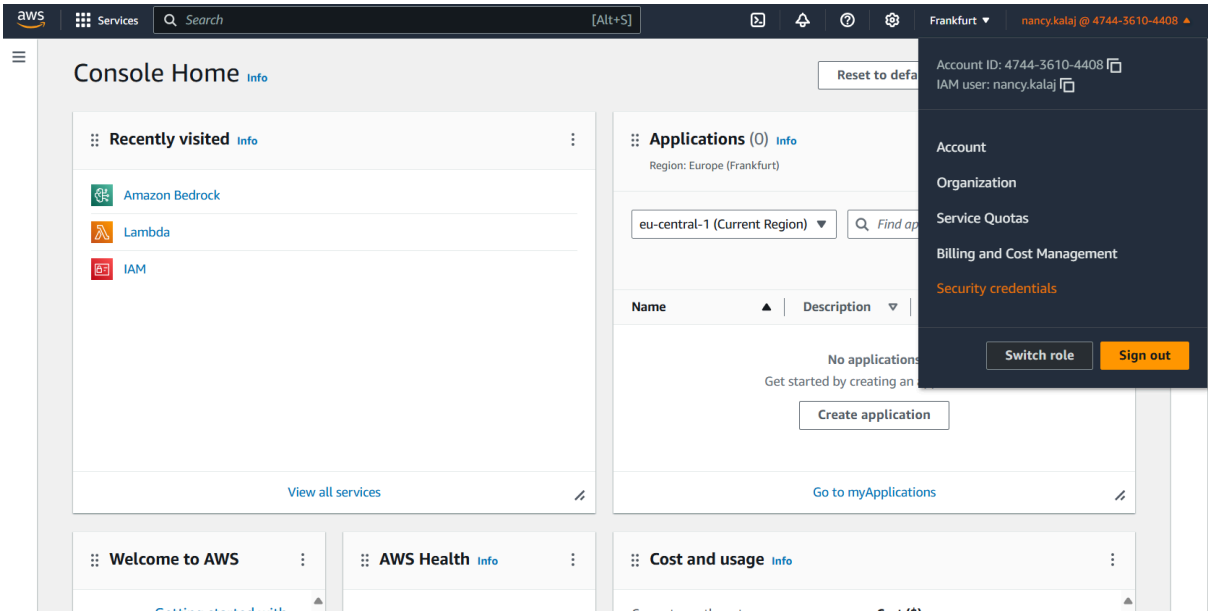
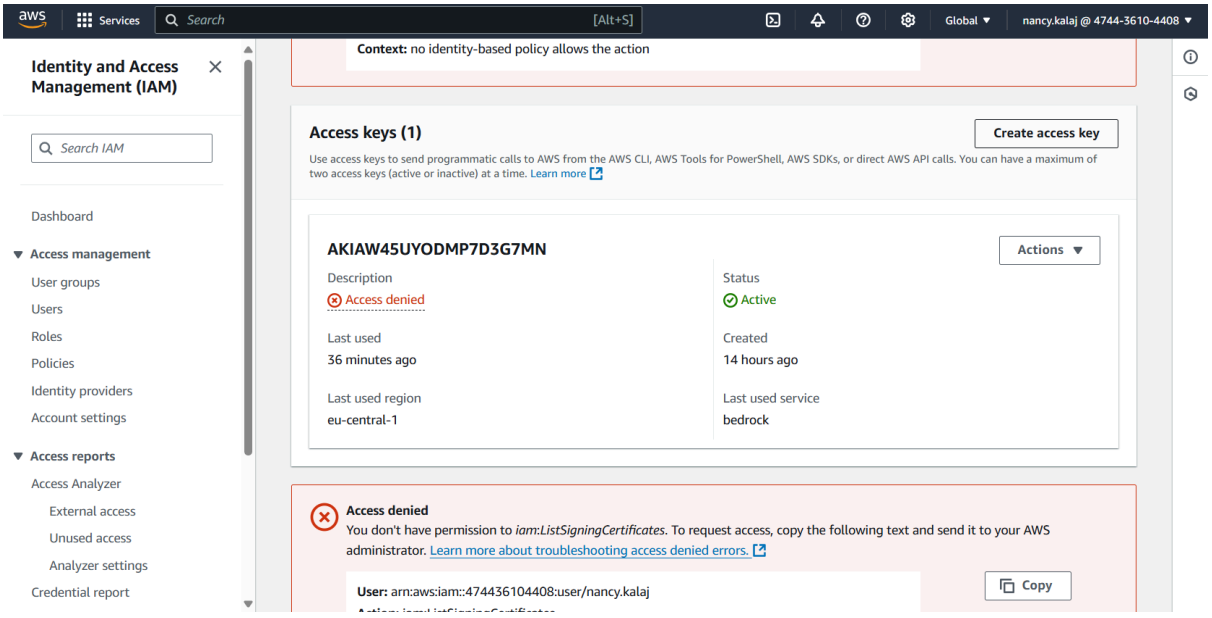
1. Login using account ID, username and password (which you'll be prompted to change after the first login)

