Muskan Gupta (J016)
Ojasvi Jain (J022)

REPORT SUMMARIZATION

Natural Language Processing

TABLE OF CONTENTS

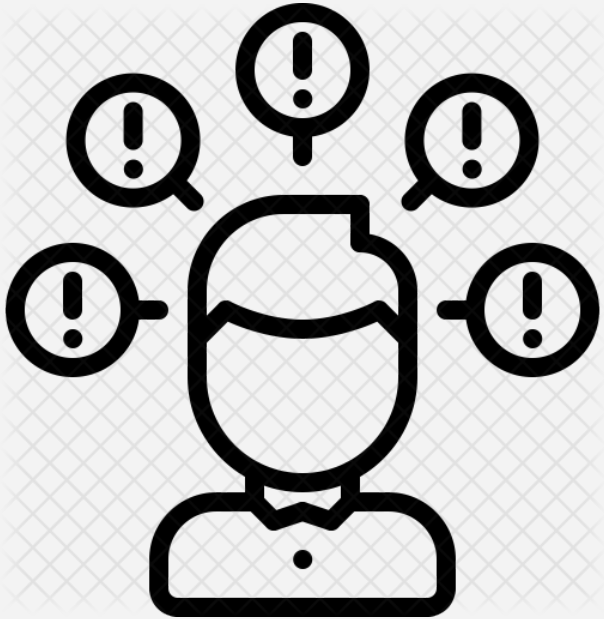
MOTIVATION

Marketing and Financial Reports generated are usually highly comprehensive. As a result of this, it becomes really difficult to go through the entire report to know the key insights in a short span of time.

Hence, creating a summarization tool will help in efficiently getting the key insights for understanding the gist of the report in a short span of time.

It will also be beneficial for people not well-versed with the technical jargon, but looking to get some key market insights from a company’s official annual reports.

PROBLEM STATEMENT



To summarize financial and marketing reports where all the text and images are got with respect to the relevant topic.

UNDERSTANDING THE PROBLEM

The problem statement can be broken down into three main tasks:

1

TEXT

Extraction of all the text is one of the most important tasks, as this is what we will summarize to get the key insights.

2

IMAGES

Next thing we need to focus on is getting the graphs and images present in the pdf report with respect to the appropriate topic.

3

STRUCTURE

Now, the main task lies in combining all of the summarised data to get our summary in a structured format to present to the user.

OUR APPROACH

The pipeline followed by us was as follows:

First, we need to focus on extraction of all of the data from the PDF, whether it is in the form of text, images or graphs.

EXTRACTION

Next, after we have successfully extracted all our data, we focus on summarization of the data, with respect to the relevant topics to have some reference.

SUMMARIZATION

The next important task is to display the data in a manner that it is an accurate representation and gives us the important content.

DISPLAY

MODEL SELECTION

The following models and libraries were best fit for our implementation:

As opposed to other summarisation algorithms like NLTK, TextRank, BERT helps us extract context with its bi-directional functionality

BERT
SUMMARISE

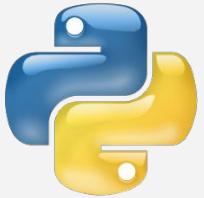
Unlike Camelot & Excalibur, a similar library for table extraction, tabula allows us to map the page numbers and get tables only from the required pages individually.

TABULA
EXTRACT TABLE

Fitz enables the user to identify images in the PDF and also extract them while mapping them back to the page numbers and hence was the best fit.

