National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Komal Vijay Bhalerao
Student ID: x20135386

School of Computing
National College of Ireland

Supervisor: Prof. Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Komal Vijay Bhalerao
Student ID:	x20135386
Programme:	MSc in Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Prof. Christian Horn
Submission Due Date:	16-08-21
Project Title:	Configuration Manual
Word Count:	1203
Page Count:	11

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:	Komal Vijay Bhalerao
Date:	15th August 2021

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

Komal Vijay Bhalerao
x20135386

1 Introduction

The configuration documentation explains how to run the implemented scripts for the present research topic. This will ensure that the code runs smoothly and without errors. This also contains information about the hardware setup of the machine upon which scripts are run, as well as the same suggested minimum configuration. Following these procedures will aid in the replication of the project's outcomes. This can then be analyzed, and further research can be done with ease.

2 System Configuration

2.1 Hardware Configuration

- Device name: LAPTOP-7CVAFID5
- Processor: Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
- Installed RAM: 8.00 GB (7.80 GB usable)
- System type: 64-bit operating system, x64-based processor

2.2 Software Configuration

The project was implemented using the Python-based Jupyter notebook IDE (Integrated Development Environment) included in the Anaconda package. The steps to execute the developed scripts are illustrated in the following sections.

3 Downloads and Installation

- **Python**
  This research project is carried out using Python. It has remarkable and noteworthy number of supporting models for Machine Learning and Deep Learning. It also has number of libraries and several modules that helps with smooth preprocessing, altering images, ease of use and implementation. Therefore, the initial requirement for running the script on the computer is to have latest version of python downloaded. This can be accomplished by browsing to the python website1 download page [1] and downloading the software installer for the desired version

[1] https://www.python.org/downloads/

based on the operating system of the computer that will be running it. Figure 1 depicts the screenshot of official python website from where the latest version can be installed. After downloading, by following the installation instruction the file must be installed.

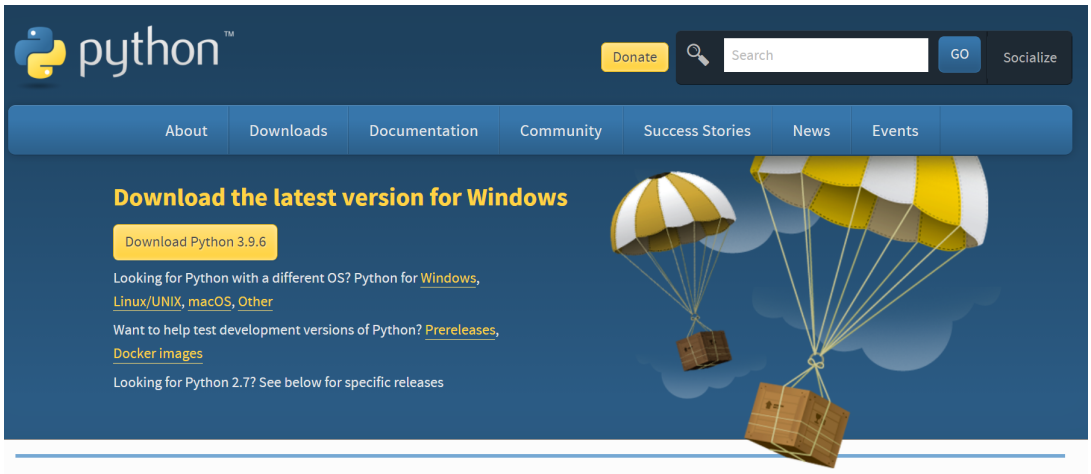

Fig.1 Download page of Python

The success of the installation can be confirmed using the 'python –version' query in the Windows command prompt. It tells you what Python version you have installed.

- **Anaconda**
  The next package to be downloaded is Anaconda. It offers a number of user-friendly Python-based IDEs that may be used for code development and viewing of outcomes. The most popular IDEs available in Anaconda Navigator on installation are Jupyter Notebook and Spyder.It can be downloaded from the offical website [2] for different OS so the OS specific installer has to be downloaded. On successful downloading and installing Anaconda Navigator, multiple IDEs are displayed that can be selected according to the need of the develpoment. Among all the IDEs available Jupyter IDE is used in this research project.

