic_df['topics'].apply(
    lambda x: unique_words(x))  # Removing duplicate words
topic_df['topics'] = topic_df['topics'].apply(
    lambda x: [' '.join(x)])  # Joining each word into a list
topic_df['topics'] = topic_df['topics'].str[0]  # Removing the list brackets
```

In [38]:
```
topic_df
```

Out[38]:

	0	1	2	3	4	5	6	7
0	'dont','NN'	'thing','NN'	'think','VBP'	'good','JJ'	'well','RB'	'get','VB'	'say','VBP'	'really','RB'
1	'window','NN'	'x','NNP'	'window','JJ'	'application','NN'	'do','VBP'	'manager','NN'	'server','NN'	'running','VBG'
2	'game','NN'	'baseball','NN'	'night','NN'	'fan','NN'	'goal','NN'	'espn','NN'	'series','NN'	'playoff','NN'

	0	1	2	3	4	5	6	7
3	'edu','NN' 'shameful','JJ'	'gebcadre','NN' 'dsl','NN'	'gebcadre','NN'	'shameful','JJ' 'surrender','NN'	'surrender','NN' 'soon','RB'	'intellect','VBP'	'intellect','VBP' 'gebcadre','NN'	'dsl','NN' 'pitt','NN'
4	'drive','NN'	'disk','NN'	'hard','JJ'	'hard','JJ' 'drive','NN'	'scsi','NN'	'floppy','JJ'	'controller','NN'	'scsi','JJ'
5	'god','NN'	'jesus','NN'	'christian','JJ'	'bible','JJ'	'god','JJ'	'church','NN'	'christ','NN'	'sin','NN'
6	'thanks','NNS'	'anyone','NN'	'know','VBP'	'anyone','NN' 'know','VBP'	'advance','NN'	'thanks','NNS' 'advance','NN'	'hi','NN'	'please','NN'
7	'car','NN'	'engine','NN'	'dealer','NN'	'oil','NN'	'speed','NN'	'auto','NN'	'tire','NN'	'owner','NN'
8	'armenian','JJ'	'turkish','JJ'	'russian','JJ'	'armenia','NN'	'genocide','NN'	'said','VBD'	'population','NN'	'turkey','NN'
9	'sale','NN'	'price','NN'	'offer','NN'	'new','JJ'	'email','NN'	'please','NN'	'condition','NN'	'interested','JJ'
10	'card','NN'	'monitor','NN'	'video','NN'	'bus','NN'	'color','NN'	'mb','NN'	'bit','NN'	'video','NN' 'card','NN'
11	'chip','NN'	'key','JJ'	'phone','NN'	'encryption','NN'	'key','NN'	'clipper','NN'	'number','NN'	'bit','NN'
12	'law','NN'	'government','NN'	'gun','NN'	'u','JJ'	'state','NN'	'crime','NN'	'federal','JJ'	'right','JJ'
13	'image','NN'	'program','NN'	'graphic','JJ'	'color','NN'	'bit','NN'	'software','NN'	'available','JJ'	'display','NN'
14	'team','NN'	'year','NN'	'player','NN'	'last','JJ'	'season','NN'	'last','JJ' 'year','NN'	'league','NN'	'play','NN'
15	'file','NN'	'directory','NN'	'zip','NN'	'format','NN'	'do','VBP'	'file','JJ'	'ftp','NN'	'disk','NN'

	0	1	2	3	4	5	6	7
16	'port','NN'	'v','NN'	'serial','JJ'	'modem','NN'	'board','NN'	'serial','JJ' 'port','NN'	'power','NN'	'pc','NN'
17	'driver','NN'	'printer','NN'	'version','NN'	'print','NN'	'font','NN'	'laser','NN'	'printer','NN' 'driver','NN'	'laser','NN' 'printer','NN'
18	'com','NN'	'edu','NN'	'list','NN'	'article','NN'	'address','NN'	'tek','NN' 'com','NN'	'tek','NN'	'ico','NN' 'tek','NN'
19	'problem','NN'	'using','VBG'	'work','NN'	'error','NN'	'machine','NN'	'running','VBG'	'fix','NN'	'solution','NN'
20	'space','NN'	'nasa','JJ'	'mission','NN'	'science','NN'	'orbit','NN'	'launch','NN'	'cost','NN'	'station','NN'
