Live Detection of Topics and Sentiment in Customer Reviews

In this project, I have come up with novel approaches to make sense of *Blenheim Palace*'s customer survey data, with the help of different Natural Language Processing (NLP) and machine learning techniques. The data with a format of having multiple quantitative and free text responses were analyzed using combinations of techniques like clustering, sentiment analysis, topic modelling and semantic analysis to bring out very specific information, which could help Blenheim understand their customer experience better.

> Blenheim Palace is a country house in Woodstock, Oxfordshire, England. It is the seat of the Dukes of Marlborough and the only non-royal, non-episcopal country house in England to hold the title of palace.

Alternative text

After an in-depth exploratory analysis of the data and getting key insights to answer set business questions, the data was used to explore the following:

- Latent Dirichlet Allocation (LDA)
- Topic modelling (BERTopic) - both manual and library implementations
- Sentiment Analysis (Lexicon based and pre-trained models)
- Semantic search and analysis
- Clustering (k-Means algorithm)
- Wordclouds
- Supervised ML models

Importing relevant libraries

In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from wordcloud import WordCloud

#NLP
import random
import re
from pprint import pprint
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel
import spacy
import pyLDAvis
import pyLDAvis.gensim_models
import nltk
from nltk.corpus import stopwords
from nltk import tokenize

from bertopic import BERTopic
from sentence_transformers import SentenceTransformer
import umap
import hdbscan
from sklearn.feature_extraction.text import CountVectorizer

# Semantic analysis
from sentence_transformers import SentenceTransformer
from sentence_transformers import util
import torch


# sentiment analysis
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from transformers import pipeline

# Machine Learning
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

In [2]:
```python
df = pd.read_excel(open('SurveyData_2022-03-04.xlsx', 'rb'),sheet_name='sheet1')
df.head(7)
```

Out[2]:

	VisitDate	SurveyTitle	VisitType	VisitFrequency	ChildrenUnder12	PartySize	StayTime	OverallNPS	CateringNPS	RetailNPS	...	Cafe_Stable
0	2019-12-31	Annual Pass Survey	Paid	I've only visited once	No	3	4 Hours	1.0	-1.0	-1	...	
1	2019-12-31	Annual Pass Survey	Free	Once every 2-3 months	No	2	2 Hours	-1.0	-1.0	-1	...	
2	2020-01-01	Annual Pass Survey	Paid	Once a year	No	1	5 Hours	0.0	0.0	0	...	
3	2020-01-01	Annual Pass Survey	Paid	I've only visited once	Yes	3	5 Hours	1.0	-1.0	-1	...	
4	2019-12-31	Annual Pass Survey	Paid	Once a year	Yes	5	5 Hours	1.0	0.0	0	...	
5	2020-01-01	Annual Pass Survey	Paid	I've only visited once	No	2	3 Hours	1.0	0.0	0	...	
6	2020-01-01	Annual Pass Survey	Paid	I've only visited once	No	3	3 Hours	1.0	0.0	0	...	

