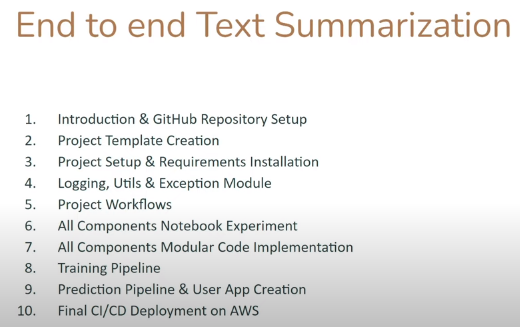
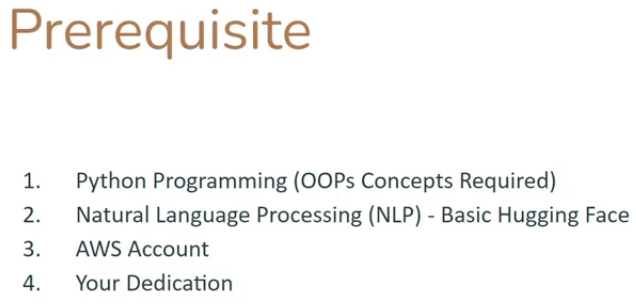
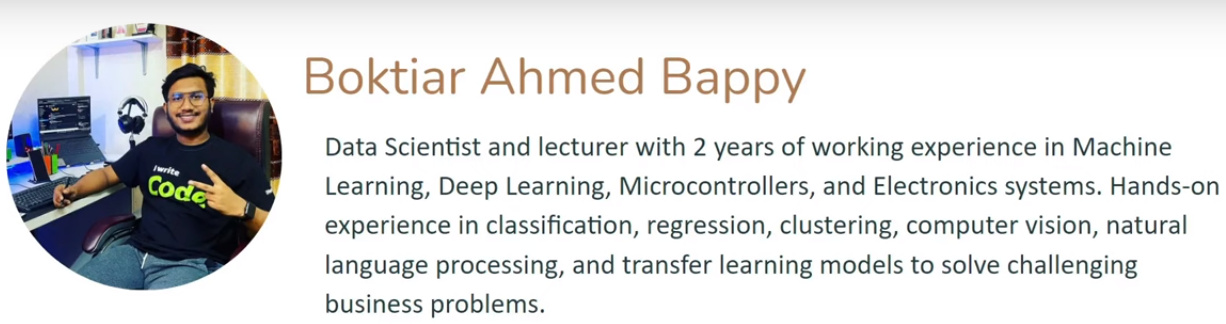
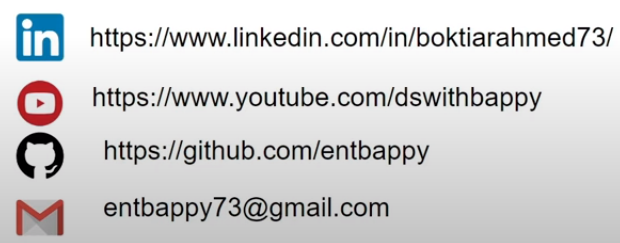
Text Summarization