21	'system','NN'	'software','NN'	'computer','NN'	'do','VBP'	'objective','JJ'	'memory','NN'	'unix','JJ'	'dx','NN'
22	'israeli','JJ'	'israel','NN'	'jew','NN'	'arab','JJ'	'palestinian','JJ'	'jewish','JJ'	'israel','JJ'	'state','NN'
23	'people','NNS'	'many','JJ'	'homosexual','JJ'	'think','VBP'	'group','NN'	'many','JJ' 'people','NNS'	'religion','NN'	'person','NN'

In [39]:
```
# Create a df with only the created topics and topic num
topic_df = topic_df['topics'].reset_index()
topic_df.columns = ['topic_num', 'topics']
```

In [40]:
```
topic_df
```

Out[40]:

	topic_num	topics
0	0	dont NN thing think VBP good JJ well RB get VB...
1	1	window NN NNP JJ application do VBP manager se...
2	2	game NN baseball night fan goal espn series pl...
3	3	edu NN shameful JJ gebcadre dsl surrender soon...
4	4	drive NN disk hard JJ scsi floppy controller
5	5	god NN jesus christian JJ bible church christ sin
6	6	thanks NNS anyone NN know VBP advance hi please
7	7	car NN engine dealer oil speed auto tire owner
8	8	armenian JJ turkish russian armenia NN genocid...
9	9	sale NN price offer new JJ email please condit...
10	10	card NN monitor video bus color mb bit
11	11	chip NN key JJ phone encryption clipper number...

	topic_num	topics
12	12	law NN government gun JJ state crime federal r...
13	13	image NN program graphic JJ color bit software...
14	14	team NN year player last JJ season league play
15	15	file NN directory zip format do VBP JJ ftp disk
16	16	port NN serial JJ modem board power pc
17	17	driver NN printer version print font laser
18	18	com NN edu list article address tek ico
19	19	problem NN using VBG work error machine runnin...
20	20	space NN nasa JJ mission science orbit launch ...
21	21	system NN software computer do VBP objective J...
22	22	israeli JJ israel NN jew arab palestinian jewi...
23	23	people NNS many JJ homosexual think VBP group ...

Merge topics to original dataset

In [41]:
```
# Creating a temp df with the url and topic num to join on
identifier = df['identifier'].tolist()

df_temp = pd.DataFrame({
    'identifier': identifier,
    'topic_num': docweights.argmax(axis=1)
})

# Merging to get the topic num with identifier
merged_topic = df_temp.merge(topic_df, on='topic_num', how='left')

# Merging with the original df
df_topics = pd.merge(df, merged_topic, on='identifier', how='left')

df_topics = df_topics.drop('text', axis=1)

df_topics.head()
```

Out[41]:

	identifier	target	title	date	clean_text	topic_num	topics
0	0	10	rec.sport.hockey	2021-12-18 14:41:06.564603	sure bashers pen fan pretty confused lack kind...	2	game NN baseball night fan goal espn series pl...
1	7	10	rec.sport.hockey	2021-12-18 14:41:06.564603	stuff deleted ok here solution problem move ca...	2	game NN baseball night fan goal espn series pl...
2	8	10	rec.sport.hockey	2021-12-18 14:41:06.564603	yeah second believe price ive trying get good ...	0	dont NN thing think VBP good JJ well RB get VB...