7 rows × 35 columns

Data Preparation and EDA

In [3]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17619 entries, 0 to 17618
Data columns (total 35 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   VisitDate                      17619 non-null  datetime64[ns]
 1   SurveyTitle                    17619 non-null  object
 2   VisitType                      17619 non-null  object
 3   VisitFrequency                 17619 non-null  object
 4   ChildrenUnder12                17619 non-null  object
 5   PartySize                      17619 non-null  object
 6   StayTime                       17619 non-null  object
 7   OverallNPS                     17618 non-null  float64
 8   CateringNPS                    8860 non-null   float64
 9   RetailNPS                      17619 non-null  int64
 10  ValueForMoney                  17618 non-null  float64
 11  VisitorSegment                 17619 non-null  object
 12  MostEnjoyed                    17606 non-null  object
 13  CouldImprove                   7059 non-null   object
 14  UseShopMore                    6426 non-null   object
 15  UseCafeMore                    4517 non-null   object
 16  Activity_TookATour             17619 non-null  int64
 17  Activity_VisitedGardens        17619 non-null  int64
 18  Activity_WalkedAroundPark      17619 non-null  int64
 19  Activity_Event                 17619 non-null  int64
 20  Activity_Cafe                  17619 non-null  int64
 21  Activity_Shop                  17619 non-null  int64
 22  Activity_Exhibition            17619 non-null  int64
 23  Activity_Other                 1517 non-null   object
 24  Cafe_Pantry                    17619 non-null  int64
 25  Cafe_Stables                   17619 non-null  int64
 26  Cafe_Pizza                     17619 non-null  int64
 27  Cafe_Orangery                  17619 non-null  int64
 28  Cafe_CoffeeStall               17619 non-null  int64
 29  Cafe_Icecream                  17619 non-null  int64
 30  Service_AdmissionsAndWelcome   17038 non-null  float64
 31  Service_TourGuide              9901 non-null   float64
 32  Service_RetailTeam             9785 non-null   float64
 33  Service_CateringTeam           12534 non-null  float64
 34  Service_AnnualPass             11694 non-null  float64
dtypes: datetime64[ns](1), float64(8), int64(14), object(12)
memory usage: 4.7+ MB
```

In [4]:
```python
df.nunique()
```

Out[4]:
```
VisitDate                         881
SurveyTitle                         1
VisitType                           2
VisitFrequency                      7
ChildrenUnder12                     2
PartySize                           6
StayTime                            6
OverallNPS                          3
CateringNPS                         2
RetailNPS                           3
ValueForMoney                      10
VisitorSegment                      7
MostEnjoyed                     13386
CouldImprove                     6115
UseShopMore                      4395
UseCafeMore                      3890
Activity_TookATour                  2
Activity_VisitedGardens             2
Activity_WalkedAroundPark           2
Activity_Event                      2
Activity_Cafe                       2
Activity_Shop                       2
Activity_Exhibition                 2
Activity_Other                   1281
Cafe_Pantry                         2
Cafe_Stables                        2
Cafe_Pizza                          2
Cafe_Orangery                       2
Cafe_CoffeeStall                    2
Cafe_Icecream                       2
Service_AdmissionsAndWelcome       10
Service_TourGuide                  10
Service_RetailTeam                 10
Service_CateringTeam               10
Service_AnnualPass                 10
dtype: int64
```

> We can see that **SurveyTitle** can be removed cause it contains the same value for all observations.

In [5]:
```python
df.drop(["SurveyTitle"],axis=1,inplace=True)
df.head(5)
```

Out[5]:

	VisitDate	VisitType	VisitFrequency	ChildrenUnder12	PartySize	StayTime	OverallNPS	CateringNPS	RetailNPS	ValueForMoney	...	Cafe_S
0	2019-12-31	Paid	I've only visited once	No	3	4 Hours	1.0	-1.0	-1	7.0	...	
1	2019-12-31	Free	Once every 2-3 months	No	2	2 Hours	-1.0	-1.0	-1	1.0	...	
2	2020-01-01	Paid	Once a year	No	1	5 Hours	0.0	0.0	0	8.0	...	
3	2020-01-01	Paid	I've only visited once	Yes	3	5 Hours	1.0	-1.0	-1	7.0	...	
4	2019-12-31	Paid	Once a year	Yes	5	5 Hours	1.0	0.0	0	8.0	...	

5 rows × 34 columns

In [6]:
```python
df["VisitDate"] = pd.to_datetime(df["VisitDate"])
```

Sorting the observations based on dates, so that we can infer better meanings in terms of events and time specific surveys

In [7]:
```python
df.sort_values(by='VisitDate', inplace=True)
df.reset_index(drop=True)
#df.drop(["index"],axis=1,inplace=True)
```

Out[7]:

	VisitDate	VisitType	VisitFrequency	ChildrenUnder12	PartySize	StayTime	OverallNPS	CateringNPS	RetailNPS	ValueForMoney	...	C
0	2018-10-10	Free	Once every 4-6 months	Yes	6+	6+ Hours	0.0	-1.0	0	8.0	...	
1	2018-12-30	Paid	Once every 4-6 months	No	4	3 Hours	1.0	-1.0	-1	9.0	...	
2	2019-01-05	Free	Once a year	Yes	4	4 Hours	1.0	0.0	1	10.0	...	
3	2019-02-08	Free	Once every 2-3 months	No	2	4 Hours	1.0	0.0	1	8.0	...	
4	2019-05-13	Paid	Once every 2-3 months	Yes	5	3 Hours	1.0	0.0	0	8.0	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
17614	2022-03-01	Free	I've only visited once	No	2	1 Hour	-1.0	0.0	-1	5.0	...	
17615	2022-03-02	Paid	Once a month	No	2	1 Hour	1.0	-1.0	-1	5.0	...	
17616	2022-03-03	Free	Once a year	No	2	2 Hours	1.0	0.0	1	10.0	...	
17617	2022-03-03	Paid	Once every 2-3 months	No	2	4 Hours	0.0	0.0	0	8.0	...	
17618	2022-03-04	Paid	I've only visited once	No	2	5 Hours	1.0	0.0	-1	9.0	...	

17619 rows × 34 columns

In [8]:
```python
# Checkpoint 1
prime_df = df.copy()
prime_df.head(5)
```

Out[8]:

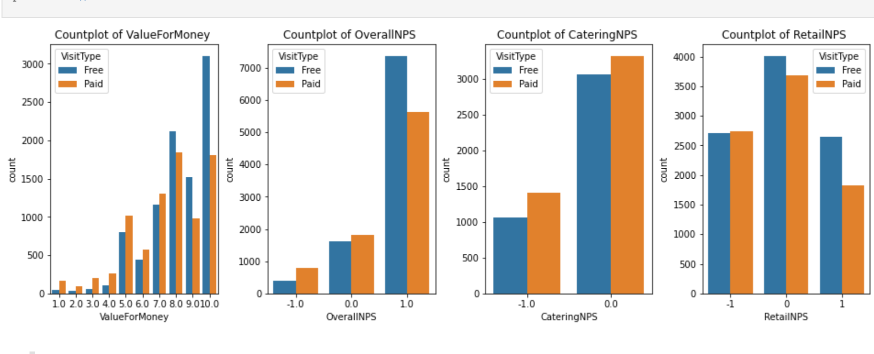
	VisitDate	VisitType	VisitFrequency	ChildrenUnder12	PartySize	StayTime	OverallNPS	CateringNPS	RetailNPS	ValueForMoney	...	Caf
1237	2018-10-10	Free	Once every 4-6 months	Yes	6+	6+ Hours	0.0	-1.0	0	8.0	...	
34	2018-12-30	Paid	Once every 4-6 months	No	4	3 Hours	1.0	-1.0	-1	9.0	...	
398	2019-01-05	Free	Once a year	Yes	4	4 Hours	1.0	0.0	1	10.0	...	
444	2019-02-08	Free	Once every 2-3 months	No	2	4 Hours	1.0	0.0	1	8.0	...	
33	2019-05-13	Paid	Once every 2-3 months	Yes	5	3 Hours	1.0	0.0	0	8.0	...	

5 rows × 34 columns

We separate the free text responses and the quantitative features. This will help us analyze the data better and will come handy further into the project.

In [9]:
```python
# free text responses
text_features = ["MostEnjoyed","CouldImprove","UseShopMore","UseCafeMore","Activity_Other"]
# ratings from 0-10
rating_features = ["ValueForMoney","Service_AdmissionsAndWelcome","Service_TourGuide","Service_RetailTeam",
                   "Service_CateringTeam","Service_AnnualPass"]
```

In [10]:
```python
plt.figure(1,figsize = (16 , 5))
n = 0
for x in ['ValueForMoney' , 'OverallNPS', "CateringNPS", "RetailNPS"]:
    n += 1
    plt.subplot(1 , 4 , n)
    plt.subplots_adjust(hspace =0.5 , wspace = 0.3)
    sns.countplot(df[x] , hue=df["VisitType"])
    plt.title('Countplot of {} '.format(x))
plt.show()
```



> Customers who visited for free, tended to be low in low ratings and high in high ratings when compared to ratings of paid customers.

In [11]:
```python
# Converting PartySize data type to integers from strings for analysis. Will denote 6+ inside analysis, now ju
pSize = []
for i in df["PartySize"]:
    if(i == "6+"):
        j = i.split("+")
        pSize.append(int(j[0]))
    else:
        j = i.split(" ")
        pSize.append(int(j[0]))
# pSize[:10]
df["PartySize"] = pSize
```

In [12]:
```python
# Converting StayTime data type to integers like above. Will denote 6+ inside analysis, now just using 6.
sTime = []
m = 0
for i in df["StayTime"]:
    sTime.append(int(list(i)[0]))
# sTime[:10]
df["StayTime"] = sTime
```

In [13]:
```python
# Checkpoint 2
prime_df = df.copy()
prime_df.head(5)
```

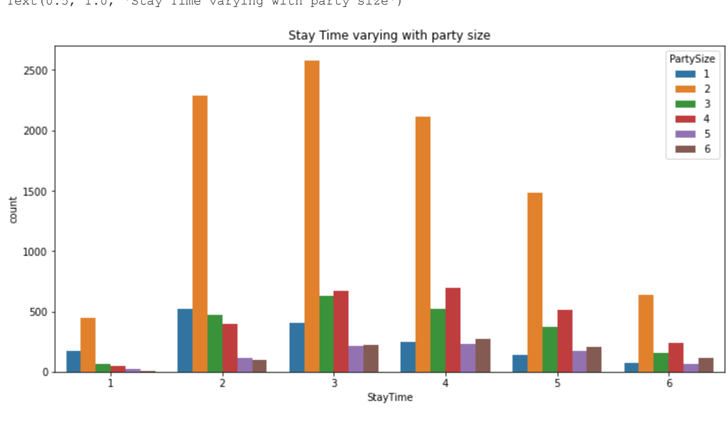
Out[13]:

	VisitDate	VisitType	VisitFrequency	ChildrenUnder12	PartySize	StayTime	OverallNPS	CateringNPS	RetailNPS	ValueForMoney	...	Cat
1237	2018-10-10	Free	Once every 4-6 months	Yes	6	6	0.0	-1.0	0	8.0	...	
34	2018-12-30	Paid	Once every 4-6 months	No	4	3	1.0	-1.0	-1	9.0	...	
398	2019-01-05	Free	Once a year	Yes	4	4	1.0	0.0	1	10.0	...	
444	2019-02-08	Free	Once every 2-3 months	No	2	4	1.0	0.0	1	8.0	...	
33	2019-05-13	Paid	Once every 2-3 months	Yes	5	3	1.0	0.0	0	8.0	...	

5 rows × 34 columns

In [14]:
```python
plt.figure(1,figsize = (12 , 6))
sns.countplot(df["StayTime"],hue=df["PartySize"])
plt.title("Stay Time varying with party size")
```

Out[14]:
```
Text(0.5, 1.0, 'Stay Time varying with party size')
```



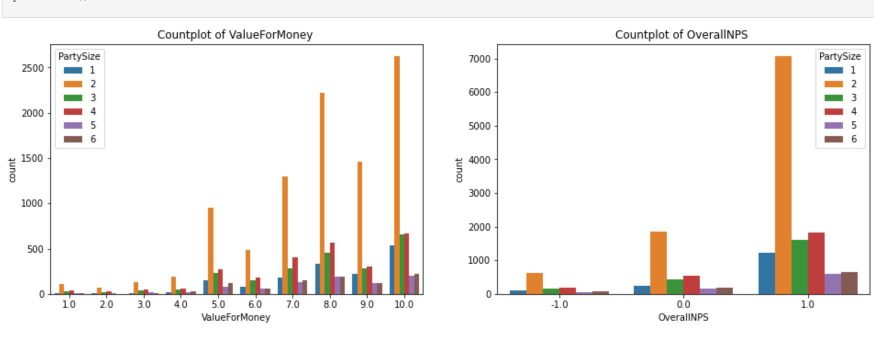
> We can see that usually party size of 2 stays longer than most. Party size of 6+ usually stay from 2-5 hours. More info can be derived from this single visualization.

In [15]:
```python
df["PartySize"].value_counts()
```

Out[15]:
```
2    9546
4    2570
3    2199
1    1557
6     916
5     831
Name: PartySize, dtype: int64
```

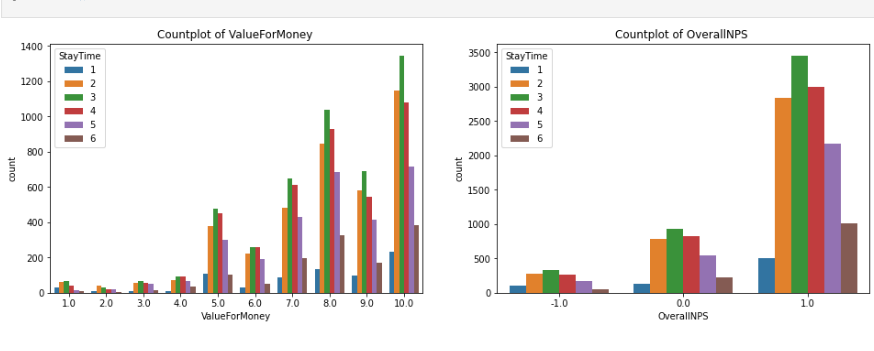
> However, my intuition was correct about the biasness of the data towards party of 2, as number of observations for party size of 2 in this dataset is much more than the other groups.

In [16]:
```python
plt.figure(1,figsize = (16 , 5))
n = 0
for x in ['ValueForMoney' , 'OverallNPS']:
    n += 1
    plt.subplot(1 , 2 , n)
    plt.subplots_adjust(hspace =0.5 , wspace = 0.2)
    sns.countplot(df[x] , hue=df["PartySize"])
    plt.title('Countplot of {} '.format(x))
plt.show()
```



> Next we see for this specific dataset, solo customers usually are low in number compared to larger groups, when it comes to giving a high rating for the experience. Again due to imbalance in sample, we see party size of two dominating the rating space. Groups of 3-4 have usually given pretty high ratings of the experience.

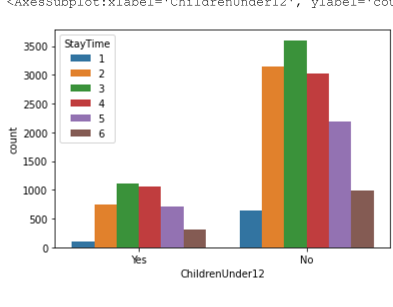
In [17]:
```python
plt.figure(1,figsize = (16 , 5))
n = 0
for x in ['ValueForMoney' , 'OverallNPS']:
    n += 1
    plt.subplot(1 , 2 , n)
    plt.subplots_adjust(hspace =0.5 , wspace = 0.2)
    sns.countplot(df[x] , hue=df["StayTime"])
    plt.title('Countplot of {} '.format(x))
plt.show()
```



> Another observation is that people who have stayed around 2-5 hours have given much higher ratings.

In [18]:
```python
sns.countplot(df["ChildrenUnder12"],hue = df["StayTime"])
```

Out[18]:
```
<AxesSubplot:xlabel='ChildrenUnder12', ylabel='count'>
```

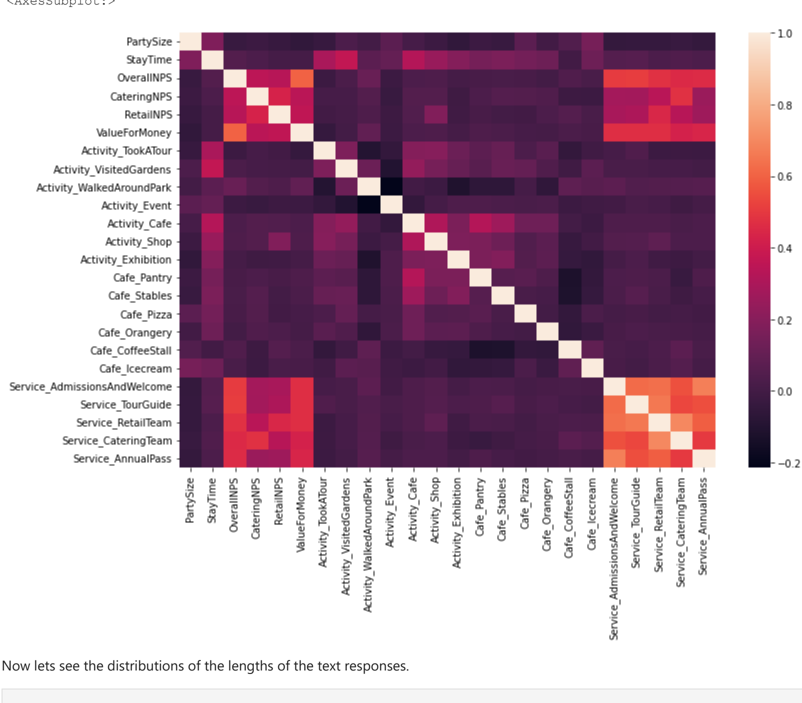


> Groups without children under 12 tended to stay longer than those with children under 12.

Plotting the heatmap of the correlations between all the features, we can see an interesting phenomenon that there are correlations between the NPS scores and the overall satisfaction of the experience with services ratings.

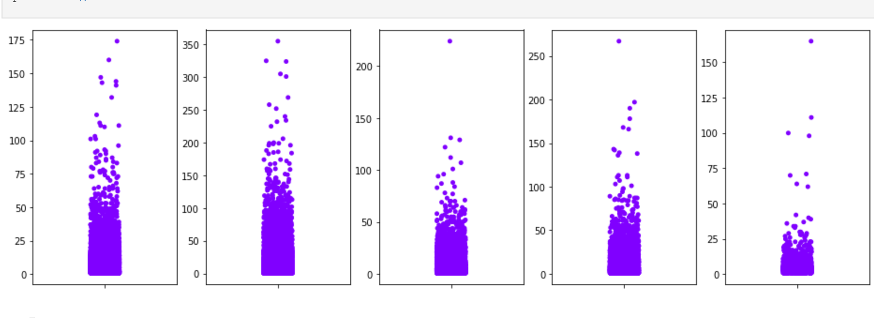
In [19]:
```python
plt.figure(1,figsize = (12 , 8))
sns.heatmap(df.corr())
```

Out[19]:
```
<AxesSubplot:>
```



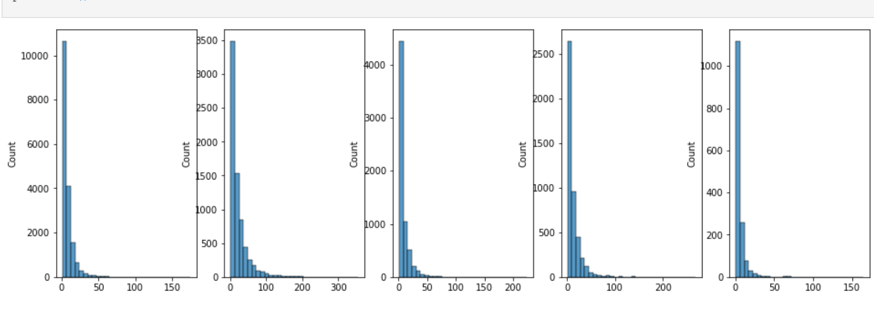
Now lets see the distributions of the lengths of the text responses.

In [20]:
```python
def response_lengths(col):
    feature = list(col.dropna())
    leng = []
    for i in feature:
        s = i.split(" ")
        leng.append(len(s))
    return leng