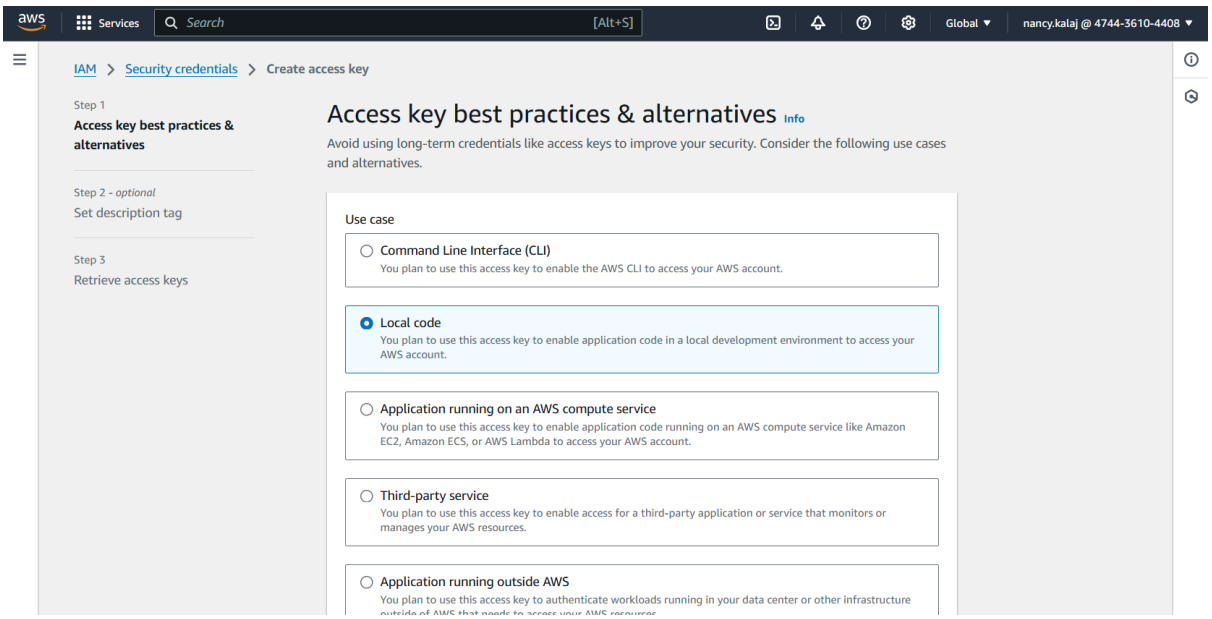
2. Switch the region to Frankfurt (eu-central-1) and click on "Security credentials" in the menu that pops up once you click on your username@accountID on the top right



3. Scroll down until you see the "Access keys" portion of the page and click on "Create access key"



4. Select "Local code", check the box asking you if you want to create an access key at the bottom of the screen and click on "Next"; afterwards insert a description (or don't, you can't see it anyways because of access restrictions) and create the key.



5. Click on the “Download csv” button to download a file containing your Access key ID and Secret access key

6. Install the aws cli; the method you should use depends on which operating system you’re on. Please refer to [Installing or updating to the latest version of the AWS CLI - AWS Command Line Interface (amazon.com)]

7. Open a terminal window in your project folder and type in “aws configure”; you’ll be prompted to enter your:
    a. Access Key ID: the access key in the csv file you downloaded
    b. Secret Access Key: the secret access key in the csv file you downloaded
    c. Default region name: eu-central-1
    d. Default output format: json

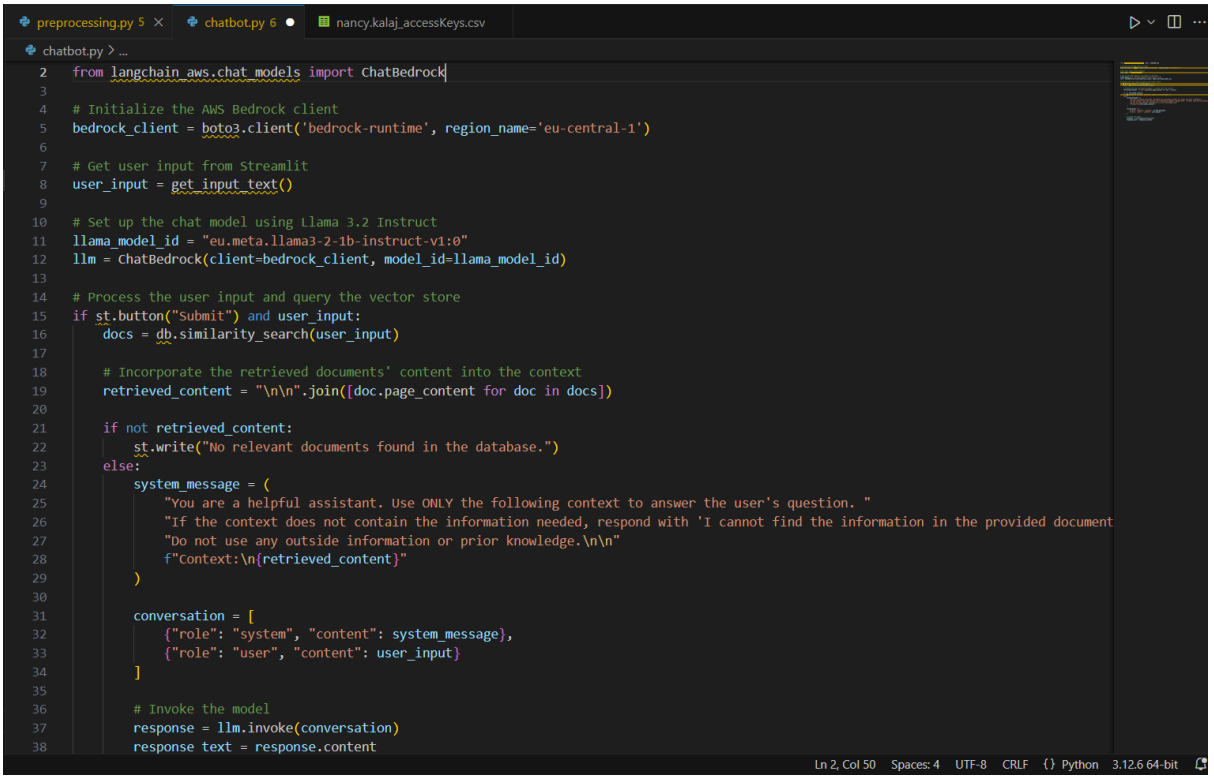
8. After setting up your credentials in this way you should be able to use the Python SDK for AWS (Boto3) to make API calls to the models, namely Amazon Titan Embeddings v2 (amazon.titan-embed-text-v2:0) and Llama 3.2 Instruct (eu.meta.llama3-2-1b-instruct-v1:0); please refer to the official documentation to understand how to format the API calls
    a. [Invoke Meta Llama 3 on Amazon Bedrock using the Invoke Model API - Amazon Bedrock]
    b. [AWS | 🦜🔗 LangChain]
    c. [ChatBedrock | 🦜🔗 LangChain]

9. I’ve updated my quick demo by using the Bedrock models, here are a couple simplified snippets for reference (note: use the langchain_aws class to import BedrockEmbeddings and ChatBedrock to work with the embedding and the chatting model, respectively). NB: I removed bits and pieces from the code, it’s meant to show you how to use Boto3 in a very simple (probably not optimal) way and therefore it is subject to change.

preprocessing.py 5 ● | chatbot.py 7 | nancy.kalaj_accessKeys.csv

preprocessing.py > embedding

```python
import boto3
import json
from langchain_aws import BedrockEmbeddings

# Initialize the AWS Bedrock Runtime client
bedrock_client = boto3.client('bedrock-runtime', region_name='eu-central-1')

# Function to invoke the Bedrock model and get embeddings
def get_embedding(client, model_id, input_text):
    # Create the request body
    payload = {
        "inputText": input_text,
        "dimensions": 512,
        "normalize": True
    }

    # Convert the request to JSON
    request_body = json.dumps(payload)

    # Invoke the model with the correct method for Bedrock runtime
    response = client.invoke_model(
        modelId=model_id,
        contentType="application/json",
        accept="application/json",
        body=request_body
    )

    # Parse and decode the response
    model_response = json.loads(response['body'].read().decode('utf-8'))
    return model_response['embedding']

# Initialize Bedrock embeddings using a wrapper class compatible with LangChain
embed_model = 'amazon.titan-embed-text-v2:0'
bedrock_embeddings = BedrockEmbeddings(client=bedrock_client, model_id=embed_model)
embedding = get_embedding(bedrock_client, embed_model, split.page_content)
```



preprocessing.py 5 | chatbot.py 6 ● | nancy.kalaj_accessKeys.csv

chatbot.py > ...

```python
from langchain_aws.chat_models import ChatBedrock

# Initialize the AWS Bedrock client
bedrock_client = boto3.client('bedrock-runtime', region_name='eu-central-1')

# Get user input from Streamlit
user_input = get_input_text()

# Set up the chat model using Llama 3.2 Instruct
llama_model_id = "eu.meta.llama3-2-1b-instruct-v1:0"
llm = ChatBedrock(client=bedrock_client, model_id=llama_model_id)

# Process the user input and query the vector store
if st.button("Submit") and user_input:
    docs = db.similarity_search(user_input)

    # Incorporate the retrieved documents' content into the context
    retrieved_content = "\n\n".join([doc.page_content for doc in docs])

    if not retrieved_content:
        st.write("No relevant documents found in the database.")
    else:
        system_message = (
            "You are a helpful assistant. Use ONLY the following context to answer the user's question. "
            "If the context does not contain the information needed, respond with 'I cannot find the information in the provided document
            "Do not use any outside information or prior knowledge.\n\n"
            f"Context:\n{retrieved_content}"
        )

        conversation = [
            {"role": "system", "content": system_message},
            {"role": "user", "content": user_input}
        ]

        # Invoke the model
        response = llm.invoke(conversation)
        response_text = response.content
```

Ln 2, Col 50 Spaces: 4 UTF-8 CRLF {} Python 3.12.6 64-bit