FITZ
EXTRACT IMAGE

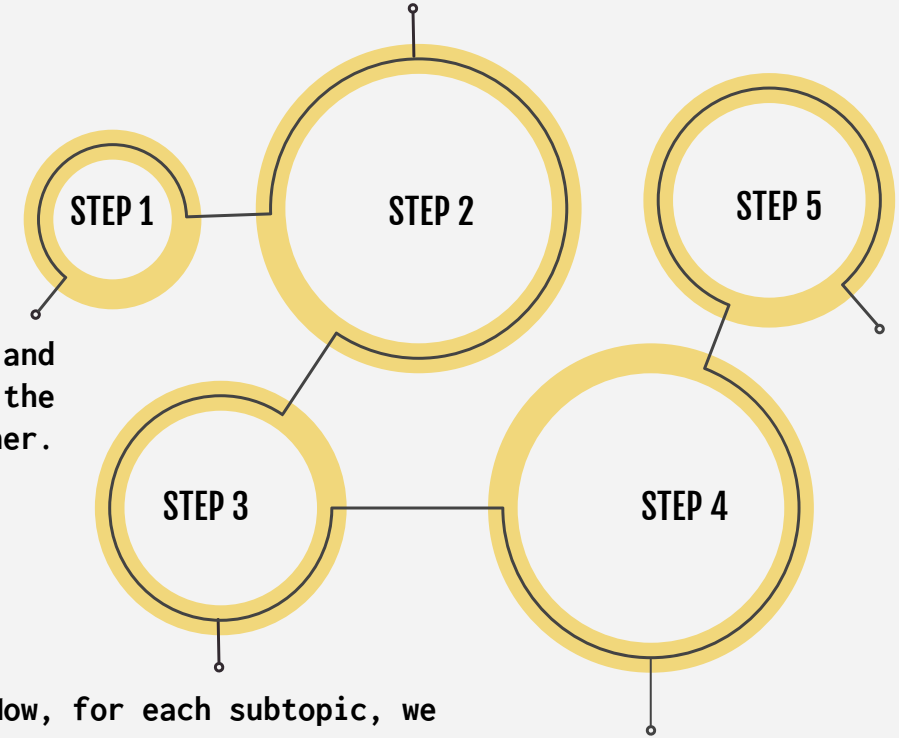
LIBRARIES USED



SR.NO.	LIBRARY:	USE:
1.	PyPDF2	This library was mainly used for extracting the text content from the PDF.
2.	re	This was used for making regular expressions.
3.	fitz	This library was used to extract all the images present in the PDF.
4.	tabula	This library was used to extract all the tabular data from the PDF.
5.	bert-extractive-summarizer	This was used to summarize all of our textual data.
6.	pandas	This was used for all of the Data Frames functionality.

OUR PROCESS

We get all the major topics, subtopics in these subsequent topics and the page numbers they start on.

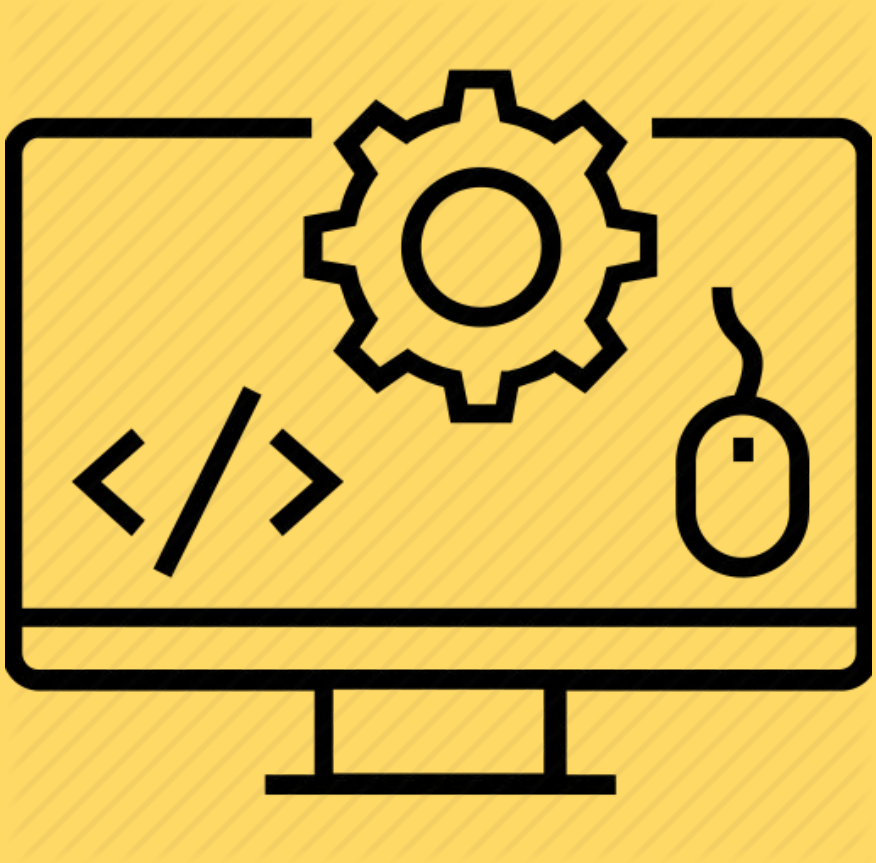


We load the PDF and extract all the content altogether.

Our final Data Frame is ready now with all the extractions, which we will use to display using Dash.

Now, for each subtopic, we extract the text content with respect to that particular subtopic.

Next, we extract all the tables and images which fall under each subtopic.