3	24	10	rec.sport.hockey	2021-12-18 14:41:06.564603	dont know exact coverage state canada covered ...	14	team NN year player last JJ season league play
4	44	10	rec.sport.hockey	2021-12-18 14:41:06.564603	nhls alltime leader goal point end season much...	14	team NN year player last JJ season league play

Check topic assignment distribution

In [42]:
```
df_topics.topic_num.value_counts()
```

Out[42]:
```
0     1895
6     1298
12    1269
23    1201
5     1193
20    1109
14    1098
9      918
13     815
19     811
2      652
22     643
18     634
7      611
```

```
21      597
16      574
10      573
4       488
11      439
15      419
1       413
17      288
8       239
3       109
Name: topic_num, dtype: int64
```

Topic 0 does not coherently identify any of of our target titles. Additionaly, this topic has been assigned the most number documents. Next steps could include adding stop words to reduce how many documents are assigned to topic 0.

NMF model evaluation

The topics most assigned to each target are listed in the 'primary_topic_prediction' column. The secondary topic prediction is also included.

In [43]:
```
# Populating columns for evaluation of nmf topics
topic_text_col = []
topic_text_col_2 = []
article_text = []
article_title = []

topic_targets = range(0, 20)

for target in topic_targets:
    df_target = df_topics.loc[df_topics['target'] == target]
    topic_num_assignment = df_target.topic_num.value_counts().index[0]
    topic_text_assignment = topic_df['topics'][topic_df['topic_num'] ==
                                               topic_num_assignment]
    topic_num_assignment_2 = df_target.topic_num.value_counts().index[1]
    topic_text_assignment_2 = topic_df['topics'][topic_df['topic_num'] ==
                                               topic_num_assignment_2]
    topic_text_col.append(topic_text_assignment)
    topic_text_col_2.append(topic_text_assignment_2)
    topic_article = df_target['clean_text'][df_target['topic_num'] ==
                                            topic_num_assignment].iloc[0]
    article_text.append(topic_article)
    title_article = df_target['title'][df_target['topic_num'] ==
                                       topic_num_assignment].iloc[0]
    article_title.append(title_article)

eval_df = pd.DataFrame({
    'sample_article_text': article_text,
    'target_title': article_title,
    'primary_topic_prediction': topic_text_col,
    'secondary_topic_prediction': topic_text_col_2
})
eval_df
```

Out[43]:

	sample_article_text	target_title	primary_topic_prediction	secondary_topic_prediction
0	oh bobby youre priceless ever tell policy bobb...	alt.atheism	5 god NN jesus christian JJ bible church ch...	23 people NNS many JJ homosexual think VBP ...
1	dont imagine real old amiga user think take lo...	comp.graphics	13 image NN program graphic JJ color bit so...	6 thanks NNS anyone NN know VBP advance hi ...
2	working visual basic v window specifically wor...	comp.os.ms-windows.misc	15 file NN directory zip format do VBP JJ f...	1 window NN NNP JJ application do VBP manag...
3	buslogic announced bt fast scsi vlb interface ...	comp.sys.ibm.pc.hardware	4 drive NN disk hard JJ scsi floppy control...	10 card NN monitor video bus color mb bit N...
4	anyone know nanao compatible popular mac video...	comp.sys.mac.hardware	10 card NN monitor video bus color mb bit N...	6 thanks NNS anyone NN know VBP advance hi ...
5	old sun get occasional use xr started console ...	comp.windows.x	1 window NN NNP JJ application do VBP manag...	13 image NN program graphic JJ color bit so...
6	counterpoint sa watt per channel warm tube sou...	misc.forsale	9 sale NN price offer new JJ email please c...	4 drive NN disk hard JJ scsi floppy control...
7	considering buying chevy gmc x fullsize pickup...	rec.autos	7 car NN engine dealer oil speed auto tire ...	0 dont NN thing think VBP good JJ well RB g...