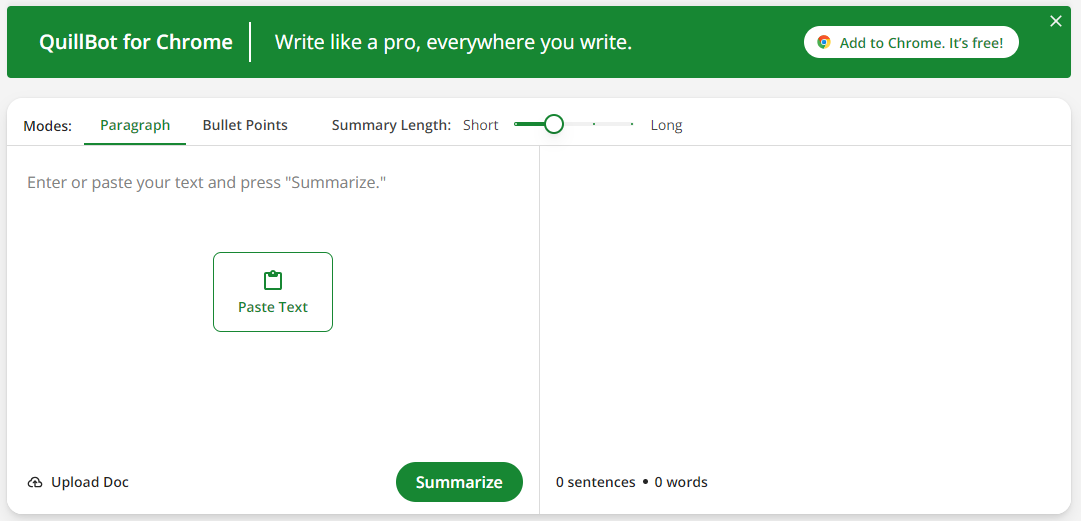


We will be using transformer based model in this project. We will be using the hugging face API. Logging, utils and exception modules are important when implementing end to end projects. It is important to perform notebook experiment of components before implementing modular code components. We will be using Python OOPS concepts. Then we will implement training pipelines. We will integrate all the training pipelines together so that when we hit a route then it starts data ingestion, data validation, data transformation, pick the model, model training, model evaluation and then in the end we will get our model. We will also implement a prediction pipeline and the user app as well. We will use the Fast API to build this simple application. In the end we will build CI/CD deployment on AWS using GitHub action.





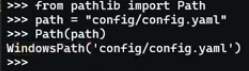
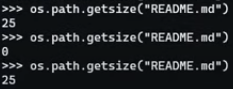




We will be doing the task from scratch. We will be using a transformer based model from hugging face which is already trained on a huge data and we will fine tune that model for our custom data.

Now we will create the project template where we will write the entire project folder structure so that we need not to create the complete structure from scratch. The logic to create the folder structure will be there in the file templete.py.

.github folder is for setting up deployment with ci/cd along with YAML files so that when we do the commit on github it will auto deploy updates to AWS.

We need to create a virtual environment first which is recommended before doing such projects.

* python -m venv textSum
* source textSum/bin/activate

Now we will write requirements.txt

transformers: This package provides state-of-the-art Natural Language Processing (NLP) models like BERT, GPT, etc., and utilities for working with them.

transformers[sentencepiece]: This extends the transformers library with support for the SentencePiece tokenizer, which is commonly used for tokenization in various NLP tasks.

datasets: This package provides access to a wide range of datasets commonly used for training and evaluating machine learning models, particularly in NLP.

sacrebleu: SacreBLEU is a popular library for computing the BLEU (Bilingual Evaluation Understudy) score, which is a metric commonly used for evaluating the quality of machine-translated text.

rouge\_score: This package provides utilities for computing the ROUGE (Recall-Oriented Understudy for Gisting Evaluation) score, which is another metric used for evaluating the quality of summaries and machine-generated text.

py7zr: Py7zr is a Python library for working with 7z archives, providing functionalities for compression and decompression.

pandas: Pandas is a powerful data manipulation and analysis library, widely used for working with structured data.

nltk: NLTK (Natural Language Toolkit) is a library for working with human language data. It provides tools for tasks like tokenization, stemming, tagging, parsing, and more.

tqdm: TQDM is a library for adding progress bars to Python code, making it easy to track the progress of iterations.