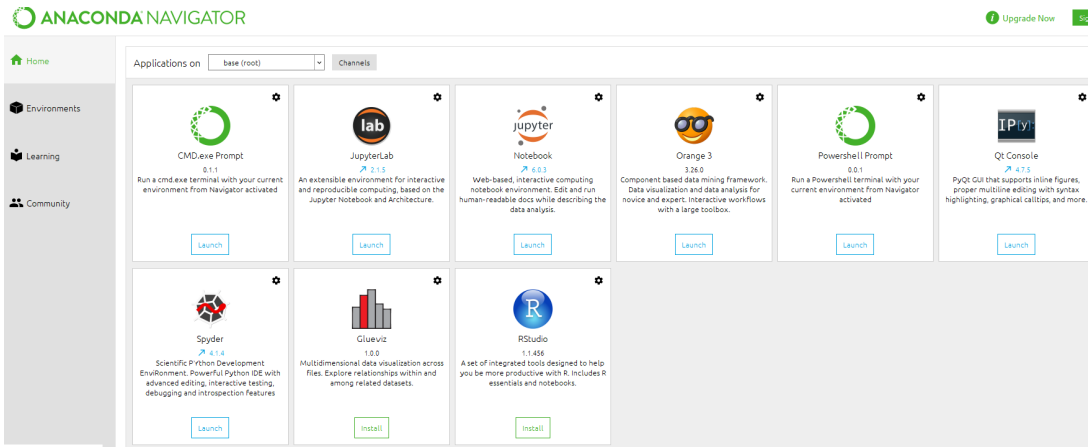

Fig.2 Anaconda Navigator

[2]https://www.anaconda.com/products/individual

- **Data Source:** This research uses the Amazon's product reviews dataset and its meta data available online [3]. The author has distributed the data into categories as there are numerous amount of products available on Amazon. From the number of categories available a subset of "All_Beauty" products are used for the research. The original data and the meta data available for all beauty products is used for the project.

4 Project development

As shown in Figure 3, Jupyter Notebook should be launched from the navigator installed. As you launch the jupyter IDE, a new tab is opened in the browser.

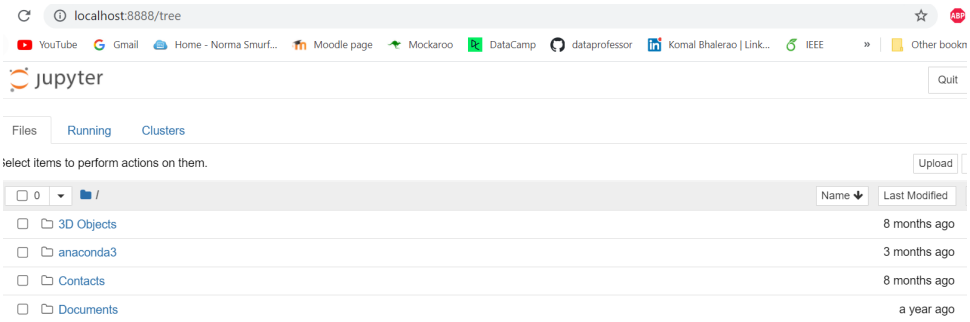

Fig.3 Jupyter Notebook home page

A new python 3 notebook can be created and suitable name for the file can be given. This is the initial stage of coding. The file that we are creating is in .ipynb format. As the project carries out the implementation of Machine Learning models, some additional libraries of python need to be installed when required. These libraries can be installed using pip command in the command prompt or on the Jupyter Notebook.
Example:
For command prompt : **pip install numpy**
For Jupyter Notebook: **!pip install numpy**

Firstly, some of the standard libraries required for building model for Sentiment Analysis are installed.The latest versions of these libraries are installed and some of the standard libraries include:

- Scikit-Learn
- Numpy
- pandas
- Sklearn
- Matplotlib
- Gensim
- nltk

[3]http://deepyeti.ucsd.edu/jianmo/amazon/index.html

Once the coding is complete, the script can be launched using the jupyter notebook command or by running the code in blocks. If there are any mistakes in the code, they will be displayed below the code block, where they can be debugged.

