Configuration Manual

MSc Research Project
Data Analytics

Shylesh Veeraraghavan Govindarajulu
Student ID: x21219249

School of Computing
National College of Ireland

Supervisor: Dr. Syed Muslim Jameel.

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shylesh Veeraraghavan Govindarajulu
Student ID:	x21219249
Programme:	Data Analytics
Year:	2022-2023
Module:	MSc Research Project
Supervisor:	Dr. Syed Muslim Jameel
Submission Due Date	14 / 08 / 2023
Project Title:	Ireland Tourism demand forecasting using social media big data with a new machine analytical approach
Word Count:	2011 words
Page count:	13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Shylesh Veeraraghavan Govindarajulu

Date: 14/08/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Shylesh Veeraraghavan Govindarajulu
X21219249

1 Introduction

This configuration manual comprises of the basic setup and requirements for building a framework for interpreting social media data, specifically Twitter for this research. This document lists the steps and the Python libraries required for extracting the tweets and preprocessing the text data. The main aim is to employ a machine-learning approach to forecast the tourist numbers who are coming to Ireland. Topic A sentiment analysis is done on a dataset of tweets to get the end user's opinions on social media. A text blob a Python library is used to conduct sentiment analysis on the dataset of tweets. Matplotlib and seaborn python libraries are used to plot the visualization of the final data.

2 Hardware Requirements

1	Device Name	MSI GF 65
2	Processor	11th Gen Intel(R) Core (TM) i5-1135G7 @ 2.40GHz 2.42 GHz
3	RAM	16.0 GB (15.8 GB usable)
4	Type	64-bit operating system, x64-based processor

3 Software Requirements

Anaconda Navigator
Jupyter Notebook or Google Colab
Python 3.6.3 version
Python libraries like keras, sci-kit learn and tensorflow

In this study, Python is the programming language which is used for the creation and evaluation of the machine learning model. The jupyter Notebook is used as the main tool for the research which was versatile and flexible with the requirements of the project. To ensure a cohesive development ecosystem replete with the necessary Python libraries, we employed Anaconda Navigator.

The Anaconda Navigator served as both our development interface and a debugger for the Python scripts. My system, running Windows 11, was equipped with the 64-bit version of Anaconda Navigator. For those interested in replicating our environment, the software can be obtained from the official Anaconda documentation: Anaconda Navigator. In depth details in relation to the specific Python libraries and their purpose of use in the research will be discussed in the forthcoming sections of this configuration manual.

https://docs.anaconda.com/free/navigator/install/

4 List of Python Libraries Installed

4.1. Data Collection and Manipulation

- **pandas:** For data analysis and manipulation.
- **requests:** For making HTTP requests (only if data is fetched via APIs).
- **Json:** For handling JSON formatted data (usually when dealing with APIs).

4.2. Topic modeling and NLP

- **Spacy:** Advanced NLP and tokenization.
- **gensim:** For topic modeling and document similarity.
- **emoji, regex:** For handling emojis and regular expressions.
- **wordcloud:** For creating word cloud visualizations.

4.3. Data Visualization

- **Matplotlib:** Basic plotting library.
- **Seaborn:** Statistical data visualization based on ‘matplotlib’.
- **Plotly:** For interactive plots**.**
- **PyLDAvis:** For interactive topic model visualization.
- **Chart_studio:** For online publishing of ‘plotly’ visualizations.

4.4. Sentiment analysis

- **Textblob:** Simple library for NLP tasks, including sentiment analysis.

4.5. Machine Learning Model Building

- **Scikit learn:** Comprehensive ML library with a range of algorithms, tools for model selection, evaluation metrics, etc.
- **xgboost:** Gradient boosting library that provides an efficient implementation of the gradient boosting algorithm.

