IST 615 Cloud Management - Project Report

MedEn: Medical Search Engine

Course Instructor:

Carlos E. Caicedo Bastidas

Project by:

Yash Senjaliya | Trishla Jain | Shaunak Edakhe | Suyash Jadhav

Table of Contents

Aim / Objective

To create a machine learning application that can recognize the relationship and pattern between distinct terms used together in medical science, a smart search engine for records containing those keywords, and, finally, an Azure machine learning pipeline to deploy and scale the program.

Project Context

We've all probably wondered why when we search for a specific term on Google, it doesn't simply return results that also contain that word, but also results that are really closely linked to it. For instance, searching Google for the term "medicine" returns results that include the word "medicine" as well as terms such as "health," "pharmacy," and "WHO." Google therefore recognizes that these phrases are connected in some way. Word embeddings have a role in this situation. Word embeddings are numerical representations of the words in a phrase that vary based on context. In this project, we have utilized word embeddings to develop a clever search engine, specifically for medical science. The project is executed or deployed using multiple services on Azure. We have also created a Docker image for our application.

Technology Stack

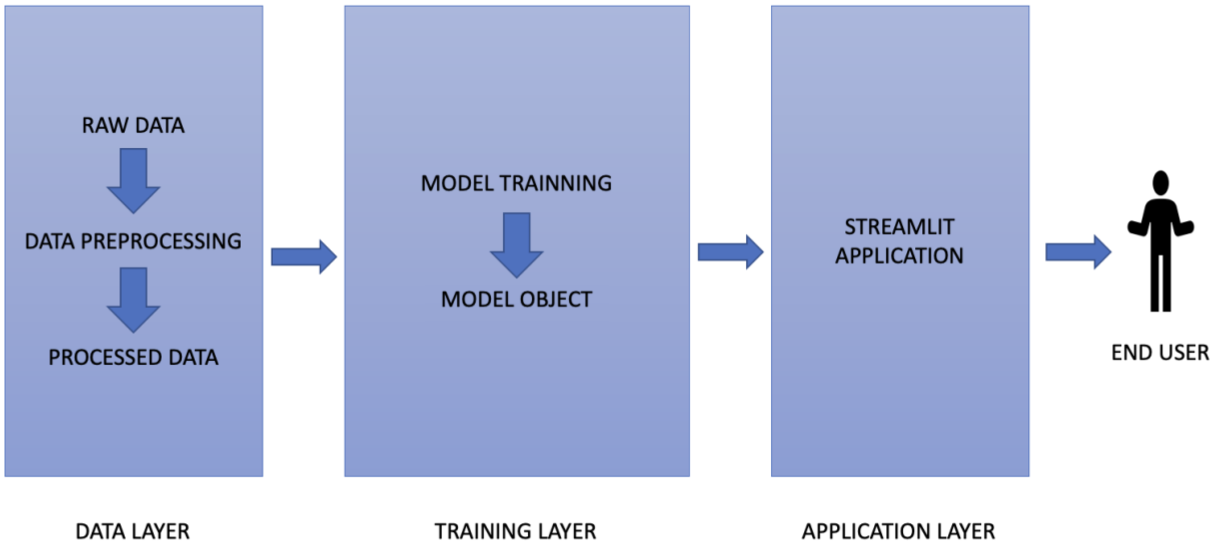
- **Language:** Python
- **Packages:** NLTK, Scikit-Learn, Pandas, Numpy, Streamlit
- **Cloud:** Azure App Services, Azure Data Factory, Azure Blob Storage, Azure Databricks, Azure Web Apps
- **Code Management**: Git, Github, Docker

Data Description

This file contains all relevant publications, datasets and clinical trials from Dimensions that are related to COVID-19. The content has been exported from Dimensions using a query in the openly accessible Dimensions application, which you can access at https://covid-19.dimensions.ai/.

Dimensions Resources. (2020). *Dimensions COVID-19 publications, datasets, and clinical trials* [Data set]. Dimensions. https://doi.org/10.6084/m9.figshare.11961063.v42

Project Flowchart



Data Layer
The data layer is the primary layer where we will perform all the basic data preprocessing steps on the data that we will obtain from the Azure Blob storage and again store the cleaned data to Azure Blob storage as a different file

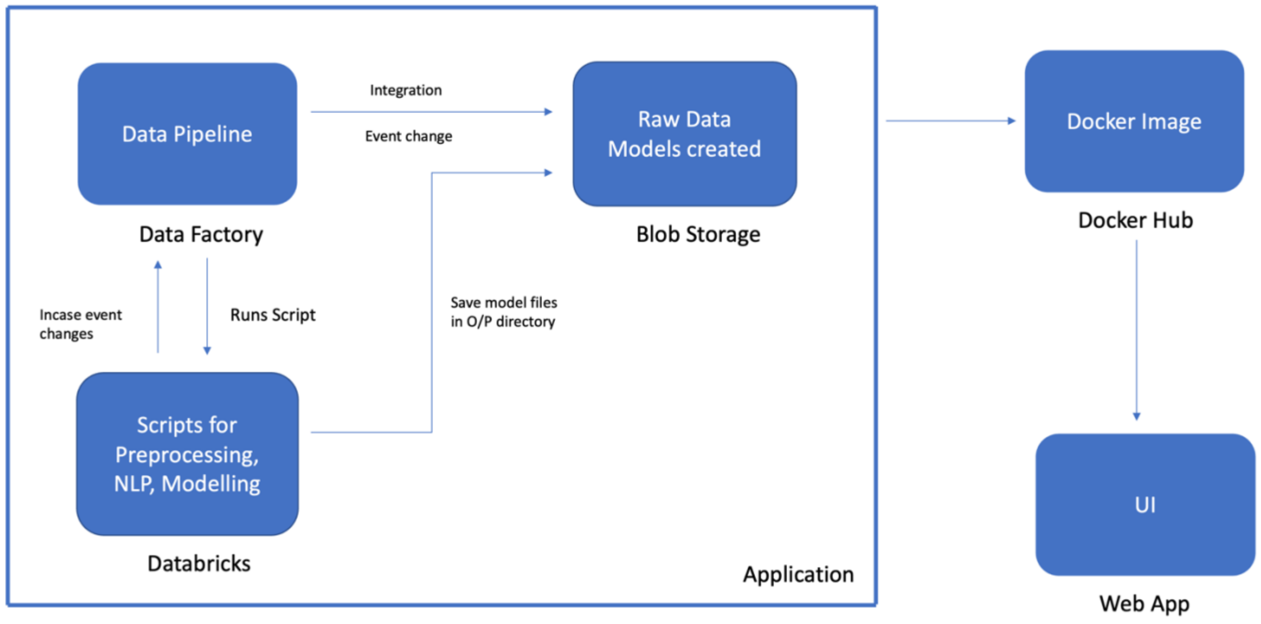
Training Layer

The training layer is the intermediary layer which involves the crucial step of model building. The model will be trained using Azure Databricks and the generated model object will be stored back to Azure Blob Storage. A trigger will also be set to the model training process to train the model periodically with new data obtained.

Application Layer

This layer involves building a user interface using Streamlit to the model built so that the users can interact with our model object. The user application is then deployed to the cloud using Azure App Services.

Cloud Services Architecture



- Firstly, we have created “**IST615Project**” as a resource group in Azure. In that resource group, we have invoked “**Azure Blob Storage**” service that stores our data.
- “**Azure Databricks**” service is used as a resource where all the Python scripts are mounted and run.

- One of the python scripts, “*authenticate.py*” contains the code that connects and integrates the Azure Blob Storage and Azure Databricks service.
- “**Azure Data Factory**” resources is leveraged to generate a “*training*” pipeline that trains the model.
- The trained model is then stored in Azure Blob Storage.
- We have integrated all our python scripts (stored in Databricks) with our Git Repository.
- This integration with Git helps in verison controlling of our code.
- Then, we created a “**Docker Image**” of the entire application which is then published into “**Docker Hub**”.
- Lastly, “**Web App**” service is created that runs the docker file and displays the User Interface (UI) of our project.