BACKEND CODE

FUNCTIONS USED

read_pdf(path):

- This function is used for reading and extracting all the information from the PDF.
- We have initialised **info** to an empty list.
- Using a for loop, we iterated through all the pages in the PDF, to read and extract the content from each page.
- After this, the content was appended to **info**, and we got the contents of the entire PDF.

```
def read_pdf(path):
    info = []
    with open(path, 'rb') as fil:
        pdf = PyPDF2.PdfFileReader(fil)
        pages = pdf.getNumPages()
        for pageno in range(pages):
            page = pdf.getPage(pageno)
            content = page.extractText()
            info.append(content)
    cont = ''
    for i in range(3, len(info)):
        cont+=info[i]
    cont = cont.replace('\n', ' ')
    return info, cont
```

FUNCTIONS USED

make_df(list1, list2):

- This function is used for creating a dataframe.
- **list1:** This is our list of topics and subtopics.
- **list2:** This is our list of page numbers corresponding to each subtopic.
- Hence, we use loops to get a Data Frame with 3 columns:
  - Topics
  - Subtopics
  - Page Numbers

```
def make_df(list1, list2):
  result = []
  k = 0
  for i in list1:
    if type(i) == str:
      topic = i
    else:
      if i:
        for j in range(len(i)):
          result.append([topic, i[j], list2[k]])
          k+=1
      else:
        result.append([topic, topic, list2[k]])
        k+=1
  df = pd.DataFrame(result,
                    columns=['topics', 'subtopics', 'page_no'])
  return df
```

FUNCTIONS USED

extract_text(column_name):

- This function is used to extract text under each corresponding subtopic.
- The way we have implemented this is by using regular expressions.
- We created regular expressions for finding the subtopic and the next subtopic in the document.
- Now, we extract all the textual content between these two subtopics to get our data for all the subtopics in the dataframe.

```
def extract_text(column_name):
  text = []
  df = make_df(index, pages)
  _, cont = read_pdf(path)
  for i in range(df.shape[0]):
    sub1 = df[column_name][i]
    try:
      sub2 = df[column_name][i+1]
    except KeyError:
      sub2 = 'lol'
    if sub2 not in cont.split(sub1, 1)[-1]:
      text.append('')
    else:
      text.append(cont.split(sub1, 1)[-1].split(sub2, 1)[0])
  return text
```

FUNCTIONS USED

extract_summary(topic):

- This function is used for summarizing our textual data under each subtopic.
- Here, we use the Bert Summarizer.
- We will understand how the summarizer works exactly in the following slide.

```
def extract_summary(topic):
  s = Summarizer()
  res = s(topic, min_length=10)
  result = ''.join(res)
  return result
```

BERT-SUMMARIZER ARCHITECTURE

B - **B**idirectional
E - **E**mbedding
R - **R**epresentations &
T - **T**ransformers

Google's BERT is a **pre-trained Transformer model** that has achieved ground-breaking performance on multiple NLP tasks.
Extractive Summarization is the task of selecting sub segments of text from the original text that would create a good summary.

This package utilizes the **HuggingFace transformers library** to run extractive summarizations.
BERT learns from unlabeled text by analyzing the preceding and subsequent text around each word (hence "bidirectional").
This works by first embedding the sentences, then running a clustering algorithm, finding the sentences that are closest to the cluster's centroids.

BERT-SUMMARIZER ARCHITECTURE

B - **B**idirectional
E - **E**mbedding
R - **R**epresentations &
T - **T**ransformers

BERT presents a few differences with the Transformer in how they handle input sequences with the objective of increasing the number of tasks the model can work on.

The transformer which performs token and positional embeddings use pre trained token embeddings and pre-learned positional embeddings.

BERT-SUMMARIZER ARCHITECTURE

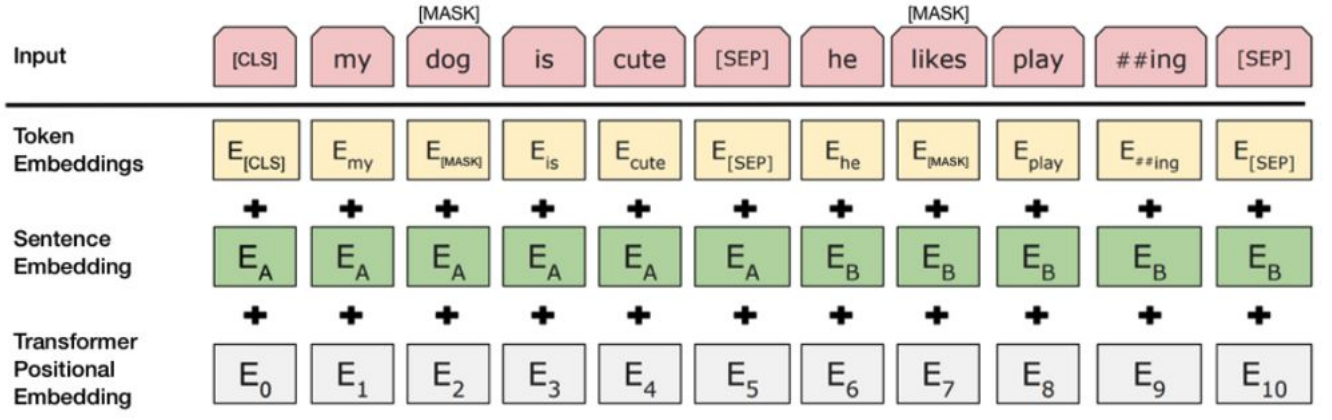
B - **B**idirectional
E - **E**mbedding
R - **R**epresentations &
T - **T**ransformers

To be able to solve two sentence problems like question answering and next sentence prediction (Pre-training Task 2), they added a [SEP] token to mark the end of a sentence and added a sentence embedding (this embedding is constant for each sentence but different across two sentences). This allows the model to easily determine where one sentence ends and the other begins and to give different meanings to different sentences.

To allow for classification tasks, the authors added a [CLS] token at the beginning of the input sequence, to be used only if classifying.

BERT-SUMMARIZER ARCHITECTURE

B - **B**idirectional
E - **E**mbedding
R - **R**epresentations &
T - **T**ransformers



FUNCTIONS USED

extract_table(topic):

- This function is used to extract the tabular data under each topic.
- This has been implemented using **tabula**.
- For each subtopic, the page it starts on, and ends on, is identified.
- After this, the tables are extracted only for these pages to get the table present under the subtopic that is selected.

```
def extract_table(topic):
  df = make_df(index, pages)
  page_start= int(df.loc[df['subtopics'] == topic,
                         'page_no'])+1
  ind= int(df[df['subtopics']==topic].index.values)
  ind+=1
  try:
    page_end= int(df.loc[df['subtopics'] ==
                         df['subtopics'][ind], 'page_no'])
  except KeyError:
    page_end= len(info)
  tables= []
  for page in (page_start, page_end):
      table = tabula.read_pdf(path, pages = page,
                              multiple_tables = True)
      tables.append(table)
  return len(tables), tables
```

TABULA OUTPUT

The table is extracted in the form of a dataframe and if there are multiple tables, a list of dataframes is made.

	Description	Budget	Estimated
0	Marketing Team Salaries	₹20,00,000	₹20,00,000
1	Sales Personnel	₹20,00,000	₹20,00,000
2	Website/App	₹7,00,000	₹5,00,000
3	Social Media Marketing:	₹40,00,000	₹40,00,000
4	a. Facebook	₹13,00,000	₹13,00,000
5	b. Instagram	₹9,00,000	₹9,00,000
6	c. Youtube	₹12,00,000	₹12,00,000
7	d. Google Ads	₹6,00,000	₹6,00,000
8	Magazine Ads (Vogue India, Elle India and Femi...	₹10,00,000	₹10,00,000
9	Celebrity Collaboration	₹25,00,000	₹24,00,000
10	PR kits	₹3,00,000	₹2,50,000
11	Free giveaways	₹50,000	₹50,000
12	Total	₹1,25,50,000	₹1,22,00,000

FUNCTIONS USED

extract_image(topic):

- This function is created to extract all the images present under a particular topic.
- This has been implemented using **fitz**.
- For each subtopic, the page it starts on, and ends on, is identified.
- After this, the images are extracted only for these pages, so we can get them with respect to the topic.
- Using fitz, these images are extracted and then stored in the PNG format.

```
def extract_image(topic):
  df = make_df(index, pages)
  doc= fitz.open(path)
  page_start= int(df.loc[df['subtopics']
                         == topic, 'page_no'])+1
  ind= int(df[df['subtopics']==topic].index.values)
  ind+=1
  print(page_start)
  page_end= int(df.loc[df['subtopics'] ==
                       df['subtopics'][ind], 'page_no'])+1
  print(page_end)
  for page in (page_start, page_end):
      for img in doc.getPageImageList(page):
          xref = img[0]
          pix = fitz.Pixmap(doc, xref)
          if pix.n < 5:
              pix.writePNG("p%s-%s.png" % (page, xref))
          else:
              pix1 = fitz.Pixmap(fitz.csRGB, pix)
              pix1.writePNG("p%s-%s.png" % (page, xref))
              pix1 = None
          pix = None
```

Dash

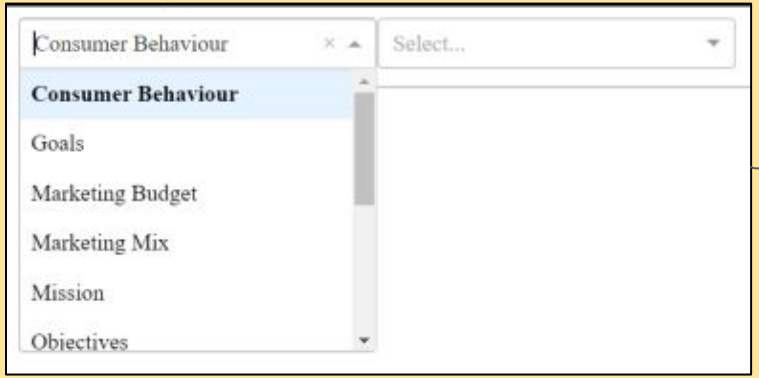
by plotly

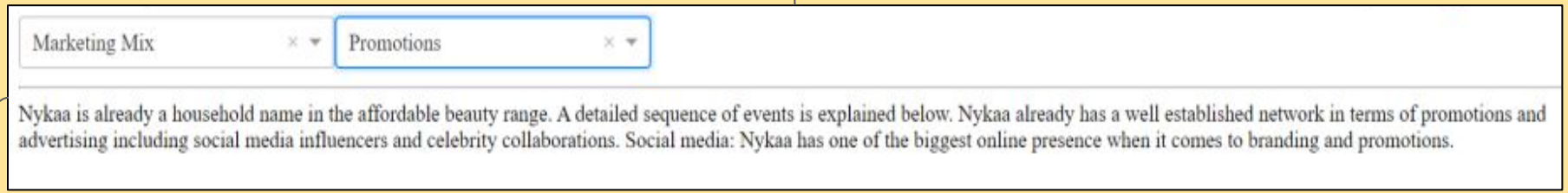
FRONTEND CODE

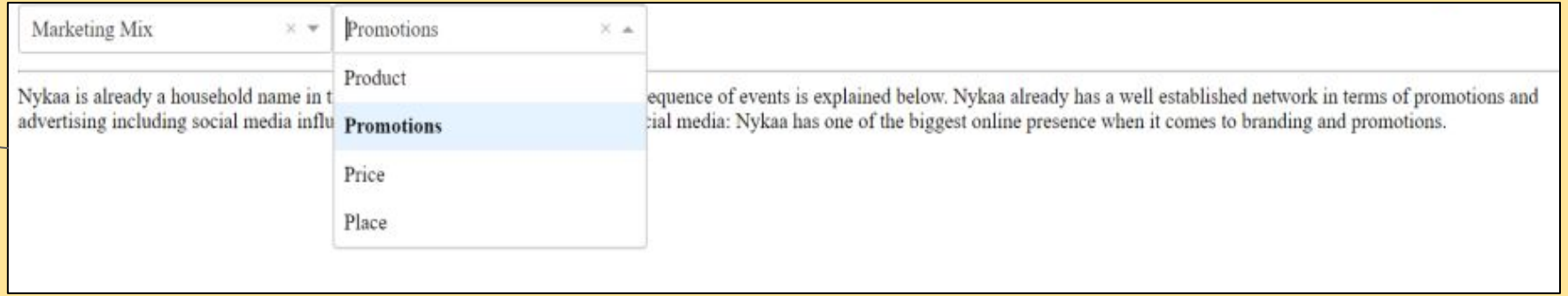
INTRODUCTION

- **Dash** is a **Python** framework for building web applications. It is a user interface library for creating analytical web applications.
- It is built on top of Flask, Plotly. js, React and React Js.
- It enables you to build reactive dashboards using pure **Python**.
- **Dash** is open source, and its apps run on the web browser. Dash is a user interface library for creating analytical web applications.

Output Screenshots







FUTURE SCOPE

1. The project is widely scalable. Essentially being able to automate the whole process by taking a PDF from the user and generating the summary with reference to the context and topic is the main future scope.
2. This entire process of automating the procedure of getting a summarization of the entire report has a lot of application as it can help individuals understand really long reports effectively and in a short period of time.
3. Although working with PDFs is a challenge, majority of the functionalities of our model cater to the different outputs that can be generated from a PDF. However, being able to use the template for getting the text in the most appropriate form is a challenge to be improved upon in the future.

THANK YOU.