PyYAML: PyYAML is a YAML parser and emitter for Python, allowing easy reading and writing of YAML files.

matplotlib: Matplotlib is a plotting library for Python, providing a MATLAB-like interface for creating a variety of plots and visualizations.

torch: Torch is the core package of PyTorch, a popular deep learning framework. It provides data structures for multi-dimensional tensors and computational graphs for building and training neural networks.

notebook: This package typically refers to Jupyter Notebook, an interactive computing environment that allows for creating and sharing documents containing live code, equations, visualizations, and narrative text.

boto3: Boto3 is the Amazon Web Services (AWS) SDK for Python. It allows Python developers to write software that makes use of services like Amazon S3, EC2, DynamoDB, and more.

mypy-boto3-s3: This is a type stub package for mypy, providing type annotations for the Boto3 library when working with Amazon S3 specifically.

python-box==6.0.2: Python-box is a simple Python dictionary-like object wrapper allowing attribute access to nested dictionaries.

ensure==1.0.2: ensure is a set of simple assertion helpers that let you write more expressive, literate, concise, and readable Pythonic code for validating conditions. It’s inspired by should.js, expect.js, and builds on top of the unittest/JUnit assert helpers.

fastapi==0.78.0: FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python type hints.

uvicorn==0.18.3: Uvicorn is a lightning-fast ASGI server, built on top of uvloop and httptools.

Jinja2==3.1.2: Jinja2 is a modern and designer-friendly templating language for Python, used to generate dynamic content for web applications and other types of applications.

Transformers library is required for using hugging face API.

We use something called as rouge matrix for evaluating the performance of text summarization model. To do that we need sacrebleu module. The metrics compare an automatically produced summary or translation against a reference or a set of references (human-produced) summary or translation. ROUGE metrics range between 0 and 1, with higher scores indicating higher similarity between the automatically produced summary and the reference.

SacréBLEU is a standard BLEU implementation that downloads and manages WMT datasets, produces scores on detokenized outputs, and reports a string encapsulating BLEU parameters, facilitating the production of shareable, comparable BLEU scores.

Torch has a GPU version as well that we can install if we have CUDA enabled GPU.

In a requirements.txt file in Python, the -e . notation specifies an editable install of the current directory. When you include -e . in your requirements.txt file and then use pip install -r requirements.txt, it installs the package as editable, meaning that changes you make in your local code will be reflected immediately without needing to reinstall the package. This is particularly useful during development when you are actively working on a Python package or module and want to test changes without repeatedly reinstalling the package. It allows you to directly work on the codebase and see the effects in real-time without the need for a reinstall step.

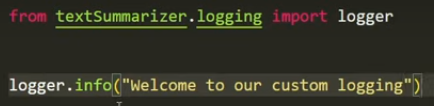
In setup.py we will write the logic for local package installation code. Once the code is there we can use pip to install requirements from requirements.txt file and also the local packages.

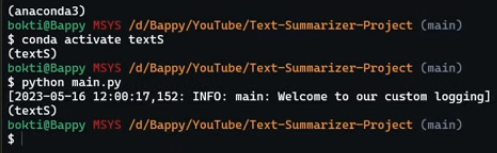
* Pip install –r requirements.txt

So, we have completed the project setup.

Now, we will work on logging, Exception & utils modules before writing the actual components.

We will start by implementing custom log in \_\_init\_\_.py file under logging on the top of logging module.





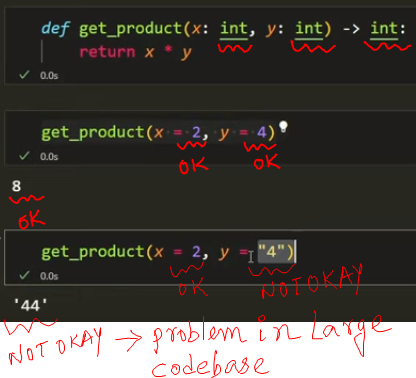




We have completed with logging. Now we will start with utils. We will not write any custom exception. We will use box exception. We will write exception in utils in common.py file.

In utils we write utility functions which are used frequently. As utils will be a module, we can import any utility function in any other file or module. Example: read and return yaml file content, create directory, return file size

ConfigBox module allows us to access dictionary values as dict.key instead of dict[“key”]



In large code base this should not happen as we have already specified type of x and y but still it is executing the code even though I am passing data of wrong datatype. The ensure\_annotation decorator notify me (throw error) about these stuff if we use this decorator in our function.

So, we have completed with logging exception and utils.

First we will try to train the model and use it in collab notebook and then we will implement the modules.

Data: <https://huggingface.co/datasets/samsum>

We have seen that how we can do the complete text summarization project in collab environment.

Now we will see the workflow. Like in this modular coding approach which file to write first and how to integrate together etc.

Workflows:

1. Update config.yaml
2. Update params.yaml
3. Update entity
4. Update the configuration manager in src config
5. update the conponents
6. update the pipeline
7. update the main.py
8. update the app.py

We need to upload the data into our github or amazon S3 etc.

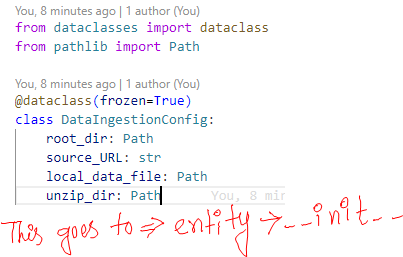
Data on Github: <https://github.com/sg13041995/Datasets/raw/main/textSummarizer_samsun.zip>

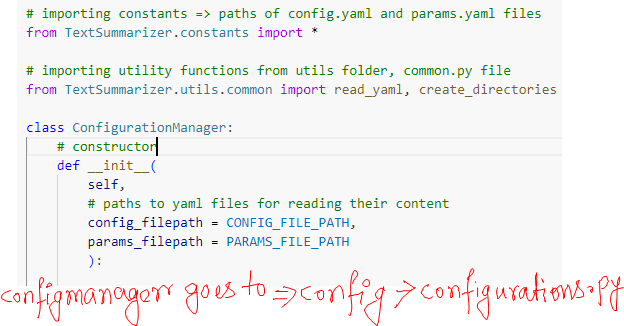
Now we will start with data ingestion component. But before that we will implement notebook experiment and once it is done we will go with component implementation.

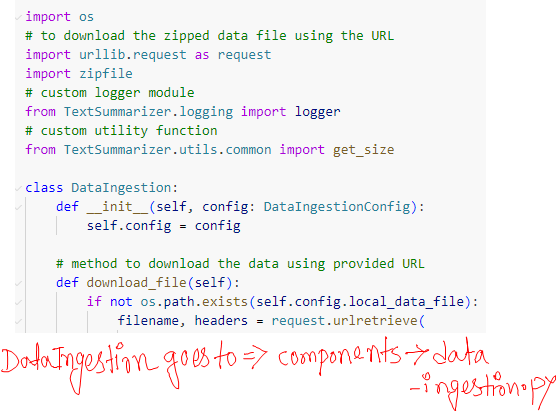
We will work on config.yaml; We will write some configuration related to atrifacts, data\_ingestion config etc. An artifact folder will be created. Artifacts persist data after a job is completed and may be used for storage of the outputs of your build process.

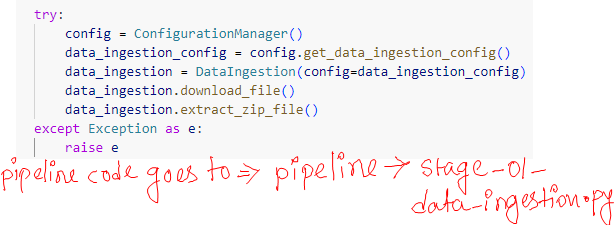
For now we will not update params.yaml because we don’t have any. We will have the params during model training.

Now we will create the entity which is nothing but the return type of a function. We can define it using dataclass. We will create the data ingestion pipeline as well but before that we will try that as a ipynb script. Once it is done we can convert that into modular code.









Finally, we need to trigger the pipeline in main.py

We have completed with data ingestion pipeline and now we will start with data validation pipeline.

Data validation means whether the proper folders like test, train, validation are available in the in the samsum\_dataset folder or not etc. We will perform the notebook experiment first and then we will start implementing the modular code.

We have kept the data validation logic very simple but we can have complex data validation logic.