1 CONTENTS

2 INTRODUCTION

This report describes the python program in which I implemented a basic parser to investigate the natural-language posts from Q&A (Question and Answering) site. The parser is able to perform basic data extraction, statistical analysis on a number of linguistic features and present the analysis results using some form of visualization.

This assignment is divided into 3 tasks –

a. Handling file contents and pre-processing
b. Building a class for data analysis
c. Analysing the file for data visualization

3 HANDING FILE CONTENTS AND PRE-PROCESSING

I began by reading in all the posts of the given dataset and conducted a number of pre-processing tasks to clean the post content (Body) needed for analysis in the subsequent tasks (Task 2 and 3). For each post, the body/content is extracted i.e., the string embedded within “Body:"..."” in each row of the XML file. After extraction of body, following pre-processing tasks are carried out:

1. Converting special character references to its original form
2. Replace special characters including “
”, “
” by a single empty space.
3. Remove all HTML tags.
4. Segregate question and answers into separate files based on post type id

There are two functions in the file:

1. preprocessLine(inputLine) for dealing with the each valid data row from the file.
2. splitFile(inputFile, outputFile_question,outputFile_answer) for reading the input file, calling preprocessLine function to process the line, and saving the cleaned questions and answers into output files.

The main file to run this task is **preprocessData.py**.

3.1 IMPORTING REQUIRED LIBRARY

This program requires the use of regular expression operator and hence I imported ‘re’ library and to perform file/directory related activities, ‘os’ library has been imported by default in the code template file.

```
# Importing required libraries
import os
import re
```

Fig 1. Code for importing re and os library

The program begins from ‘__main__’ function where splitFile function is called by passing variables (data.xml, question.txt and answer.txt). Below is the code snippet.

```
if __name__ == "__main__":
    f_data = "data.xml"
    f_question = "question.txt"
    f_answer = "answer.txt"
    splitFile(f_data, f_question, f_answer)
```

Fig 2. Calling splitFile function

3.2 SPLITFILE FUNCTION

The splitFile function accepts three arguments namely data.xml, question.txt and answer.txt. This function opens all the files, reads each line from the data.xml file and checks if there is any row id. If there is row id, post type id is determined and saved in the variable ‘postType’ and then the body/content of the row is extracted using search and group functionality of re library. This extracted body/content of the row is then passed to preprocessLine function. The preprocessLine function returns the clean body after pre processing of that particular line. Once clean body is obtained, on the basis of postType, the post is segregated as a question or as an answer and stored in question.txt and answer.txt respectively. Once all the lines in the input file are processed and are segregated, all the files are closed. Below is the code snippet of the splitFile function.

```
# Splitting the inputFile into output question and answer file
# Inputs : inputFile(data.xml),outputFile_question(question.txt),outputFile_answer(answer.txt)-> string data type
def splitFile(inputFile, outputFile_question, outputFile_answer):

    input_handle = open(inputFile, "r", encoding="utf-8")              # To open data.xml file in read mode
    questionFile = open(outputFile_question, "a", encoding="utf-8")    # To open question.txt file in append mode
    answerFile = open(outputFile_answer, "a", encoding="utf-8")        # To open answer.txt file in append mode

    for line in input_handle:
        # Search for row id in data.xml file
        row = re.search('row Id="(.*?)"', line)
        if row is not None:
            # Search for post type id to determine if entry is a question or an answer
            postType = re.search('PostTypeId="(.*?)"', line)
            postType = postType.group(1).strip()

            # Search for body/content and pass it to preprocessLine method
            body = re.search('Body="(.*?)" />', line)
            body = body.group(1).strip()
            cleanBody = preprocessLine(body)

            # Segregate the content as a question or an answer based on post type id and dump into respective files
            if postType == '1':
                questionFile.write(cleanBody)  # To dump content into the file
                questionFile.write("\n")

            elif postType == '2':
                answerFile.write(cleanBody)  # To dump content into the file
                answerFile.write("\n")

    questionFile.close()    # To close question.txt file
    answerFile.close()      # To close answer.txt file
    input_handle.close()    # To close data.xml file
```

Fig 3. splitFile Function Content

3.3 PREPROCESSLINE FUNCTION

The preprocessLine function accepts one argument – input line which is the line from the data.xml file. This function checks for character references (defined in the table 1) in each line and replaces their original form. Once the character references are handled, all the html tags are removed and then the cleaned input line is returned as function output.