5 Data collection

```
In [1]: pip install snscrape

        Requirement already satisfied: snscrape in c:\users\msi\anaconda3\lib\site-packages (0.5.0.20230113)
        Requirement already satisfied: beautifulsoup4 in c:\users\msi\anaconda3\lib\site-packages (from snscrape) (4.11.1)
        Requirement already satisfied: requests[socks] in c:\users\msi\anaconda3\lib\site-packages (from snscrape) (2.27.1)
        Requirement already satisfied: filelock in c:\users\msi\anaconda3\lib\site-packages (from snscrape) (3.6.0)
        Requirement already satisfied: lxml in c:\users\msi\anaconda3\lib\site-packages (from snscrape) (4.8.0)
        Requirement already satisfied: soupsieve>1.2 in c:\users\msi\anaconda3\lib\site-packages (from beautifulsoup4->snscrape) (2.3.
        1)
        Requirement already satisfied: idna<4,>=2.5 in c:\users\msi\anaconda3\lib\site-packages (from requests[socks]->snscrape) (3.3)
        Requirement already satisfied: certifi>=2017.4.17 in c:\users\msi\anaconda3\lib\site-packages (from requests[socks]->snscrape)
        (2021.10.8)
        Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\msi\anaconda3\lib\site-packages (from requests[socks]->sns
        crape) (2.0.4)
        Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\msi\anaconda3\lib\site-packages (from requests[socks]->snscrap
        e) (1.26.9)
        Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in c:\users\msi\anaconda3\lib\site-packages (from requests[socks]->snscra
        pe) (1.7.1)
        Note: you may need to restart the kernel to use updated packages.

In [2]: import snscrape.modules.twitter as sntwitter
        import pandas as pd

In [7]: query = "ireland Ireland lang:en until:2015-12-31 since:2015-01-01"
        tweets = []
        limit = 10000

In [8]: for tweet in sntwitter.TwitterSearchScraper(query).get_items():

            # print(vars(tweet))
            # break
            if len(tweets) == limit:
                break
            else:
                tweets.append([tweet.date, tweet.content])
```

Figure 1: Tweets extracted from twitter

Tweets from the year 2012 to 2016 with the keyword Ireland has been collected and for each year 10000 tweets were collected using python library snscrape. The tweets of these five years are merged together in SQL server.

6 Data cleaning

The dataset used in this research consists of text data which are tweets extracted from Twitter, so the process of data cleaning involves text preprocessing which is the step used in the process of NLP.

Text preprocessing is a crucial step in many natural language processing (NLP) and machine learning tasks. The exact preprocessing steps often depend on the specific task at hand. However, a general framework for text preprocessing typically includes the following steps:

The first step of text preprocessing is text cleaning which involves removing the URLs, special characters, and punctuation. This process also involves conversion to lowercase letters and removing numbers.

```
# Step 2: Text cleaning
def clean_text(text):
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text)
    # Remove special characters and punctuation
    text = re.sub(r'[^\w\s]', '', text)
    # Convert to lowercase
    text = text.lower()
    # Remove numbers
    text = re.sub(r'\d+', '', text)
    return text
```

Figure 2: Text cleaning

Lemmatization is a linguistic process that involves converting a word to its base or root form. This is especially valuable in natural language processing (NLP) tasks to reduce the number of distinct words or tokens in a text and to understand the essential meaning of the content. This process is carried out in the cleaned text.

```
In [23]:

# Apply tokenizer
df['lemma_tokens'] = df['lemmas_back_to_text'].apply(tokenize)

In [24]:

# Create a id2word dictionary
id2word = Dictionary(df['lemma_tokens'])
print(len(id2word))

48191

In [25]:

# Filtering Extremes
id2word.filter_extremes(no_below=2, no_above=.99)
print(len(id2word))

16940

In [26]:

# Creating a corpus object
corpus = [id2word.doc2bow(d) for d in df['lemma_tokens']]
```

Figure 3: Lemmatization

Text is broken into individual words or tokens which is a part of text preprocessing. The `tokenize` function processes a given text to extract semantic units, or tokens, from it. Initially, the function removes URLs using regular expressions. It then employs a series of `re. sub()` operations to eliminate various non-alphanumeric characters, punctuation, words containing numbers, and specific symbols like '@', '!', and '$'. Several `strip()` methods are tested sequentially to remove certain trailing or leading characters like ', ', '?', '!', '"', and '. '. Finally, the text is converted to lowercase and split into individual words or tokens. The resulting list of tokens is then returned.

```python
# Tokenizer function
def tokenize(text):
    """
    Parses a string into a list of semantic units (words)
    Args:
        text (str): The string that the function will tokenize.
    Returns:
        list: tokens parsed out
    """
    # Removing url's
    pattern = r"http\S+"

    tokens = re.sub(pattern, "", text) # https://www.youtube.com/watch?v=O2onA4r5UaY
    tokens = re.sub('[^a-zA-Z 0-9]', '', text)
    tokens = re.sub('[%s]' % re.escape(string.punctuation), '', text) # Remove punctuation
    tokens = re.sub('\w*\d\w*', '', text) # Remove words containing numbers
    tokens = re.sub('@*!*\$*', '', text) # Remove @ ! $
    tokens = tokens.strip(',') # TESTING THIS LINE
    tokens = tokens.strip('?') # TESTING THIS LINE
    tokens = tokens.strip('!') # TESTING THIS LINE
    tokens = tokens.strip("'") # TESTING THIS LINE
    tokens = tokens.strip(".") # TESTING THIS LINE

    tokens = tokens.lower().split() # Make text lowercase and split it

    return tokens
```

Figure 4: Tokenization

7 Topic modeling

7.1. LDA base model

The provided code initiates a topic modeling task using the LDA (Latent Dirichlet Allocation) method from the `gensim` library. After initializing the LDA model with the `LdaMulticore` function on a given corpus and specifying 5 topics, the model's derived topics are fetched with `base_model. print_topics()`. This output is parsed to extract the most representative words for each topic using regular expressions. The top words for each topic are then combined into a space-separated string and stored in a list. Finally, the code loops through the topics list, displaying each topic's number and its corresponding top words in a user-friendly format.

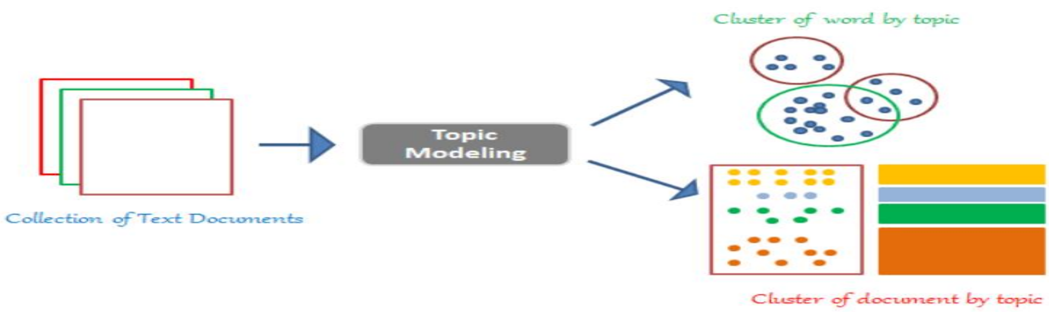

Figure 5: Topic modeling using LDA

```
In [27]:
# Instantiating a Base LDA model
base_model = LdaMulticore(corpus=corpus, num_topics=5, id2word=id2word, workers=12, passes=5)

In [28]:
# Filtering for words
words = [re.findall(r'"([^"]*)"',t[1]) for t in base_model.print_topics()]

In [29]:
# Create Topics
topics = [' '.join(t[0:10]) for t in words]

In [30]:
# Getting the topics
for id, t in enumerate(topics):
    print(f"------ Topic {id} ------")
    print(t, end="\n\n")

------ Topic 0 ------
ireland not northern m uk come live know s right

------ Topic 1 ------
ireland northern m amp s brexit not love irish state

------ Topic 2 ------
ireland job hire come good time jobfairy apply irish travel

------ Topic 3 ------
ireland year day good love new northern christmas c talk

------ Topic 4 ------
ireland new year go amp m today northern not day
```

Figure 6: LDA base model

```
------ Topic 0 ------
ireland not northern m uk come live know s right

------ Topic 1 ------
ireland northern m amp s brexit not love irish state

------ Topic 2 ------
ireland job hire come good time jobfairy apply irish travel

------ Topic 3 ------
ireland year day good love new northern christmas c talk

------ Topic 4 ------
ireland new year go amp m today northern not day
```

Figure 7: Topics extracted from base LDA model

7.2. Hyperparameter tuning

The code starts by transforming a DataFrame column `df['lemmas_back_to_text']` into a document-term matrix using the `CountVectorizer`. This matrix is then processed using the Latent Dirichlet Allocation (LDA) model for topic modeling. To find the best hyperparameters for the LDA model, grid search is employed with specified parameters for the number of topics (`n_components`) and the learning decay (`learning_decay`). Using the `GridSearchCV` class, the optimal LDA model is determined from a combination of provided hyperparameters. Once the best model is found, its perplexity—a measure of how well the model predicts the sample—is printed, which can help evaluate the model's quality on the given data.

```
In [44]:

vectorizer = CountVectorizer()
data_vectorized = vectorizer.fit_transform(df['lemmas_back_to_text'])

In [45]:
# Define Search Param
search_params = {'n_components': [10, 15, 20, 25, 30], 'learning_decay': [.5, .7, .9]}

In [46]:
# Init the Model
lda = LatentDirichletAllocation()

In [47]:
# Init Grid Search Class
model = GridSearchCV(lda, param_grid=search_params)

In [48]:
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import LatentDirichletAllocation

In [49]:
# Create a GridSearchCV instance
grid_search = GridSearchCV(lda, search_params, cv=None)

In [50]:
# Fit the GridSearchCV instance
grid_search.fit(data_vectorized)
Out[50]:
GridSearchCV(estimator=LatentDirichletAllocation(),
             param_grid={'learning_decay': [0.5, 0.7, 0.9],
                         'n_components': [10, 15, 20, 25, 30]})

In [51]:
# Best Model
best_lda_model = grid_search.best_estimator_

In [52]:
# Perplexity
print("Model Perplexity: ", best_lda_model.perplexity(data_vectorized))
Model Perplexity:  3663.2120524271145
```

Figure 8: Grid search

7.3. Important parameters selection to build the final LDA model

This code initializes an instance of the LDA (Latent Dirichlet Allocation) model using the `LdaMulticore` method from the `gensim` library. This method is specifically optimized to run on multiple CPU cores. Here's the breakdown of the provided parameters:

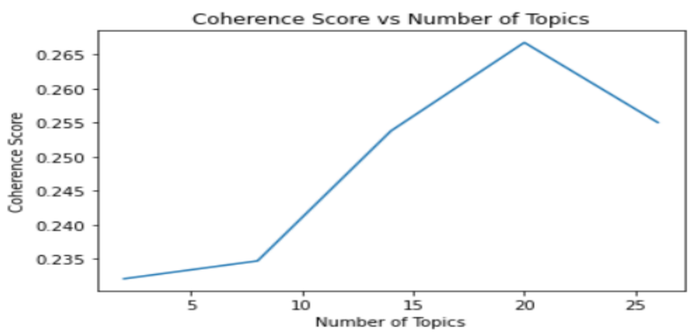

Figure 9: Coherence Score vs Number of Topics

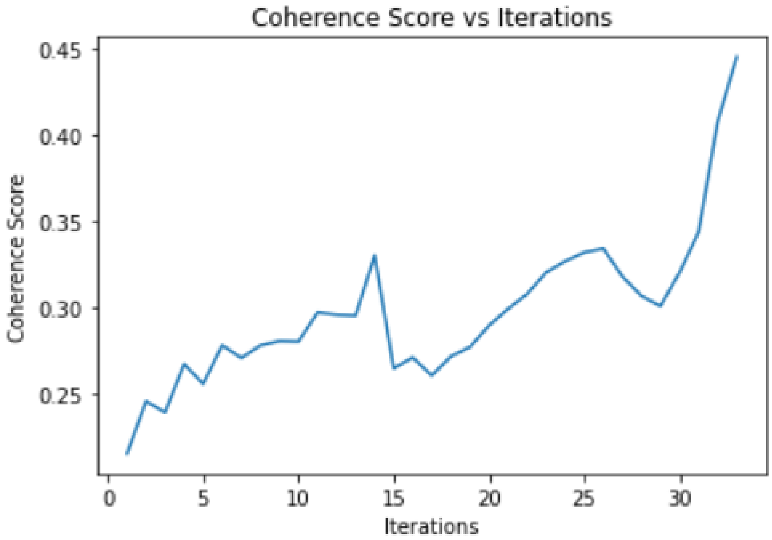

Figure 10: Coherence Score vs Number of Iterations

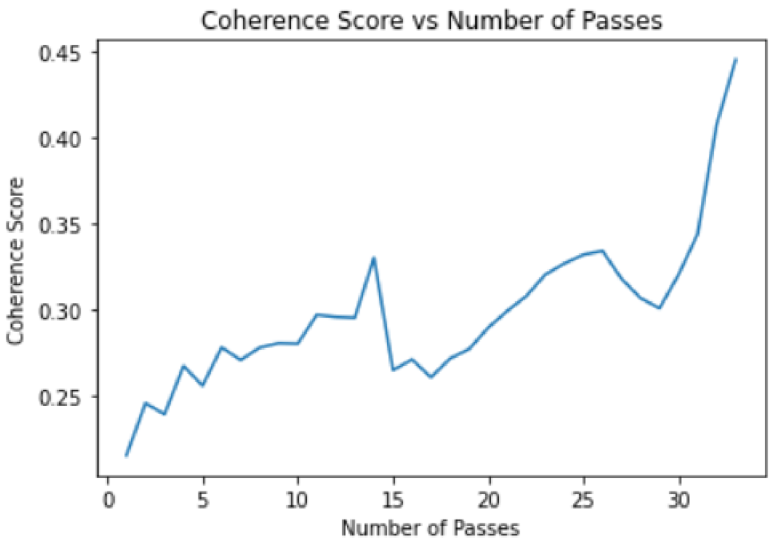

Figure 11: Coherence Score vs Number of Passes

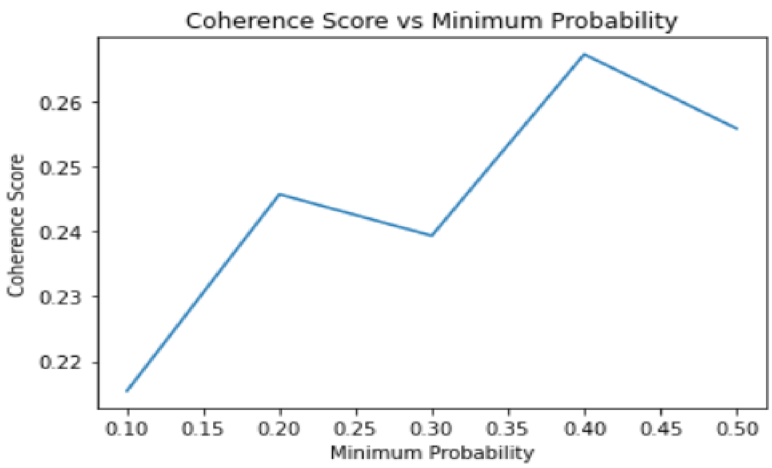

Figure 12: Coherence Score vs Minimum probability

7.3. Final LDA model

In [112]:

```
final_model = LdaMulticore(corpus=corpus,
                           id2word=id2word,
                           num_topics=15,
                           random_state=42,
                           chunksize=2000,
                           passes=30,
                           decay=0.9,
                           iterations=30)
```

Figure 13: Final LDA model

- corpus=corpus : The dataset being passed to the model. In topic modeling, a corpus is a collection of documents.
- id2word=id2word : A mapping from word IDs to words. This helps the model know the vocabulary of the corpus and is used for interpreting topics.
- num_topics=15 : Specifies that the model should identify 15 distinct topics within the provided corpus.
- random_state=42 : A seed for the random number generator to ensure reproducibility. Using the same seed will give the same results across different runs with the same data.
- chunksize=2000 : The number of document samples the training algorithm will use in each update. Larger chunk sizes speed up the training at the expense of memory.
- passes=30 : The number of times the entire corpus will be processed. Multiple passes can help in achieving a more accurate topic distribution, especially for larger corpora.
- decay=0. 9: A hyperparameter that controls the learning rate in the online learning method. Values closer to 1 will give more weight to newer batches of documents, while values closer to 0 will give more weight to older batches.
- iterations=30: The maximum number of times the model will iterate over each document's topic distribution during the E-step of the algorithm.

In summary, the code initializes a more finely-tuned LDA model using the `LdaMulticore` function from the `gensim` library. This model aims to discover 15 distinct topics in the given corpus with the specified hyperparameters for training.

A topic distance visualisation of 15 topics extracted using the pyLDAvis as shown below.

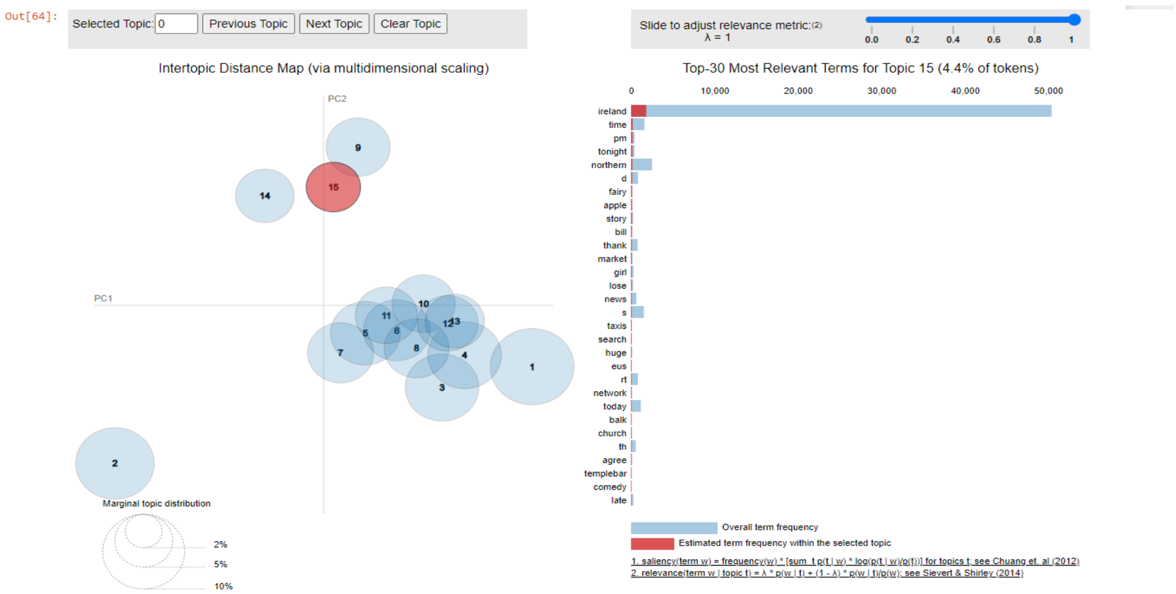

Figure 14: Topic distance visualization

Topics

Jobs
Weather
Travel
City
Irish whiskey
Christmas
Football
Business environment
Party
Accomodation
Guinness
Castle
Beach
New year
Marketing

Figure 15: Topics extracted

The text describes a visualization of 15 topics derived from tweets via the LDA model, with each topic represented by a circle whose size indicates its frequency. On the visualization's top-right, an adjustable Lambda (λ) value, set to 0. 6, dictates term importance within topics. Beside this, the 30 most defining terms of each topic are displayed. Topics are named by discerning a theme from their most frequent words, then vetted for relevance to tourism. Out of the original 15, only 15 topics were deemed pertinent to tourism after this evaluation.

8 Sentiment analysis

```
df['sentiment_score'] = df['cleaned_text'].apply(lambda x: TextBlob(x).sentiment.polarity)
```

In [9]:

```
# Define a function to map sentiment scores to sentiment labels
def get_sentiment_label(score):
    if score > 0:
        return 'Positive'
    elif score < 0:
        return 'Negative'
    else:
        return 'Neutral'
```

In [10]:

```
# Create a new column to store sentiment labels
df['sentiment_label'] = df['sentiment_score'].apply(get_sentiment_label)
```

In [11]:

```
average_sentiment = df.groupby(['year', 'topic'])['sentiment_score'].mean()
```

In [12]:

```
# Print the average sentiment scores for each year
print(average_sentiment)
```

```
year  topic
2012  Accomodation             0.084315
      Beach                    0.096475
      Business_environment     0.096970
      Castle                   0.127609
      Christmas                0.036421
                                 ...
2016  Marketing                0.131685
      New_year                 0.103870
      Party                    0.135216
      Travel                   0.078834
      Weather                  0.085697
Name: sentiment_score, Length: 75, dtype: float64
```

Figure 16: Sentiment analysis using the Topics extracted

The code processes a DataFrame, `df`, to analyze the sentiment of its `cleaned_text` column using the `TextBlob` library. For each entry, it computes a sentiment polarity score, ranging from -1 (negative) to 1 (positive), storing this in a new column `sentiment_score`. Subsequently, a function `get_sentiment_label` classifies these scores into 'Positive', 'Negative', or 'Neutral' categories, which are then added to the DataFrame in the `sentiment_label` column. The code concludes by calculating the average sentiment score for each combination of year and topic within the dataset, printing these average sentiment values for analysis.

```
year  topic
2012  Accomodation            0.084315
      Beach                   0.096475
      Business_environment    0.096970
      Castle                  0.127609
      Christmas               0.036421
      City                    0.126911
      Football                0.119702
      Guinness                0.152036
      Irish_whiskey           0.114448
      Jobs                    0.074316
      Marketing               0.152155
      New_year                0.075823
      Party                   0.108749
      Travel                  0.068532
      Weather                 0.166184
Name: sentiment_score, dtype: float64
```

```
year  topic
2013  Accomodation            0.108027
      Beach                   0.097330
      Business_environment    0.093878
      Castle                  0.115747
      Christmas              -0.021324
      City                    0.126083
      Football                0.103697
      Guinness                0.117023
      Irish_whiskey           0.147238
      Jobs                    0.065545
      Marketing               0.101842
      New_year                0.047901
      Party                   0.038654
      Travel                  0.080978
      Weather                 0.072784
Name: sentiment_score, dtype: float64
```

```
year  topic
2014  Accomodation            0.104757
      Beach                   0.091660
      Business_environment    0.113549
      Castle                  0.107202
      Christmas               0.061265
      City                    0.125734
      Football                0.035144
      Guinness                0.165301
      Irish_whiskey           0.101134
      Jobs                    0.070339
      Marketing               0.192960
      New_year                0.082843
      Party                   0.117541
      Travel                  0.064872
      Weather                 0.076887
Name: sentiment_score, dtype: float64
```

```
year  topic
2015  Accomodation            0.110905
      Beach                   0.087082
      Business_environment    0.096184
      Castle                  0.135335
      Christmas               0.034769
      City                    0.159984
      Football                0.143018
      Guinness                0.113907
      Irish_whiskey           0.078979
      Jobs                    0.056790
      Marketing               0.128027
      New_year                0.023103
      Party                   0.112730
      Travel                  0.056944
      Weather                 0.091246
Name: sentiment_score, dtype: float64
```

```
year  topic
2016  Accomodation            0.092492
      Beach                   0.121487
      Business_environment    0.126670
      Castle                  0.122464
      Christmas               0.045462
      City                    0.116594
      Football                0.094098
      Guinness                0.148729
      Irish_whiskey           0.135020
      Jobs                    0.110687
      Marketing               0.131685
      New_year                0.103870
      Party                   0.135216
      Travel                  0.078834
      Weather                 0.085697
Name: sentiment_score, dtype: float64
```

Figure 17: Mean sentiment score of the topics extracted

9 Tourist demand forecasting using machine learning models

The tourist arrival numbers column contains data taken from the Ireland tourism website https://www.tourismireland.com/ and the data visualisations are shown below.

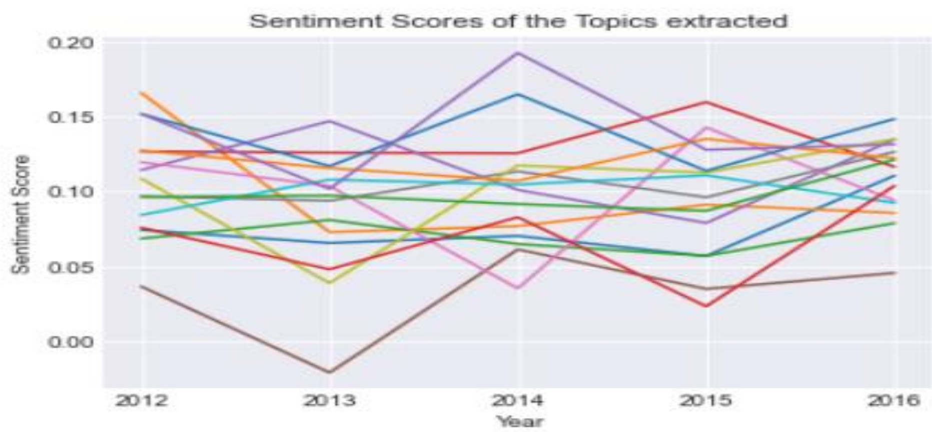

Figure 18: Mean sentiment score of the topics extracted for five years

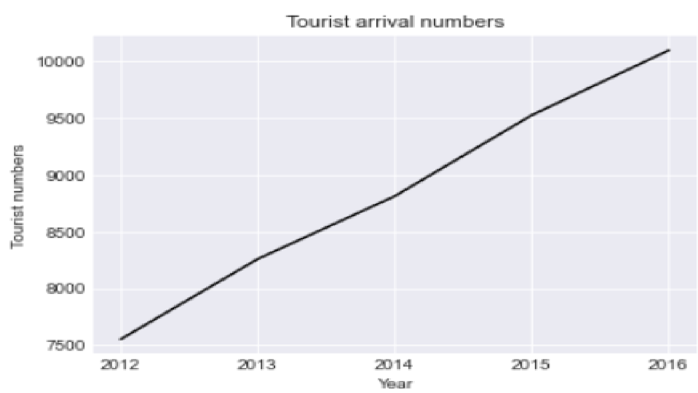

Figure 19: Tourist arrival numbers for five years

The code assigns a subset of columns from the `data` DataFrame to `X`, representing the features, while setting the 'Tourist arrival numbers' column as the target variable `y` for potential modeling or analysis.

```
# Separate the features (X) and the target variable (y)
X = data[['Year', 'Jobs', 'Weather', 'Travel', 'City', 'Irish_whiskey', 'Christmas', 'Fo
       'Beach', 'New_year', 'Marketing']]
y = data['Tourist arrival numbers']
```

```
In [17]:
loo = LeaveOneOut()
```

Figure 20: Dependent and Independent variable

Four supervised machine learning models linear regression, random forest, SVR and XGboost were built to forecast the tourist numbers coming to Ireland. The performance metrics used for the evaluation of this models are MAE and MAPE. The results and the plots of the different models are shown below.

Performance of the forecasting models

Model	MAE (mean)	MAPE (mean)
Linear regression	59	0.6 %
Random forest	818	8.1 %
SVR	1554	15 %
XGboost	315	3.1 %

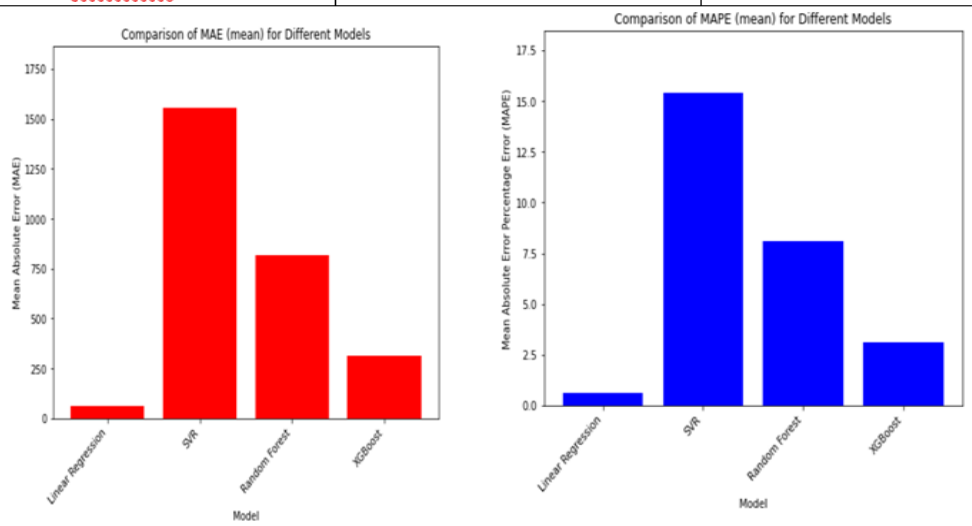

Figure 21: Comparison of MAE and MAPE of different models

References

Anaconda Navigator (no date). Available at: https://docs.anaconda.com/free/navigator/index.html (Accessed: 1 August 2023).

Scikit Library (no date). Available at: https://pypi.org/project/scikit-learn/ (Accessed: 1 August 2023).

Ireland tourism. Available at: https://www.tourismireland.com/(Accessed: 1 August 2023).