1. Azure Blob Storage

About:

Azure Blob Storage helps you create data lakes for your analytics needs and provides storage to build powerful cloud-native and mobile apps. Whether it is images, audio, video, logs, configuration files, or sensor data from an IoT array, data needs to be stored in a way that can be easily accessible for analysis purposes, and Azure Storage provides options for each one of these possible use cases. Blob Storage is Microsoft Azure’s service for storing binary large objects or blobs which are typically composed of unstructured data such as text, images, and videos, along with their metadata. Blobs are stored in directory-like structures called “containers.”

The blob service includes:

- Blobs, which are the data objects of any type
- Containers, which wrap multiple blobs together
- Azure storage account, which contains all your Azure storage data objects

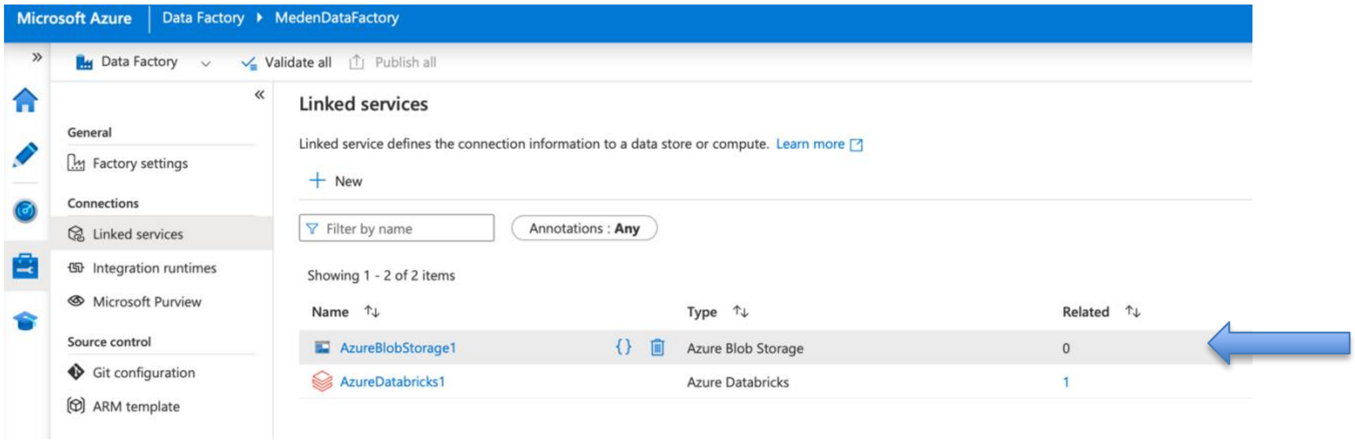
Much of what data consumers do with storage is focused on dealing with unstructured data such as logs, files, images, videos, etc. Using Azure's blob storage is a way to overcome the challenge of having to deploy different database systems for different types of data. Blob storage provides users with strong data consistency, storage and access flexibility that adapts to the user's needs, and it also provides high availability by implementing geo-replication.

Integration with other services:

- Azure Blob Storage Integration with Azure Databricks

```
# one time execution only
dbutils.fs.mount(
    source="wasbs://meden@meden.blob.core.windows.net",
    mount_point = "/mnt/data",
    extra_configs =
{"fs.azure.account.key.meden.blob.core.windows.net":'hfWeCi5qS
)
```

- Azure Blob Storage Integration with Azure Data Factory



Linked services

Linked service defines the connection information to a data store or compute. Learn more

Showing 1 - 2 of 2 items

Name	Type	Related
AzureBlobStorage1	Azure Blob Storage	0
AzureDatabricks1	Azure Databricks	1

2. Azure Databricks

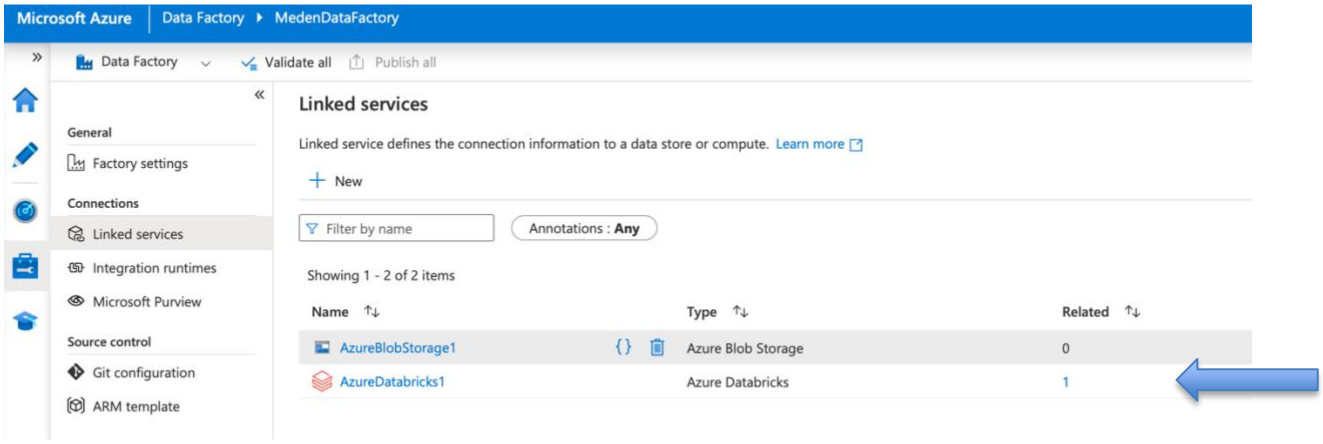
About:
The Azure Databricks Lakehouse Platform provides a unified set of tools for building, deploying, sharing, and maintaining enterprise-grade data solutions at scale. Azure Databricks integrates with cloud storage and security in your cloud account and manages and deploys cloud infrastructure on your behalf. The Azure Databricks platform architecture is composed of two primary parts: the infrastructure used by Azure Databricks to deploy, configure, and manage the platform and services, and the customer-owned infrastructure managed in collaboration by Azure Databricks and your company. Unlike many enterprises database companies, Azure Databricks does not force you to migrate your data into proprietary storage systems to use the platform. Instead, you configure a Azure Databricks workspace by configuring secure integrations between the Azure Databricks platform and your cloud account, and then Azure Databricks deploys ephemeral compute clusters using cloud resources in your account to process and store data in object storage and other integrated services you control.

The most common use cases of Azure Databricks are

- Building an enterprise data Lakehouse
- ETL & Data Engineering
- Machine Learning, AI & Data Science
- Data Warehousing, Analytics, & BI
- Data Governance & Security Data Sharing
- DevOps, CI/CD
- Real-time Streaming Analytics

Integration with other services:

- Azure Databricks Integration with Azure Data Factory



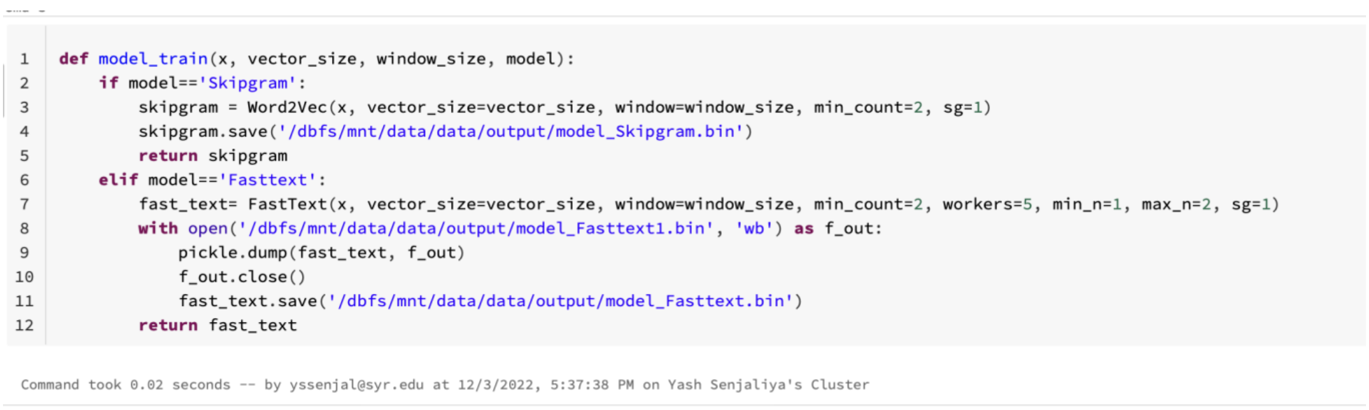
Linked services

Linked service defines the connection information to a data store or compute. Learn more