	sample_article_text	target_title	primary_topic_prediction	secondary_topic_prediction
8	ed u argue gyroscope etc throughly understand ...	rec.motorcycles	0 dont NN thing think VBP good JJ well RB g...	7 car NN engine dealer oil speed auto tire ...
9	dont buy think thing colored large degree prec...	rec.sport.baseball	14 team NN year player last JJ season leagu...	2 game NN baseball night fan goal espn seri...
10	dont know exact coverage state canada covered ...	rec.sport.hockey	14 team NN year player last JJ season leagu...	2 game NN baseball night fan goal espn seri...
11	long mean use encryption im sold	sci.crypt	11 chip NN key JJ phone encryption clipper ...	12 law NN government gun JJ state crime fed...
12	subject say use digital signal dont see couldn...	sci.electronics	6 thanks NNS anyone NN know VBP advance hi ...	16 port NN serial JJ modem board power pc N...
13	according previous poster seek doctor assistan...	sci.med	0 dont NN thing think VBP good JJ well RB g...	20 space NN nasa JJ mission science orbit l...
14	phil didnt early jet fighter also think phil g...	sci.space	20 space NN nasa JJ mission science orbit l...	0 dont NN thing think VBP good JJ well RB g...
15	let word christ dwell richly teach admonish an...	soc.religion.christian	5 god NN jesus christian JJ bible church ch...	23 people NNS many JJ homosexual think VBP ...
16	reported canadian paper thursday april think s...	talk.politics.guns	12 law NN government gun JJ state crime fed...	23 people NNS many JJ homosexual think VBP ...
17	way antisemite mean antijewish antiall persons...	talk.politics.mideast	22 israeli JJ israel NN jew arab palestinia...	8 armenian JJ turkish russian armenia NN ge...
18	proof entire private sector vastly inefficient...	talk.politics.misc	12 law NN government gun JJ state crime fed...	23 people NNS many JJ homosexual think VBP ...
19	christian mean someone belief divinity jesus s...	talk.religion.misc	5 god NN jesus christian JJ bible church ch...	0 dont NN thing think VBP good JJ well RB g...

Based on target titles, all but two target titles were assigned a coherent topic on the first try: **sci.med and rec.motorcycles**. Fortunately, secondary topics assigned to these two target titles are coherent!

Evaluation

Our originally stated hypothesis was:

'A model can be derived to input news related language material, so it can reliably organize the material into coherent topics.'

After text preprocessing and model selection, the resulting findings are:

1. We can reliably identify coherent topics at a rate of 90%. This is significantly better than our model-less baseline of 5%.
2. We have identified an abundance of key words (e.g. law, team, price, god, space, card) that are the most significant in contributing to identification of topics.

Next steps

1. POS-tagging with only nouns (and perhaps verbs and adjectives) may further reduce noise
2. Improve baseline model by searching for key word(s) and clustering into topics
3. Add more stop words to reduce topic 0

Build a visualization using LDA model

Unfortunately, the pyLDAvis topic model visualizer does not support NMF. Thus, we will build a quick LDA model using the same parameters as the above model. The result will not be an exact match to the above topics, but the visualization nonetheless will help us to further evaluate our results.