# a list to store the collective lists of frequencies of the individual text features
text_cols = []
for c in text_features:
    text_cols.append(response_lengths(df[c]))

# finally plotting now
plt.figure(1,figsize = (16 , 5))
n = 0
for x in text_cols:
    n += 1
    plt.subplot(1 , 5 , n)
    plt.subplots_adjust(hspace =0.5 , wspace = 0.2)
    sns.stripplot(y= x,palette=["#8000ff"],jitter=1)
    #plt.title('Countplot of {} '.format(x))
plt.show()
```



> the strip plots showing the lengths of the text responses in each of the columns

In [21]:
```python
plt.figure(1,figsize = (16 , 5))
n = 0
for x in text_cols:
    n += 1
    plt.subplot(1 , 5 , n)
    plt.subplots_adjust(hspace =0.5 , wspace = 0.2)
    #sns.kdeplot(x)
    sns.histplot(x,bins=30)
    #plt.title('Countplot of {} '.format(x))
plt.show()
```



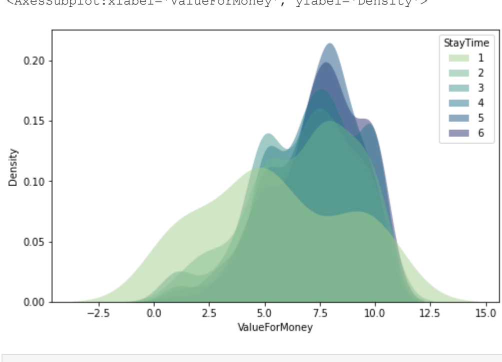
> the hist plots showing the frequencies of the different responses of varying length

In [22]:
```python
df["VisitFrequency"].value_counts()
```

Out[22]:
```
I've only visited once      5532
Once every 2-3 months       3053
Once a month                2444
Once a fortnight or more    2141
Once every 4-6 months       1992
Less often                  1336
Once a year                 1121
Name: VisitFrequency, dtype: int64
```

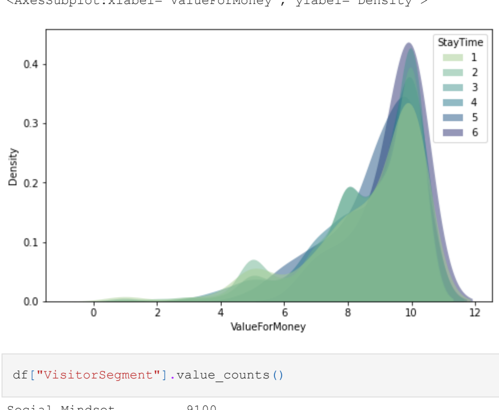
In [23]:
```python
# Trying to understand relationship between StayTime and VisitFrequency. (1)
plt.figure(figsize=(8,5))
once_visit = df[df["VisitFrequency"]=="Less often"]
sns.kdeplot(once_visit["ValueForMoney"],hue=once_visit["StayTime"],fill=True, common_norm=False, palette="crest
           alpha=.5, linewidth=0)
```

Out[23]:
```
<AxesSubplot:xlabel='ValueForMoney', ylabel='Density'>
```



In [24]:
```python
# Trying to understand relationship between StayTime and VisitFrequency. (2)
plt.figure(figsize=(8,5))
once_visit = df[df["VisitFrequency"]=="Once a fortnight or more"]
sns.kdeplot(once_visit["ValueForMoney"],hue=once_visit["StayTime"],fill=True, common_norm=False, palette="crest
           alpha=.5, linewidth=0)
```

Out[24]:
```
<AxesSubplot:xlabel='ValueForMoney', ylabel='Density'>
```



In [25]:
```python
df["VisitorSegment"].value_counts()
```

Out[25]:
```
Social Mindset         9100
Tick Box               3303
Special Focus          2089
Child Engagement       1330
Broadening Horizons    1223
Topic Interest          308
Big Kid                 266
Name: VisitorSegment, dtype: int64
```

Topic Modelling

What is topic modelling?

Topic modeling is a method for unsupervised classification of documents, similar to clustering on numeric data, which finds some natural groups of items (topics) even when we're not sure what we're looking for.

Latent Dirichlet Allocation (LDA)

> In natural language processing, Latent Dirichlet Allocation (LDA) is a generative statistical model that explains a set of observations through unobserved groups, and each group explains why some parts of the data are similar. It is one of the most popular topic modeling methods. Each document is made up of various words, and each topic also has various words belonging to it. The aim of LDA is to find topics a document belongs to, based on the words in it.

In [27]:
```python
# getting the text responses for preprocessing before topic modelling
most_enjoyed = list(df["MostEnjoyed"].dropna())
could_improve = list(df["CouldImprove"].dropna())
use_shop_more = list(df["UseShopMore"].dropna())
use_cafe_more = list(df["UseCafeMore"].dropna())
activity_other = list(df["Activity_Other"].dropna())
```

In [35]:
```python
nltk.download('stopwords')
stop_words = stopwords.words('english')
stop_words.extend(["I'm","like","even","get","goes","would","yes",
               "so","also","I'll","thing","im","let","still","yet","one",
               "want","never","much","really","know","go","since","make","could","I've","day","tell","happen","
              "without","with","back","nothing","something","got","get","told","everything","around","before","
              "think","way","getting","thinking","said","say","told","already","give","gave","can't","anything"
              "good","keep","away","thought","makes","make","things","many","good","don't","dont","days","away"
               "everyday","called","every","bad","seem","almost","another","lot","cant","asked","part","self","
               "find","found","next","point","living","made","either","put","fucking","year","little","take","y
               "probably","wanna","mean","might","person","hour","people","everyone","u","ago","ever","week","w
               "maybe","someone","mom","month","head","actually","guess","happy","used","die","gonna","stuff","
               "happiness","big","world","better","left","live","feel","last","felt","wanted","want","honestly"
               "sure","later","feels","start","least","less","more","most","two","ive","I'd"])
nlp = spacy.load("en_core_web_sm")

# Creating helper functions for data preparation as we'll be using them later in different nlp techniques
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in stop_words] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV','PROPN']):
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out

def data_preprocess(col):
    # sentence tokenizing now
    nltk.download('punkt')
    sentences = []
    for i in col:
        s = tokenize.sent_tokenize(i)
        for j in s:
            sentences.append(j)

    # split sentences to arrays of its words
    data_words = []
    for sen in sentences:
        data_words.append(gensim.utils.simple_preprocess(str(sen), deacc=True))

    # remove stopwords
    data_words_nostops = remove_stopwords(data_words)

    # lemmatization
    data_lemmatized = lemmatization(data_words_nostops, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV','PROPN'])
    return data_lemmatized
```

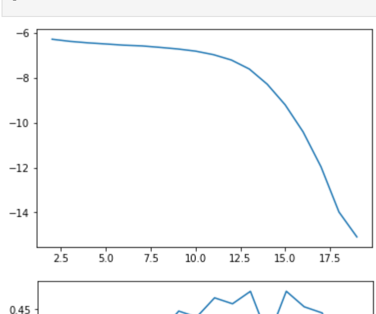
```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Tawhid\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

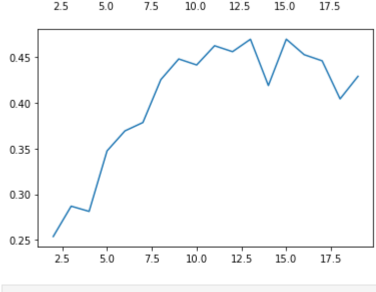
Lets work an example with ALL **MostEnjoyed** responses.

In [36]:
```python
ME_lemmatized = data_preprocess(most_enjoyed)
id2word = corpora.Dictionary(ME_lemmatized)
texts = ME_lemmatized
corpus = [id2word.doc2bow(text) for text in texts]
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Tawhid\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [38]:
```python
# Before choosing the final model, we determine the best params
perp=[]
cohe=[]
for k in range(2,20):
    lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,id2word=id2word,num_topics=k, random_state=100,up
                                               chunksize=100,passes=10,alpha='auto',per_word_topics=True)
    perp.append(lda_model.log_perplexity(corpus))
    coherence_model_lda = CoherenceModel(model=lda_model, texts=ME_lemmatized, dictionary=id2word, coherence='c
    cohe.append(coherence_model_lda.get_coherence())