Showing 1 - 2 of 2 items

Name	Type	Related
AzureBlobStorage1	Azure Blob Storage	0
AzureDatabricks1	Azure Databricks	1

- Azure Databricks Integration with Azure Blob Storage



```
def model_train(x, vector_size, window_size, model):
    if model=='Skipgram':
        skipgram = Word2Vec(x, vector_size=vector_size, window=window_size, min_count=2, sg=1)
        skipgram.save('/dbfs/mnt/data/data/output/model_Skipgram.bin')
        return skipgram
    elif model=='Fasttext':
        fast_text= FastText(x, vector_size=vector_size, window=window_size, min_count=2, workers=5, min_n=1, max_n=2, sg=1)
        with open('/dbfs/mnt/data/data/output/model_Fasttext1.bin', 'wb') as f_out:
            pickle.dump(fast_text, f_out)
            f_out.close()
            fast_text.save('/dbfs/mnt/data/data/output/model_Fasttext.bin')
        return fast_text
```

```
Command took 0.02 seconds -- by yssenjal@syr.edu at 12/3/2022, 5:37:38 PM on Yash Senjaliya's Cluster
```

The above script demonstrates the trained model is stored in Azure Blob Storage

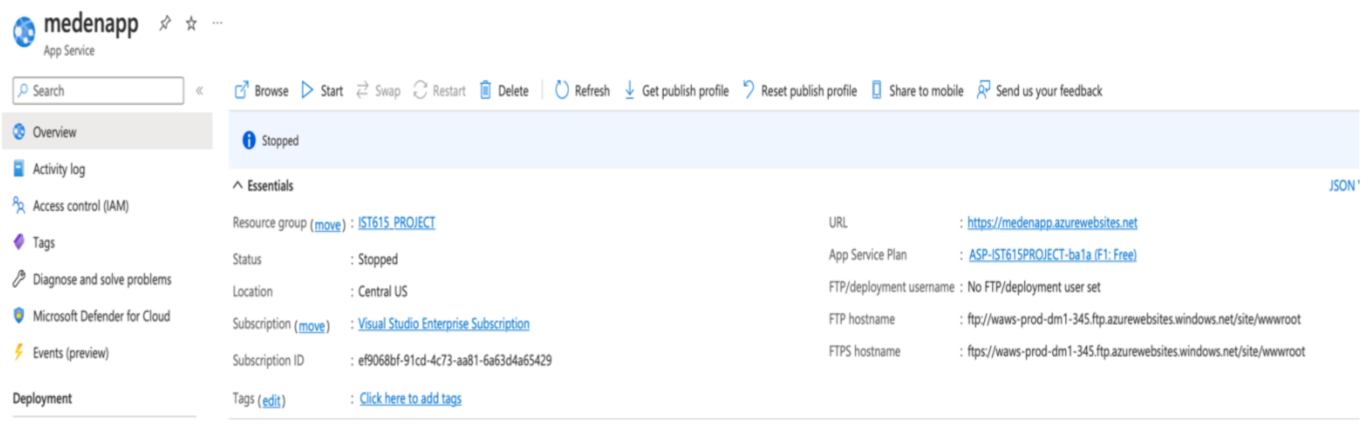
3. Azure Data Factory

About:
Big data requires a service that can orchestrate and operationalize processes to refine enormous stores of raw data into actionable business insights. Azure Data Factory is a managed cloud service that's built for these complex hybrid extract-transform-load (ETL), extract-load-transform (ELT), and data integration projects. It offers a code-free UI for intuitive authoring and single-pane-of-glass monitoring and management. We can lift and shift existing SSIS packages to Azure and run them with full compatibility in Data Factory.
Azure Data Factory can be used for:

- Supporting data migrations
- Getting data from a client's server or online data to an Azure Data Lake
- Carrying out various data integration processes
- Integrating data from different ERP systems and loading it into Azure Synapse for reporting.

4. Azure Web Apps

Web App service is created that runs the docker file and displays the User Interface (UI) of our project.



Stopped

Essentials

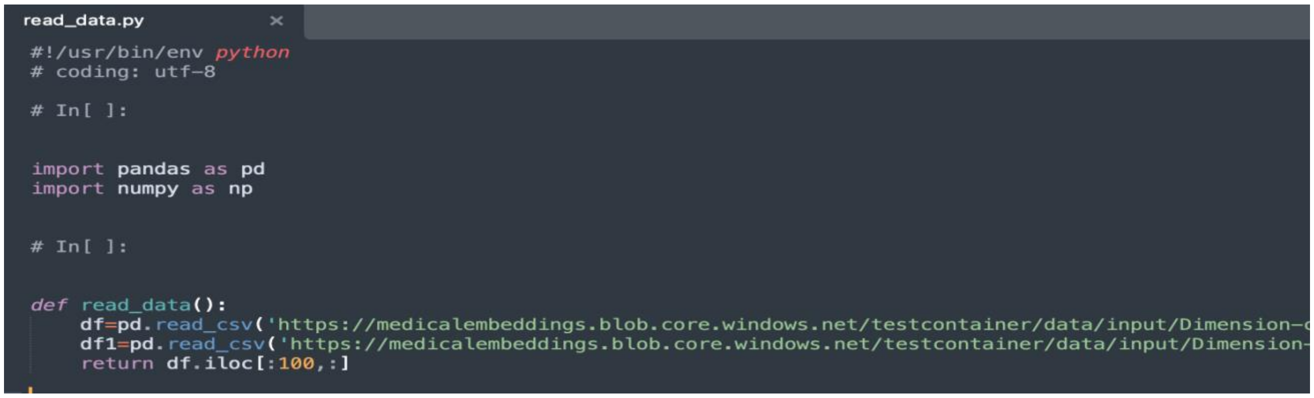
Resource group (move)	: IST615_PROJECT	URL	: https://medenapp.azurewebsites.net
Status	: Stopped	App Service Plan	: ASP-IST615PROJECT-ba1a (F1: Free)
Location	: Central US	FTP/deployment username	: No FTP/deployment user set
Subscription (move)	: Visual Studio Enterprise Subscription	FTP hostname	: ftp://waws-prod-dm1-345.ftp.azurewebsites.windows.net/site/wwwroot
Subscription ID	: ef9068bf-91cd-4c73-aa81-6a63d4a65429	FTPS hostname	: ftps://waws-prod-dm1-345.ftp.azurewebsites.windows.net/site/wwwroot
Tags (edit)	: Click here to add tags		

Natural Language Processing Scripts

We have seven different Python scripts that are stored in Azure Databricks.

1. ***read_data.py***

   This script reads the data which is stored in Azure Blob Storage.