In [44]:
```
# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=dictionary,
                                            num_topics=best_num_topics,
                                            random_state=42,
                                            update_every=1,
                                            chunksize=2000,
                                            passes=5,
                                            alpha='auto',
                                            per_word_topics=True)
```

In [45]:
```
# Print key words in each topic
pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(13,
  '0.053*"(\'period\',\'NN\')" + 0.027*"(\'michael\',\'NN\')" + '
  '0.022*"(\'eve\',\'VBP\')" + 0.021*"(\'psychological\',\'JJ\')" + '
  '0.019*"(\'daniel\',\'NN\')" + 0.017*"(\'worship\',\'NN\')" + '
  '0.015*"(\'shift\',\'NN\')" + 0.014*"(\'liberty\',\'NN\')" + '
  '0.012*"(\'michael\',\'JJ\')" + 0.010*"(\'qualify\',\'VB\')"'),
 (21,
  '0.078*"(\'drug\',\'NN\')" + 0.069*"(\'gun\',\'NN\')" + '
  '0.062*"(\'christian\',\'NN\')" + 0.026*"(\'water\',\'NN\')" + '
  '0.016*"(\'wright\',\'VBD\')" + 0.014*"(\'stone\',\'NN\')" + '
  '0.012*"(\'reform\',\'NN\')" + 0.012*"(\'gun\',\'JJ\')" + '
  '0.012*"(\'medicine\',\'NN\')" + 0.012*"(\'rise\',\'NN\')"'),
 (17,
  '0.030*"(\'condemnation\',\'NN\')" + 0.021*"(\'bond\',\'NN\')" + '
  '0.021*"(\'satellite\',\'NN\')" + 0.017*"(\'food\',\'NN\')" + '
  '0.016*"(\'brave\',\'VBP\')" + 0.015*"(\'win\',\'VBP\')" + '
  '0.014*"(\'win\',\'NN\')" + 0.013*"(\'suspect\',\'VBP\')" + '
  '0.013*"(\'sake\',\'NN\')" + 0.012*"(\'born\',\'JJ\')"'),
 (15,
  '0.027*"(\'moon\',\'NN\')" + 0.017*"(\'green\',\'JJ\')" + '
  '0.017*"(\'red\',\'JJ\')" + 0.016*"(\'deletion\',\'NN\')" + '
  '0.014*"(\'black\',\'JJ\')" + 0.013*"(\'mm\',\'NN\')" + '
  '0.013*"(\'expressed\',\'VBN\')" + 0.011*"(\'monthly\',\'JJ\')" + '
  '0.011*"(\'cursor\',\'NN\')" + 0.010*"(\'g\',\'NN\')"'),
 (8,