Table 1: Character Reference Transformation.

Character reference	Original form
&amp;	&
&quot;	"
&apos;	'
&gt;	>
&lt;	<

```python
# Method to preprocess the line that is read from data.xml file
# Input : inputLine (string)
def preprocessLine(inputLine):

    # Initializing required flag value
    signFlag = True

    # Initializing character reference transformation dictionary
    characterReferenceTransformation = dict()

    # Setting key-value pair for dictionary
    characterReferenceTransformation = {'&amp;': '&', '&quot;': '"', '&apos;': '\'', '&gt;': '>', '&lt;': '<', '&#xA;': '_', '&#xD;': ' '}

    # Replace the keys with values in inputLine
    while signFlag:
        signFlag = any(sign in inputLine for sign, value in characterReferenceTransformation.items())
        if signFlag is True:
            for sign, value in characterReferenceTransformation.items():
                inputLine = inputLine.replace(sign, value)

    # Using re library functionality to remove xml/html tags
    clean = re.compile('<.*?>')
    inputLine = re.sub(clean, '', inputLine)

    # Stripping the excess spaces in the inputLine after preprocessing
    inputLine = inputLine.strip()

    # Returning the processed inputLine
    return inputLine
```

Fig 4. preprocessLine Function Content

4 BUILDING A CLASS FOR DATA ANALYSIS

The main task here is to further parse the given row of the data in XML format with object-oriented programming. The main file to run this task is **parser.py**.

Similar to the task 1, the program imports re library and preprocessLine function from preprocessData_30513669.py file. Below is the screenshot.

```python
# Importing preprocessLine method from preprocessData_30513669.py
from preprocessData_30513669 import preprocessLine

# Importing required libraries
import re
```

Fig 5. Importing re library and preprocessLine function(task1)

4.1 CLASS METHODS

__init__ method is used to initialise object variables such as inputString (each line from data.xml file), ID (row id from a line), type (post type id from a line), dateQuarter (extract year and quarter from the creation date) and cleanBody (preprocessed line).

__str__ method returns the row id, post type id, creation date quarter and cleaned body in a human friendly readable manner. Below is the code snippet for the two methods.

```
class Parser:

    # Initializing object variables
    def __init__(self, inputString):
        self.inputString = inputString
        self.ID = self.getID()
        self.type = self.getPostType()
        self.dateQuarter = self.getDateQuarter()
        self.cleanBody = self.getCleanedBody()

    # To print in user friendly manner in order of row id, post type id. creation date quarter and cleaned body
    def __str__(self):
        if self.getID() != "":
            endString = "Row ID : " + self.getID() + "\n" + "Post Type ID : " + self.getPostType() + "\n" + "Creation Date Quarter :" + \
                        self.getDateQuarter() + "\n" + "Cleaned Content : " + self.getCleanedBody() + "\n\n"
        else:
            endString = ""
        return endString
```

Fig 6. __init__ and __str__ methods

getID method reads the line from the input string and returns the row id.

getPostType method reads the line from the input string and returns the post type id. Below is the code snippet for the above two methods.

```
# To get the row id of the inputString
def getID(self):
    row = re.search('row Id="(.*?)"', self.inputString)
    if row is not None:
        row = row.group(1).strip()
    else:
        row = ""
    return row

# To get post type id of the inputString
def getPostType(self):
    if self.getID() != "":
        postType = re.search('PostTypeId="(.*?)"', self.inputString)
        postType = postType.group(1).strip()
    else:
        postType = ""
    return postType
```

Fig 7. getID and getPostType methods

getDateQuarter method reads the line from the input string and returns the concatenated value of year and quarter based on creation date. Below is the code snippet.

```
# To get creation data quarter from inputString
def getDateQuarter(self):
    if self.getID() != "":
        # Search for creation date in format yyyy-mm-dd
        creationDate = re.search('CreationDate="(.*?)T', self.inputString)
        creationDate = creationDate.group(1).strip()

        # Search for year from creation date in format yyyy
        year = re.search('(^[0-9]{4})-', creationDate)
        year = year.group(1).strip()

        # Search for month from creation date in format mm
        month = re.search('-([0-9]{2})-', creationDate)
        month = month.group(1).strip()

        # To categorize a month as part of a quarter of a year
        if month == '01' or month == '02' or month == '03':
            month = 'Q1'
        elif month == '04' or month == '05' or month == '06':
            month = 'Q2'
        elif month == '07' or month == '08' or month == '09':
            month = 'Q3'
        elif month == '10' or month == '11' or month == '12':
            month = 'Q4'
        creationDate = year + month
    else:
        creationDate = ""
    return creationDate
```

Fig 8. getDateQuarter method

getCleanedBody method reads the line from the input string and returns the cleaned body by passing the line to preprocessLine method from task 1. Below is the code snippet.

```
# To get cleaned body from inputString
def getCleanedBody(self):
    if self.getID() != "":
        body = re.search('Body="(.*?)" />', self.inputString)
        body = body.group(1).strip()
    else:
        body = ""

    # Passing the body as argument to the preprocessLine method from preprocessData_30513669.py
    cleanBody = preprocessLine(body)
    return cleanBody
```

Fig 9. getCleanedBody method

getVocabularySize method fetches the cleaned body from above method and then removes the punctuations. After that, it counts the number of unique words in the line and returns the count as the output for a particular line. Below is the code snippet.

```
# To get vocabulary size of inputString
def getVocabularySize(self):

    # Initializing a wordList (list)
    wordList = []

    if self.getID() != "":
        # To convert cleaned body into lower case
        cleanedBody = self.getCleanedBody().lower()

        # Removing all the punctuations from the cleaned body
        nonPunctuationCleanedBody = re.sub(r'[!"#\$%&\'\(\)\*\+,-\./:;<=>?@\[\\\]\^_`{|}~]', '', cleanedBody)

        # To convert string into a list by splitting the string based on space
        nonPunctuationCleanedBodyList = nonPunctuationCleanedBody.split()

        # To store unique words from nonPunctuationCleanedBodyList
        for word in nonPunctuationCleanedBodyList:
            if word not in wordList:
                wordList.append(word)
        vocabularySize = len(wordList)
    else:
        vocabularySize = None

    return vocabularySize
```

Fig 10. getVocabularySize method

5 ANALYSING THE FILE FOR DATA VISUALIZATION

This task uses two functions to visualise the statistics as some form of graphs. The implementation of these two functions makes use of the external Python packages, such as NumPy and Matplotlib in order to create the suitable graphs for comparing the statistics collected for posts. The main file to run this task is **dataVisualization.py**.

Similar to the task 1 and 2, the program initializes imports numpy, matplotlib libraries and class Parser from parser_30513669.py file.

```
# Importing class Parser from parser_30513669.py
from parser_30513669 import Parser

# Importing required libraries
import numpy as np
import matplotlib.pyplot as plt
```

Fig 11. Importing required libraries and class Parser from task 2

5.1 VISUALIZEVOCABULARYSIZEDISTRIBUTION METHOD

This method takes two arguments – inputFile (data.xml) and outputImage (vocabularySizeDistribution.png). This method counts the vocabulary size for each row in the file data.xml and then displays a bar chart to visualize the distribution of the vocabulary size of all posts. The x-axis is the vocabulary size, and the y-axis represents the number of posts with certain vocabulary size. For the x-axis, the vocabulary size is in the range 0-9, 10-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90-99, others (left inclusive or > 100). The final plot is saved into a file named as “vocabularySizeDistribution.png”.

Below is the code snippet for this method.

```
# Importing class Parser from parser_30513669.py
from parser_30513669 import Parser

# Importing required libraries
import numpy as np
import matplotlib.pyplot as plt


# Method to draw bar graph to visualize the distribution of the vocabulary size of all posts
# Inputs : inputFile (data.xml) and outputImage (wordNumberDistribution.png)
def visualizeWordDistribution(inputFile, outputImage):

    # Initializing variables to plot graph
    width = 20
    height = 20
    pixel = 160

    # To open data.xml file
    input_handle = open(inputFile, "r", encoding="utf-8")

    # Setting x axis (vocabulary range)
    vocabularyRange = ('0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79', '80-89', '90-99', 'Others')
    lenOfVocabularyRange = np.arange(len(vocabularyRange))

    # Initializing noOfLines list to store total number of lines per vocabulary range
    noOfLines = [0] * lenOfVocabularyRange
```

Fig 12. Initialising required variables, opening file, setting x axis and initialising number of lines list

```
# Initializing noOfLines list to store total number of lines per vocabulary range
noOfLines = [0] * lenOfVocabularyRange

for line in input_handle:
    parseLine = Parser(line)
    if parseLine.getID() != "":
        vocabularySize = parseLine.getVocabularySize()
        if 0 <= vocabularySize <= 9:
            noOfLines[0] += 1
        elif 10 <= vocabularySize <= 19:
            noOfLines[1] += 1
        elif 20 <= vocabularySize <= 29:
            noOfLines[2] += 1
        elif 30 <= vocabularySize <= 39:
            noOfLines[3] += 1
        elif 40 <= vocabularySize <= 49:
            noOfLines[4] += 1
        elif 50 <= vocabularySize <= 59:
            noOfLines[5] += 1
        elif 60 <= vocabularySize <= 69:
            noOfLines[6] += 1
        elif 70 <= vocabularySize <= 79:
            noOfLines[7] += 1
        elif 80 <= vocabularySize <= 89:
            noOfLines[8] += 1
        elif 90 <= vocabularySize <= 99:
            noOfLines[9] += 1
        elif vocabularySize >= 100:
            noOfLines[10] += 1

input_handle.close()        # To close data.xml file
```

Fig 13. Increasing count of number of lines that have vocabulary size in a particular range and closing the file

```
# Setting size and dpi of the graph
plt.figure(figsize=(width, height), dpi=pixel)

# Plotting bar graph vocabulary size vs number of posts in a range
plt.bar(lenOfVocabularyRange, noOfLines, align='center')
plt.xticks(lenOfVocabularyRange, vocabularyRange, rotation=30)
plt.ylabel('Number Of Posts Within Vocabulary Size')
plt.xlabel('Vocabulary Size')
plt.title('Vocabulary Size Distribution')

# Saving the output image file into wordNumberDistribution.png
plt.savefig(outputImage)
#plt.show()

# Clear the plot so that next graph can be displayed without overlapping
plt.clf()
```

Fig 14. Setting, plotting and saving the graph

wordNumberDistribution.png

Bar graph for word number distribution

5.2 VISUALIZEPOSTNUMBERTREND METHOD

This method takes two arguments – inputFile (data.xml) and outputImage (postNumberTrend.png). This method displays the trend of the post number in the Q&A site. The number of questions and answers in each quarter is fetched and then the data is displayed in form of a line chart. The graph consists of questions as well as answers trend. The final plot is saved into a file named as "postNumberTrend.png". Below is the code snippet for the above method.

```
# Method to display trend of the post numbers in the Q&A site
# Inputs : inputFile (data.xml) and outputImage (postNumberTrend.png)
def visualizePostNumberTrend(inputFile, outputImage):

    # Initializing variables to plot graph
    width = 20
    height = 20
    pixel = 160

    # Initializing quarter year list
    quarter = []

    # Setting x axis (year - quarter)
    quarter = ['2015Q2', '2015Q3', '2015Q4', '2016Q1', '2016Q2', '2016Q3', '2016Q4', '2017Q1', '2017Q2', '2017Q3', '2017Q4',
               '2018Q1', '2018Q2', '2018Q3', '2018Q4', '2019Q1', '2019Q2', '2019Q3', '2019Q4']

    question = [0] * len(quarter)
    answer = [0] * len(quarter)


    input_handle = open(inputFile, "r", encoding="utf-8")  # To open data.xml file
    for line in input_handle:
        parseLine = Parser(line)
        if parseLine.getID() != "":
            if parseLine.getPostType() == '1':
                question[quarter.index(parseLine.getDateQuarter())] += 1
            elif parseLine.getPostType() == '2':
                answer[quarter.index(parseLine.getDateQuarter())] += 1

    input_handle.close()        # To close data.xml file
```

Fig 15. Initializing variables, quarter year list, setting x axis and fetching post type id

```
# Setting size and dpi of the graph
plt.figure(figsize=(width, height), dpi=pixel)

# Plotting line chart to annotate the number of posts in each quarter
plt.plot(quarter, question, color='green', label='Question')
plt.plot(quarter, answer, color='orange', label='Answer')
plt.xlabel('Year - Quarter')
plt.ylabel('Number Of Questions and Answers')
plt.title('Post Number Trend')
plt.legend()

# Saving the output image file into postNumberTrend.png
plt.savefig(outputImage)
#plt.show()

# Clear the plot so that next graph can be displayed without overlapping
plt.clf()
```

Fig 16. Setting, plotting and saving the graph

postNumberTrend.png

Line graph for post number trend

6 ASSUMPTIONS

Following assumptions have been made in the tasks.

6.1 TASK 1

1. Files are expected to be placed in same folder.
2. The body/content exists in a line only if row id exists.
3. There are only seven character reference transformations to be done as per the given table.
4. The tags considered are the ones starting with ‘<’ and ending with ‘>’ and the content in between them is removed.

6.2 TASK 2

1. Similar to that of task 1. Also, in getVocabularySize method, the punctuations considered to be removed are as follows: *!"#$%&'()*+,-.:;<=>?@\[]^_`{|}~*.

6.3 TASK 3

1. For the first method which displays graph for vocabulary size, the x axis range is considered to be as follows: '0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79', '80-89', '90-99', 'Others' to avoid overlap between highest number in a group and lowest number in the next group.
2. For the second method which displays line graph for the post trend, the x axis range is considered as follows: '2015Q2', '2015Q3', '2015Q4', '2016Q1', '2016Q2', '2016Q3', '2016Q4', '2017Q1', '2017Q2', '2017Q3', '2017Q4', '2018Q1', '2018Q2', '2018Q3', '2018Q4', '2019Q1', '2019Q2', '2019Q3', '2019Q4'.

7 BENEFITS OF THIS CODE

The following are the benefits of this code :

7.1 TASK 1

1. This program uses the flag which helps is easy execution of the loops.
2. Dictionary is used to store the character reference transformations. To add new character references, user can simply add new character reference as key and the original form as its value.
3. For any operation, row id is checked. If row id is present, operations are carried out.
4. re library is used for pattern searching and replacing which is easy to understand and implement

7.2 TASK 2

1. In every class method, row id is checked first and then the operations are performed. By doing so, I eliminated the possibility of checking the lines which do not have body/content in it based on the assumption.

7.3 TASK 3

1. This program uses numpy and matplotlib libraries.
2. For any operation, row id is checked. If row id is present, operations are carried out.
3. The height, width and pixels of the graph is set and can also be altered as desired by the user.

8 LIMITATIONS OF THIS CODE

In task 3, both the methods have their x-axis defined in a hard-coded manner. As an alternative, the x-axis can be generated dynamically by reading the maximum and minimum value of the count and increase in steps of 10 for vocabulary size distribution method whereas the x axis can be generated dynamically by reading all the unique values of the creation date quarter with the help of getDateQuarter method from class Parser for post trend method.

But upon doing so, there is a need to iterate throughout the file to get the desired values. This may in turn affect the efficiency or timing of the execution of this code if the data set has millions of records.

As this dataset has about 10,000 records, file operations such as opening, writing and closing are possible quickly. Hence, the above alternative could be implemented for this specific case.