# plotting the perplexity and coherence of the trialed models
plt.plot(range(2,20),perp)
plt.show()
plt.plot(range(2,20),cohe)
plt.show()
```



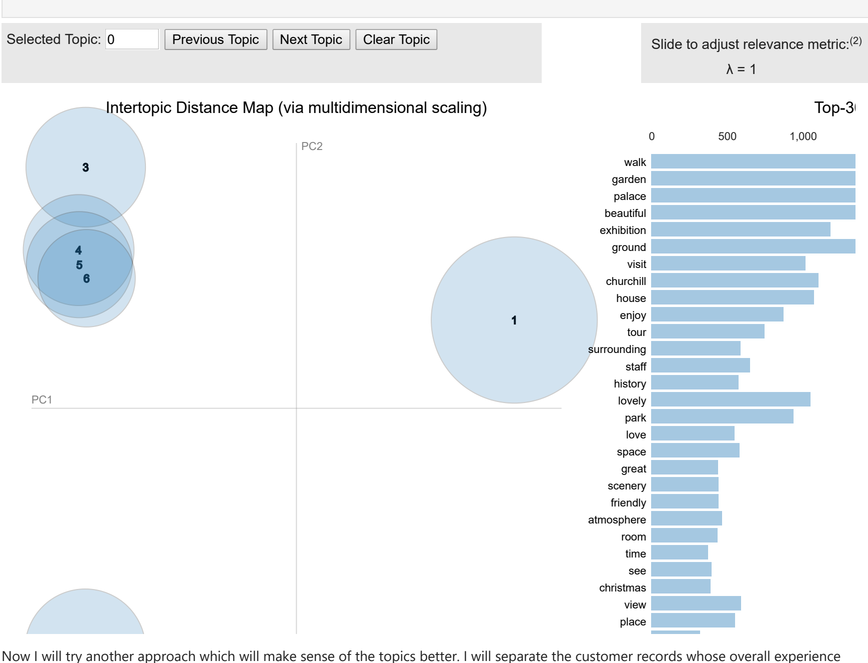


In [39]:
```python
# final LDA model chosen
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,id2word=id2word,num_topics=6, random_state=100,update
                                           chunksize=200,passes=10,alpha='auto',per_word_topics=True)
pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(0,
  '0.161*"palace" + 0.107*"exhibition" + 0.100*"churchill" + 0.068*"tour" + '
  '0.053*"space" + 0.026*"stable" + 0.024*"cafe" + 0.022*"open" + '
  '0.020*"trail" + 0.017*"tree"'),
 (1,
  '0.099*"house" + 0.081*"enjoy" + 0.043*"atmosphere" + 0.040*"room" + '
  '0.037*"see" + 0.036*"christmas" + 0.030*"amazing" + 0.025*"whole" + '
  '0.025*"experience" + 0.023*"child"'),
 (2,
  '0.052*"scenery" + 0.044*"time" + 0.033*"dog" + 0.032*"area" + 0.031*"look" '
  '+ 0.031*"maze" + 0.030*"guide" + 0.028*"helpful" + 0.026*"site" + '
  '0.026*"environment"'),
 (3,
  '0.110*"visit" + 0.070*"staff" + 0.059*"love" + 0.048*"friendly" + '
  '0.029*"blenheim" + 0.029*"nice" + 0.029*"pass" + 0.026*"interesting" + '
  '0.019*"quiet" + 0.019*"explore"'),
 (4,
  '0.082*"surrounding" + 0.080*"history" + 0.061*"great" + 0.035*"pleasant" + '
  '0.033*"tea" + 0.029*"family" + 0.027*"lot" + 0.027*"food" + 0.025*"friend" '
  '+ 0.024*"landscape"'),
 (5,
  '0.146*"walk" + 0.138*"garden" + 0.112*"beautiful" + 0.085*"ground" + '
  '0.050*"lovely" + 0.045*"park" + 0.028*"view" + 0.026*"place" + 0.022*"lake" '
  '+ 0.016*"relax"')]
```

In [40]:
```
pyLDAvis.enable_notebook()
visualization = pyLDAvis.gensim_models.prepare(lda_model, corpus, id2word)
visualization
```

Out[40]: Selected Topic: 0 Previous Topic Next Topic Clear Topic Slide to adjust relevance metric:(2) λ = 1



Now I will try another approach which will make sense of the topics better. I will separate the customer records whose overall experience was the worst from those with the best experience. Then applying LDA on them separately can help us better understand, what customers are saying from the **"MostEnjoyed"** response. Obviously, customers with more positive overall experience may have a different responses than their counterpart.
We divide the customers based on the extreme value of their experience based on the data at hand. We use the metric "ValueForMoney" to do this.

In [42]:
```
most_satisfied_customers = df[df["ValueForMoney"] == 10.0]
least_satisfied_customers = df[df["ValueForMoney"] == 1.0]
```

In [43]:
```
most_enjoyed_10 = list(most_satisfied_customers["MostEnjoyed"].dropna())
most_enjoyed_1 = list(least_satisfied_customers["MostEnjoyed"].dropna())
```

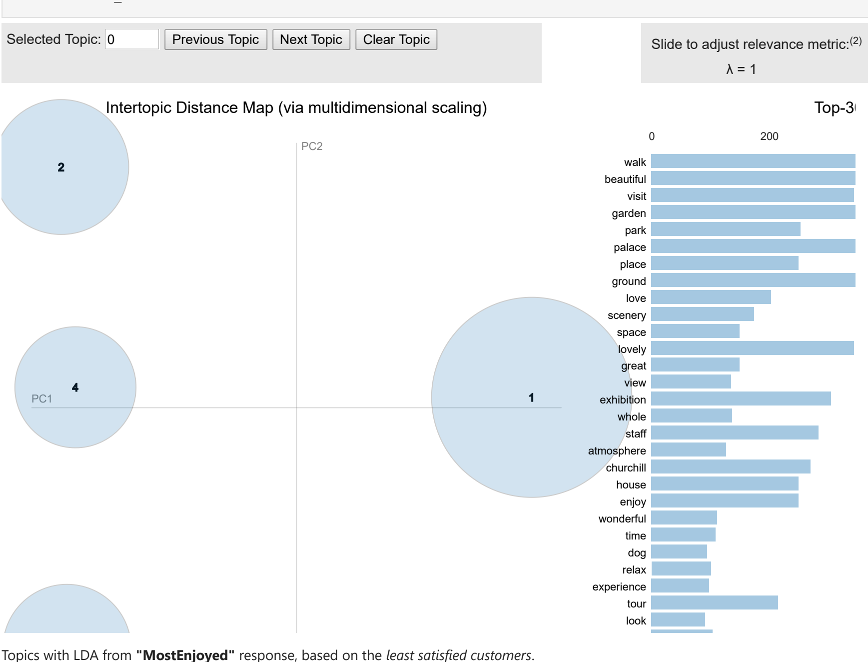
Topics with LDA from **"MostEnjoyed"** response, based on the *most satisfied customers*.

In [44]:
```
ME_lemmatized_10 = data_preprocess(most_enjoyed_10)
id2word_10 = corpora.Dictionary(ME_lemmatized_10)
texts_10 = ME_lemmatized_10
corpus_10 = [id2word_10.doc2bow(text) for text in texts_10]

lda_model_10 = gensim.models.ldamodel.LdaModel(corpus=corpus_10,id2word=id2word_10,num_topics=4, random_state=1
                                        chunksize=100,passes=10,alpha='auto',per_word_topics=True)
pprint(lda_model_10.print_topics())
doc_lda_10 = lda_model_10[corpus_10]
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Tawhid\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[(0,
  '0.079*"visit" + 0.057*"place" + 0.040*"scenery" + 0.026*"wonderful" + '
  '0.025*"time" + 0.024*"christmas" + 0.023*"room" + 0.023*"blenheim" + '
  '0.015*"weather" + 0.014*"tea"'),
 (1,
  '0.084*"walk" + 0.082*"beautiful" + 0.072*"garden" + 0.056*"palace" + '
  '0.047*"ground" + 0.036*"lovely" + 0.032*"exhibition" + 0.030*"staff" + '
  '0.028*"churchill" + 0.026*"house"'),
 (2,
  '0.043*"space" + 0.038*"view" + 0.027*"dog" + 0.026*"look" + 0.023*"beauty" '
  '+ 0.022*"interesting" + 0.022*"pass" + 0.021*"excellent" + 0.021*"peace" + '
  '0.019*"informative"'),
 (3,
  '0.064*"park" + 0.052*"love" + 0.038*"great" + 0.035*"whole" + '
  '0.032*"atmosphere" + 0.026*"relax" + 0.025*"experience" + 0.022*"round" + '
  '0.021*"event" + 0.020*"cafe"')]
```

In [45]:
```
pyLDAvis.enable_notebook()
visualization_10 = pyLDAvis.gensim_models.prepare(lda_model_10, corpus_10, id2word_10)
visualization_10
```

Out[45]: Selected Topic: 0 Previous Topic Next Topic Clear Topic Slide to adjust relevance metric:(2) λ = 1



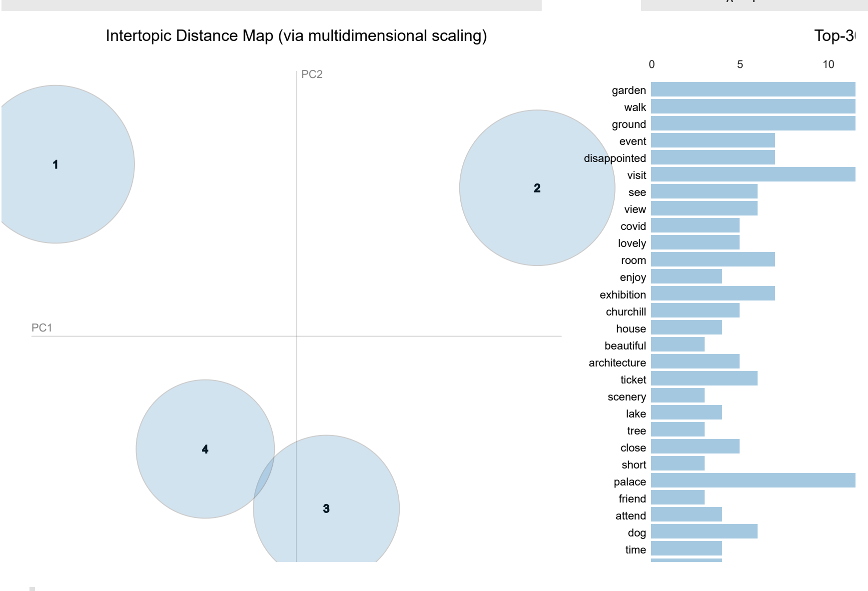
Topics with LDA from **"MostEnjoyed"** response, based on the *least satisfied customers*.

In [46]:
```
ME_lemmatized_1 = data_preprocess(most_enjoyed_1)
id2word_1 = corpora.Dictionary(ME_lemmatized_1)
texts_1 = ME_lemmatized_1
corpus_1 = [id2word_1.doc2bow(text) for text in texts_1]

lda_model_1 = gensim.models.ldamodel.LdaModel(corpus=corpus_1,id2word=id2word_1,num_topics=4, random_state=100,
                                        chunksize=100,passes=10,alpha='auto',per_word_topics=True)
pprint(lda_model_1.print_topics())
doc_lda_10 = lda_model_1[corpus_1]
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Tawhid\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[(0,
  '0.107*"garden" + 0.042*"palace" + 0.023*"see" + 0.017*"close" + '
  '0.015*"lake" + 0.013*"site" + 0.012*"cancel" + 0.012*"park" + 0.012*"book" '
  '+ 0.011*"advice"'),
 (1,
  '0.047*"ground" + 0.034*"visit" + 0.018*"open" + 0.017*"enjoy" + '
  '0.017*"ticket" + 0.016*"beautiful" + 0.015*"scenery" + 0.014*"short" + '
  '0.013*"expensive" + 0.013*"space"'),
 (2,
  '0.050*"walk" + 0.033*"visit" + 0.024*"disappointed" + 0.021*"room" + '
  '0.021*"view" + 0.020*"exhibition" + 0.020*"palace" + 0.019*"lovely" + '
  '0.017*"churchill" + 0.016*"architecture"'),
 (3,
  '0.030*"event" + 0.019*"covid" + 0.016*"dog" + 0.015*"time" + 0.015*"tree" + '
  '0.014*"friend" + 0.011*"cafe" + 0.010*"ticket" + 0.009*"book" + '
  '0.009*"palace"')]
```

In [47]:
```
pyLDAvis.enable_notebook()
visualization_1 = pyLDAvis.gensim_models.prepare(lda_model_1, corpus_1, id2word_1)
visualization_1
```

Out[47]: Selected Topic: 0 Previous Topic Next Topic Clear Topic Slide to adjust relevance metric:(2) λ = 1



> After separating the customers, we see that while we do get better interpretable topics, its still not much improvement. So next, I will use another method of Topic Modelling to compare the results with this methodology.

BERTopic

> BERTopic is a topic modeling technique that leverages 🤗 transformers and c-TF-IDF to create dense clusters allowing for easily interpretable topics whilst keeping important words in the topic descriptions. BERTopic supports guided, (semi-) supervised, and dynamic topic modeling. Please check out the fantastic work by Maarten Grootendorst here.

As part of this method, we need full sentences and if the sentences are cleaned, its better. so basically reconverting from lemmatized versions.

In [50]:
```
def cleaned_reconvert(lem):
    cleaned_sentences = []
    for i in lem:
        k = ""
        for j in i:
            k+=j+" "
        cleaned_sentences.append(k)
    return cleaned_sentences