  '0.060*"(\'tax\',\'NN\')" + 0.024*"(\'enemy\',\'NN\')" + '
  '0.023*"(\'evil\',\'JJ\')" + 0.023*"(\'curse\',\'NN\')" + '
  '0.019*"(\'module\',\'NN\')" + 0.018*"(\'camera\',\'NN\')" + '
  '0.017*"(\'sword\',\'NN\')" + 0.015*"(\'auto\',\'NN\')" + '
  '0.014*"(\'picture\',\'NN\')" + 0.012*"(\'opposed\',\'JJ\')"'),
 (2,
  '0.026*"(\'search\',\'NN\')" + 0.019*"(\'weapon\',\'NN\')" + '
  '0.014*"(\'german\',\'JJ\')" + 0.014*"(\'plane\',\'NN\')" + '
  '0.012*"(\'france\',\'NN\')" + 0.012*"(\'greek\',\'JJ\')" + '
  '0.011*"(\'south\',\'JJ\')" + 0.010*"(\'dc\',\'NN\')" + '
  '0.010*"(\'commit\',\'NN\')" + 0.009*"(\'saturday\',\'JJ\')"'),
 (9,
  '0.033*"(\'space\',\'NN\')" + 0.022*"(\'university\',\'NN\')" + '
  '0.019*"(\'research\',\'NN\')" + 0.013*"(\'mission\',\'NN\')" + '
  '0.013*"(\'study\',\'NN\')" + 0.013*"(\'new\',\'JJ\')" + '
  '0.012*"(\'center\',\'NN\')" + 0.011*"(\'conference\',\'NN\')" + '
  '0.010*"(\'national\',\'JJ\')" + 0.009*"(\'page\',\'NN\')"'),
 (20,
  '0.049*"(\'marriage\',\'NN\')" + 0.040*"(\'grace\',\'NN\')" + '
  '0.036*"(\'holy\',\'JJ\')" + 0.035*"(\'shall\',\'MD\')" + '
  '0.029*"(\'male\',\'JJ\')" + 0.026*"(\'testament\',\'NN\')" + '
  '0.018*"(\'object\',\'JJ\')" + 0.017*"(\'traditional\',\'JJ\')" + '
  '0.014*"(\'active\',\'JJ\')" + 0.014*"(\'believer\',\'NN\')"'),
 (0,
  '0.028*"(\'canon\',\'NN\')" + 0.012*"(\'extreme\',\'JJ\')" + '
  '0.011*"(\'rocket\',\'NN\')" + 0.010*"(\'darren\',\'NN\')" + '
  '0.010*"(\'human\',\'NN\')" + 0.009*"(\'campus\',\'NN\')" + '
  '0.008*"(\'region\',\'NN\')" + 0.007*"(\'hate\',\'VBP\')" + '
  '0.007*"(\'attitude\',\'JJ\')" + 0.006*"(\'mass\',\'NN\')"'),
 (18,
  '0.106*"(\'jesus\',\'NN\')" + 0.083*"(\'christ\',\'NN\')" + '
  '0.023*"(\'spirit\',\'NN\')" + 0.020*"(\'p\',\'NN\')" + '
  '0.018*"(\'b\',\'NN\')" + 0.015*"(\'myers\',\'NNS\')" + '
  '0.015*"(\'st\',\'NN\')" + 0.014*"(\'e\',\'NN\')" + 0.013*"(\'new\',\'JJ\')" '
  '+ 0.012*"(\'c\',\'NN\')"'),
 (5,
  '0.041*"(\'game\',\'NN\')" + 0.024*"(\'year\',\'NN\')" + '
