Evaluation of the Christmasyness of a Pseudo Random Podcast Sample

UoL Programming with Data 2022 Midterm Project

by *Richard Neuboeck*

Introduction

The annual event of Christmas is not only a reason to put lights on trees and wrapping boxes in coloured paper but is actually a festivity of Christian religion celebrating the birth of Jesus Christ. However over many years parts of this celebration have been ingrained in western lifestyle so nowadays it can (also) be seen as common cultural event that is observed not only by Christians but people from other religions or no religious affiliation as well. Another common cultural phenomenon that has only a very short history but already a large footprint is podcasts. Recorded audio like old-timey radio shows but on demand. They can be downloaded and played on almost all modern devices and are ubiquitous in our society. In the following research I'm trying to figure out how Christmas influences podcasts by determining how "christmasy" a podcast is. "Christmasy" is defined by the sum of the usage frequency of single words or word combinations that are mostly associated with Christmas time.

Project Background and Data Relevance

The Internet offers a wide variety of (more or less dubious) "How Christmasy are You?" quizzes (example 1, example 2, example 3). But aside from that there is no (easily findable) research that tries to quantify Christmas related word usage.

In detail my analysis is only concerned with the word usage frequency of specific words and word combinations as defined by a compiled list of terms linked with Christmas. The analysis will be limited to podcasts in English and not be extended to other languages. Only at least monthly recurring media in form of podcasts are tested.

On a large scale the proposed analysis could have some obvious benefits:

- people obsessed with Christmas could use the data to find new sources of holiday cheer
- businesses and advertisement agencies could use the analysis to pinpoint when and where to place specific ads
- a trend history of Christmas in podcasts could be established (when analysed over a longer time periode)

The proposed analysis would be applicable to any text in theory but it makes much more sense to analyse something that is published periodically to see a trend over time. Podcasts are immensely popular, as of 2021 there have been more than 2 million podcasts online and they offer a great variability. This huge pool of data and regular users make podcasts of particular interest. I could not find podcasts that were dedicated to Christmas and posted episodes all year long. Therefore my choice of sources fell to popular comedy and learning podcasts.

Data Origin

The list of words and word combinations that are associated with Christmas has been compiled from multiple sources in part by manually assembling a list and by scraping word lists from web sites directly. The lists are combined and only words that show up more than one time will be used.

Podcasts are published in RSS feeds. These represent a catalogue of all the episodes of a particular podcast over time and point to the downloads for the data (audio or video) files. The feeds can be easily downloaded and parsed to retrieve selected podcast episodes. For this project a command line tool called getpodcast has been used for the data acquisition. The particular sources for the selected podcasts are Maximumfun.org (using simplecast as provider), the BBC (using their own resources to provide the podcasts) and 'A Week of Mornings' is self published via Soundcloud.

For this particular analysis five podcasts have been chosen. Three from the list of podcasts I'm listening too on a regular basis and two randomly chosen because they sounded interesting. The five podcasts in no particular order are:

- A week of Mornings : A compilation of the radio host Monte Belmonte's morning shows during one whole week without proprietary content (music, commercials).
- Judge John Hodgman : A comedic court show adjudicating real-life disputes in a fake court room.
- Shmanners : Wife and husband discussing extraordinary etiquette for ordinary occasions (paraphrased from their podcast).
- We Got This with Mark and Hal : Small debates definitively settled by two actors.
- You're dead to me : Comedy podcast about history.

Obviously the format of a podcast file which is usually mp3 is not directly suited for natural language processing. Even though some of the mentioned podcasts offer a transcript in form of a PDF not all of them do. The format of the PDFs also varies widely and they can not be easily scraped efficiently. So one way to use podcasts is an additional processing step using a speech recognition software called Whisper which is an open sourced trained neural net from the OpenAI project to transcribe audio to text. This part is out of the scope of this Jupyter Notebook. The downloaded podcasts sum up to almost 50GB of data and the transcription process took about a week. Included in the submitted archive containing this Jupyter Notebook are the raw text files that represent the results of the transcription step. Everything that is

described in this Notebook operates on these raw text files but for the completeness of this paper the transcriptions steps are described as well.

Potential other Data Sources

Other recurring media that fits the necessities (published at least monthly, roughly equal length) for the proposed analysis would be for example:

- blog posts : the main advantage is that they are already available in text form but very different HTML document structures would make it more difficult to extract the relevant part of a post fully automatic on a large scale
- (online) news paper articles : the same advantages and disadvantages apply as for blog posts with the added disadvantage that news papers are either pay walled or littered with ads which make scraping the articles more challenging.

Pipeline

The pipeline can be split in

- data collection
  - manually assembling and scraping Christmas words lists
  - pick suitable podcasts that are released on a regular basis, at least once a month, have episodes of almost equal length and are in English
  - find the RSS feed of said podcasts
  - configure getpodcast to download episodes
  - use Whisper to transcribe audio to text
  - assembling a collection of Christmas related terms
- data pre-processing
  - text conversion to lower case
  - removal of newline characters
  - depending on the analysis step removal of unwanted words (stopwords) and characters (punctuation marks)
- data analysis
  - exploratory analysis : word frequency analysis per podcast
  - Christmas word detection per episode and frequency count
- data evaluation
  - extraction of relevant values
  - graph generation

Ethics

The Podcasts in question are licensed to their respective networks or a person but are freely available to listen to. Further processing of the podcast data is not mentioned on the Maximumfun network pages and the self published podcast page. The BBC is very clear that it is not allowed but they offer special access for educational institutions. In any case I started inquiries to use specific podcast

episodes. So far I've only heard back from the BBC. I'm still going ahead with the analysis counting on a delayed reply since this paper will remain unpublished anyway (otherwise I would need to contact the podcast providers again to obtain appropriate permissions).

> Hi Richard,
>
> Thank you for contacting BBC Enquiries.
>
> I understand you wish to transcribe You're Dead To Me podcast part of your language analysis research. This will feature part of your unpublished project for the University of London. With regard to all rights owned by the BBC in this programme, we would have no objection to it being transcribed, purely for these educational purposes.
>
> This does not apply to publishing material outside of the education setting.
>
> Thank you for raising this request with the BBC, we wish you all the best with your research project.
>
> Kind regards, Ciaran Black BBC Enquiries Team

Downloads for this research happened only once therefore resembling regular usage of a free to listen to podcast. So I did not put any unusual strain on the network providers bandwidth or data limit.

This research focuses on Christmas which not solely a Christian religious happening but has reached a level of non religious cultural event status. However there are many equally relevant cultural/religious ceremonies that should be considered for further research to satisfy other interests and for completeness.

An obvious bias is that the chosen podcasts are produced in western culture and therefore are most likely to present the searched for terms. Also the necessity of using podcasts spoken in English is a definite bias.

Considering the results of the analysis increasing the popularity of a podcast is probably in their interest. However there could be negative economic repercussions from advertisers that do not want to have their ad presented on a podcast that shows the specific properties the analysis will provide.

Data

As already explained the podcast raw data (aka mp3s) needed processing to make them available for further analysis and to work within the limits of the assignment. A problem in the data set is commercials. On modern podcast platforms these are injected dynamically and can not be easily screened out automatically. The seasonality of the ads present an additional problem as those ads are in autumn

holiday related. Often they are also spoken by the same person that hosts the podcast making it even more difficult as there is no change in the voice that could be used as a marker. For some podcasts it's possible to find certain word combinations that mark the start and stop of commercials but this can not be generalised to all podcasts. In the following analysis a rough attempt to remove ads has been tried but at least for a subset it is clear that there was no obvious mention of Christmas in an ad break that would skew the results.

The Christmas words list has been tailored to the occasion but there are still words that are used in everyday sentences. In this regard false positives are to be expected but since each matching word sums up to an overall score a threshold value will be set after the initial data exploration to account for this offset.

Environment Setup and Data Preparation

To make the input uniform for further analysis all text has been changed to lower case, unwanted words (stopwords) and characters (punctuation) have been removed and contractions fixed. The next cell will setup the environment. Details to what and why will be given as comments in the code of the cell.

The directory `data_raw` contains the unmodified podcast transcripts in the directory `podcasts` and the Christmas word list sources in `xmas_words`. Immediately inside the `podcasts` directory are sub directories named after the podcasts and inside those directories the text files following the naming scheme `YYYY.MM.DD episode title.mp3.txt`.

Pre Jupyter Processing

Command line (bash) to download the podcasts (started within a Python virtual environment)

```
(getpodcast.venv) [hawk@den getpodcast]$ python
getpodcasts.py --run
```

Command line (bash) to transcribe audio to text using Whisper (started within a Python virtual environment)

```
(whisper.venv) [hawk@den podcasts]$ for i in *; do cd $i;
echo $i; find . -type f -name '*.mp3' -exec whisper --
model base --language en {} \; ; cd ..; done
```

Jupyter Tests and Initial Setup

The Jupyter Notebook has been tested on

- MacOS Ventura (Intel) (homebrew)
- Fedora 37
- and for backward compatibility CentOS7

For these tests *requirements.txt* has been modified and no version numbers are present.

All tests have been done in a Python3 virtual environment setup as follows:

```
[hawk@den]$ python -m venv pwd.venv
[hawk@den]$ cd pwd.venv
[hawk@den]$ source bin/activate
(pwd.venv) [hawk@den]$ pip install jupyterlab
(pwd.venv) [hawk@den]$ unzip ../midterm.zip
(pwd.venv) [hawk@den]$ jupyter lab
```

In [ ]:
```
# installing the necessary dependencies
import sys

!{sys.executable} -m pip install -r requirements.txt
```

In [2]:
```
# preparing the python environment
import os
from pathlib import Path
import re
import nltk
import contractions
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statistics

nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /Users/hawk/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /Users/hawk/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[2]:
```
True
```

In [8]:
```
# preparing podcast transcripts for analysis

# function for data sanitation and tokenization
def santok(data):
    # check if it's a string
    if isinstance(data, str):
        # convert to lower case and remove leading and trailing whitespac
        data = data.lower().strip()
        # subsitute newlines with space
        data = re.sub(r'\n', ' ', data)
        # remove known ad inserts
        data = re.sub(r'(\. [\w\s\']*?(quick )??break.*?(welcome|are|re)
        # fix contractions
        data = contractions.fix(data)
        # tokenize the data by word
        data = nltk.word_tokenize(data)
        # remove punctuation marks
        data = [ word for word in data if word.isalpha() ]
        # remove stop words
        stop_words = set(nltk.corpus.stopwords.words('english'))
```

```
        data = [ word for word in data if not word in stop_words ]
        # return the clean word array
        return data
    # if nothing else return an empty string
    return '';


# data structure to store the working texts
# { dict of podcast names } -> { dict of month } -> [ array of podcast ep
casts = {}

print('Sorry. This takes a while.')

# data directory has the following layout
# ./data_raw
#   |-------- /podcasts
#   |          |---------- /JJHO
#   |          |            |------- /YYYY.MM.DD title.mp3.txt
#   |          |            |------- ...
#   |          |---------- ...
#   |-------- /xmas_words
#
# get the podcast raw data
# go through all entries in the podcasts directory
count = 0
for entry in Path('./data_raw/podcasts').iterdir():
    # check if the entry is a directory (which it should be)
    if entry.is_dir():
        # go through all items in the podcast directory
        for item in Path('./data_raw/podcasts/' + entry.name).iterdir():
            # get the month an title from the file name using a regex
            match = re.match(r'\d{4}\.(\d{2})\.\d{2}\s(.+)\.mp3\.txt', it
            # check if a match exists
            if match:
                # if so read the data
                with open(item, 'r') as f:
                    data = f.read()
                # prepare the data structure and store the data
                if entry.name not in casts.keys(): casts[entry.name] = {}
                if match.group(1) not in casts[entry.name].keys(): casts[
                casts[entry.name][match.group(1)].append({ 'title' : matc
                count += 1

# count all words, not unique words!
words = 0
for pod in casts.keys():
    for month in casts[pod].keys():
        for entry in casts[pod][month]:
            words += len(entry['text'])

print('Read', count, 'files from', len(casts.keys()), 'podcasts summing u
# sample output
print('Sample output')
#for entry in casts['JJHO']['02']:
#    print(entry['title'])
#    print(entry['text'][:10])
#
print(casts['JJHO']['01'][2]['title'])
print(casts['JJHO']['01'][2]['text'][:10])
```

```
Sorry. This takes a while.
Read 240 files from 5 podcasts summing up 1055189 words.
Sample output
Maine's Holiday Boy
['welcome', 'judge', 'john', 'hodgman', 'podcast', 'bailiff', 'jesse',
'thorn', 'chambers', 'week']
```

In [9]:

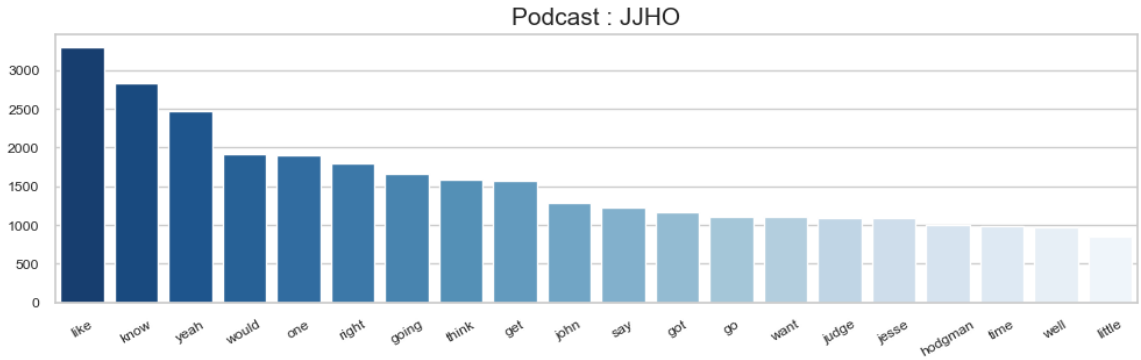
```
# preparing Christmas word list

# function to get words from an html table
# clean everything up and return a dictionary
def get_words(url):
    tbl = pd.read_html(url)
    # concatenate the result of the list comprehension
    # which extracts only the first column of the table in question
    res = pd.concat([t[t.columns[0]] for t in tbl])
    # convert to pandas data frame -> why is it not a df already?
    res = pd.DataFrame(res)
    # manually set name of the column for easier addressing
    res = res.set_axis(['words'], axis='columns')
    # remove all rows that have 'Word' as content
    res = res.loc[res['words'] != 'Word']
    # return dictionary comprehension from datafram to list conversion
    return { x.lower() : x_words.get(x.lower().strip(), 0) + 1 for x in r

# initialize an empty dictionary
x_words = {}

# words

# read manually assembled word list and add that to the dictionary
with open('data_raw/xmas_words/xmas_words2.txt', 'r') as file:
    # line by line
    for line in file:
        # set the key as stripped and lower case string
        key = line.strip().lower().strip()
        # add word to the dictionary
        x_words[key] = x_words.get(key, 0) + 1

# add additional sources
# source https://grammar.yourdictionary.com/word-lists/popular-christmas-
x_words.update(get_words('data_raw/xmas_words/popular-christmas-words-fro
# source https://ielts.idp.com/prepare/article-full-list-100-popular-engl
x_words.update(get_words('data_raw/xmas_words//article-full-list-100-popu

print('Initial dictionary size:', len(x_words))

# sort dictionary by value descending
x_words = { k: v for k, v in sorted(x_words.items(), key=lambda item: ite

# remove items from the dictionary that only appear trice
x_words = {k: v for k, v in x_words.items() if v > 3}

# after reviewing the results the following words are excluded from the l
exclude_list = [ 'star', 'holiday', 'tradition' ]
x_words = { k: v for k, v in x_words.items() if k not in exclude_list }

print('Cleaned up dictionary size:', len(x_words))
print('The ten most common words associated with Christmas:', list(x_word
```

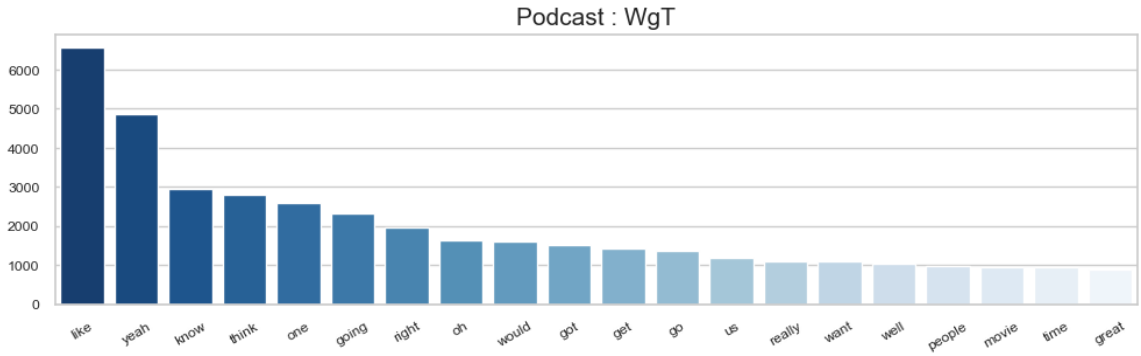
```
Initial dictionary size: 335
Cleaned up dictionary size: 24
The ten most common words associated with Christmas: ['christmas', 'rudo
lph', 'scrooge', 'tidings', 'vacation', 'yuletide', 'eggnog', 'fruitcak
e', 'holly', 'mistletoe']
```

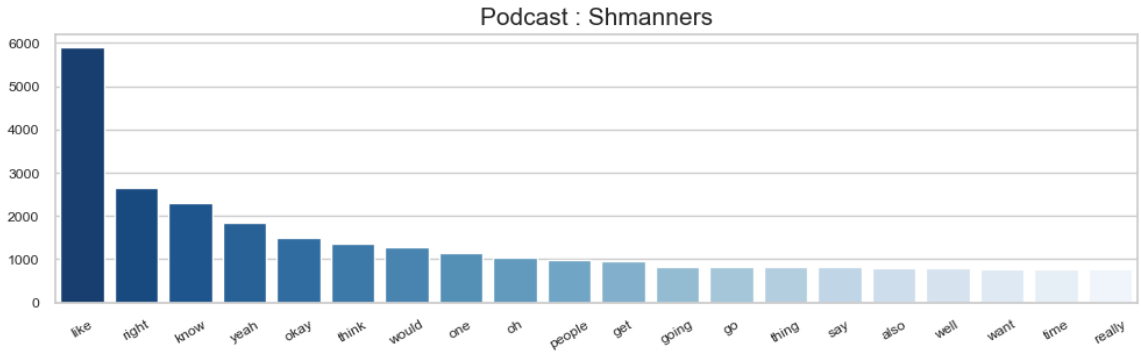
Data Exploration

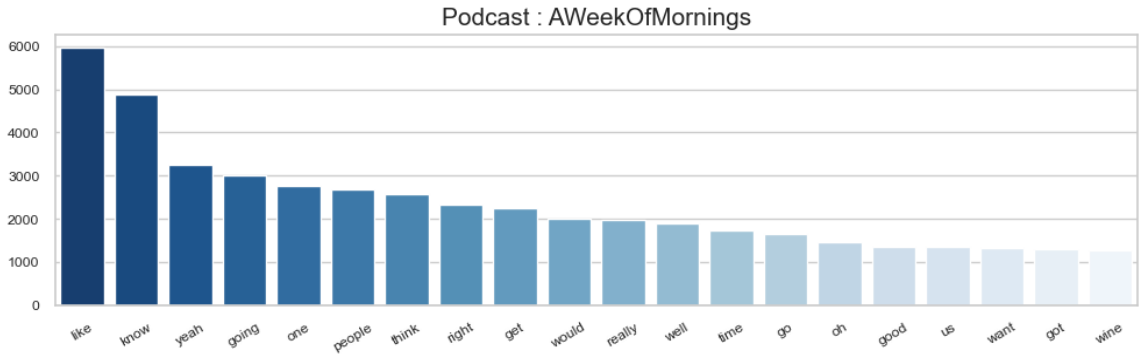
In [10]:

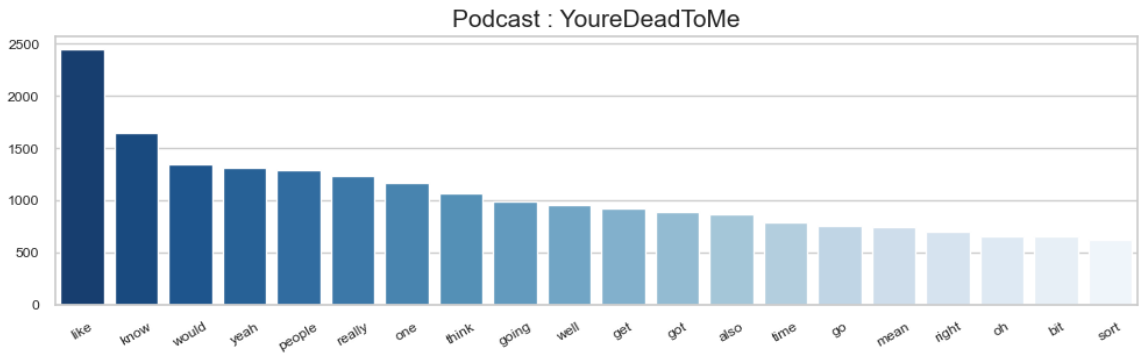
```python
# combine all episodes for freqency distribiution analysis

# function to do the frequency analysation and plotting
def analyse_and_plot(words, title, pout=False):
    # calc frequency distribution from the most common 50 words
    all_fdist = nltk.FreqDist(words).most_common(20)

    # convert to pandas series for easier handling
    all_fdist = pd.Series(dict(all_fdist))

    if pout:
        print(all_fdist)

    # seaborn defaults
    sns.set(font_scale=0.8)
    sns.set_style("whitegrid")

    # setting up figure
    fig, ax = plt.subplots(figsize=(12,3))

    # plotting using seaborn and pandas
    all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax,
    all_plot.axes.set_title(title, fontsize=15)
    plt.xticks(rotation=30)
    plt.show()

# all the words from all podcasts
over_all_words = []

# loop over all podcasts
for pod in casts.keys():
    all_words = []

    # loop over all month and episodes
    for month in casts[pod]:
        for episode in casts[pod][month]:
            all_words.extend(episode['text'])

    # add all the words from this podcast to the over all collection
    over_all_words.extend(all_words)

    # analyse and plot the current podcast
    analyse_and_plot(all_words, 'Podcast : ' + pod)

# analyse and plot everything
analyse_and_plot(over_all_words, 'Everything')
```

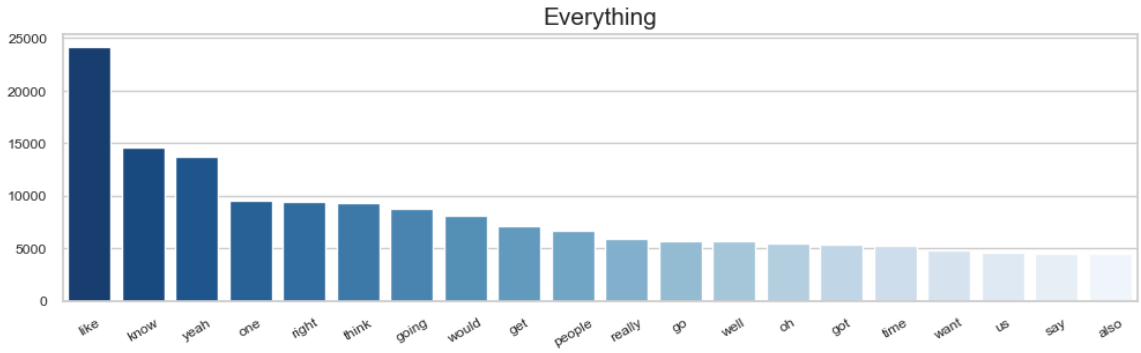

Podcast : JJHO
3000
2500
2000
1500
1000
500
0
like
know
yeah
would
one
right
going
think
get
john
say
got
go
want
judge
jesse
hodgman
time
well
little


Podcast : WgT
6000
5000
4000
3000
2000
1000
0
like
yeah
know
think
one
going
right
oh
would
got
get
go
us
really
want
well
people
movie
time
great


Podcast : Shmanners
6000
5000
4000
3000
2000
1000
0
like
right
know
yeah
okay
think
would
one
oh
people
get
going
go
thing
say
also
well
want
time
really


Podcast : AWeekOfMornings
6000
5000
4000
3000
2000
1000
0
like
know
yeah
going
one
people
think
right
get
would
really
well
time
go
oh
good
us
want
got
wine


Podcast : YoureDeadToMe
2500
2000
1500
1000
500
0
like
know
would
yeah
people
really
one
think
going
well
get
got
also
time
go
mean
right
oh
bit
sort



The frequency distribution analysis of words usage per podcast and then over all podcasts showed that the words *like*, *yeah* and *know* seem to be the most used ones no matter if the podcast is comedy, news or history. Also interesting that there is no difference in word usage between American English (JJHO, Shammers, WgT and AWeekofMornings) and UK English (YoureDeadToMe) visible in the single podcast graphs.

Some further exploration of the token lists shows obvious errors in the transcription process where one and the same introduction has different outcomes. It is entirely possible that this word mangling could also happen to spoken Christmas related words. More data gathering would be necessary to find out how much of the text corpus has been transcribed wrong. For this analysis this problem has been noted and ignored in the next steps.

```
['welcome', 'judge', 'john', 'hodgman', 'podcast', 'bale',
'jesse', 'thorn', 'chambers', 'week']
['welcome', 'judge', 'john', 'hodgman', 'podcast',
'bailiff', 'jesse', 'thorne', 'chambers', 'week']
['welcome', 'judge', 'john', 'hodgman', 'podcast', 'bill',
'jesse', 'thorn', 'chambers', 'week']
['welcome', 'judge', 'john', 'hodgman', 'podcast',
'bailiff', 'jesse', 'thorn', 'chambers', 'week']
```

Analysis

In [11]:
```
# calculate the christmasy score per podcast episode

x_word_list = x_words.keys()

# return the indices of list items in list_to_search that match the item_
def find_indices(list_to_search, item_to_find):
    indices = []
    # go over all items in the list
    for idx, val in enumerate(list_to_search):
        # if the list item matches the search remember the index
        if val == item_to_find:
            indices.append(idx)
    return indices

# calculate the (raw) Christmas score
# since Christmas words are not always single words but some times
# word combinations this function searches for the index of the
```

```python
# match of the first word of a combination and then tries to
# expand the match
def x_score(words):
    score = 0
    match = []
    all_indices = []
    for entry in x_word_list:
        parts = entry.split()
        indices = find_indices(words, parts[0])
        if len(indices) > 0:
            all_indices.extend(indices)
            if len(parts) == 1:
                score += 1
                match.append(parts[0])
            else:
                full_match = True
                for p in range(1, len(parts)):
                    for i in indices:
                        if i + p < len(words):
                            if words[i + p] != parts[p] or not full_match
                                full_match = False
                        else:
                            full_match = False
                if full_match:
                    score += 1
                    match.append(entry)
    # calc term freqency and bring value in plottable range
    score = int((score / len(words)) * 10000)
    return score, match, all_indices

# a function that returns the mean distance between christmas related wor
# and standard deviation
# after some exploratory analysis it has not been used in the report thou
# because the values did not support the hypothesis of clustered christma
def x_term_stats(indices):
    distances = []
    for i in range(0, len(indices)):
        for l in range(i+1, len(indices)):
            distances.append(abs(indices[i] - indices[l]))
    if len(distances) > 1: return sum(distances) / len(distances), statis
    return 0

# count of podcasts that are christmasy
cast_count = 0

print('This will take a bit again.')

# loop over all podcasts and episodes
for pod in casts.keys():
    for month in sorted(casts[pod].keys()):
        for episode in casts[pod][month]:
            # get the christmasy score and matches
            score, match, indices = x_score(episode['text'])
            if score:
                # store the score and match info per podcast
                episode['score'] = score
                episode['match'] = match
                episode['indices'] = indices
                episode['stats'] = x_term_stats(indices)
                cast_count += 1
```

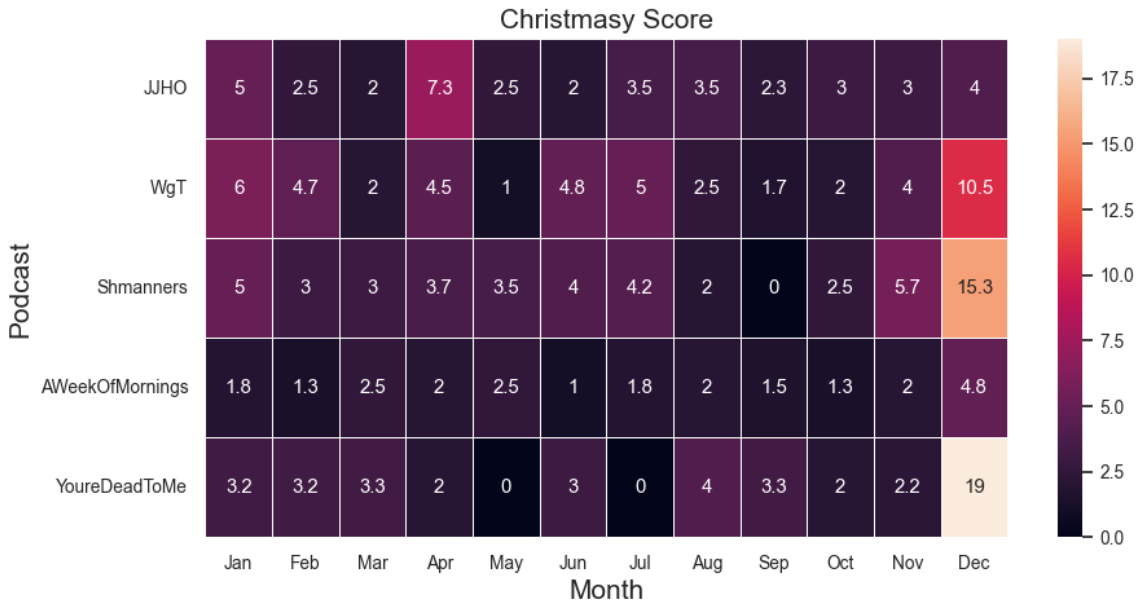
```
                # sample output
                #print(score, pod, month, episode['title'], match)
                #if pod == 'JJHO' and month == '04':
                #    print(score, pod, month, episode['title'], match)
                #print(score, pod, month, episode['title'], episode['stat

print('Found', cast_count, 'podcasts to be christmasy.')
```

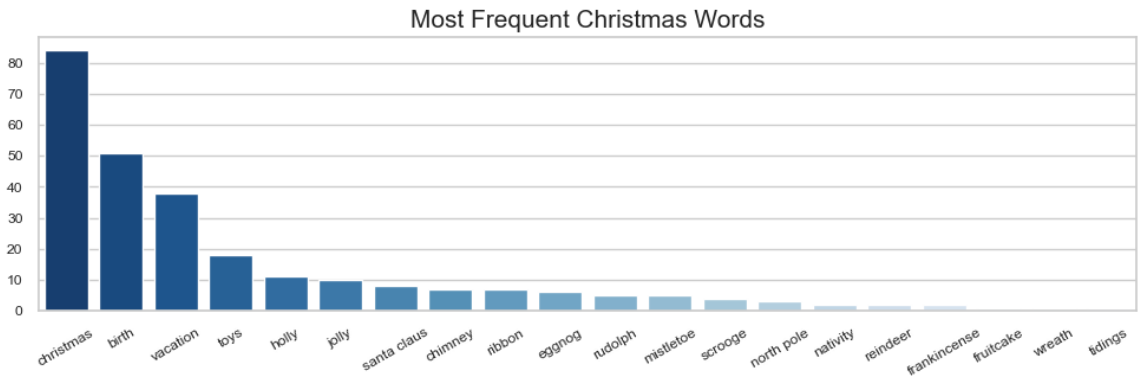
```
This will take a bit again.
Found 157 podcasts to be christmasy.
```

In [12]:

```
# build data matrix from x_scores

# initialize empty matrix (will be 2D when done)
matrix = []
high = 0
matches = []

# loop over every podcast
for pod in casts.keys():
    # initialize empty row list
    row = []
    # loop over all month in this case SORTED for the final plot
    #for month in sorted(casts[pod].keys()):
    for m in range(1, 13):
        month = str(m)
        # prefix with zero if necessary
        if len(str(m)) == 1: month = '0' + str(m)
        x_sum = 0
        episode_count = 0;
        # check if the key exists
        if month in casts[pod].keys():
            for episode in casts[pod][month]:
                # sum up all episode x_scores
                if 'score' in episode.keys():
                    score = episode['score']
                    # store highest value for plot
                    #if score > high: high = score
                    x_sum += score
                    matches.extend(episode['match'])
                    episode_count += 1
        # take the mean of the sum over all episodes in this month
        if episode_count: x_sum = x_sum / episode_count
        # store highest value for plot
        if x_sum > high: high = x_sum
        # threshold would emphasis coloration
        #if x_sum < 50: x_sum *= 0.5
        #if x_sum > 4.5: x_sum *= 1.5
        # add the score to the row
        row.append(round(x_sum, 1))
    # add the row to the matrix
    matrix.append(row)

# axis podcast names
cast_names = casts.keys()
# axis month names
months = [ 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
# prepare the matrix as dataframe
matrix_df = pd.DataFrame(matrix, columns=months, index=cast_names)
```

```
# set up plot
plt.figure(figsize=(10, 5))
sns.set(font_scale=0.9)

# create seaborn heat map
hm = sns.heatmap(matrix_df,
            #cmap='coolwarm',
            annot=True,
            fmt='.5g',
            vmax=high,
            linewidths=.5,
            linecolor='white')

# add some addinional labels
plt.xlabel('Month',fontsize=15)
plt.ylabel('Podcast',fontsize=15)
hm.axes.set_title('Christmasy Score', fontsize=15)

plt.show()

analyse_and_plot(matches, 'Most Frequent Christmas Words')
```





Conclusion

For the first analysis attempt almost the entire Christmas word list was used. The result was surprising at first. The heat map for all podcasts and all month showed only high temperature entries. On second thought it became bluntly obvious. Even

though the Christmas word list contains only words that are associated with Christmas many of those words also are in day to day use. Therefore a lot of matches in the podcast corpus that are not meant to be Christmas related have shown up. To account for this the word list has been trimmed down appropriately. Still there was an unanticipated high percentage of high temp entries in the heat map visible but curiously only for podcasts from the Maxfun network. In this analysis attempt dynamic ad insertion was the culprit. The download of the podcasts happened when seasonal ads have already been released and inserted therefore skewing the results. Skimming through the podcasts to find start and stop words for commercial breaks made it possible to compile a simple regular expression to filter those ads out.

The final analysis attempt proves the expected result of an increase in Christmas related word usage and therefore Christmasyness in December in all podcasts. Obviously in some podcasts more than in others. Some podcasts also show higher values in January. The hosts and guests were still talking about Christmas at that time increasing the hit score. The general level of Christmasyness visible for all podcasts through out the year shows an overlap of the Christmas word list with words in regular use. This could easily be dealt with by setting a threshold. There is one outlier in the *JJHO* podcast in April. But a closer look showed that the episodes in this month scored relatively high in term frequency with clear Christmas related terms. So even thought it's an outlier it is because of Christmas content which proves that the analysis works correct.

```
[score] [podcast] [month] [title] [matches]
5 JJHO 04 A Donut of One's Own ['christmas', 'birth',
'santa claus']
8 JJHO 04 This Illustrated Man ['christmas', 'birth',
'chimney']
```

Finally the frequency distribution of the words from the Christmas word list matched to the podcasts highlights that *Christmas* is indeed the most used word closely followed by *birth* and *vacation*. Aside from the name of the holiday (*Christmas*) there is still talk about the religious aspect (*birth*) even though Christmas can be seen as general cultural event now. Over all the results confirm the assumptions that December and to a degree January are the two month were Christmas is in our minds a lot.

Further Research Is Needed