Gen AI project cold mailing

cold email generator using Llama

- Llama 3.1
- chroma ADB vector store
- Lang chain
- Streamlit

Problem to be solved : Software service companies like TCS Infosys has clients like Nike, JP Morgan and Kroger to help build software's and to acquire such projects sales team uses techniques to cold email .

Suppose if client company hiring people with requirements

= we gonna extract text from career page→ using LLM Llama 3.1 → job={'role' : 'senior soft eng.', 'skills' : ['2 years experience in React']}→ in Json format then feed to chroma DB which is vector database and relevant portfolios for companys

Lang chain Llama 3.1 is open source → we gonna be using groq cloud which allows to run lama 3.1 as using is locally slows down inference .As Groq uses lpu language processing which makes faster inference to process int faster than the normal

Step 1 :- Playground

Step 2 :- Add api keys → create api → Name it anything and submit → copy the url of password which is below and it is to be not shared to any one

pass key :

2. gsk_

3. [redacted] →
   running

command prompt to go to juypter notebook

C:\Users\javeed>python -m notebook

- create a new folder and open editor file of python 3.1

```
from langchain_groq import ChatGroq

llm = ChatGroq(
temperature=0,
groq_api_key="[redacted]",
# ✅ your valid key
model_name="llama-3.1-8b-instant"  # ✅ supported and active model
)

response = llm.invoke("The first person to land on the moon was ...")
print(response.content)
```

- Now we gonna use chromadb as its light weight and open source
- go to documentation

pip install chromadb
 into command prompt

In jupyter create a new file name it and open editor file and run this

```
import chromadb
client = chromadb.Client()
collection = client.create_collection(name="my_collection")
```

```
collection.add(
documents=[
"This document is about India",
"This document is about Karnataka"
],
ids = ['id1','id2']
)

all_docs = collection.get()
all_docs

documents = collection.get(ids=["id1"])
documents

results = collection.query(
query_texts = ['Query is about Chhole Bhature'],
n_results = 2
)
results
```

[17]:

```
{'ids': [['id2', 'id1']],
 'embeddings': None,
 'documents': [['This document is about Karnataka',
   'This document is about India']],
 'uris': None,
 'included': ['metadatas', 'documents', 'distances'],
 'data': None,
 'metadatas': [[None, None]],
 'distances': [[1.4757359027862549, 1.5452191829681396]]}
```

```
results = collection.query(
```

query_texts = ['Query is about national flag'],

n_results = 2

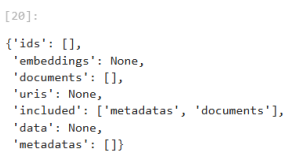
)
results

```
[18]:

{'ids': [['id1', 'id2']],
 'embeddings': None,
 'documents': [['This document is about India',
   'This document is about Karnataka']],
 'uris': None,
 'included': ['metadatas', 'documents', 'distances'],
 'data': None,
 'metadatas': [[None, None]],
 'distances': [[1.4389841556549072, 1.6159753799438477]]}
```

in above two sequence data probability takes shift according to the query we provide

vector data base will help to semantic search and tie up to other querys like this document is about india and in india people eat chhole bhature

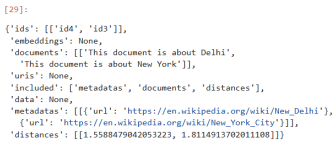
collection.delete(ids=all_docs['ids'])
collection.get()



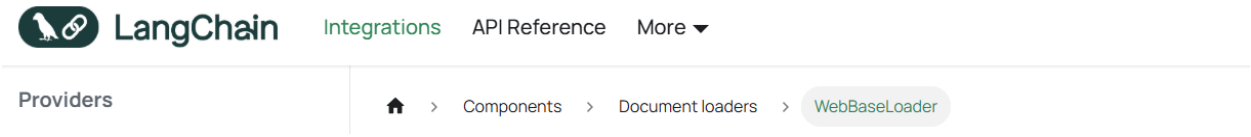
collection.add(
documents=[

"This document is about New York",
"This document is about Delhi"
],
ids = ["id3","id4"],
metadatas = [
{"url": "https://en.wikipedia.org/wiki/New_York_City"},
{"url": "https://en.wikipedia.org/wiki/New_Delhi"}
]
)

results = collection.query(
query_texts = ["Query is about Chhole Bhature"],
n_results = 2
)
results



In lang chain website under components /document loaders/webbaseloader is which allows the link to be accessed