  '0.021*"(\'team\',\'NN\')" + 0.017*"(\'player\',\'NN\')" + '
  '0.012*"(\'last\',\'JJ\')" + 0.011*"(\'play\',\'NN\')" + '
  '0.010*"(\'baseball\',\'NN\')" + 0.009*"(\'season\',\'NN\')" + '
  '0.008*"(\'league\',\'NN\')" + 0.007*"(\'fan\',\'NN\')"'),
 (7,
  '0.045*"(\'thanks\',\'NNS\')" + 0.037*"(\'please\',\'NN\')" + '
  '0.034*"(\'anyone\',\'NN\')" + 0.022*"(\'post\',\'NN\')" + '
  '0.020*"(\'interested\',\'JJ\')" + 0.019*"(\'email\',\'NN\')" + '
  '0.017*"(\'know\',\'VBP\')" + 0.014*"(\'looking\',\'VBG\')" + '
  '0.013*"(\'information\',\'NN\')" + 0.013*"(\'please\',\'VB\')"'),
 (23,
  '0.033*"(\'window\',\'NN\')" + 0.018*"(\'program\',\'NN\')" + '
  '0.015*"(\'file\',\'NN\')" + 0.015*"(\'do\',\'VBP\')" + '
  '0.011*"(\'image\',\'NN\')" + 0.011*"(\'software\',\'NN\')" + '
  '0.010*"(\'problem\',\'NN\')" + 0.010*"(\'driver\',\'NN\')" + '
  '0.010*"(\'work\',\'NN\')" + 0.010*"(\'using\',\'VBG\')"'),
 (19,
```

```
 '0.033*"(\'law\',\'NN\')" + 0.017*"(\'people\',\'NNS\')" + '
 '0.010*"(\'u\',\'JJ\')" + 0.009*"(\'death\',\'NN\')" + '
 '0.008*"(\'case\',\'NN\')" + 0.008*"(\'life\',\'NN\')" + '
 '0.007*"(\'right\',\'JJ\')" + 0.007*"(\'state\',\'NN\')" + '
 '0.007*"(\'judge\',\'NN\')" + 0.007*"(\'power\',\'NN\')"'),
(12,
 '0.017*"(\'problem\',\'NN\')" + 0.013*"(\'system\',\'NN\')" + '
 '0.009*"(\'b\',\'NN\')" + 0.009*"(\'power\',\'NN\')" + '
 '0.008*"(\'control\',\'NN\')" + 0.008*"(\'work\',\'NN\')" + '
 '0.007*"(\'paul\',\'JJ\')" + 0.007*"(\'use\',\'NN\')" + '
 '0.006*"(\'line\',\'NN\')" + 0.005*"(\'two\',\'CD\')"'),
(22,
 '0.029*"(\'word\',\'NN\')" + 0.026*"(\'god\',\'NN\')" + '
 '0.022*"(\'sin\',\'NN\')" + 0.017*"(\'book\',\'NN\')" + '
 '0.015*"(\'lord\',\'NN\')" + 0.013*"(\'paul\',\'NN\')" + '
 '0.012*"(\'faith\',\'NN\')" + 0.011*"(\'man\',\'NN\')" + '
 '0.011*"(\'also\',\'RB\')" + 0.010*"(\'god\',\'JJ\')"'),
(10,
 '0.060*"(\'x\',\'NNP\')" + 0.013*"(\'bit\',\'NN\')" + 0.008*"(\'x\',\'JJ\')" '