As the process of running the model begins, converting and fetching the data in a pandas dataframe is required. As the source file was in json.gz format and the meta data was a large file so in order to decompress the files gzip is imported and then it is converted to dataframe as shown in figure 4.

```
import gzip

def parse(path):
  g = gzip.open(path, 'rb')
  for l in g:
    yield eval(l)

def getDF(path):
  i = 0
  df = {}
  for d in parse(path):
    df[i] = d
    i += 1
  return pd.DataFrame.from_dict(df, orient='index')

dfmeta = getDF('C:/Users/KOMAL BHALERAO/OneDrive/Desktop/beauty/meta_All_Beauty.json.gz')
```

Fig.4

4.1 Text Pre-processing

```
In [1]: # Dataframe
        import pandas as pd

        # Array
        import numpy as np

        # Decompress the file
        import gzip

        # Visualizations
        import matplotlib.pyplot as plt
        from matplotlib.colors import ListedColormap
        import seaborn as sns
        import matplotlib.colors as colors
        %matplotlib inline

        # Datetime
        from datetime import datetime

        ## Warnings
        import warnings
        from scipy import stats
        warnings.filterwarnings('ignore')

        # Large dataset
        import dask.bag as db
```

Fig.5

Figure 5 depicts the imported libraries required for the model development.

```
In [9]: product_reviews=pd.merge(review_df,dfmeta,on='asin',how='left')
```

Figure 6.

This block shows the merging of original data and meta data.

```
[13]: #Dropping Unnecessary columns

      review_df1=product_reviews.drop(['reviewerName','image','verified','feature','also_view','similar_item', 'imageURL','imageURLHi
```

```
[22]: review_df2['review_Text'] = review_df2[['summary', 'reviewText']].apply(lambda x: " ".join(str(y) for y in x if str(y) != 'nan
      review_df3 = review_df2.drop(['reviewText', 'summary'], axis = 1)
      review_df3.head(5)
```

```
[23]: #Retriving a subset of "perfume" from all the beauty products

      review_df4 = review_df3[review_df3["title"].str.contains("perfume|perfumes|Perfume|Perfumes")]
```

Figure 7.

In the first block, unnecessary columns that bring no value are discarded.

Second block depicts the merging of two columns 'Summary' and 'reviewText' to a new column named 'review_Text'.

In the next block, a subset of perfume is retrieved from the entire set by passing a search string of "perfume—perfumes—Perfume—Perfumes".

```
In [27]: #Classification of ratings in good or bad

         good_rate = len(review_df4[review_df4['Ratings'] >= 3])
         bad_rate = len(review_df4[review_df4['Ratings'] < 3])

         # Printing rates and their total numbers
         print ('Good ratings : {} reviews for perfumes'.format(good_rate))
         print ('Bad ratings : {} reviews for perfumes'.format(bad_rate))
```

```
Good ratings : 68 reviews for perfumes
Bad ratings : 8 reviews for perfumes
```

```
In [28]: #Applying the classification to rating column
         review_df3['rating_class'] = review_df3['Ratings'].apply(lambda x: 'bad' if x < 3 else'good')
         review_df3.head()
```

Figure 8.

In figure 8, the ratings provided by the user is classified as good or bad. The ratings are in the range of 1-5 where 1 is the lowest and 5 is the highest rating. The rating which is less than 3 is classified as 'Bad' and the rating which is equal to or greater than 3 is classified as 'Good'. In the second block, the classified rating is applied to the rating column and a rating_class column is formed.

```
In [31]:
#Total reviews
total = len(review_df4)
print ("Number of reviews: ",total)
print ()
```

```
================================================
Number of reviews:  16234
```

```
In [32]:
#Unique reviewers
print ("Number of unique reviewers: ",len(review_df4.reviewer_id.unique()))
reviewer_prop = float(len(review_df4.reviewer_id.unique())/total)
print ("Prop of unique reviewers: ",round(reviewer_prop,3))
print ()
```

```
Number of unique reviewers:  13762
Prop of unique reviewers:  0.848
```

```
In [33]:
#Unique Products
print ("Number of unique products: ", len(review_df4.product_id.unique()))
product_prop = float(len(review_df4.product_id.unique())/total)
print ("Prop of unique products: ",round(product_prop,3))
print ()
```

```
Number of unique products:  1128
Prop of unique products:  0.069
```

Figure 9.

Figure 9 depicts some of the statistical analysis carried out in the project.

```
In [36]:
plt.figure(figsize=(12,8))
review_df4['ratings'].value_counts().sort_index().plot(kind='bar')
plt.title('Distribution of Rating')
plt.xlabel('Rating')
plt.ylabel('Number of Reviews')
```

```
Out[36]: Text(0, 0.5, 'Number of Reviews')
```

Figure 10.