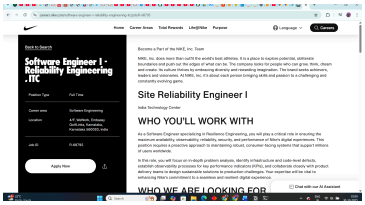


to extract data from it

from langchain_community.document_loaders import WebBaseLoader

loader = WebBaseLoader("https://www.example.com/")

load this to juypter note and add link of the job post from careers



https://careers.nike.com/software-engineer-i-reliability-engineering-itc/job/R-68795

pip install langchain-community

```
from langchain_groq import ChatGroq

llm = ChatGroq(
temperature=0,
groq_api_key="gsk_[redacted]"
# ✅ your valid key
model_name="llama-3.1-8b-instant"  # ✅ supported and active model
)

response = llm.invoke("The first person to land on the moon was ...")
print(response.content)
```

```
from langchain_community.document_loaders import WebBaseLoader

loader = WebBaseLoader("https://careers.nike.com/software-engineer-i-reliability-engineering-itc/job/R-68795")
page_data = loader.load().pop().page_content
print(page_data)
```

now go to chat gpt and ask this to get code

i will give you a job posting from that extract the skills, role and description in a JSON format job description without preamble(copy paste job description)

now back to juypter

chain_extract = prompt_extract | llm → acts like a pipeline to llm

write this:

```
from langchain_core.prompts import PromptTemplate
prompt_extract = PromptTemplate.from_template(
"""
### SCRAPED TEXT FROM WEBSITE:
{page_data}
### INSTRUCTION:
The scraped text is from the career's page of a website.
Your job is to extract the job postings and return them in JSON format containing following keys: 'role', 'experience', 'skills' and 'description'.
Only return the valid JSON.
### VALID JSON (NO PREAMBLE):
"""
)
chain_extract = prompt_extract | llm
res = chain_extract.invoke(input={'page_data':page_data})
print(res.content)
```

```
{
  "Software Engineer I - Reliability Engineering, ITC": {
    "role": "Software Engineer I - Reliability Engineering, ITC",
    "experience": "1-3 years of relevant professional experience in lieu of a degree will be considered",
    "skills": [
      "Java",
      "Node.js",
      "React",
      "Angular",
      "Scala",
      "Python",
      "Golang",
      "DNS",
      "networking",
      "virtualization",
      "Linux operating systems",
      "Docker",
      "serverless architectures",
      "NoSQL database",
      "RESTful APIs",
      "ServiceNow",
      "Jira",
      "Jenkins",
      "Splunk",
      "New Relic",
      "SignalFX"
    ],
    "description": "As a Software Engineer specializing in Resilience Engineering, you will play a critical role in ensuring the maximum availabilit
y, observability, reliability, security, and performance of Nike’s digital experiences."
  }
```


`type(res.content)`

shows str format

now we need to make it in Json Format

```
from langchain_core.output_parsers import JsonOutputParser
json_parser = JsonOutputParser()
json_res = json_parser.parse(res.content)
json_res
```

`type(json_res)`

→ dict

add this file into juypter home page by importing folder to main profile

my_portfolio.csv

```
import os
print(os.getcwd())
```

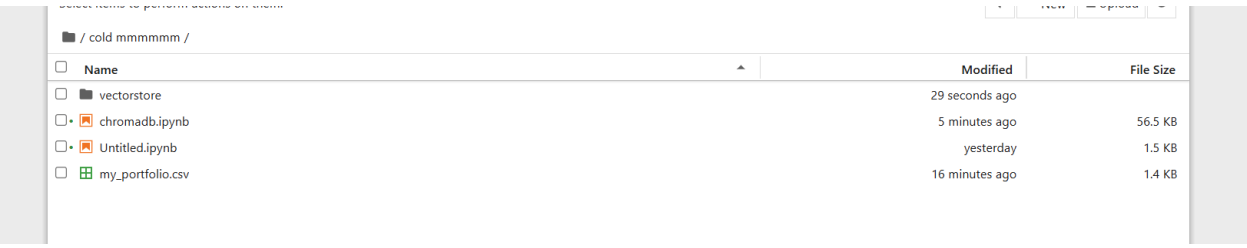
→ helps find directory so you can add file to that directory

```
C:\Users\javeed\cold mmmmmm
```

```
import pandas as pd
df = pd.read_csv("my_portfolio.csv")
df
```

if we use client it creates chroma file in its memory and when use PersistentClient it gives file in its disk

```
import uuid
import chromadb

client= chromadb.PersistentClient('vectorstore')
collection= client.get_or_create_collection(name="portfolio")

ifnot collection.count():
for _, rowin df.iterrows():
        collection.add(documents=row["Techstack"],
                       metadatas={"links": row["Links"]},
                       ids=[str(uuid.uuid4())])
```



now for the experience asked in the job description

```
links= collection.query(query_texts=job['skills'], n_results=2).get('metadatas', [])
links
```

write this

***** note: use this which is below

links = collection.query(query_texts=job['skills'], n_results=2).get('metadatas', [])
links

or

links = collection.query(query_texts= ["Experience in Python","Expertice in React Native"], n_results = 2).get('metadatas')
links

print(json_res.keys())

job = json_res["Software Engineer I - Reliability Engineering, ITC"]
print(job['skills'])

Get the first (and only) job dictionary from json_res

first_key = list(json_res.keys())[0]
job = json_res[first_key]

Now you can safely access its details

print(job['skills'])
print(job['role'])

```
print(job['description'])
```

```
prompt_email = PromptTemplate.from_template(
"""
### JOB DESCRIPTION:
{job_description}

### INSTRUCTION:
You are Javeed, an Engineering graduate specializing in Electronics and IoT systems.
You have developed several real-time automation and embedded projects including:
- IoT-based forest fire prevention system
- Automatic door locker system using microcontroller
- Gas detection and alert system
- Smart street lighting project using Arduino and sensors

You are skilled in C, C++, Python, and embedded programming, with hands-on exper
ience using
microcontrollers (AT89C51, ESP32), GSM/GPS modules, temperature and smoke sen
sors,
and real-time system integration. You also possess a solid understanding of
software engineering concepts including Data Structures, Algorithms, Database Man
agement Systems (DBMS),
Operating Systems, and Computer Networks.

Your task is to write a professional cold email to the client regarding the job mentione
d above,
explaining how your technical background, project experience, and software proficie
ncy make you a
strong fit for the role. Highlight your ability to apply both hardware and software kno
wledge to
real-world automation and IoT solutions.

Also include the most relevant ones from the following portfolio links: {link_list}

Remember, you are Javeed, an engineering graduate with expertise in IoT, automatio
```

```
n, and software development.
Do not provide a preamble.

### EMAIL (NO PREAMBLE):
"""
```

)

```
chain_email = prompt_email | llm
res = chain_email.invoke({"job_description": str(job), "link_list": links})
print(res.content)
```

chain by pass two paramaterrs :

job description and link list

```
print(res.content)
```

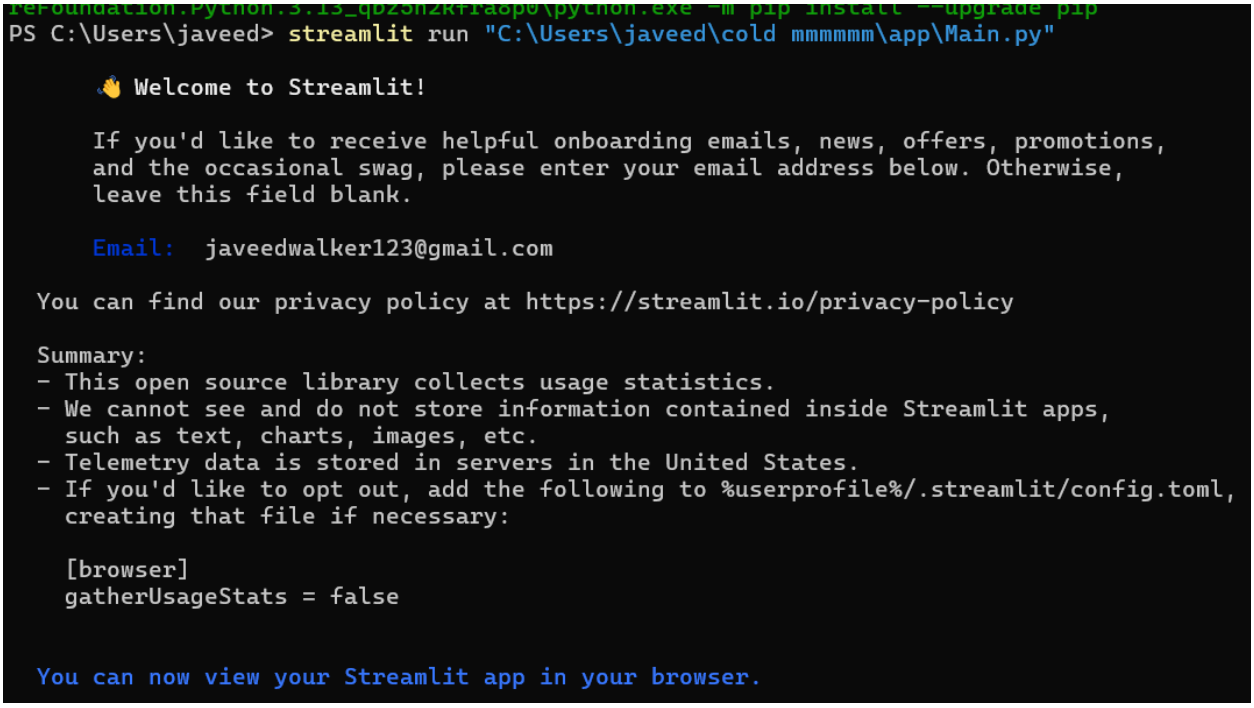
```
Subject: Application for Software Engineer I - Reliability Engineering Role at Nike

Dear Hiring Manager,

I am writing to express my interest in the Software Engineer I - Reliability Engineering role at Nike, as advertised on [Job Board/Source]. With a stro
ng background in electronics and IoT systems, I am confident that my technical expertise and project experience make me a strong fit for this position.

As a graduate in Electronics and IoT systems, I have developed a solid understanding of software engineering concepts, including Data Structures, Algor
ithms, Database Management Systems (DBMS), Operating Systems, and Computer Networks. My proficiency in programming languages such as C, C++, Python, an
d embedded programming has enabled me to design and implement various real-time automation and IoT projects.

Some of my notable projects include an IoT-based forest fire prevention system, an automatic door locker system using a microcontroller, a gas detectio
n and alert system, and a smart street lighting project using Arduino and sensors. These projects have not only honed my skills in hardware and softwar
e integration but also demonstrated my ability to apply technical knowledge to real-world automation and IoT solutions.

In addition to my technical expertise, I have also developed a strong proficiency in software development. I have hands-on experience with microcontrol
lers (AT89C51, ESP32), GSM/GPS modules, temperature and smoke sensors, and real-time system integration. My software skills include proficiency in Jav
a, Node.js, React, Angular, Python, Golang, and other relevant technologies.

I have also showcased my software development skills through various portfolio links, including:

- Java: https://example.com/java-portfolio
- Android: https://example.com/android-portfolio
- React: https://example.com/react-portfolio
- Full-stack JS: https://example.com/full-stack-js-portfolio
- Angular: https://example.com/angular-portfolio
- Python: https://example.com/ml-python-portfolio
- Kotlin Android: https://example.com/kotlin-android-portfolio
- DevOps: https://example.com/devops-portfolio

I am particularly drawn to this role at Nike because of the opportunity to apply my technical expertise to ensure the maximum availability, observabili
ty, reliability, security, and performance of Nike's digital experiences. My experience in IoT and automation has prepared me well for the challenges o
f this role, and I am excited about the prospect of joining a team that shares my passion for innovation and technical excellence.

Thank you for considering my application. I would welcome the opportunity to discuss my qualifications further and explain in greater detail why I am t
```