 '+ 0.008*"(\'use\',\'NN\')" + 0.007*"(\'data\',\'NNS\')" + '
 '0.007*"(\'resource\',\'NN\')" + 0.007*"(\'entry\',\'NN\')" + '
 '0.006*"(\'x\',\'NN\')" + 0.006*"(\'source\',\'NN\')" + '
 '0.006*"(\'set\',\'VBN\')"'),
(6,
 '0.025*"(\'church\',\'NN\')" + 0.023*"(\'christian\',\'JJ\')" + '
 '0.022*"(\'god\',\'NN\')" + 0.015*"(\'people\',\'NNS\')" + '
 '0.011*"(\'may\',\'MD\')" + 0.010*"(\'believe\',\'VBP\')" + '
 '0.010*"(\'belief\',\'NN\')" + 0.010*"(\'question\',\'NN\')" + '
 '0.009*"(\'religion\',\'NN\')" + 0.007*"(\'many\',\'JJ\')"'),
(11,
 '0.008*"(\'new\',\'JJ\')" + 0.008*"(\'also\',\'RB\')" + '
 '0.007*"(\'year\',\'NN\')" + 0.007*"(\'number\',\'NN\')" + '
 '0.006*"(\'money\',\'NN\')" + 0.006*"(\'use\',\'NN\')" + '
 '0.006*"(\'could\',\'MD\')" + 0.005*"(\'may\',\'MD\')" + '
 '0.005*"(\'used\',\'VBN\')" + 0.005*"(\'cost\',\'NN\')"'),
(4,
 '0.013*"(\'think\',\'VBP\')" + 0.012*"(\'people\',\'NNS\')" + '
 '0.012*"(\'know\',\'VBP\')" + 0.011*"(\'thing\',\'NN\')" + '
 '0.010*"(\'dont\',\'NN\')" + 0.010*"(\'good\',\'JJ\')" + '
 '0.009*"(\'say\',\'VBP\')" + 0.009*"(\'well\',\'RB\')" + '
 '0.008*"(\'way\',\'NN\')" + 0.008*"(\'really\',\'RB\')"')]
```

In [46]:
```
pyLDAvis.enable_notebook()

# feed the LDA model into the pyLDAvis instance
lda_viz = gensimvis.prepare(lda_model, corpus, dictionary)
```

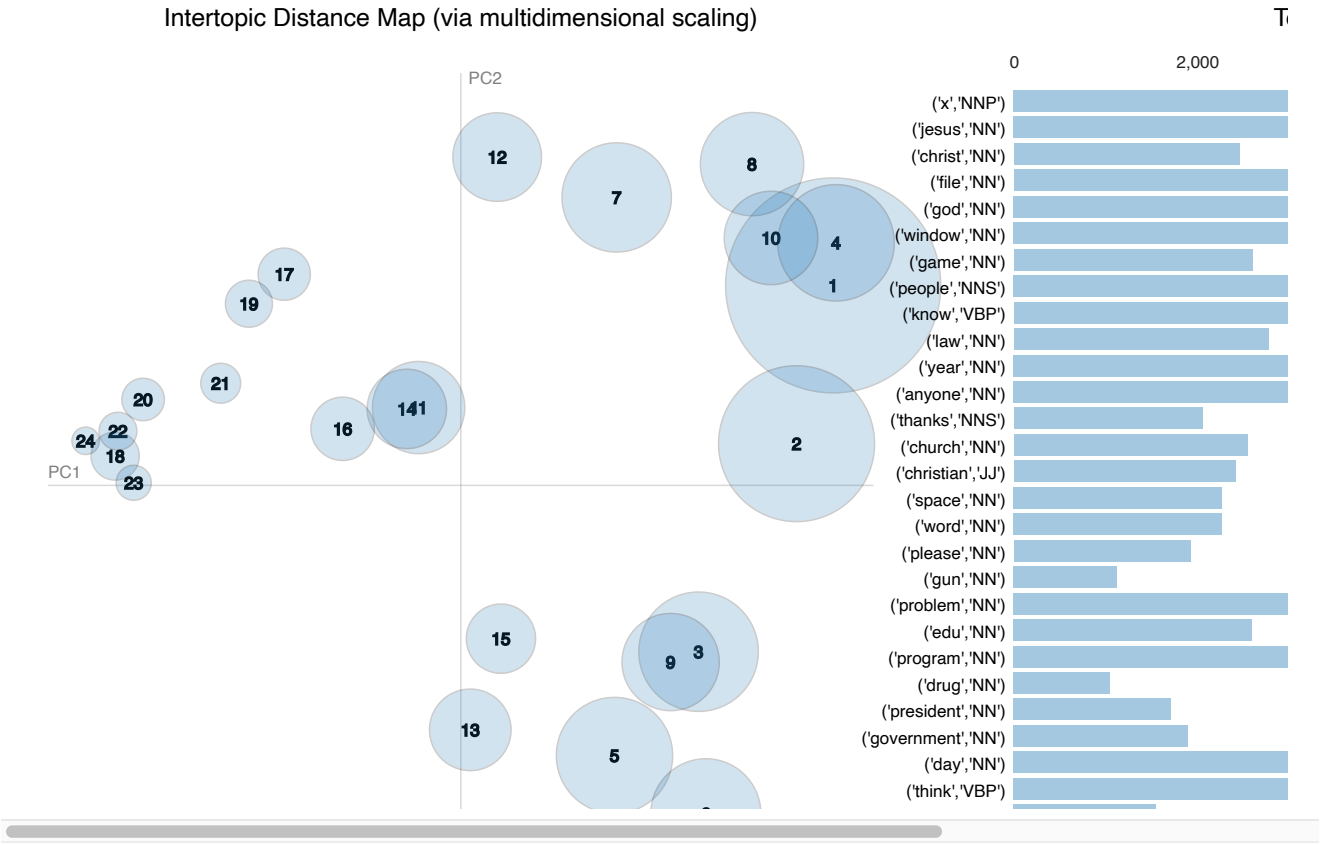
In [47]:
```
lda_viz
```

Out[47]:

Selected Topic: 0 Previous Topic Next Topic Clear Topic

Slide to adjust relevance metric (2)

λ = 1



Each circle represents one of 24 topics. The bigger the circle, the more documents had been assigned to that topic. Topics further apart are more distinguished from each other. An optimal pyLDAvis will have large circles with not much overlapping. We can see that our LDA model performed well in creating distunguished topics. However, overlapping topics such as (10,4,1), (11,14), (3,9) show us there are multiple topics identifying similar documents.

In [ ]: