CWordTM Toolkit Usage on the Holy Bible (CUV)

This Jupyter notebook demonstrates how to use the package "CWordTM" on the Holy Bible (Chinese Union Version - Traditional Chinese):

1. Utility Features
2. Text Visualization - Word Cloud
3. Pivot Table
4. OT Quotes
5. Topic Modeling - BERTopic

CWordTM Toolkit's Documentation: https://cwordtm.readthedocs.io

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Import the Package CWordTM
import cwordtm
from cwordtm import *
```

1. Utility Features

In [3]:
```
# Load the whole Bible (Chinese Union Version)
bible = "cuv.csv"
cdf = util.load_word(bible, info=True)
```

```
Loading file 'C:\Dev\Anaconda3\envs\aiml\lib\site-packages\cwordtm\data\cuv.csv' ...

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31102 entries, 0 to 31101
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   book       31102 non-null  object
 1   book_no    31102 non-null  int64 
 2   chapter    31102 non-null  int64 
 3   verse      31102 non-null  int64 
 4   text       31102 non-null  object
 5   testament  31102 non-null  int64 
 6   category   31102 non-null  object
 7   cat        31102 non-null  object
 8   cat_no     31102 non-null  int64 
dtypes: int64(5), object(4)
memory usage: 2.1+ MB
```

Extract Partial Scripture

In [4]:
```
util.bible_cat_info()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[4]:

	category	cat	book_list	nbooks
0	Torah	tor	Gen Exo Lev Num Deu	5
1	OT History	oth	Jos Jug Rut 1Sa 2Sa 1Ki 2Ki 1Ch 2Ch Ezr Neh Est	12
2	Ketuvim	ket	Job Psm Pro Ecc Son	5
3	Major Prophets	map	Isa Jer Lam Eze Dan	5
4	Minor Prophets	mip	Hos Joe Amo Oba Jon Mic Nah Hab Zep Hag Zec Mal	12
5	Gospel	gos	Mat Mak Luk Jhn	4
6	NT History	nth	Act	1
7	Pauline Epistles	pau	Rom 1Co 2Co Gal Eph Phl Col 1Ts 2Ts 1Ti 2Ti Ti...	13
8	General Epistles	epi	Heb Jas 1Pe 2Pe 1Jn 2Jn 3Jn Jud	8
9	Apocalypse	apo	Rev	1

In [5]:
```
# Extract Gospels (The first four book in NT)
gos = util.extract(cdf, category='gos')
gos.head()
```

Out[5]:

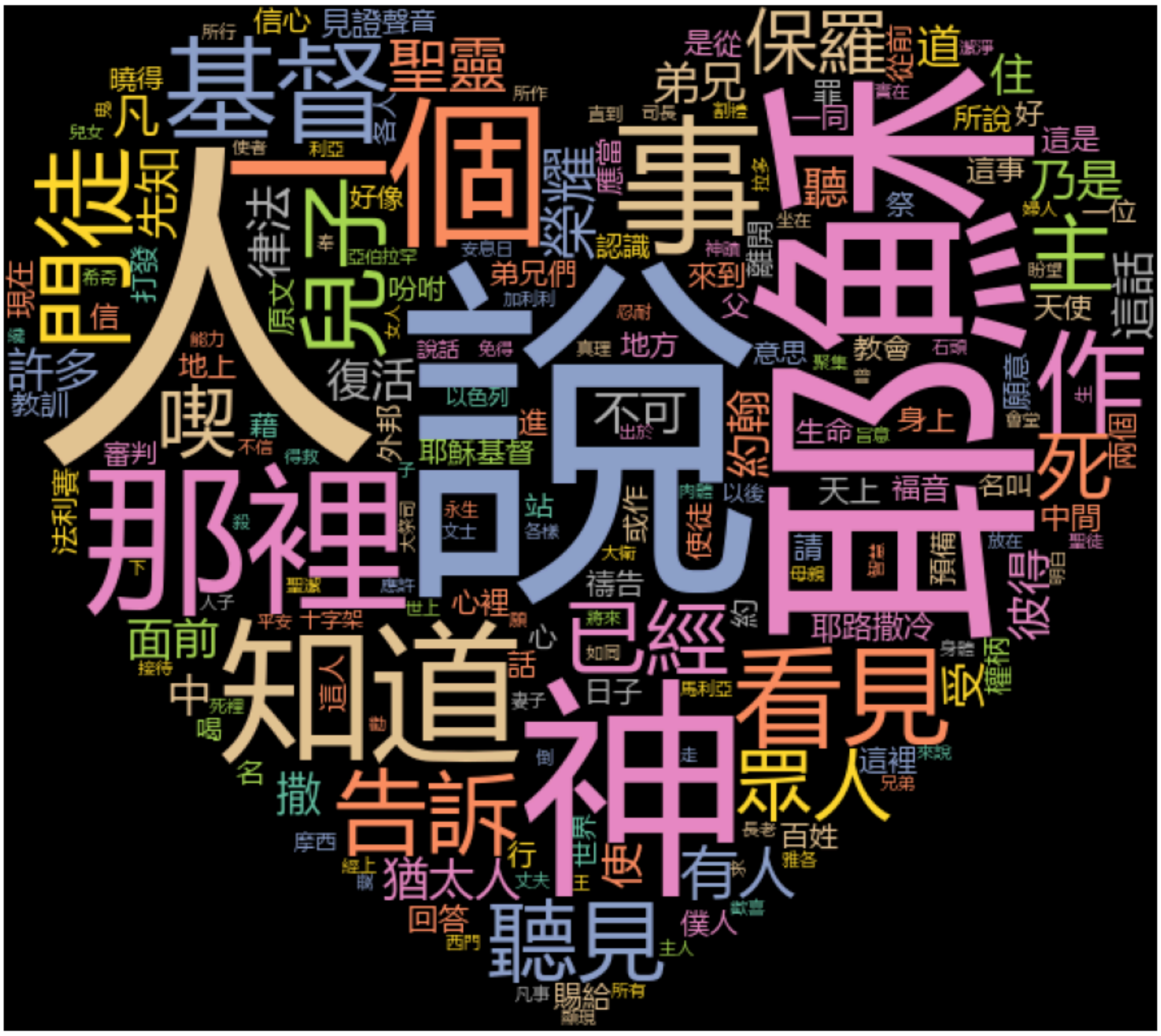
	book	book_no	chapter	verse	text	testament	category	cat	cat_no
23145	Mat	40	1	1	亞伯拉罕的後裔、大衛的子孫、耶穌基督的家譜．〔後裔子孫原文都作兒子下同〕	1	Gospel	gos	5
23146	Mat	40	1	2	亞伯拉罕生以撒．以撒生雅各．雅各生猶大和他的弟兄．	1	Gospel	gos	5
23147	Mat	40	1	3	猶大從他瑪氏生法勒斯和謝拉．法勒斯生希斯崙．希斯崙生亞蘭．	1	Gospel	gos	5
23148	Mat	40	1	4	亞蘭生亞米拿達．亞米拿達生拿順．拿順生撒門．	1	Gospel	gos	5
23149	Mat	40	1	5	撒門從喇合氏生波阿斯．波阿斯從路得氏生俄備得．俄備得生耶西．	1	Gospel	gos	5

2. Text Visualization - Word Cloud

In [6]:
```
# Extract the NT Scripture for Word Cloud
text_list = util.get_text_list(cdf[cdf.testament==1])   # Load New Testament Scripture

# Use internal image mask
viz.chi_wordcloud(text_list, bg='black', image=1)
```

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\User\AppData\Local\Temp\jieba.cache
Loading Chinese vocabulary 'C:\Dev\Anaconda3\envs\aiml\lib\site-packages\cwordtm\data\bible_voca
b.txt' ...
Loading model cost 1.461 seconds.
Prefix dict has been built successfully.
Building prefix dict from C:\Dev\Anaconda3\envs\aiml\lib\site-packages\cwordtm\dictionary\dict.tx
t.big.txt ...
Loading model from cache C:\Users\User\AppData\Local\Temp\jieba.ufaf52121053d30f6b6740fa1422773b
4.cache
Loading model cost 2.480 seconds.
Prefix dict has been built successfully.
C:\Dev\Anaconda3\envs\aiml\lib\site-packages\wordcloud\wordcloud.py:106: MatplotlibDeprecationWar
ning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two minor releas
es later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap(obj)`` instead.
  self.colormap = plt.cm.get_cmap(colormap)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [7]:
```
# Show source code of a function without execution
viz.chi_wordcloud(text_list, bg='black', image=1, code=2)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
def chi_wordcloud(docs, figsize=(15, 10), bg='white', image=0):
    """Prepare and show a Chinese wordcloud

    :param docs: The collection of Chinese documents for preparing a wordcloud,
        default to None
    :type docs: pandas.DataFrame
    :param figsize: Size (width, height) of word cloud, default to (15, 10)
    :type figsize: tuple, optional
    :param bg: The background color (name) of the wordcloud, default to 'white'
    :type bg: str, optional
    :param image: The filename of the presribed image as the mask of the wordcloud,
        or 1/2/3/4 for using an internal image (heart / disc / triangle / arrow),
        default to 0 (No image mask)
    :type image: int or str, optional
    """

    util.set_lang('chi')
    diction = util.get_diction(docs)

    masks = ['heart.jpg', 'disc.jpg', 'triangle.jpg', 'arrow.jpg']

    if image == 0:
        mask = None
    elif image in [1, 2, 3, 4]:  # Internal image file
        img_file = files('cwordtm.images').joinpath(masks[image-1])
        mask = np.array(Image.open(img_file))
    elif isinstance(image, str) and len(image) > 0:
        mask = np.array(Image.open(image))
    else:
        mask = None

    font_file = files('cwordtm.data').joinpath('msyh.ttc')
    wordcloud = WordCloud(background_color=bg, colormap='Set2',
                          mask=mask, font_path=str(font_file)) \
                    .generate_from_frequencies(frequencies=diction)

    plot_cloud(wordcloud, figsize=figsize)


>> cwordtm.util.set_lang
def set_lang(lang='en'):
    """Sets the prescribed language (English or Chinese (Traditional))
    for further text processing.

    :param lang: The prescribed language for text processing, where
        'en' stands for English or 'chi' for Traditonal Chinese,
        default to 'en'
    :type lang: str, optional
    """

    global glang, stops
    glang = lang
    if glang == 'en':  # English
        stops = set(stopwords.words("english"))
    else:  # Chinese (Traditional)
        add_chi_vocab()
        stops = chi_stops()
        chi_flag = True

>> cwordtm.util.get_diction
def get_diction(docs):
    """Determines which is the target language, English or Chinese,
    in order to build a dictionary of words with their frequencies.

    :param docs: The collection of documents, default to None
    :type docs: pandas.DataFrame or list
    :return: The dictionary of words with their frequencies
    :rtype: dict
    """

    if glang == 'en':
        return get_diction_en(docs)
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js
        return get_diction_chi(docs)
```

```
>> cwordtm.viz.plot_cloud
def plot_cloud(wordcloud, figsize):
    """Plot the prepared 'wordcloud'
    :param wordcloud: The WordCloud object for plotting, default to None
    :type wordcloud: WordCloud object
    :param figsize: Size (width, height) of word cloud, default to None
    :type figsize: tuple
    """

    plt.figure(figsize=figsize)
    plt.imshow(wordcloud)
    plt.axis("off");
```

3. Pivot Table

Show Bible Scripture Statistics through a Pivot Table

In [8]:
```
util.set_rows()
pivot.stat(cdf, chi=True)
```

```
Book category information can be shown by invoking 'util.bible_cat_info()'
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[8]:

			chapter	verse	text
category	**book_no**	**book**			
Torah	**1**	**Gen**	50	1533	51460
	2	**Exo**	40	1213	40057
	3	**Lev**	27	859	29228
	4	**Num**	36	1288	41654
	5	**Deu**	34	959	35904
Sub-Total			187	5852	198303
OT History	**6**	**Jos**	24	658	25794
	7	**Jug**	21	618	24375
	8	**Rut**	4	85	3362
	9	**1Sa**	31	810	32218
	10	**2Sa**	24	695	26697
	11	**1Ki**	22	816	30530
	12	**2Ki**	25	719	29626
	13	**1Ch**	29	942	30231
	14	**2Ch**	36	822	33445
	15	**Ezr**	10	280	10094
	16	**Neh**	13	406	14739
	17	**Est**	10	167	6571
Sub-Total			249	7018	267682
Ketuvim	**18**	**Job**	42	1070	24294
	19	**Psm**	150	2461	64303
	20	**Pro**	31	915	19986
	21	**Ecc**	12	222	7228
	22	**Son**	8	117	3979
Sub-Total			243	4785	119790
Major Prophets	**23**	**Isa**	66	1292	50630
	24	**Jer**	52	1364	58005
	25	**Lam**	5	154	4721
	26	**Eze**	48	1273	49218
	27	**Dan**	12	357	14646
Sub-Total			183	4440	177220
Minor Prophets	**28**	**Hos**	14	197	7455
	29	**Joe**	3	73	2642
	30	**Amo**	9	146	5487
	31	**Oba**	1	21	838
	32	**Jon**	4	48	1675
	33	**Mic**	7	105	4307
	34	**Nah**	3	47	1716
	35	**Hab**	3	56	1974
	36	**Zep**	3	53	2216

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

			chapter	verse	text
category	**book_no**	**book**			
	37	**Hag**	2	38	1484
	38	**Zec**	14	211	8235
	39	**Mal**	4	55	2658
Sub-Total			67	1050	40687
Gospel	**40**	**Mat**	28	1071	32774
	41	**Mak**	16	678	20482
	42	**Luk**	24	1151	35526
	43	**Jhn**	21	878	27378
Sub-Total			89	3778	116160
NT History	**44**	**Act**	28	1007	33762
Sub-Total			28	1007	33762
Pauline Epistles	**45**	**Rom**	16	433	14365
	46	**1Co**	16	437	14154
	47	**2Co**	13	257	9260
	48	**Gal**	6	149	4817
	49	**Eph**	6	155	4591
	50	**Phl**	4	104	3408
	51	**Col**	4	95	3312
	52	**1Ts**	5	89	2983
	53	**2Ts**	3	47	1624
	54	**1Ti**	6	113	3579
	55	**2Ti**	4	83	2639
	56	**Tit**	3	46	1544
	57	**Phm**	1	25	717
Sub-Total			87	2033	66993
General Epistles	**58**	**Heb**	13	303	10428
	59	**Jas**	5	108	3534
	60	**1Pe**	5	105	3905
	61	**2Pe**	3	61	2343
	62	**1Jn**	5	105	3779
	63	**2Jn**	1	13	463
	64	**3Jn**	1	15	478
	65	**Jud**	1	25	1030
Sub-Total			34	735	25960
Apocalypse	**66**	**Rev**	22	404	15606
Sub-Total			22	404	15606
	Total		1189	31102	1062163

4. OT Quotes

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Identify Cited Sources in OT Scripture for Some NT verses

In [9]:
```
rom10 = util.extract2(cdf, 'Rom 10')
quot.show_quot(rom10, lang='chi')
```

```
Loading Chinese vocabulary 'C:\Dev\Anaconda3\envs\aiml\lib\site-packages\cwordtm\data\bible_voca
b.txt' ...
Loading file 'C:\Dev\Anaconda3\envs\aiml\lib\site-packages\cwordtm\data\cuv.csv' ...
Building prefix dict from C:\Dev\Anaconda3\envs\aiml\lib\site-packages\cwordtm\dictionary\dict.tx
t.big.txt ...
Loading model from cache C:\Users\User\AppData\Local\Temp\jieba.ufaf52121053d30f6b6740fa1422773b
4.cache
Loading file 'C:\Dev\Anaconda3\envs\aiml\lib\site-packages\cwordtm\data\book_categories.csv' ...
( 1) 羅 10:5 摩西寫著說、『人若行那出於律法的義、就必因此活著。』
Loading model cost 2.516 seconds.
Prefix dict has been built successfully.
( 2) 羅 10:6 惟有出於信心的義如此說、『你不要心裡說、誰要升到天上去呢．就是要領下基督來．
( 3) 羅 10:8 他到底怎麼說呢．他說、『這道離你不遠、正在你口裡、在你心裡。』就是我們所傳信主的道。
( 4) 羅 10:11 經上說、『凡信他的人、必不至於羞愧。』
( 5) 羅 10:13 因為『凡求告主名的、就必得救。』
( 6) 羅 10:15 若沒有奉差遣、怎能傳道呢．如經上所記、『報福音傳喜信的人、他們的腳蹤何等佳美。』
( 7) 羅 10:16 只是人沒有都聽從福音．因為以賽亞說、『主阿、我們所傳的有誰信呢。』
     -> 0.6723 賽 53:1    我們所傳的、〔或作所傳與我們的〕有誰信呢．耶和華的膀臂向誰顯露呢。
( 8) 羅 10:18 但我說、人沒有聽見麼．誠然聽見了．『他們的聲音傳遍天下、他們的言語傳到地極。』
( 9) 羅 10:19 我再說、以色列人不知道麼．先有摩西說、『我要用那不成子民的、惹動你們的憤恨．我要用那無知的
民、觸動你們的怒氣。』
     -> 0.5403 申 32:21    他們以那不算為神的、觸動我的憤恨、以虛無的神、惹了我的怒氣、我也要以那不成子民
的、觸動他們的憤恨、以愚昧的國民、惹了他們的怒氣。
(10) 羅 10:20 又有以賽亞放膽說、『沒有尋找我的、我叫他們遇見．沒有訪問我的、我向他們顯現。』
     -> 0.6651 賽 65:1    素來沒有訪問我的、現在求問我．沒有尋找我的、我叫他們遇見．沒有稱為我名下的、我
對他們說、我在這裡、我在這裡。
(11) 羅 10:21 至於以色列人、他說、『我整天伸手招呼那悖逆頂嘴的百姓。』
     -> 0.6086 賽 65:2    我整天伸手招呼那悖逆的百姓、他們隨自己的意念行不善之道．
```

5. Topic Modeling

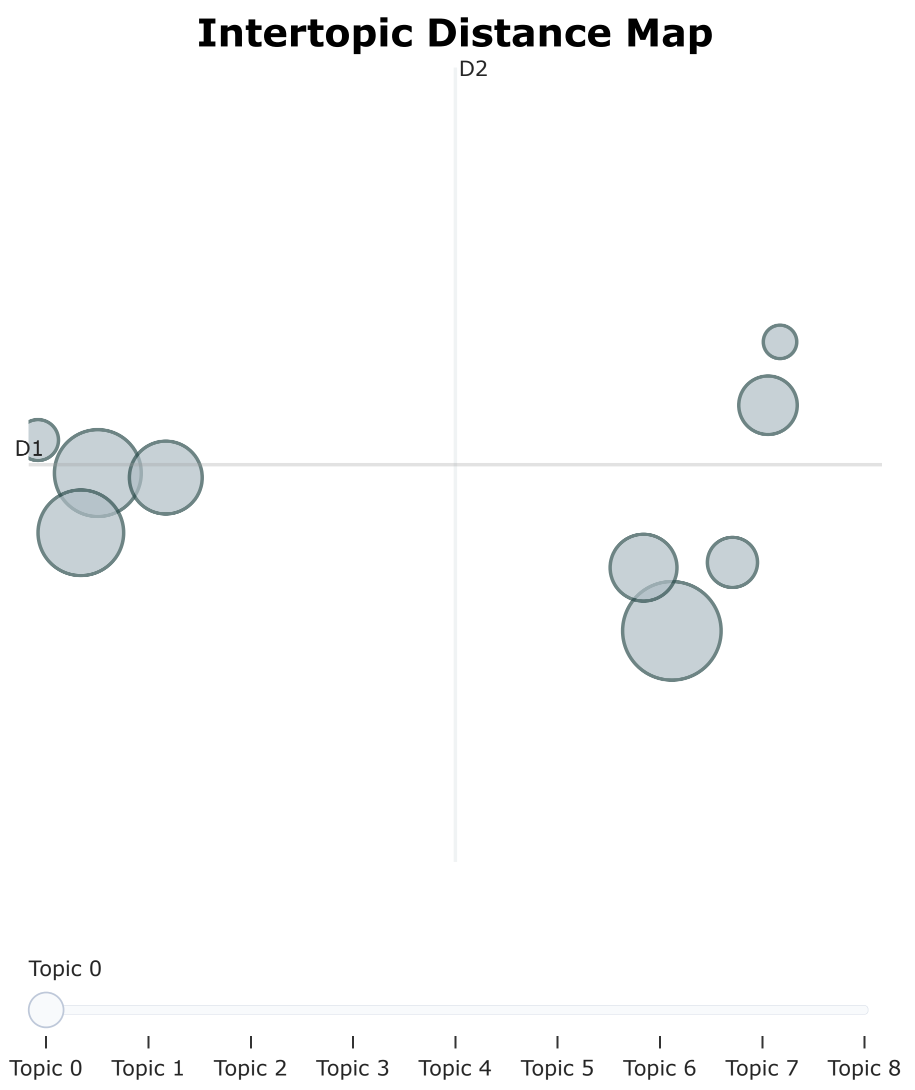
In [10]:
```
import warnings
warnings.filterwarnings('ignore')
```

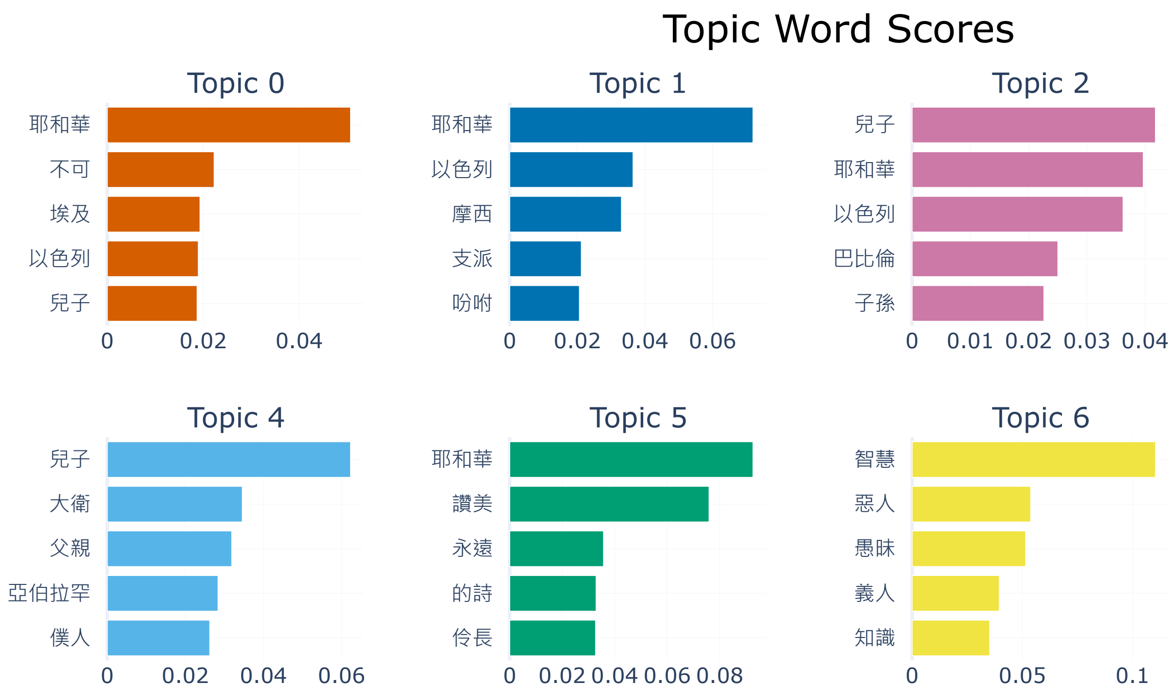
BERTopic Modeling

In [11]:
```
# Build a BERTopic model on the OT Scripture
btm = tm.btm_process(bible, cat=1, chi=True, eval=True, timing=True)
```

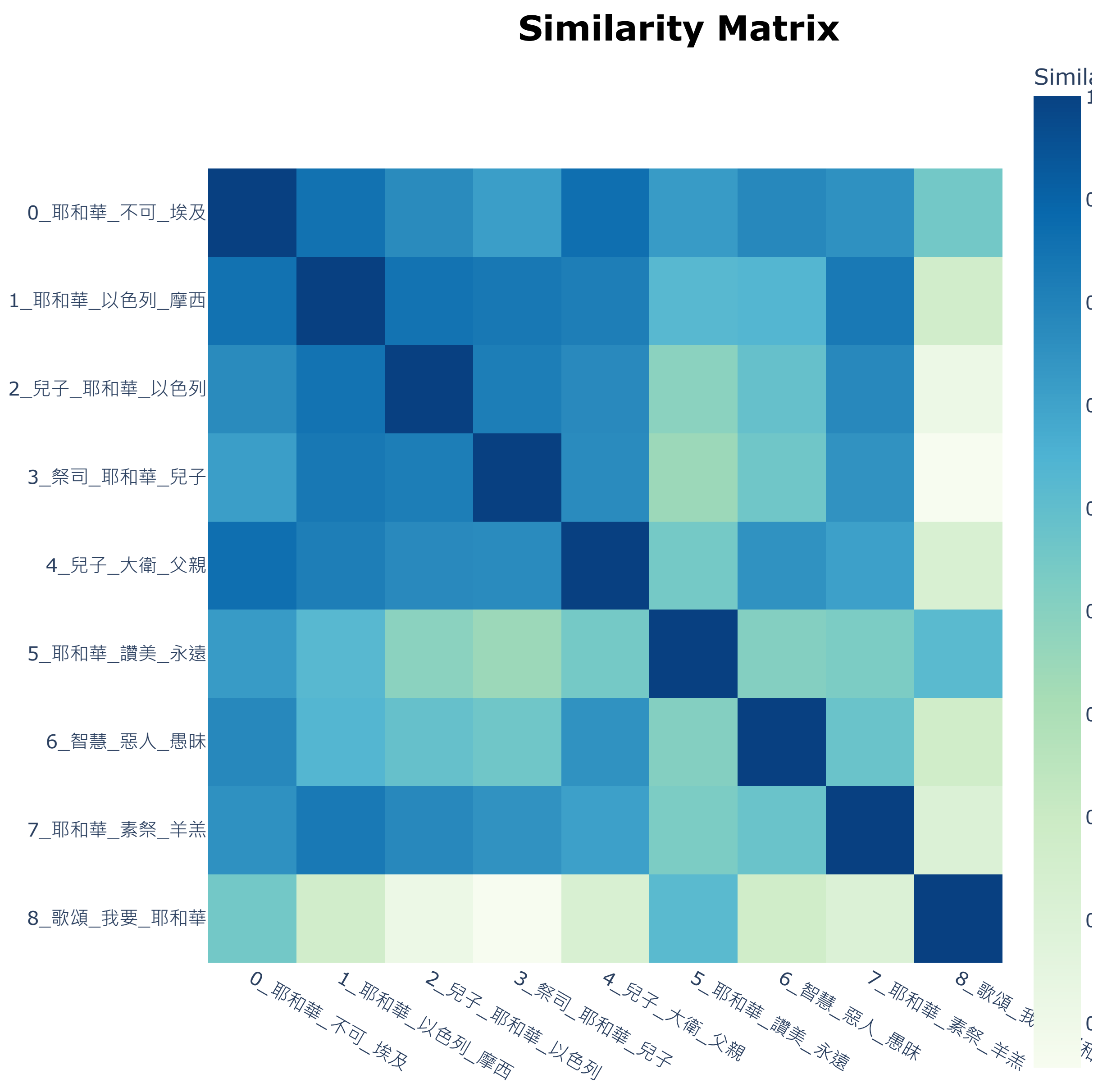
```
Loading Bible 'C:\Dev\Anaconda3\envs\aiml\lib\site-packages\cwordtm\data\cuv.csv' ...
Corpus loaded!
Chinese text preprocessed!
Text trained!

Topics from BERTopic Model:
Topic 0: 耶和華 | 不可 | 埃及 | 以色列 | 兒子 | 雅各 | 地上 | 面前 | 沒有 | 我要
Topic 1: 耶和華 | 以色列 | 摩西 | 支派 | 吩咐 | 沒有 | 百姓 | 面前 | 埃及 | 屬城
Topic 2: 兒子 | 耶和華 | 以色列 | 巴比倫 | 子孫 | 耶路撒冷 | 猶大 | 大衛 | 萬軍 | 耶利米
Topic 3: 祭司 | 耶和華 | 兒子 | 以色列 | 潔淨 | 利未人 | 子孫 | 燔祭 | 一隻 | 亞倫
Topic 4: 兒子 | 大衛 | 父親 | 亞伯拉罕 | 僕人 | 雅各 | 以掃 | 女子 | 以撒 | 女兒
Topic 5: 耶和華 | 讚美 | 永遠 | 的詩 | 伶長 | 公義 | 之詩 | 交與 | 大衛 | 上行
Topic 6: 智慧 | 惡人 | 愚昧 | 義人 | 知識 | 日光 | 之下 | 虛空 | 言語 | 正直
Topic 7: 耶和華 | 素祭 | 羊羔 | 同獻 | 獻給 | 一隻 | 不可 | 奠祭 | 摩西 | 火祭
Topic 8: 歌頌 | 我要 | 耶和華 | 歡呼 | 萬民 | 稱謝 | 詩歌 | 救恩 | 細拉 | 稱讚

Model Evaluation Scores:
  Coherence: 0.31795382397281424

BERTopic Model Visualization:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js





Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



```
If no visualization is shown,
  you may execute the following commands one-by-one:
    btm.model.visualize_topics()
    btm.model.visualize_barchart()
    btm.model.visualize_heatmap()

Finished 'btm_process' in 247.8438 secs
```

In [12]:
```
# Show source code of the btm_process without execution
btm = tm.btm_process(bible, cat=1, chi=True, eval=True, timing=True, code=2)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
def btm_process(doc_file, num_topics=10, source=0, text_col='text', cat=0, chi=False, group=True,
eval=False):
    """Pipelines the BERTopic modeling.

    :param doc_file: The filename of the prescribed text file to be loaded,
        default to None
    :type doc_file: str
    :param num_topics: The number of topics to be modeled, default to 10
    :type num_topics: int, optional
    :param source: The source of the prescribed document file ('doc_file'),
        where 0 refers to internal store of the package and 1 to external file,
        default to 0
    :type source: int, optional
    :param text_col: The name of the text column to be extracted, default to 'text'
    :type text_col: str, optional
    :param cat: The category indicating a subset of the Scripture to be loaded, where
        0 stands for the whole Bible, 1 for OT, 2 for NT, or one of the ten categories
        ['tor', 'oth', 'ket', 'map', 'mip', 'gos', 'nth', 'pau', 'epi', 'apo'] (See
        the package's internal file 'data/book_cat.csv'), default to 0
    :type cat: int or str, optional
    :param chi: The flag indicating whether the text is processed as Chinese (True)
        or English (False), default to False
    :type chi: bool, optional
    :param group: The flag indicating whether the loaded text is grouped by chapter,
        default to True
    :type group: bool, optional
    :param eval: The flag indicating whether the model evaluation results will be shown,
        default to False
    :type eval: bool, optional
    :return: The pipelined BTM
    :rtype: cwordtm.tm.BTM object
    """

    btm = BTM(doc_file, num_topics, chi)
    if source == 0:
        btm.docs = load_bible(btm.textfile, cat=cat, group=group)
    else:
        btm.docs = load_text(btm.textfile, text_col=text_col)

    print("Corpus loaded!")

    if chi:
        btm.preprocess_chi()
        print("Chinese text preprocessed!")
        btm.fit_chi()
    else:
        btm.preprocess()
        print("Text preprocessed!")
        btm.fit()

    print("Text trained!")

    btm.show_topics()

    if eval:
        print("\nModel Evaluation Scores:")
        btm.evaluate()

    btm.viz()

    return btm


>> cwordtm.tm.BTM
class BTM:
    """The BTM object for BERTopic modeling.

    :cvar num_topics: The number of topics to be modeled, default to 10
    :vartype num_topics: int
    :ivar textfile: The filename of the text file to be processed
    :vartype textfile: str
    :ivar chi: The flag indicating whether the processed text is in Chinese or not,
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js Chinese or False for English
    :vartype chi: bool
    :ivar num_topics: The number of topics set for the topic model
    :vartype num_topics: int
    :ivar docs: The collection of the original documents to be processed
    :vartype docs: pandas.DataFrame or list
    :ivar pro_docs: The collection of documents, in form of list of lists of words
        after text preprocessing
    :vartype pro_docs: list
    :ivar dictionary: The dictionary of word ids with their tokenized words
        from preprocessed documents ('pro_docs')
    :vartype dictionary: gensim.corpora.Dictionary
    :ivar corpus: The list of documents, where each document is a list of tuples
        (word id, word frequency in the particular document)
    :vartype corpus: list
    :ivar model: The BERTopic model object
    :vartype model: bertopic.BERTopic
    :ivar embed: The flag indicating whether the BERTopic model is trained
        with the BERT pretrained model
    :vartype embed: bool
    :ivar bmodel: The BERT pretrained model
    :vartype bmodel: transformers.BertModel
    :ivar bt_vectorizer: The vectorizer extracted from the BERTopic model
        for model evaluation
    :vartype bt_vectorizer: sklearn.feature_extraction.text.CountVectorizer
    :ivar bt_analyzer: The analyzer extracted from the BERTopic model
        for model evaluation
    :vartype bt_analyzer: functools.partial
    :ivar cleaned_docs: The list of documents (string) built by grouping
        the original documents by the topics created from the BERTopic model
    :vartype cleaned_docs: list
    """

    def __init__(self, textfile, num_topics, chi=False, embed=True):
        """Constructor method.
        """

        self.textfile = textfile
        self.num_topics = num_topics
        self.chi = chi
        self.docs = None
        self.pro_docs = None
        self.dictionary = None
        self.corpus = None
        self.model = None

        self.embed = embed
        self.bmodel = None
        self.bt_vectorizer = None
        self.bt_analyzer = None
        self.cleaned_docs = None


    def preprocess(self):
        """Process the original English documents (cwordtm.tm.BTM.docs)
        by invoking cwordtm.tm.process_text, and build a dictionary and
        a corpus from the preprocessed documents for the BERTopic model.
        """

        self.pro_docs = [process_text(doc) for doc in self.docs]

        for i, doc in enumerate(self.pro_docs):
            self.pro_docs[i] += ["_".join(w) for w in ngrams(doc, 2)]
            # self.pro_docs[i] += ["_".join(w) for w in ngrams(doc, 3)]

        # Create a dictionary and corpus for the BERTopic model
        self.dictionary = corpora.Dictionary(self.pro_docs)
        self.corpus = [self.dictionary.doc2bow(doc) for doc in self.pro_docs]


    def preprocess_chi(self):
        """Process the original Chinese documents (cwordtm.tm.BTM.docs)
        by tokenizing text, removing stopwords, and building a dictionary
        and a corpus from the preprocessed documents for the BERTopic model.
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
        # Build stop words
        stop_file = files('cwordtm.data').joinpath("tc_stopwords_2.txt")
        stopwords = [k[:-1] for k in open(stop_file, encoding='utf-8')\
                     .readlines() if k != '']

        # Tokenize"the text using Jieba
        dict_file = files('cwordtm.data').joinpath("user_dict_4.txt")
        jieba.load_userdict(str(dict_file))
        docs = [jieba.cut(doc) for doc in self.docs]

        # Replace special characters
        docs = [[word.replace('\u3000', ' ') for word in doc] \
                                     for doc in docs]

        # Remove stop words
        self.pro_docs = [' '.join([word for word in doc if word not in stopwords]) \
                                        for doc in docs]

        self.pro_docs = [doc.split() for doc in self.pro_docs]

        # Create a dictionary and corpus
        self.dictionary = corpora.Dictionary(self.pro_docs)
        self.corpus = [self.dictionary.doc2bow(doc) for doc in self.pro_docs]


    def fit(self):
        """Build the BERTopic model for English text with the created corpus
        and dictionary.
        """

        j_pro_docs = [" ".join(doc) for doc in self.pro_docs]

        if self.embed:
            self.bmodel = BertModel.from_pretrained('bert-base-uncased')
            self.model = BERTopic(language='english',
                                  calculate_probabilities=True,
                                  embedding_model=self.bmodel,
                                  nr_topics=self.num_topics)
        else:
            self.model = BERTopic(language='english',
                                  calculate_probabilities=True,
                                  nr_topics=self.num_topics)

        _, _ = self.model.fit_transform(j_pro_docs)


    def fit_chi(self):
        """Build the BERTopic model for Chinese text with the created corpus
        and dictionary.
        """

        j_pro_docs = [" ".join(doc) for doc in self.pro_docs]

        if self.embed:
            self.bmodel = BertModel.from_pretrained('bert-base-chinese')
            self.model = BERTopic(language='chinese (traditional)',
                                  calculate_probabilities=True,
                                  embedding_model=self.bmodel,
                                  nr_topics=self.num_topics)
        else:
            self.model = BERTopic(language='chinese (traditional)',
                                  calculate_probabilities=True,
                                  nr_topics=self.num_topics)

        _, _ = self.model.fit_transform(j_pro_docs)


    def show_topics(self):
        """Shows the topics with their keywords from the built BERTopic model.
        """

        print("\nTopics from BERTopic Model:")
                                    topic_freq().Topic:
            if topic == -1: continue
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
            twords = [word for (word, _) in self.model.get_topic(topic)]
            print(f"Topic {topic}: {' | '.join(twords)}")


    def pre_evaluate(self):
        """Prepare the original documents per built topic for model evaluation.
        """

        doc_df = pd.DataFrame({"Document": self.docs,
                       "ID": range(len(self.docs)),
                       "Topic": self.model.topics_})
        documents_per_topic = doc_df.groupby(['Topic'], \
                             as_index=False).agg({'Document': ' '.join})
        self.cleaned_docs = self.model._preprocess_text(\
                              documents_per_topic.Document.values)

        # Extract vectorizer and analyzer from BERTopic
        self.bt_vectorizer = self.model.vectorizer_model
        self.bt_analyzer = self.bt_vectorizer.build_analyzer()


    def evaluate(self):
        """Computes and outputs the coherence score.
        """

        try:
            self.pre_evaluate()

            # Extract features for Topic Coherence evaluation
            # words = self.bt_vectorizer.get_feature_names_out()
            tokens = [self.bt_analyzer(doc) for doc in self.cleaned_docs]

            self.dictionary = corpora.Dictionary(tokens)
            self.corpus = [self.dictionary.doc2bow(doc) for doc in tokens]

            topic_words = [[words for words, _ in self.model.get_topic(topic)]
                            for topic in range(len(set(self.model.topics_))-1)]

            coherence = CoherenceModel(topics=topic_words, texts=tokens, corpus=self.corpus,
                                                            dictionary=self.dictionary, coherence
='c_v')\
                        .get_coherence()

            if math.isnan(coherence):
                print("** No coherence score computed!")
            else:
                print(f"  Coherence: {coherence}")
        except:
            print("** No coherence score computed!")


    def viz(self):
        """Visualize the built BERTopic model through Intertopic Distance Map,
        Topic Word Score Charts, and Topic Similarity Matrix.
        """

        print("\nBERTopic Model Visualization:")

        # Intertopic Distance Map
        try:
            self.model.visualize_topics().show()
        except:
            print("** No Intertopic Distance Map shown for your text!")

        # Visualize Terms (Topic Word Scores)
        try:
            self.model.visualize_barchart().show()
        except:
            print("** No chart of Topic Word Scores shown for your text!")

        # Visualize Topic Similarity
        try:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
                               map().show()
        except:
                print("** No heatmap of Topic Similarity shown for your text!")

            print("  If no visualization is shown,")
            print("    you may execute the following commands one-by-one:")
            print("      btm.model.visualize_topics()")
            print("      btm.model.visualize_barchart()")
            print("      btm.model.visualize_heatmap()")
            print()


        def save(self, file):
            """Saves the built BERTopic model to the specified file.

            :param file: The name of the file to store the built model, default to None
            :type file: str
            """

            if file is None or len(file.strip())==0:
                print("No valid filename has been specifid!")
                return

            if file.split('.')[-1] == file:
                file += '.pickle'

            self.model.save(file, serialization="pickle")
            print(f"BERTopic model has been stored in {file!r}.")


        def load(self, file):
            """Loads the stored BERTopic model from the specified file.

            :param file: The name of the file to be loaded, default to None
            :type file: str
            :return: The loaded BERTopic model
            :rtype: bertopic._bertopic.BERTopic
            """

            if file is None or len(file.strip())==0:
                print("No valid filename has been specifid!")
                return

            if file.split('.')[-1] == file:
                file += '.pickle'

            return BERTopic.load(file)

>> cwordtm.tm.load_bible
def load_bible(textfile, cat=0, group=True):
    """Loads and returns the Bible Scripture from the prescribed internal
    file ('textfile').

    :param textfile: The package's internal Bible text from which the text is loaded,
        either World English Bible ('web.csv') or Chinese Union Version (Traditional)
        ('cuv.csv'), default to None
    :type textfile: str
    :param cat: The category indicating a subset of the Scripture to be loaded, where
        0 stands for the whole Bible, 1 for OT, 2 for NT, or one of the ten categories
        ['tor', 'oth', 'ket', 'map', 'mip', 'gos', 'nth', 'pau', 'epi', 'apo'] (See
        the package's internal file 'data/book_cat.csv'), default to 0
    :type cat: int or str, optional
    :param group: The flag indicating whether the loaded text is grouped by chapter,
        default to True
    :type group: bool, optional
    :return: The collection of Scripture loaded
    :rtype: pandas.DataFrame
    """

    # textfile = "web.csv"
    scfile = files('cwordtm.data').joinpath(textfile)
    print("Loading Bible '%s' ..." %scfile)
    df = pd.read_csv(scfile)

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js 'map', 'mip',\
                         'gos', 'nth', 'pau', 'epi', 'apo']
    cat = str(cat)
    if cat == '1' or cat == 'ot':
        df = util.extract(df, testament=0)
    elif cat == '2' or cat == 'nt':
        df = util.extract(df, testament=1)
    elif cat in cat_list:
        df = util.extract(df, category=cat)

    if group:
        # Group verses into chapters
        df = df.groupby(['book_no', 'chapter'])\
                        .agg({'text': lambda x: ' '.join(x)})\
                .reset_index()

    df.text = df.text.str.replace('  ', '')
    return list(df.text)

>> cwordtm.tm.load_text
def load_text(textfile, text_col='text'):
    """Loads and returns the list of documents from the prescribed file ('textfile').

    :param textfile: The prescribed text file from which the text is loaded,
        default to None
    :type textfile: str
    :param text_col: The name of the text column to be extracted, default to 'text'
    :type text_col: str, optional
    :return: The list of documents loaded
    :rtype: list
    """

    # docs = pd.read_csv(textfile)
    docs = util.load_text(textfile)
    return list(docs[text_col])
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js