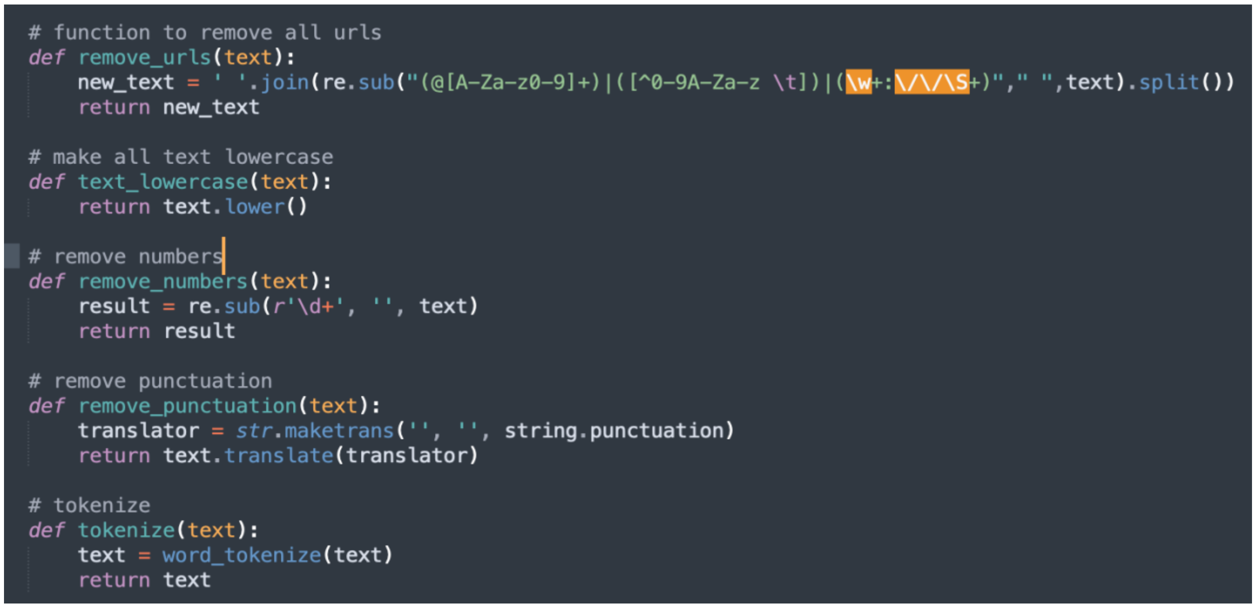


read_data.py

```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:


import pandas as pd
import numpy as np


# In[ ]:


def read_data():
    df=pd.read_csv('https://medicalembeddings.blob.core.windows.net/testcontainer/data/input/Dimension-c
    df1=pd.read_csv('https://medicalembeddings.blob.core.windows.net/testcontainer/data/input/Dimension-
    return df.iloc[:100,:]
```

2. ***preprocessing.py***

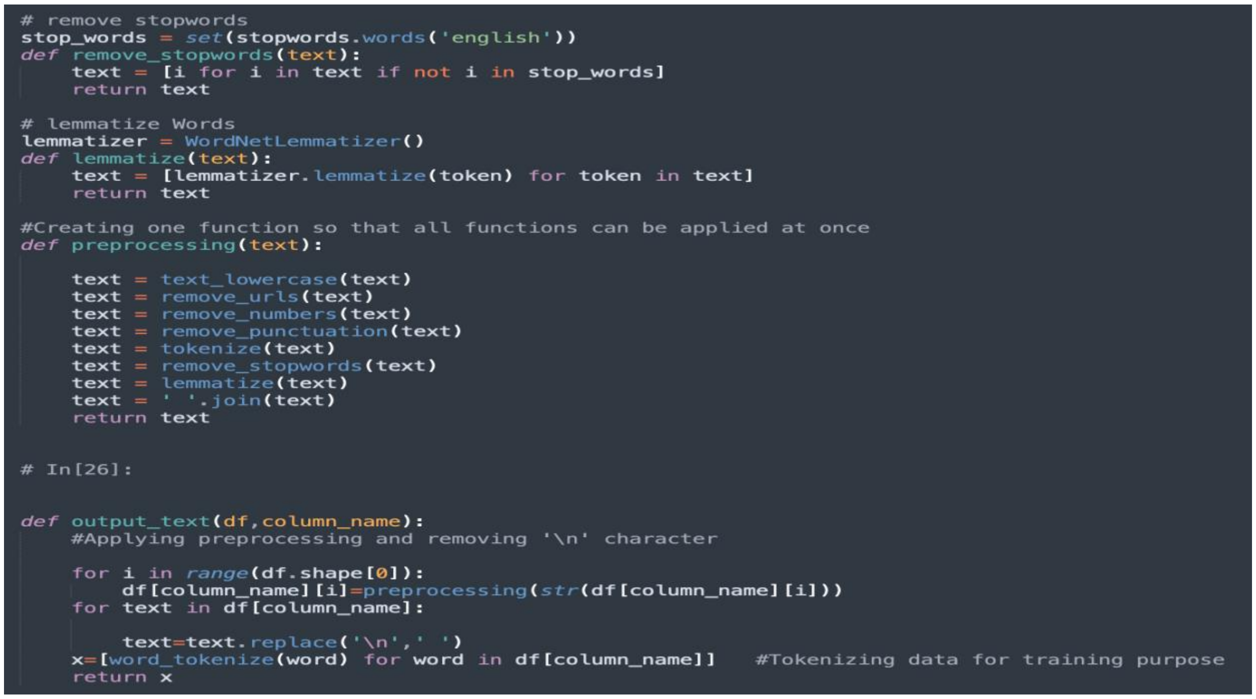
   Different functions are used to pre-process the data that includes removing the URLs, converting text to lowercase, removing punctuation, and tokenizing the words.



```
# function to remove all urls
def remove_urls(text):
    new_text = ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)"," ",text).split())
    return new_text

# make all text lowercase
def text_lowercase(text):
    return text.lower()

# remove numbers
def remove_numbers(text):
    result = re.sub(r'\d+', '', text)
    return result

# remove punctuation
def remove_punctuation(text):
    translator = str.maketrans('', '', string.punctuation)
    return text.translate(translator)

# tokenize
def tokenize(text):
    text = word_tokenize(text)
    return text
```

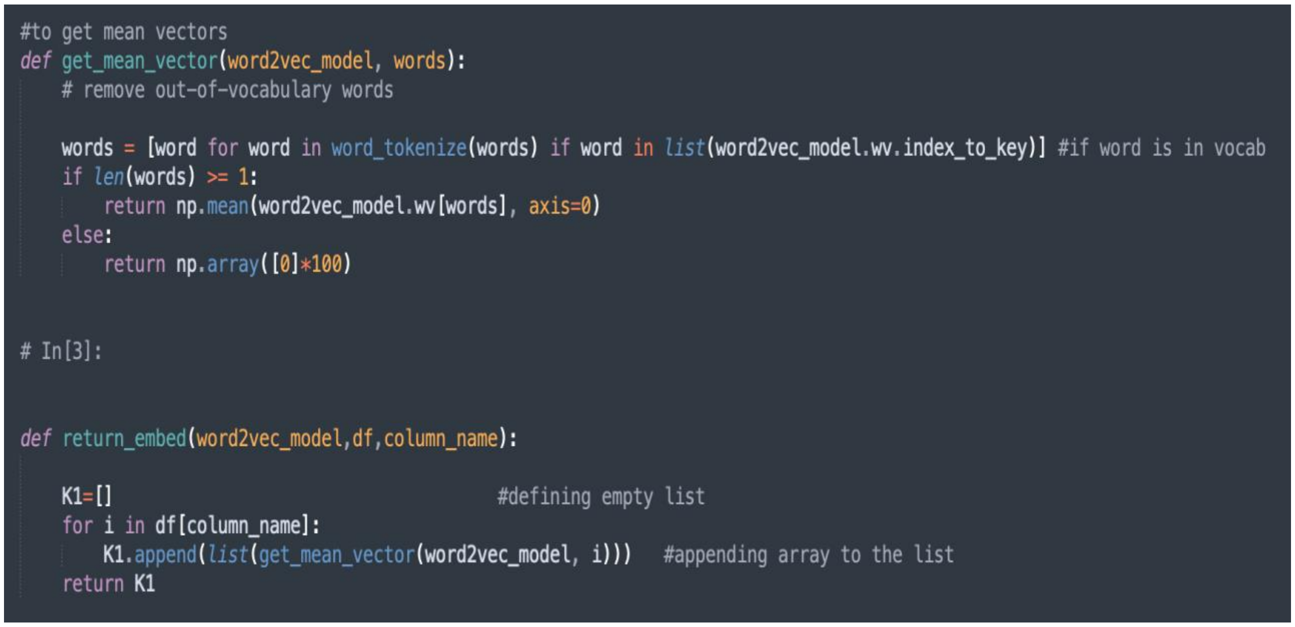
Once the data is tokenized, stop words such as “is”, “that” are removed using and finally lemmatized using functions.

Lastly, all the pre-processing functions are called together, and the output is displayed.



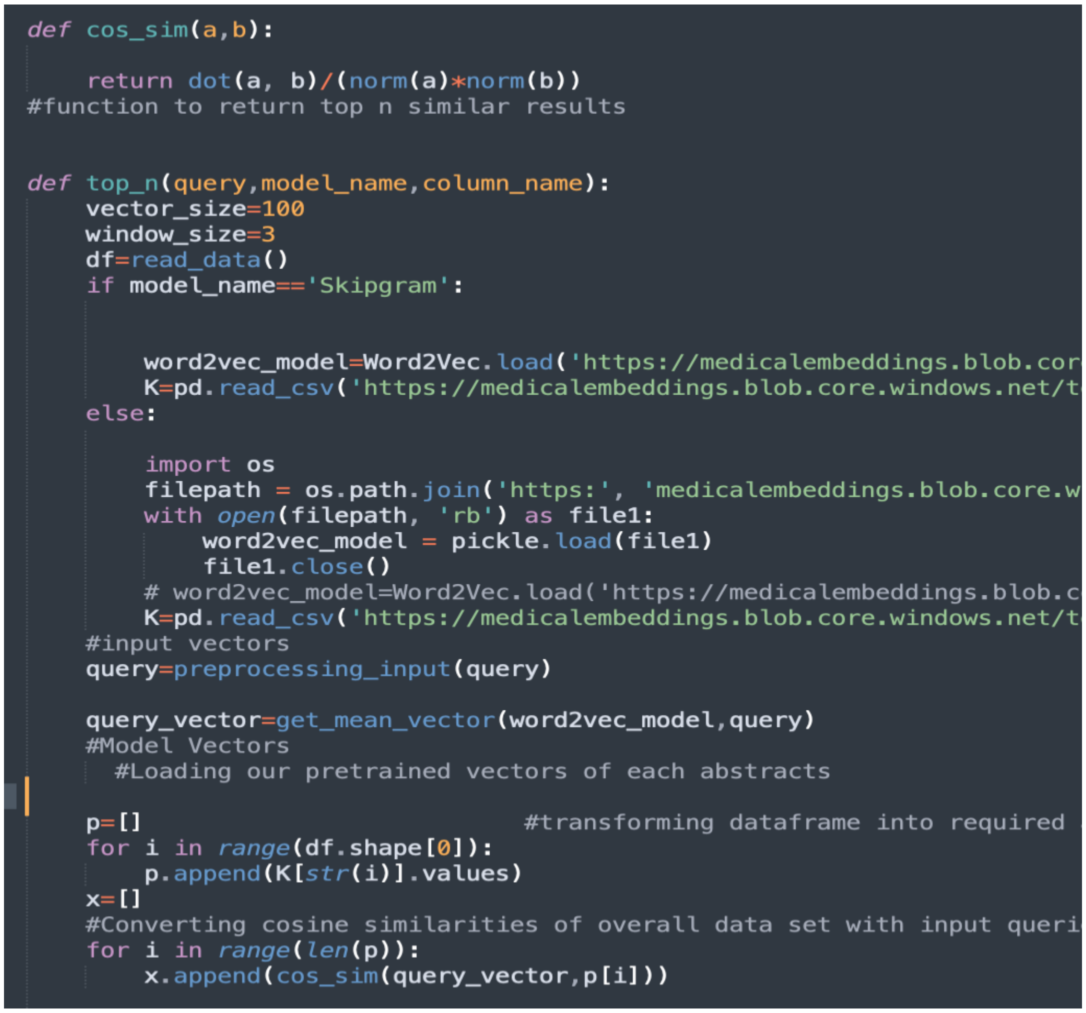
```python
# remove stopwords
stop_words = set(stopwords.words('english'))
def remove_stopwords(text):
    text = [i for i in text if not i in stop_words]
    return text

# lemmatize Words
lemmatizer = WordNetLemmatizer()
def lemmatize(text):
    text = [lemmatizer.lemmatize(token) for token in text]
    return text

#Creating one function so that all functions can be applied at once
def preprocessing(text):

    text = text_lowercase(text)
    text = remove_urls(text)
    text = remove_numbers(text)
    text = remove_punctuation(text)
    text = tokenize(text)
    text = remove_stopwords(text)
    text = lemmatize(text)
    text = ' '.join(text)
    return text


# In[26]:


def output_text(df,column_name):
    #Applying preprocessing and removing '\n' character

    for i in range(df.shape[0]):
        df[column_name][i]=preprocessing(str(df[column_name][i]))
    for text in df[column_name]:

        text=text.replace('\n',' ')
    x=[word_tokenize(word) for word in df[column_name]]   #Tokenizing data for training purpose
    return x
```

3. ***return_embed.py***
   In this script, we have leveraged the “Word2Vec Model” that converts the tokenized words into vector format. Lastly, the list of tokenized words in vectored format are returned.



```
#to get mean vectors
def get_mean_vector(word2vec_model, words):
    # remove out-of-vocabulary words

    words = [word for word in word_tokenize(words) if word in list(word2vec_model.wv.index_to_key)] #if word is in vocab
    if len(words) >= 1:
        return np.mean(word2vec_model.wv[words], axis=0)
    else:
        return np.array([0]*100)


# In[3]:


def return_embed(word2vec_model,df,column_name):

    K1=[]                                   #defining empty list
    for i in df[column_name]:
        K1.append(list(get_mean_vector(word2vec_model, i)))   #appending array to the list
    return K1
```

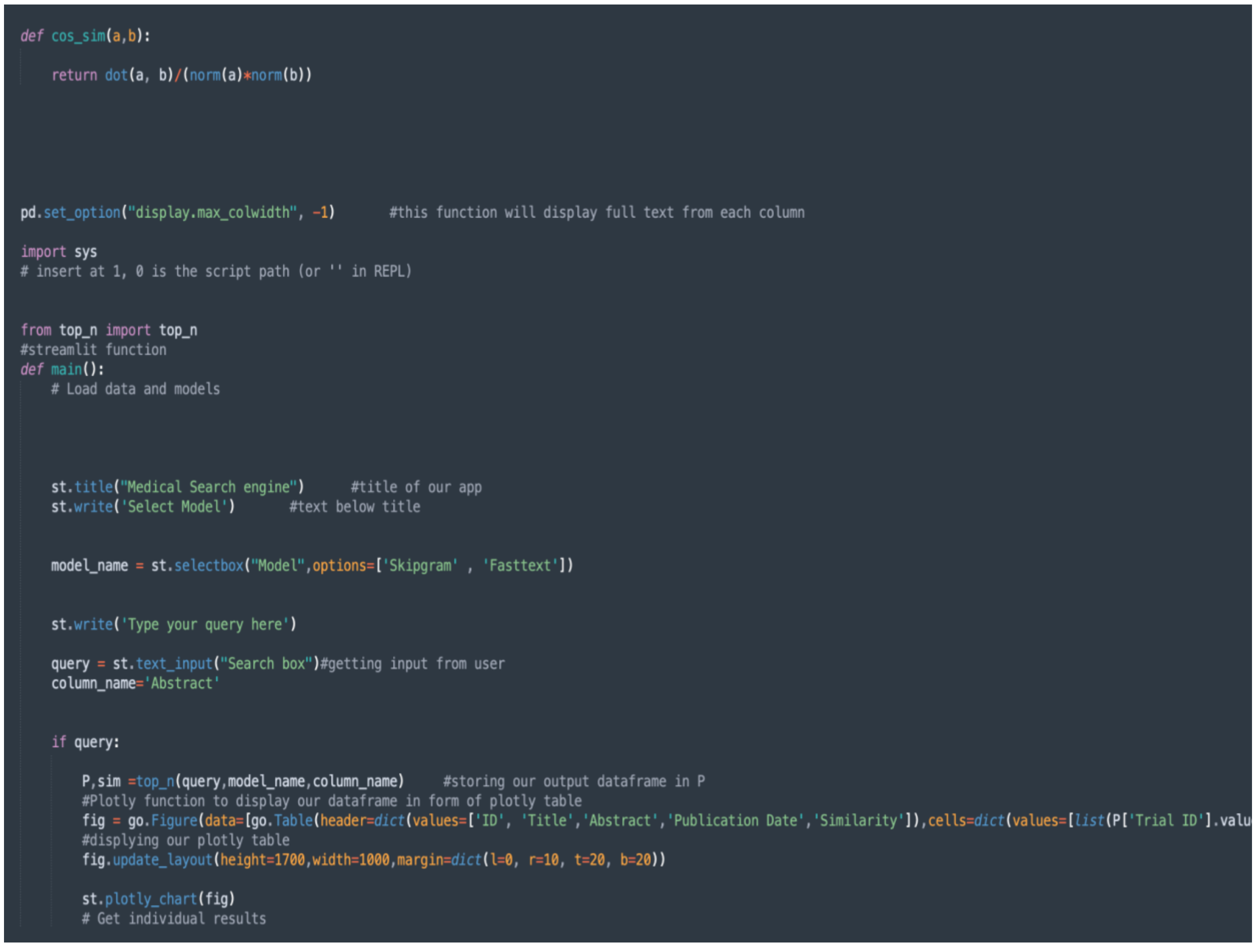
4. ***top_n.py***
   We have used cosine similarity that is calculates/finds the nearest word to the query entered by the user. Next, top_n function is used that returns the top hundred (100) results related to the query entered.



```
def cos_sim(a,b):

    return dot(a, b)/(norm(a)*norm(b))
#function to return top n similar results


def top_n(query,model_name,column_name):
    vector_size=100
    window_size=3
    df=read_data()
    if model_name=='Skipgram':


        word2vec_model=Word2Vec.load('https://medicalembeddings.blob.cor
        K=pd.read_csv('https://medicalembeddings.blob.core.windows.net/t
    else:

        import os
        filepath = os.path.join('https:', 'medicalembeddings.blob.core.w
        with open(filepath, 'rb') as file1:
            word2vec_model = pickle.load(file1)
            file1.close()
        # word2vec_model=Word2Vec.load('https://medicalembeddings.blob.c
        K=pd.read_csv('https://medicalembeddings.blob.core.windows.net/t
    #input vectors
    query=preprocessing_input(query)

    query_vector=get_mean_vector(word2vec_model,query)
    #Model Vectors
      #Loading our pretrained vectors of each abstracts

    p=[]                        #transforming dataframe into required
    for i in range(df.shape[0]):
        p.append(K[str(i)].values)
    x=[]
    #Converting cosine similarities of overall data set with input queri
    for i in range(len(p)):
        x.append(cos_sim(query_vector,p[i]))
```

5. *app.py*

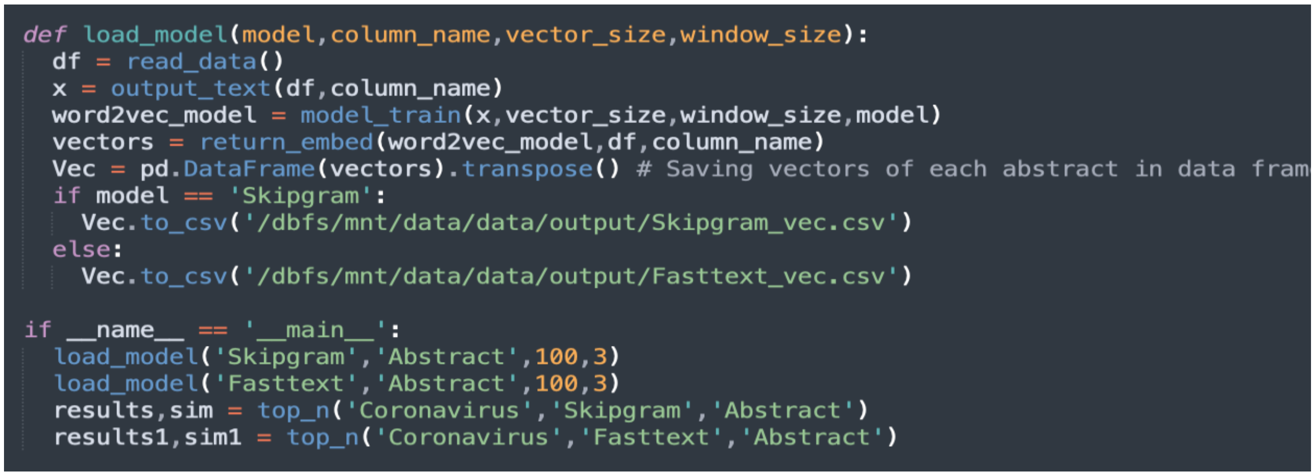
This script is used to display the actual User Interface (UI) of the project.



```
def cos_sim(a,b):
    return dot(a, b)/(norm(a)*norm(b))




pd.set_option("display.max_colwidth", -1)       #this function will display full text from each column

import sys
# insert at 1, 0 is the script path (or '' in REPL)


from top_n import top_n
#streamlit function
def main():
    # Load data and models




    st.title("Medical Search engine")      #title of our app
    st.write('Select Model')       #text below title


    model_name = st.selectbox("Model",options=['Skipgram' , 'Fasttext'])


    st.write('Type your query here')

    query = st.text_input("Search box")#getting input from user
    column_name='Abstract'


    if query:

        P,sim =top_n(query,model_name,column_name)     #storing our output dataframe in P
        #Plotly function to display our dataframe in form of plotly table
        fig = go.Figure(data=[go.Table(header=dict(values=['ID', 'Title','Abstract','Publication Date','Similarity']),cells=dict(values=[list(P['Trial ID'].valu
        #displying our plotly table
        fig.update_layout(height=1700,width=1000,margin=dict(l=0, r=10, t=20, b=20))

        st.plotly_chart(fig)
        # Get individual results
```

6. *main.py*

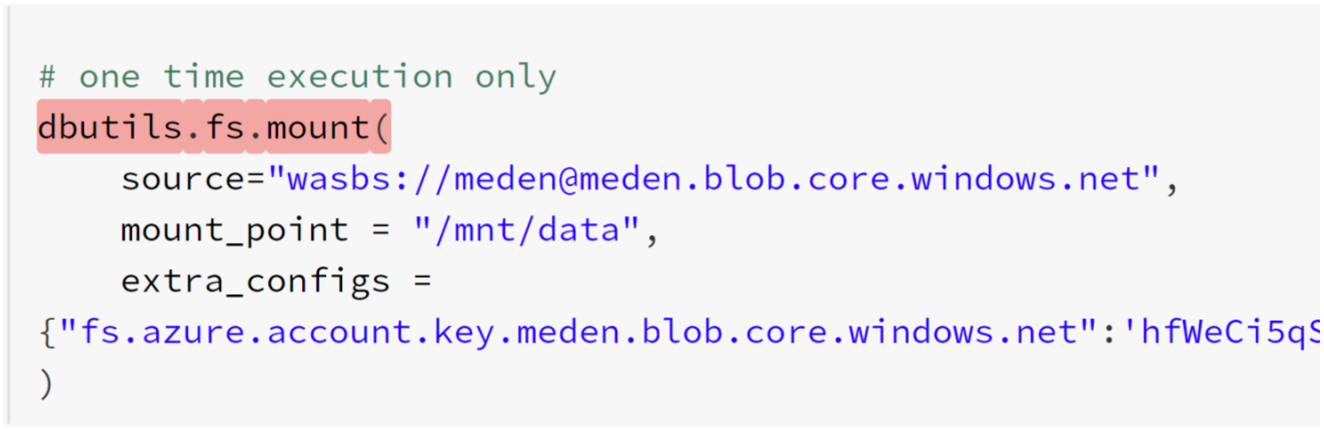
This is the master script that calls the rest of the scripts.



```
def load_model(model,column_name,vector_size,window_size):
    df = read_data()
    x = output_text(df,column_name)
    word2vec_model = model_train(x,vector_size,window_size,model)
    vectors = return_embed(word2vec_model,df,column_name)
    Vec = pd.DataFrame(vectors).transpose() # Saving vectors of each abstract in data fram
    if model == 'Skipgram':
      Vec.to_csv('/dbfs/mnt/data/data/output/Skipgram_vec.csv')
    else:
      Vec.to_csv('/dbfs/mnt/data/data/output/Fasttext_vec.csv')

if __name__ == '__main__':
    load_model('Skipgram','Abstract',100,3)
    load_model('Fasttext','Abstract',100,3)
    results,sim = top_n('Coronavirus','Skipgram','Abstract')
    results1,sim1 = top_n('Coronavirus','Fasttext','Abstract')
```

7. ***authentication.py***

   This script is used to create a mount point for the storage that reads the data stored in Azure Blob Storage and to connect to Azure Databricks.

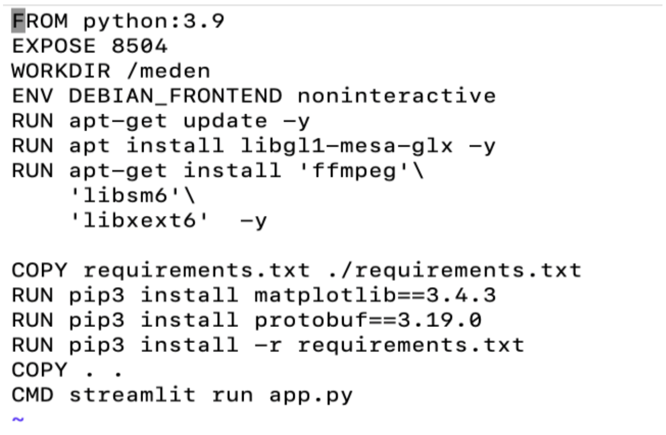


```
# one time execution only
dbutils.fs.mount(
    source="wasbs://meden@meden.blob.core.windows.net",
    mount_point = "/mnt/data",
    extra_configs =
{"fs.azure.account.key.meden.blob.core.windows.net":'hfWeCi5q
)
```

Git Integration / Docker

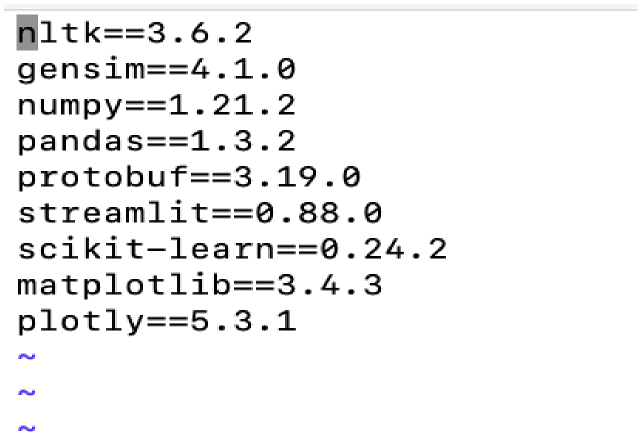
Git repo link: https://github.com/trishh088/Meden

Docker File



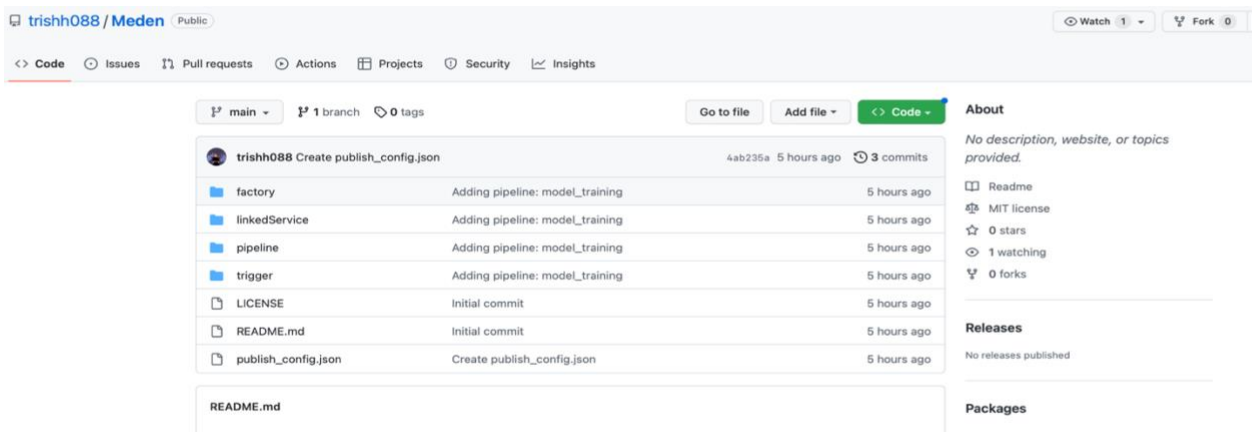
```
FROM python:3.9
EXPOSE 8504
WORKDIR /meden
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update -y
RUN apt install libgl1-mesa-glx -y
RUN apt-get install 'ffmpeg'\
    'libsm6'\
    'libxext6'  -y

COPY requirements.txt ./requirements.txt
RUN pip3 install matplotlib==3.4.3
RUN pip3 install protobuf==3.19.0
RUN pip3 install -r requirements.txt
COPY . .
CMD streamlit run app.py
~
```

Requirements



```
nltk==3.6.2
gensim==4.1.0
numpy==1.21.2
pandas==1.3.2
protobuf==3.19.0
streamlit==0.88.0
scikit-learn==0.24.2
matplotlib==3.4.3
plotly==5.3.1
~
~
~
```

The above script of docker image file installs the necessary requirements used for running the application.



trishh088 Create publish_config.json		4ab235a 5 hours ago 3 commits
factory	Adding pipeline: model_training	5 hours ago
linkedService	Adding pipeline: model_training	5 hours ago
pipeline	Adding pipeline: model_training	5 hours ago
trigger	Adding pipeline: model_training	5 hours ago
LICENSE	Initial commit	5 hours ago
README.md	Initial commit	5 hours ago
publish_config.json	Create publish_config.json	5 hours ago

README.md

About

No description, website, or topics provided.

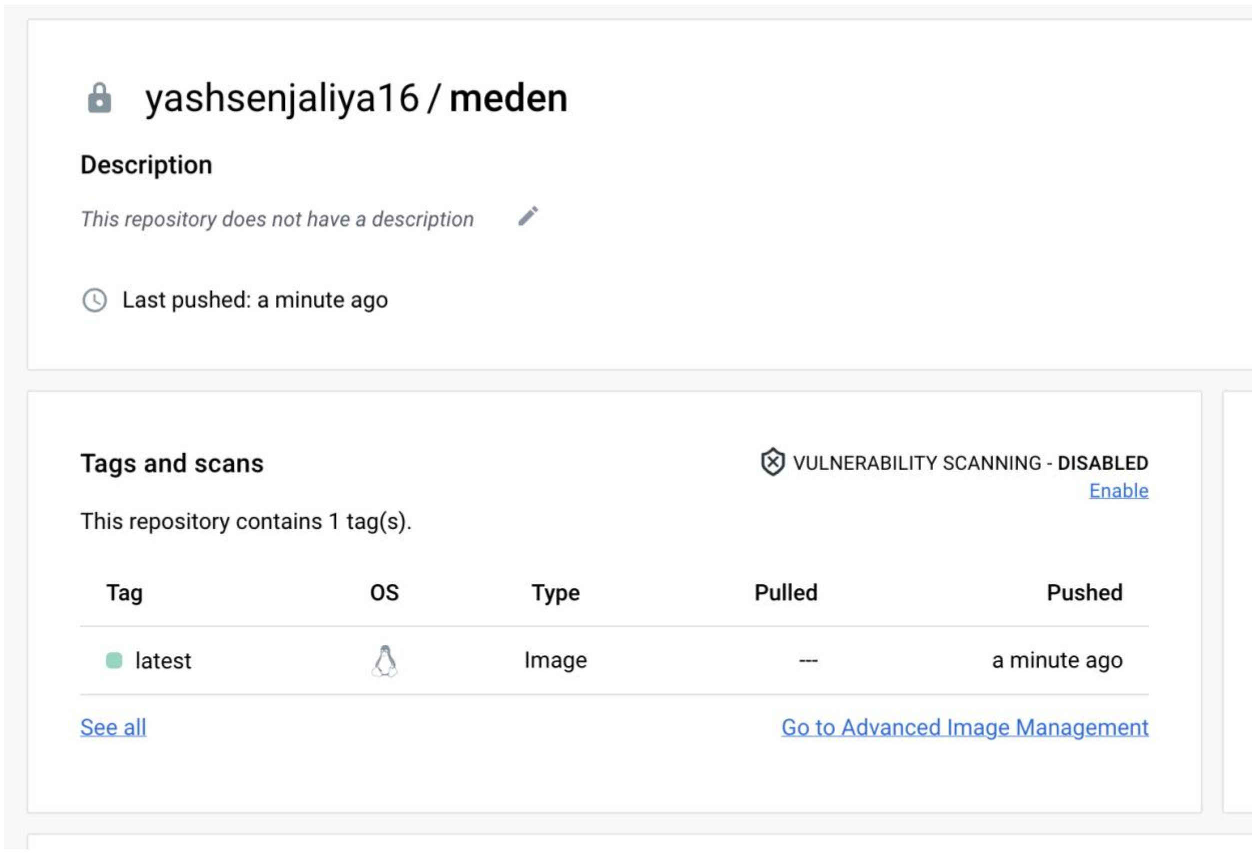
Readme
MIT license
0 stars
1 watching
0 forks

Releases

No releases published

Packages

The above image shows the GitHub Repository for our application



yashsenjaliya16 / **meden**

Description

This repository does not have a description

Last pushed: a minute ago

Tags and scans

VULNERABILITY SCANNING - **DISABLED**
Enable

This repository contains 1 tag(s).

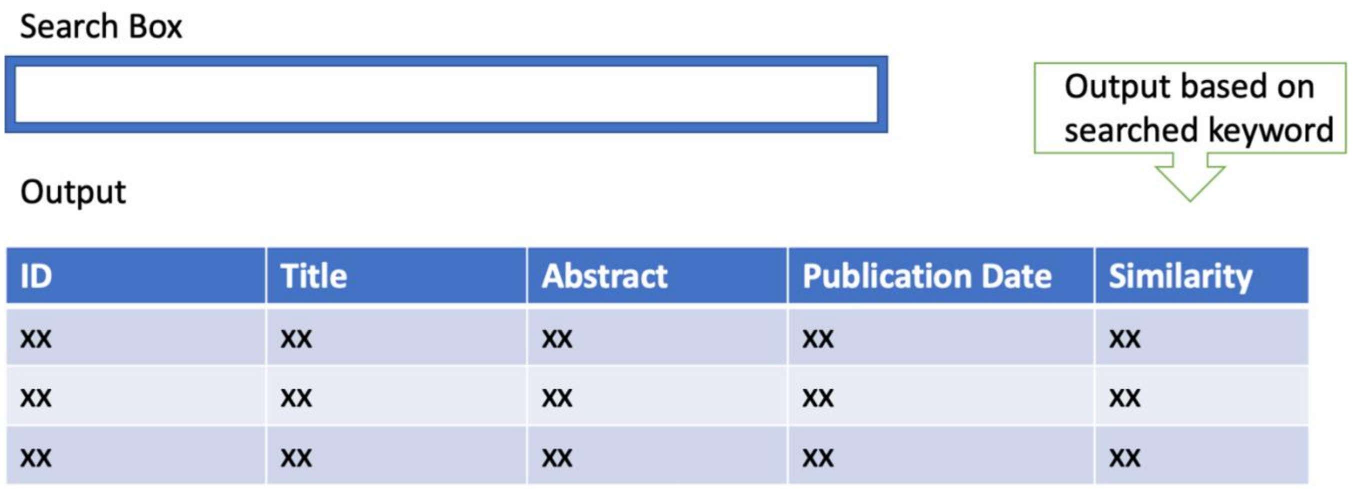
Tag	OS	Type	Pulled	Pushed
latest		Image	---	a minute ago

See all

Go to Advanced Image Management

Application User Interface (UI)

The application will display the related research papers closest associated with the searched keyword. Search results will be in the form of a list.



ID	Title	Abstract	Publication Date	Similarity
xx	xx	xx	xx	xx
xx	xx	xx	xx	xx
xx	xx	xx	xx	xx

- In the “Search” box, the user is expected to input a query or a keyword and hit “Enter”.
- The application starts running and will display the Top 100 search results related to that particular query or a keyword.

Issues Encountered

- As far as the utmost requirements of this project were concerned, integration of all possible Cloud Services altogether was a daunting task for obvious reasons. Particularly, in our case, we faced issues connecting and integrating Azure Blob Storage and Azure Databricks service. We were unable to mount and authenticate the python scripts that were stored in Azure Databricks service.

- Essentially one of the major parts of our project was Containerization. After the application was built, it was difficult to understand to push the entire application with cloud services integrated to each other, to Docker Hub.

- As we came building towards the end of our project, most of our time was utilized in solving UI deploying issue. Our application was crashed suddenly as the package "streamlit" responsible for displaying the UI was not running.

- As a part of our effort towards completing this project, we mounted all NLP python scripts on Azure Databricks service. One of the python scripts, "main.py" which is the master script that calls the rest of the scripts was unable to function. This again acted as a major roadblock in our project's progress.

Future Scope

- Considering the inner complexity and dependencies in our project, there is an opportunity to enhance and upgrade the look and feel of the User Interface (UI) that would contribute to building the entire project more user friendly.

- Similarly, a live interactive dashboard could be built using Microsoft’s PowerBI that would integrate and connect very efficiently with other Azure’s services. So along with UI enhancements, a user will have an ability to interact with the system.

- To improve the overall efficiency of the project, the NLP python scripts can be optimized by working on reducing the time complexity thus tremendously bolstering the overall accuracy of the model.

References

Microsoft. (n.d.). *Azure Data Factory Documentation - Azure Data Factory*. Microsoft Learn. https://learn.microsoft.com/en-us/azure/data-factory/

Microsoft. (n.d.-a). *Azure Blob Storage documentation*. Microsoft Learn. https://learn.microsoft.com/en-us/azure/storage/blobs/

Microsoft. (n.d.-c). *Azure Databricks documentation*. Microsoft Learn. https://learn.microsoft.com/en-us/azure/databricks/

Deepanshi. (2022, December 5). *Text preprocessing NLP: Text preprocessing in NLP with python codes*. Analytics Vidhya. Retrieved December 5, 2022, from https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/

Satarupa GuhaSatarupa Guha 1, phyroxphyrox 2, binarymaxbinarymax 2, Aviral MathurAviral Mathur 2733 bronze badges, & Jinhua WangJinhua Wang 1. (1961, April 1). *How to use word2vec to calculate the similarity distance by giving 2 words?* Stack Overflow. Retrieved December 5, 2022, from https://stackoverflow.com/questions/21979970/how-to-use-word2vec-to-calculate-the-similarity-distance-by-giving-2-words