This is used to plot bar chart which shows the total number of ratings for each class

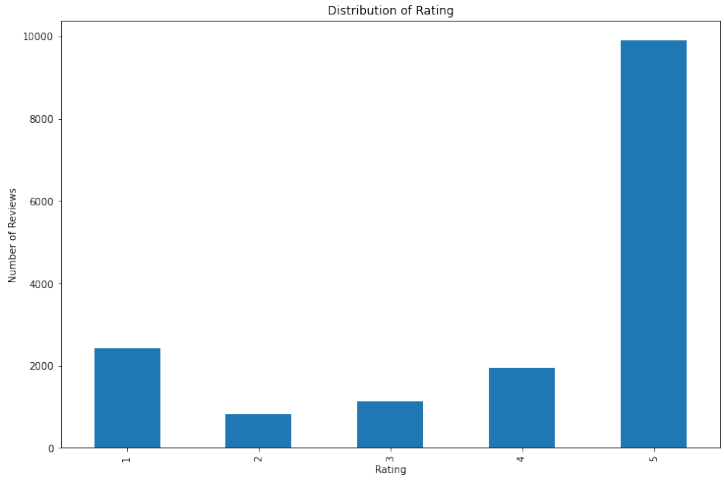

Figure 11.

```
In [39]: import nltk
         nltk.download('punkt')
         nltk.download('stopwords')
         nltk.download('wordnet')
         from nltk.corpus import stopwords
         from bs4 import BeautifulSoup
         import unicodedata
         #import contractions
         #from contractions import CONTRACTION_MAP
         from nltk.stem import WordNetLemmatizer
         from nltk.tokenize import word_tokenize, sent_tokenize, regexp_tokenize
         from nltk.stem import PorterStemmer, WordNetLemmatizer
         import re
```

```
In [42]: def strip_html(text):
             soup = BeautifulSoup(text, "html.parser")
             return soup.get_text()

         def remove_between_square_brackets(text):
             return re.sub('\[[^]]*\]', '', text)

         def denoise_text(text):
             text = strip_html(text)
             text = remove_between_square_brackets(text)
             return text


         # special_characters removal
         def remove_special_characters(text, remove_digits=True):
             pattern = r'[^a-zA-z0-9\s]' if not remove_digits else r'[^a-zA-z\s]'
             text = re.sub(pattern, '', text)
             return text


         def remove_non_ascii(words):
             """Remove non-ASCII characters from list of tokenized words"""
             new_words = []
             for word in words:
                 new_word = unicodedata.normalize('NFKD', word).encode('ascii', 'ignore').decode('utf-8', 'ignore')
                 new_words.append(new_word)
             return new_words
```

Figure 12

The above block depicts some of libraries imported and defined functions for text pre-processing.

4.2 Model Development

```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import learning_curve
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer, HashingVectorizer
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from  sklearn.metrics import precision_recall_fscore_support
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from gensim.models import Word2Vec
from tqdm import tqdm
```

Figure 13

Importing all the libraries required for Model building

```
In [58]: df1['rating_class'] = df1['rating_class'].apply(lambda x: 0 if x == 'bad' else 1)

In [59]: # Splitting the Data Set into Train and Test Sets
         X = df1['clean_text']
         y = df1['rating_class']

In [60]: # Splitting Dataset into train and test set with a ratio of 75(train):25(test)
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

In [61]: print ('Train Set Shape\t\t:{}\nTest Set Shape\t\t:{}'.format(X_train.shape, X_test.shape))

         Train Set Shape         :(12175,)
         Test Set Shape          :(4059,)
```

Figure 14

In this block, the data is split into test and train in the ratio of 75:25.

Following blocks of code shows the models defined for various algorithms.

Support Vector Machine:

```
In [72]: #model function calling for SVM with CountVector
         modeling(SVC())

         # Assign y_pred to a variable for further process
         y_pred_cv_svc = y_pred
```

Figure 21

Naive Bayes:

```
In [69]: # model function calling for Naive Bayes with CountVector
         modeling(MultinomialNB())

         # Assign y_pred to a variable for further process
         y_pred_cv_nb = y_pred
```

Figure 22

Logistic Regression:

```
In [66]: # model function calling for logistic regression with CountVector
         modeling(LogisticRegression(multi_class = 'multinomial', solver = 'newton-cg',
                                     class_weight = 'balanced', C = 0.1, n_jobs = -1, random_state = 42))

         # Assigning y_pred to a variable
         y_pred_cv_logreg = y_pred
```

Figure 23

```
In [65]: def modeling(Model, Xtrain = count_vect_train, Xtest = count_vect_test):



             model = Model

             # To fit classifier to train set
             model.fit(Xtrain, y_train)

             global y_pred
             #To predict test results
             y_pred = model.predict(Xtest)

             # assigning f1 score to variable
             score = f1_score(y_test, y_pred, average = 'weighted')

             # Printing evaluation metric
             print("f1 score: {}".format(score))
```

Figure 15

The block above depicts the global function defined for modeling and how the data is trained.

```
In [64]:
         # Create the word vector with CountVectorizer
         count_vect = CountVectorizer(ngram_range=(1,1))
         count_vect_train = count_vect.fit_transform(X_train)
         count_vect_train = count_vect_train.toarray()
         count_vect_test = count_vect.transform(X_test)
         count_vect_test = count_vect_test.toarray()
```

```
In [81]: # Create the word vector with TF-IDF Vectorizer
         tfidf_vect = TfidfVectorizer(ngram_range=(1, 1))
         tfidf_vect_train = tfidf_vect.fit_transform(X_train)
         tfidf_vect_train = tfidf_vect_train.toarray()
         tfidf_vect_test = tfidf_vect.transform(X_test)
         tfidf_vect_test = tfidf_vect_test.toarray()
```

Figure 18

The above blocks shows the creation of word vectors with CountVectorizer and TF-IDF Vectorizer

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title = 'Confusion matrix',
                          cmap = plt.cm.ocean):
    """
    Create a confusion matrix plot for 'good' and 'bad' rating values
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis = 1)[:, np.newaxis]
    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title, fontsize = 20)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, fontsize = 20)
    plt.yticks(tick_marks, classes, fontsize = 20)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt), horizontalalignment = "center",
                 color = "white" if cm[i, j] < thresh else "black", fontsize = 40)

    plt.tight_layout()
    plt.ylabel('True Label', fontsize = 30)
    plt.xlabel('Predicted Label', fontsize = 30)

    return plt
```

The above block is a function for creation and displaying confusion matrix

```
In [68]: # Print confusion matrix for logistic regression with countvectorizer
         disp_confusion_matrix(y_pred_cv_logreg, "Logistic Regression")
```

Figure 16

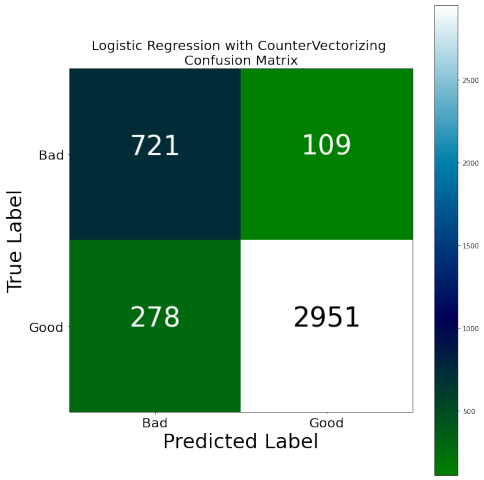

Figure 17

The above block is a function for generating confusion matrix

```
In [75]: # Function for converting the "classification report" results to a dataframe
         def pandas_classification_report(y_true, y_pred):
             metrics_summary = precision_recall_fscore_support(
                     y_true=y_true,
                     y_pred=y_pred)

             avg = list(precision_recall_fscore_support(
                     y_true=y_true,
                     y_pred=y_pred,
                     average='weighted'))

             metrics_sum_index = ['precision', 'recall', 'f1-score', 'support']
             class_report_df = pd.DataFrame(
                 list(metrics_summary),
                 index=metrics_sum_index)

             support = class_report_df.loc['support']
             total = support.sum()
             avg[-1] = total

             class_report_df['weighted avg'] = avg

             return class_report_df.T
```

Figure 19

The above block shows how the model is evaluated in the form of classification report.

```
In [67]: print(classification_report(y_test, y_pred_cv_logreg))
```

	precision	recall	f1-score	support
0	0.72	0.87	0.79	830
1	0.96	0.91	0.94	3229
accuracy			0.90	4059
macro avg	0.84	0.89	0.86	4059
weighted avg	0.91	0.90	0.91	4059

Figure 20

This block generates the classification report for the model using multiple evaluation metrics.