def topic_model_with_BERT(txt):
    topic_model = BERTopic()
    reconverted = cleaned_reconvert(txt)
    topics, probs = topic_model.fit_transform(reconverted)
    return topic_model
```

"MostEnjoyed" based on the *most satisfied customers*.

In [51]:
```
topic_model = topic_model_with_BERT(ME_lemmatized_10)
topic_model.get_topic_info()
```

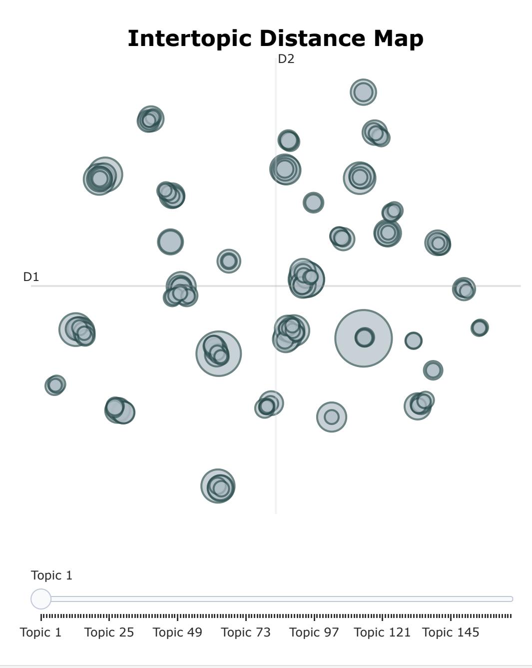
Out[51]:

	Topic	Count	Name
0	-1	845	-1_exercise_guide_archery_place
1	0	281	0_thst_everthe_abc_
2	1	215	1_staff_friendly_helpful_knowledgeable
3	2	133	2_blenheim_love_doorstep_circumstance
4	3	85	3_garden_bored_pease_surround
...	...	...	...
163	162	11	162_pleasant_rather_isolate_recent
164	163	11	163_meeting_friend_meet_surrey
165	164	11	164_pizza_cafe_cook_dining
166	165	11	165_relaxing_stress_overly_arboretum
167	166	10	166_tour_buggy_servanys_morning

168 rows × 3 columns

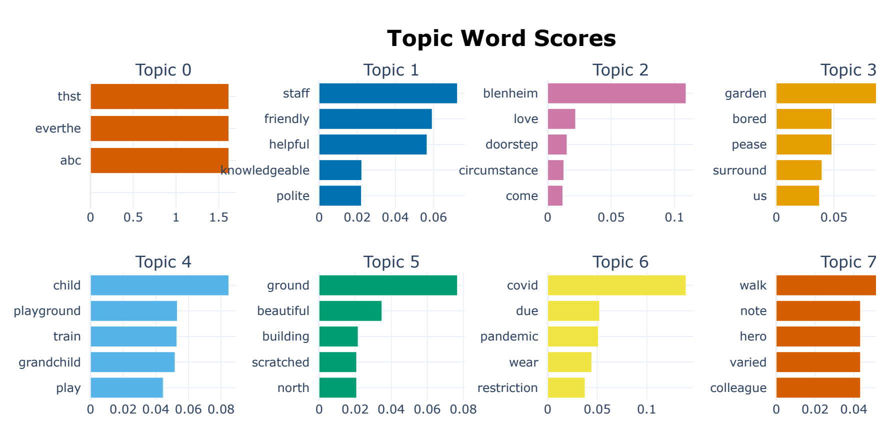
In [52]:
```
topic_model.visualize_topics()
```

```
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



In [53]:
```
topic_model.visualize_barchart()
```

```
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



In [54]:
```
topic_model.get_topic(5)
```

Out[54]:
```
[('ground', 0.0766683151091111),
 ('beautiful', 0.034764413719840695),
 ('building', 0.02157128432388542),
 ('scratched', 0.020735172068032555),
 ('north', 0.020735172068032555),
 ('imagine', 0.020735172068032555),
 ('foot', 0.020735172068032555),
 ('elation', 0.020735172068032555),
 ('january', 0.020735172068032555),
 ('sparse', 0.020735172068032555)]
```

"MostEnjoyed" based on the *least satisfied customers*.

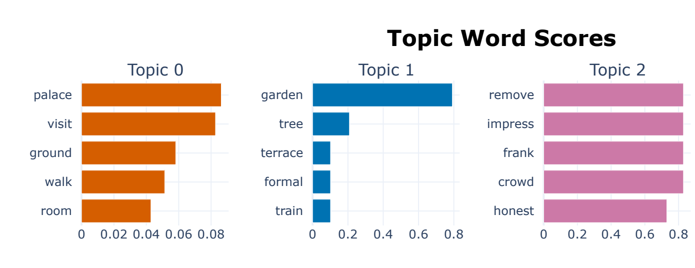
In [55]:
```
topic_model = topic_model_with_BERT(ME_lemmatized_1)
topic_model.get_topic_info()
```

Out[55]:

	Topic	Count	Name
0	0	212	0_palace_visit_ground_walk
1	1	48	1_garden_tree_terrace_formal
2	2	35	2_remove_impress_frank_crowd

In [56]:
```
topic_model.visualize_barchart()
```

```
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



In [57]:
```
topic_model.get_topic(1)
```

Out[57]:
```
[('garden', 0.7914257153085086),
 ('tree', 0.2081530999698498),
 ('terrace', 0.10122748095755532),
 ('formal', 0.10122748095755532),
 ('train', 0.10122748095755532),
 ('landscape', 0.10122748095755532),
 ('picnic', 0.10122748095755532),
 ('cafe', 0.08762072538330316),
 ('beautiful', 0.08326123998793991),
 ('open', 0.06633473440147733)]
```

> It seems that using BERTOPIC gives much better results, however, in terms of the low number of customers with bad experiences, limited topics have been extracted for their "MostEnjoyed" response. So, I will now use the **full** "MostEnjoyed" data to do topic modelling and see the results.

"MostEnjoyed" responses for *ALL customers*.

In [59]:
```
topic_model = topic_model_with_BERT(ME_lemmatized)
topic_model.get_topic_info()
```

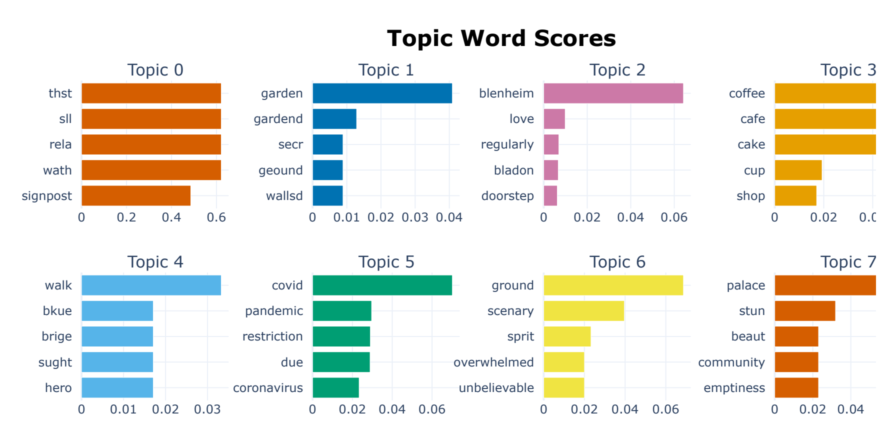
Out[59]:

	Topic	Count	Name
0	-1	2877	-1_tree_lake_daughter_enjoy
1	0	778	0_thst_sll_rela_wath
2	1	520	1_garden_gardend_secr_geound
3	2	306	2_blenheim_love_regularly_bladon
4	3	257	3_coffee_cafe_cake_cup
...	...	...	...
465	464	10	464_outside_pasty_uncrowded_space
466	465	10	465_quite_quiteness_restricted_fortunate
467	466	10	466_giant_legion_british_attraction
468	467	10	467_statue_horse_sculpture_bronze
469	468	10	468_exercise_daily_stat_shopping

470 rows × 3 columns

In [60]:
```
topic_model.visualize_barchart()
```

```
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



> Now that we have established that BERTOPIC is doing a better depiction of extracting topics from this dataset, let's try to understand how the **"CouldImprove"** suggestions differ between customers who have had positive and negative experiences. But this time, instead of using the extreme ends of the experiences like 1 and 10, I will use values greater and smaller than 5 for **"ValueForMoney"** feature, to get a more holistic view.

In [62]:
```
customers_rated_above_5 = df[df["ValueForMoney"] > 5.0]
customers_rated_below_5 = df[df["ValueForMoney"] <= 5.0]

could_improve_above_5 = list(customers_rated_above_5["CouldImprove"].dropna())
could_improve_below_5 = list(customers_rated_below_5["CouldImprove"].dropna())
```

"CouldImprove" topics from satisfied customers who rated **"ValueForMoney"** > 5.0

In [63]:
```
topic_model = topic_model_with_BERT(data_preprocess(could_improve_above_5))
topic_model.get_topic_info()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Tawhid\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

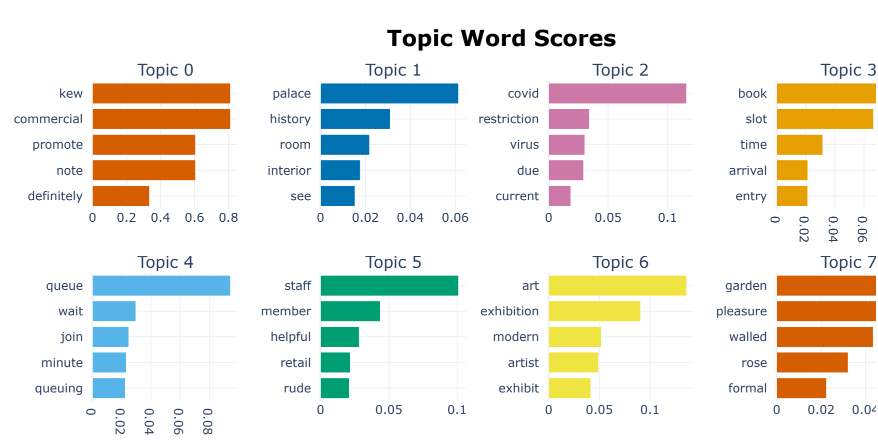
Out[63]:

	Topic	Count	Name
0	-1	2291	-1_pass_walk_park_car
1	0	1142	0_kew_commercial_promote_note
2	1	172	1_palace_history_room_interior
3	2	152	2_covid_restriction_virus_due
4	3	147	3_book_slot_time_arrival
...	...	...	...
165	164	11	164_lead_formal_allow_dog
166	165	10	165_water_feature_fountain_empty
167	166	10	166_van_coffee_grandchildren_charge
168	167	10	167_blenheim_hygene_abramovic_ceo
169	168	10	168_signage_directional_norton_scanning

170 rows × 3 columns

In [64]:
```
topic_model.visualize_barchart()
```

```
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



"CouldImprove" topics from satisfied customers who rated **"ValueForMoney"** > 5.0

In [65]:
```
topic_model = topic_model_with_BERT(data_preprocess(could_improve_below_5))
topic_model.get_topic_info()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Tawhid\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

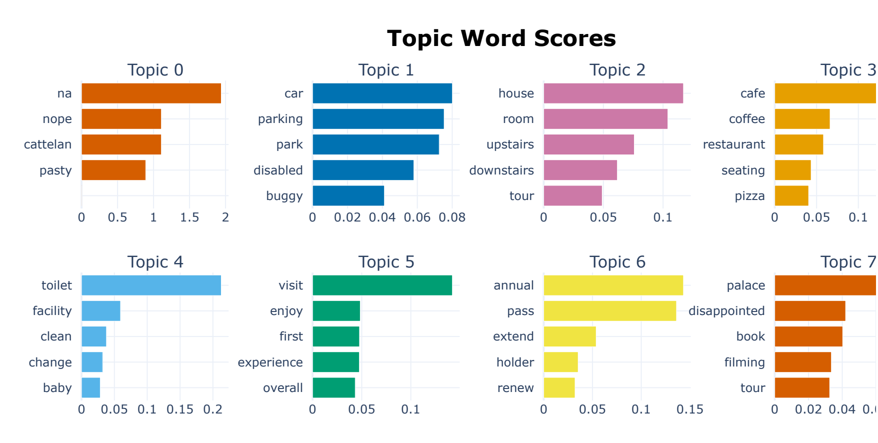
Out[65]:

	Topic	Count	Name
0	-1	826	-1_garden_time_entry_pass
1	0	172	0_na_nope_cattelan_pasty
2	1	114	1_car_parking_park_disabled
3	2	89	2_house_room_upstairs_downstairs
4	3	77	3_cafe_coffee_restaurant_seating
...	...	...	...
56	55	12	55_apart_vast_course_garden
57	56	12	56_friend_membership_free_bring
58	57	10	57_book_slot_advance_allow
59	58	10	58_signage_garden_clear_additionally
60	59	10	59_queue_allot_slot_time

61 rows × 3 columns

In [66]:
```
topic_model.visualize_barchart()
```

```
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```

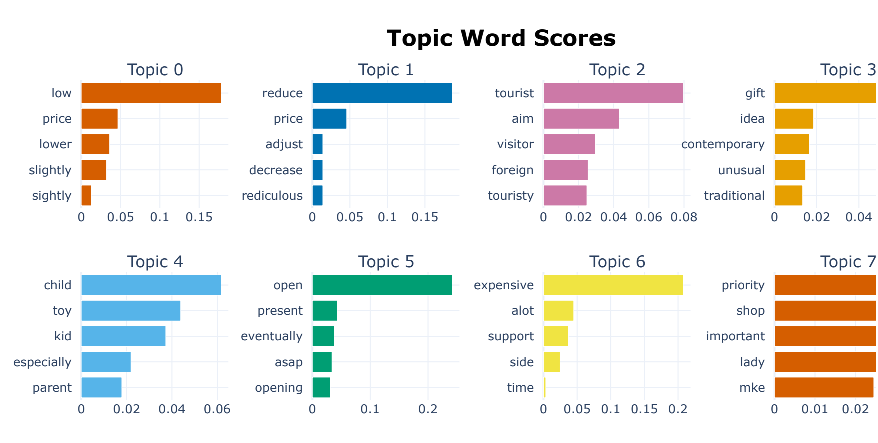


> We can see much difference in the types of improvement suggestions based on customers with varying experiences. Next, we will do topic modelling on **"UseShopMore"** responses. However, while its already been established that based on experiences, the responses and topics vary, here we will use *all customer responses* for this feature.

Topics from **"UseShopMore"** responses.

In [68]:
```
topic_model = topic_model_with_BERT(data_preprocess(use_shop_more))
topic_model.visualize_barchart()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Tawhid\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



> Interesting to see topics like "shortage of items" has been highlighted from the topic list. Overall, good representations of topics have been done here.

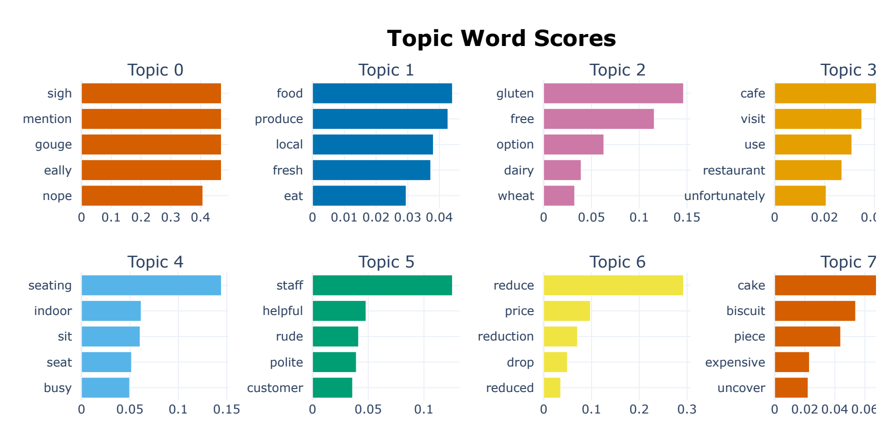
In [69]:
```
topic_model.get_topic(9)
```

Out[69]:
```
[('blenheim', 0.08889054187696037),
 ('specific', 0.034875931905330275),
 ('generic', 0.023113573325216106),
 ('merchandise', 0.02071139367258528),
 ('instead', 0.01900573386081291),
 ('elsewhere', 0.016833112778025006),
 ('honey', 0.016344362606233988),
 ('name', 0.015409048883477404),
 ('relate', 0.014862080097442842),
 ('focus', 0.014625911924585688)]
```

Finally, we do the same for **"UseCafeMore"**.

In [70]:
```
topic_model = topic_model_with_BERT(data_preprocess(use_cafe_more))
topic_model.visualize_barchart()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Tawhid\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



For the final text response from **"ActivityOther"**, I will use a third approach to topic modelling which would be a custom implementation of the above BERTopic approach in the next section. I will then try to compare the performances of these techniques to bring out underlying topics.

Transformers, c-TF-IDF, UMAP, and HDBSCAN

In [72]:

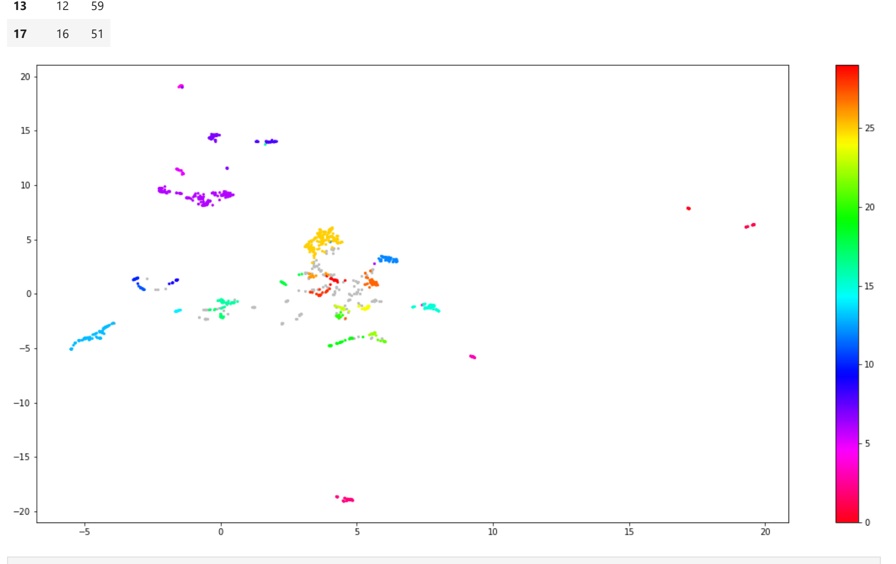
```python
# helper functions
def c_tf_idf(documents, m, ngram_range=(1, 1)):
    count = CountVectorizer(ngram_range=ngram_range, stop_words="english").fit(documents)
    t = count.transform(documents).toarray()
    w = t.sum(axis=1)
    tf = np.divide(t.T, w)
    sum_t = t.sum(axis=0)
    idf = np.log(np.divide(m, sum_t)).reshape(-1, 1)
    tf_idf = np.multiply(tf, idf)
    return tf_idf, count

def extract_top_n_words_per_topic(tf_idf, count, docs_per_topic, n=20):
    words = count.get_feature_names()
    labels = list(docs_per_topic.Topic)
    tf_idf_transposed = tf_idf.T
    indices = tf_idf_transposed.argsort()[:, -n:]
    top_n_words = {label: [(words[j], tf_idf_transposed[i][j]) for j in indices[i]][::-1] for i, label in enume
    return top_n_words

def extract_topic_sizes(df):
    topic_sizes = (df.groupby(['Topic'])
                     .Doc
                     .count()
                     .reset_index()
                     .rename({"Topic": "Topic", "Doc": "Size"}, axis='columns')
                     .sort_values("Size", ascending=False))
    return topic_sizes

def custom_topic_model(txt):
    model = SentenceTransformer('distilbert-base-nli-mean-tokens')
    sents = txt
    embeddings = model.encode(sents, show_progress_bar=True)
    umap_embeddings = umap.UMAP(n_neighbors=15, n_components=5, metric='cosine').fit_transform(embeddings)
    cluster = hdbscan.HDBSCAN(min_cluster_size=15,metric='euclidean', cluster_selection_method='eom').fit(umap_
    # Prepare data
    umap_data = umap.UMAP(n_neighbors=15, n_components=2, min_dist=0.0, metric='cosine').fit_transform(embeddir
    result = pd.DataFrame(umap_data, columns=['x', 'y'])
    result['labels'] = cluster.labels_

    # Visualize clusters
    fig, ax = plt.subplots(figsize=(20, 10))
    outliers = result.loc[result.labels == -1, :]
    clustered = result.loc[result.labels != -1, :]
    plt.scatter(outliers.x, outliers.y, color='#BDBDBD', s=5)
    plt.scatter(clustered.x, clustered.y, c=clustered.labels, s=5, cmap='hsv_r')
    plt.colorbar()

    docs_df = pd.DataFrame(sents, columns=["Doc"])
    docs_df['Topic'] = cluster.labels_
    docs_df['Doc_ID'] = range(len(docs_df))
    docs_per_topic = docs_df.groupby(['Topic'], as_index = False).agg({'Doc': ' '.join})

    tf_idf, count = c_tf_idf(docs_per_topic.Doc.values, m=len(sents))
    top_n_words = extract_top_n_words_per_topic(tf_idf, count, docs_per_topic, n=20)
    topic_sizes = extract_topic_sizes(docs_df)
    return top_n_words, topic_sizes
```

In [73]:

```python
#for_comparison = data_preprocess(could_improve_below_5)
activity_other = list(df["Activity_Other"].dropna())
top_n_words, topic_sizes = custom_topic_model(activity_other)
topic_sizes.head(10)
```

Out[73]:

	Topic	Size
0	-1	217
26	25	183
7	6	180
14	13	104
28	27	62
8	7	60
16	15	59
3	2	59
13	12	59
17	16	51



In [74]:

```python
top_n_words[12][:20]
```

Out[74]:

```
[('annual', 0.7982219354292401),
 ('pass', 0.5713598125602665),
 ('renew', 0.4220143838591523),
 ('renewed', 0.2528834031727084),
 ('passes', 0.2495153686273509),
 ('privilege', 0.09960333019061517),
 ('obtain', 0.08647052348167253),
 ('membership', 0.07684347930722885),
 ('12th', 0.06140132221100544),
 ('card', 0.05764701565444835),
 ('collected', 0.05764701565444835),
 ('collect', 0.05498329276137631),
 ('15', 0.05498329276137631),
 ('tickets', 0.05291714876772622),
 ('online', 0.05291714876772622),
 ('paid', 0.05122898620481923 5),
 ('got', 0.04980166509530758),
 ('ticket', 0.04747467964826215),
 ('day', 0.04055602777946381),
 ('saver', 0.033909675830317274)]
```

> This method seems to do a good job extracting some meaningful topics for **"Activity_Other"** responses. For instance, topic 15 basically is a topic about children activities, and topic 12 talks about memberships and their concerns.

After performing the topic modelling using various techniques, it is evident that while some methods give out good results, it is quite difficult to interpret whole topics and results are not very accurate. So now, using **semantic search techniques**, I will try an experiment to refine the extracted topics, using a preconceived notion or abstract notion from above topic modelling techniques. Using an idea from the outputs of the above models, I will use semantic search with **vector embeddings**, to bring together documents containing those words. THEN, I will do topic modelling again to see if we get better results. Because currently, there is a lot of information loss. Afterwards, I will try another approach to see using sentiment analysis to see if we get some better representations of the underlying topics and compare the methods.

Semantic Search and Sentiment Analysis

> Semantic search applies user intent, context, and conceptual meanings to match a user query to the corresponding content. It uses vector search and machine learning to return results that aim to match a user's query, even when there are no word matches. Basically embedding similar sentences together based on specified terms/sentences.

In [107…

```python
def similar_sents(txt,queries,count):
    embedder  = SentenceTransformer('all-MiniLM-L6-v2')
    corpus_embeddings = embedder.encode(txt, convert_to_tensor=True)
    dat_exp = []
    top_k = min(count, len(txt))

    for query in queries:
        query_embedding = embedder.encode(query, convert_to_tensor=True)
        # We use cosine-similarity and torch.topk to find the highest 10 scores
        cos_scores = util.cos_sim(query_embedding, corpus_embeddings)[0]
        top_results = torch.topk(cos_scores, k=top_k)

#         print("\n\n======================\n\n")
#         print("Query:", query)
#         print("\nTop most similar sentences in corpus:")

        for score, idx in zip(top_results[0], top_results[1]):
            dat_exp.append(txt[idx])
            #print(txt[idx], "(Score: {:.4f})".format(score))
    return dat_exp

# helper function to get the sentiments for responses in a dataframe
def get_sentiment_df(txt):
    sentiments = SentimentIntensityAnalyzer()
    dat_exp_df = pd.DataFrame(txt)
    dat_exp_df["Positive"] = [sentiments.polarity_scores(i)["pos"] for i in dat_exp_df[0]]
    dat_exp_df["Negative"] = [sentiments.polarity_scores(i)["neg"] for i in dat_exp_df[0]]
    dat_exp_df["Neutral"] = [sentiments.polarity_scores(i)["neu"] for i in dat_exp_df[0]]
    return dat_exp_df

# helper function to produce word cloud which will be used later.
def generate_word_cloud(txt,max_words):
    plt.figure(figsize=(12,8))
    cloudtext = ""
    for i in txt:
        cloudtext += i+" "

    wordcloud = WordCloud(background_color="black",colormap="Set2",
                          max_words=max_words, contour_width=0,collocations=True)# Generate a word cloud

    wordcloud.generate(cloudtext) # Visualize the word cloud
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
```

In [77]:

```python
dat_exp = similar_sents(could_improve_above_5,["weather"],50)
topic_model = topic_model_with_BERT(data_preprocess(dat_exp))
topic_model.get_topic_info()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Tawhid\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[77]:

	Topic	Count	Name
0	0	61	0_weather_rain_today_visit
1	1	12	1_shame_none_excellent_cool

In [78]:

```python
topic_model.get_topic(0)
```

Out[78]:

```
[('weather', 0.22514624284623488),
 ('rain', 0.10389989486674246),
 ('today', 0.08196258046224283),
 ('visit', 0.06487867846465215),
 ('guarantee', 0.06487867846465215),
 ('cold', 0.06487867846465215),
 ('control', 0.06487867846465215),
 ('cover', 0.06487867846465215),
 ('improve', 0.05529992817056725),
 ('shade', 0.05529992817056725)]
```

Now I will use the sentiments. There are some inconsistencies with the sentiments, and it is the prime area of future improvement for this analysis.

In [79]:

```python
dat_exp_df = get_sentiment_df(dat_exp)
weather_exp = dat_exp_df[dat_exp_df["Negative"]>0.000]
weather_exp
```

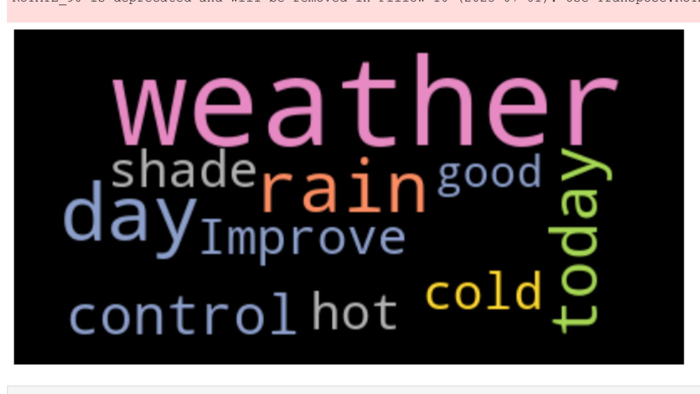
Out[79]:

	0	Positive	Negative	Neutral
11	No weather was not good but that's England	0.000	0.355	0.645
15	Pity unable to control weather	0.000	0.355	0.645
22	The only bad thing about today was the weather.	0.000	0.304	0.696
23	No, we have a lovely day despite the inclement...	0.292	0.169	0.538
24	Unfortunately you can't control the weather! ...	0.072	0.064	0.864
26	Some where to sit in bad weather	0.000	0.368	0.632
27	Maybe if you could guarantee the weather altho...	0.285	0.075	0.641
30	A rainy day policy	0.000	0.394	0.606
31	Stop the rain:)	0.000	0.524	0.476
34	Stop the rain?!\n\nIndoor / under cover (open ...	0.000	0.284	0.716
35	Not really, thoroughly enjoyable day out so lo...	0.000	0.221	0.779
36	The line fir the house tour was too long on a ...	0.000	0.073	0.927
37	No rain! The weather did spoil the visit. Its ...	0.000	0.248	0.752
41	Unfortunately the weather was against us, but ...	0.105	0.047	0.848
44	No excellent day out	0.468	0.278	0.253
45	We have visited twice in the past week and bee...	0.065	0.085	0.850
46	Queing inm the hot sun is not for everyone... ...	0.000	0.151	0.849
47	None. Always enjoy my visits, even if weather...	0.000	0.226	0.774

In [80]:

```python
generate_word_cloud(dat_exp_df[0],10)
```

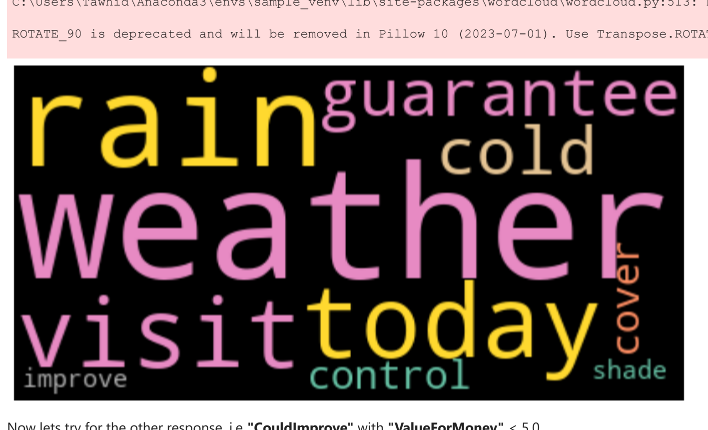
```
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:492: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
```



In [81]:

```python
tm = []
for i in topic_model.get_topic(0):
    tm.append(i[0])
generate_word_cloud(tm,10)
```

```
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:492: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:513: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:492: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:513: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:492: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:513: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
```



Now lets try for the other response, i.e **"CouldImprove"** with **"ValueForMoney"** < 5.0

In [82]:

```python
dat_exp2 = similar_sents(could_improve_below_5,["toilet"],50)
topic_model2 = topic_model_with_BERT(data_preprocess(dat_exp2))
topic_model2.get_topic_info()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Tawhid\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[82]:

	Topic	Count	Name
0	0	41	0_toilet_facility_park_visit
1	-1	26	-1_toilet_use_open_hand
2	1	14	1_toilet_thank_sign_poo

I will now try another way to conduct sentiment analysis using **pre-trained models** from **Huggingface**.

> A pre-trained model is a saved network that was previously trained on a large dataset. We either use the pretrained model as is or use **transfer learning** to customize this model to a given task.

I will be using the **bert-base-multilingual-uncased-sentiment** model which is finetuned for sentiment analysis on product reviews. It predicts the sentiment of the review as a number of stars (between 1 and 5). The most negative sentiments are held by 1 star labels and the most positive in the sample is denoted by 5 stars.

In [83]:

```python
sentiment_pipeline = pipeline(task = 'sentiment-analysis',
                    model = 'nlptown/bert-base-multilingual-uncased-sentiment',
                    tokenizer = 'nlptown/bert-base-multilingual-uncased-sentiment')
sent_list = sentiment_pipeline(dat_exp2)
toilet_exp = []
for i,k in enumerate(sent_list):
    if(k["label"]=="1 star"):
        toilet_exp.append(dat_exp2[i])
```

```
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\transformers\image_utils.py:188: DeprecationWarnin
g:

BILINEAR is deprecated and will be removed in Pillow 10 (2023-07-01). Use Resampling.BILINEAR instead.
```

In [84]:

```python
sent_list[:5]
```

Out[84]:

```
[{'label': '4 stars', 'score': 0.3061443567276001},
 {'label': '3 stars', 'score': 0.305232971906662},
 {'label': '3 stars', 'score': 0.305232971906662},
 {'label': '4 stars', 'score': 0.3785192668437958},
 {'label': '1 star', 'score': 0.2314562052488327}]
```

> We see that this method of using sentiments to get the most negative responses, does work quite well to bring out major concerns in comparison to the above method of **rule based sentiment analysis**.

In [85]:

```python
toilet_exp
```

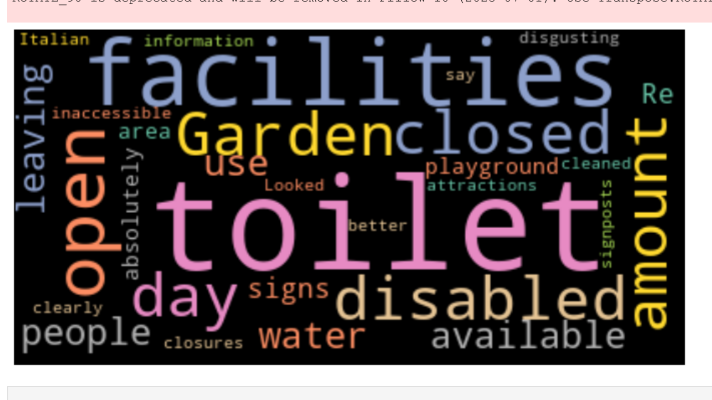
Out[85]:

```
['open the toilets',
 'Re open the toilets',
 'No signs for toilets',
 "Toilets at the playground area were absolutely disgusting. Looked like they hadn't been cleaned all day.",
 'More toilets - better information about closures and signposts to attractions such as the Italian Garden shou
ld clearly say that they are inaccessible.  I am disabled and it took an enormous amount of effort and pain to
walk to that garden... and all for nought!',
 'More toilet facilities! And to not close the toilet facilities before the venue has actually closed! A lot of
people were leaving 15 minutes before closing and wanted to use the toilet facilities before setting off on the
ir journey home but the block of 6 toilets had closed leaving only the 1 disabled toilet open on the whole sit
e, this caused a very long and slow moving queue of people needing to use that 1 toilet!',
 'The provision of toilet facilities was lamentable. To have merely half a dozen or so Porterloos available for
the amount of visitors you had on the day we visited was disgraceful. No hand washing facilities using soap and
water, never mind hot water, were available.']
```

In [108…

```python
generate_word_cloud(toilet_exp,30)
```

```
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:492: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:492: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:492: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
```



In [87]:

```python
tm2 = []
for i in topic_model2.get_topic(0):
    tm2.append(i[0])
generate_word_cloud(tm2,10)
```

```
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
C:\Users\Tawhid\Anaconda3\envs\sample_venv\lib\site-packages\wordcloud\wordcloud.py:512: DeprecationWarning:
ROTATE_90 is deprecated and will be removed in Pillow 10 (2023-07-01). Use Transpose.ROTATE_90 instead.
```

Clustering

> An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labeled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.
>
> Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

In [88]:

```python
cluster_features = ["PartySize","StayTime","ValueForMoney","Activity_TookATour","Activity_VisitedGardens","Acti
                    "Activity_Event","Activity_Cafe","Activity_Shop","Activity_Exhibition",
                    "Cafe_Pantry","Cafe_Stables","Cafe_Pizza","Cafe_Orangery","Cafe_CoffeeStall","Cafe_Icecrean
                    "Service_AdmissionsAndWelcome","Service_TourGuide","Service_RetailTeam","Service_CateringTe
df_for_clustering = df[cluster_features]
df_for_clustering.head(5)
```

Out[88]:

	PartySize	StayTime	ValueForMoney	Activity_TookATour	Activity_VisitedGardens	Activity_WalkedAroundPark	Activity_Event
1237	6	6	8.0	0	0	0	1
34	4	3	9.0	0	0	0	1
398	4	4	10.0	0	0	1	1
444	2	4	8.0	1	0	0	0
33	5	3	8.0	0	0	1	1

5 rows × 21 columns

We cant have null values, so dropping the null for all the features.

In [89]:

```python
df_for_clustering = df_for_clustering.dropna()
df_for_clustering.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5363 entries, 1237 to 17617
Data columns (total 21 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   PartySize                     5363 non-null   int64
 1   StayTime                      5363 non-null   int64
 2   ValueForMoney                 5363 non-null   float64
 3   Activity_TookATour            5363 non-null   int64
 4   Activity_VisitedGardens       5363 non-null   int64
 5   Activity_WalkedAroundPark     5363 non-null   int64
 6   Activity_Event                5363 non-null   int64
 7   Activity_Cafe                 5363 non-null   int64
 8   Activity_Shop                 5363 non-null   int64
 9   Activity_Exhibition           5363 non-null   int64
 10  Cafe_Pantry                   5363 non-null   int64
 11  Cafe_Stables                  5363 non-null   int64
 12  Cafe_Pizza                    5363 non-null   int64
 13  Cafe_Orangery                 5363 non-null   int64
 14  Cafe_CoffeeStall              5363 non-null   int64
 15  Cafe_Icecream                 5363 non-null   int64
 16  Service_AdmissionsAndWelcome  5363 non-null   float64
 17  Service_TourGuide             5363 non-null   float64
 18  Service_RetailTeam            5363 non-null   float64
 19  Service_CateringTeam          5363 non-null   float64
 20  Service_AnnualPass            5363 non-null   float64
dtypes: float64(6), int64(15)
memory usage: 921.8 KB
```

In [90]:

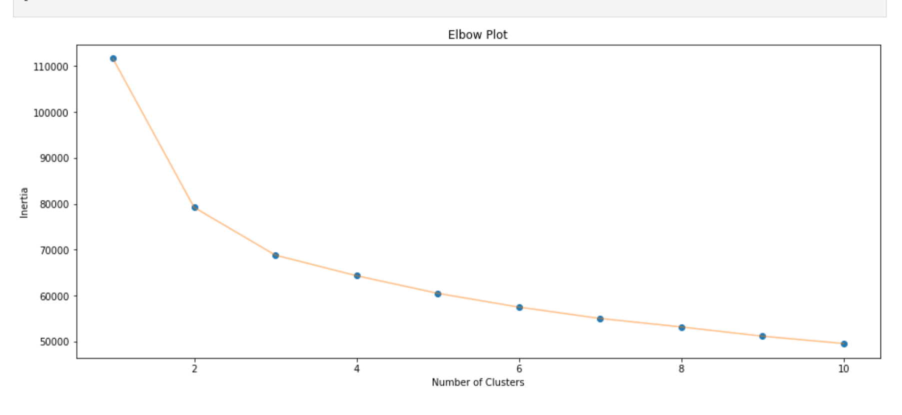
```python
X1 = df_for_clustering.iloc[: , :].values
inertia = []
for n in range(1 , 11):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10 ,max_iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan') )
    algorithm.fit(X1)
    inertia.append(algorithm.inertia_)
```

Lets plot the **elbow plot** to pick an optimum number of clusters.

> In cluster analysis, the elbow method is a heuristic used in determining the number of clusters in a data set.

In [109…

```python
plt.figure(1 , figsize = (15 ,6))
plt.plot(np.arange(1 , 11) , inertia , 'o')
plt.plot(np.arange(1 , 11) , inertia , '-' , alpha = 0.5)
plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')
plt.title("Elbow Plot")
plt.show()
```



> From the elbow plot we see that it is safe to use **k=4** for this problem, as the first major flattening out starts at k=4.

In [92]:

```python
algorithm = (KMeans(n_clusters = 4 ,init='k-means++', n_init = 10 ,max_iter=300, tol=0.0001,  random_state= 111
algorithm.fit(X1)
labels1 = algorithm.labels_
centroids1 = algorithm.cluster_centers_

#assigning the labels in the dataframe
df_for_clustering["cluster"] = labels1
```

In [93]:

```python
df_for_clustering[df_for_clustering["cluster"]==3]
```

Out[93]:

	PartySize	StayTime	ValueForMoney	Activity_TookATour	Activity_VisitedGardens	Activity_WalkedAroundPark	Activity_Event
1237	6	6	8.0	0	0	0	1
444	2	4	8.0	1	0	0	0
33	5	3	8.0	0	0	1	1
82	4	6	4.0	0	0	0	0
92	4	6	7.0	1	1	1	0
...	...	...	...	...	...	...	...
17551	2	4	7.0	1	1	1	0
17566	2	3	8.0	0	1	0	0
17579	4	4	6.0	0	1	1	0
17600	5	2	8.0	0	0	1	0
17604	2	1	5.0	0	0	0	0

1163 rows × 22 columns

Machine Learning

We will try to model **"ValueForMoney"** with other quantitative features and will try to understand the importances of the features. The algorithms I will use here are:

- Decision Tree Classifier
- Random Forest Classifier
- Gradient Boosting Classifier
- K-Neighbors Classifier
- Support Vector Classifier

This is just an experiment to see what sort of results we get and if they actually make sense.

In [94]:

```python
model_features = df.copy()
model_features.drop(["MostEnjoyed","CouldImprove","UseShopMore","UseCafeMore","Activity_Other",
                     "VisitDate","VisitFrequency","VisitorSegment"], axis=1,inplace=True)
```

In [95]:

```python
model_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17619 entries, 1237 to 17618
Data columns (total 26 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   VisitType                     17619 non-null  object
 1   ChildrenUnder12               17619 non-null  object
 2   PartySize                     17619 non-null  int64
 3   StayTime                      17619 non-null  int64
 4   OverallNPS                    17618 non-null  float64
 5   CateringNPS                   8860 non-null   float64
 6   RetailNPS                     17619 non-null  int64
 7   ValueForMoney                 17618 non-null  float64
 8   Activity_TookATour            17619 non-null  int64
 9   Activity_VisitedGardens       17619 non-null  int64
 10  Activity_WalkedAroundPark     17619 non-null  int64
 11  Activity_Event                17619 non-null  int64
 12  Activity_Cafe                 17619 non-null  int64
 13  Activity_Shop                 17619 non-null  int64
 14  Activity_Exhibition           17619 non-null  int64
 15  Cafe_Pantry                   17619 non-null  int64
 16  Cafe_Stables                  17619 non-null  int64
 17  Cafe_Pizza                    17619 non-null  int64
 18  Cafe_Orangery                 17619 non-null  int64
 19  Cafe_CoffeeStall              17619 non-null  int64
 20  Cafe_Icecream                 17619 non-null  int64
 21  Service_AdmissionsAndWelcome  17038 non-null  float64
 22  Service_TourGuide             9901 non-null   float64
 23  Service_RetailTeam            9785 non-null   float64
 24  Service_CateringTeam          12534 non-null  float64
 25  Service_AnnualPass            11694 non-null  float64
dtypes: float64(8), int64(16), object(2)
memory usage: 3.6+ MB
```

We encode the following features.

In [96]:

```python
model_features = pd.get_dummies(model_features, columns = ["VisitType","ChildrenUnder12"], drop_first = True)
model_features.dropna(inplace=True)
```

In [97]:

```python
X = model_features.drop(["ValueForMoney"],axis=1)
y = model_features[["ValueForMoney"]]
```

In [98]:

```python
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2, random_state=42)
print("The shape of X_train is ", X_train.shape)
print("The shape of X_test is ",X_test.shape)
print("The shape of y_train is ",y_train.shape)
print("The shape of y_test is ",y_test.shape)
```

```
The shape of X_train is  (2604, 25)
The shape of X_test is  (652, 25)
The shape of y_train is  (2604, 1)
The shape of y_test is  (652, 1)
```

In [100…

```python
classifiers=[['Decision Tree Classification :',DecisionTreeClassifier()],
       ['Random Forest Classification :',RandomForestClassifier()],
       ['Gradient Boosting Classification :', GradientBoostingClassifier()],
       ['K-Neighbors Classification :',KNeighborsClassifier()],
       ['Support Vector Classification :',SVC()]]
cla_pred=[]
for name, model in classifiers:
    model=model
    model.fit(X_train,y_train)
    predictions = model.predict(X_test)
    cla_pred.append(accuracy_score(y_test,predictions))
    print(name,accuracy_score(y_test,predictions))
```

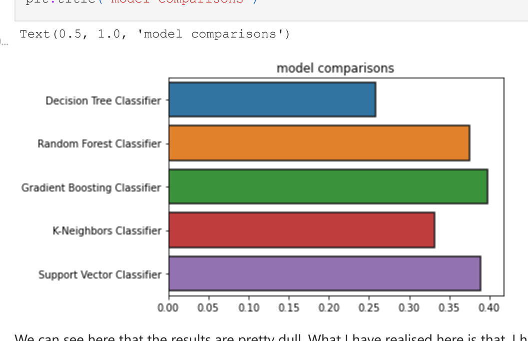
```
Decision Tree Classification : 0.25766871165644173
Random Forest Classification : 0.37576687116564417
Gradient Boosting Classification : 0.39723926380368096
K-Neighbors Classification : 0.3312883435582822
Support Vector Classification : 0.3895705521472393
```

In [110…

```python
y_ax=['Decision Tree Classifier',
      'Random Forest Classifier',
      'Gradient Boosting Classifier',
      'K-Neighbors Classifier',
      'Support Vector Classifier']
x_ax = cla_pred
sns.barplot(x_ax,y_ax,linewidth=1.5,edgecolor="0.1")
plt.title("model comparisons")
```

Out[110…

```
Text(0.5, 1.0, 'model comparisons')
```



We can see here that the results are pretty dull. What I have realised here is that, I have treated it as a multiclass classification problem, and with features that don't really contribute much to the learning parameters. I have identified some ways to have had a better modelling:

- Instead of trying to model **"ValueForMoney"** with 10 distinct values/classes, we could turn it into a binary classification problem. For example, we could have two classes *satisfied* and *dissatisfied*, based on their scoring and try to **then** try to model it.
- Another approach could be creating a scoring mechanism from the five free text responses, and incorporating them to the model and then try to predict the customer experience quantitatively.
- Further work could involve using SHAP values, to facilitate the explanation of how much a single feature affects the prediction.

Conclusion

With that, the project comes to an end. This project has brought out some really interesting insights that Blenheim could use to get rich insights from their survey data. The in-depth exploratory analysis has explained some customer behaviours that are not visible normally. With the help of natural language processing techniques such as topic modelling, sentiment analysis and semantic analysis, and using them in combination extracted some rich topics from the large text corpuses. The K-Means clustering algorithm proved to be useful by grouping similar customers effectively. All in all, Blenheim could use this project as a starting point to develop real time analysis frameworks for understanding their free text responses, which are pivotal to understanding where the ogranisation is placed in customers'minds.

Thank you!