C:\Users\javeed\cold mmmmmm\app

download pycharam to restructure the order to publish it online now create new folder and name it as app in the path

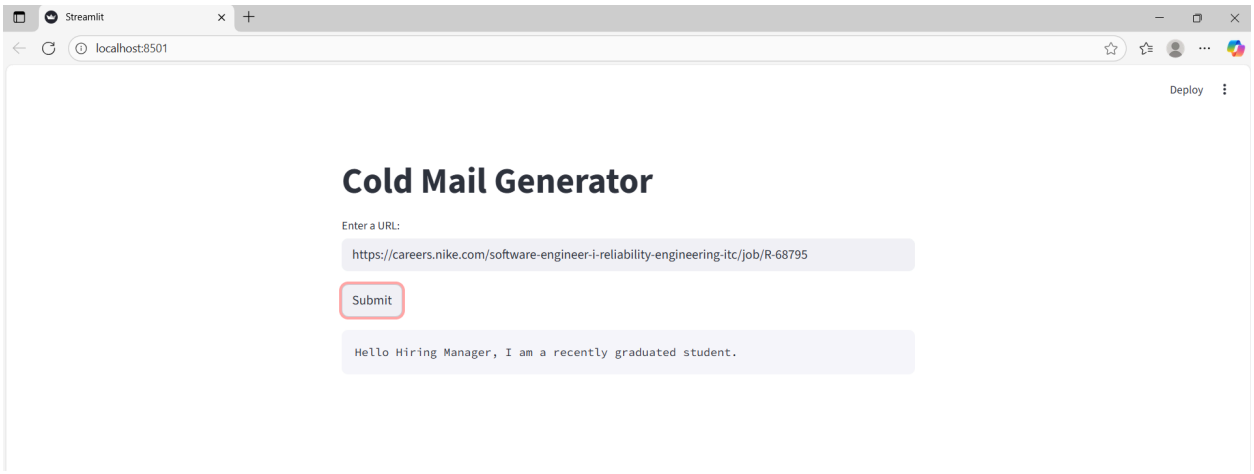
now in pycharm run the file named cold mmm to load project and now create new python file named main.py now in command prompt run pip install streamli to download the extension which builds faster ui environment

```
import streamlit as stst.title("Cold Mail Generator")url_input = st.text_input(    "Enter a URL:",    value="https://careers.nike.com/software-engineer-i-reliability-engineering-itc/job/R-68795")submit_button = st.button("Submit")if submit_button:    st.code("Hello Hiring Manager, I am a recently graduated student.", language='markdown')
```



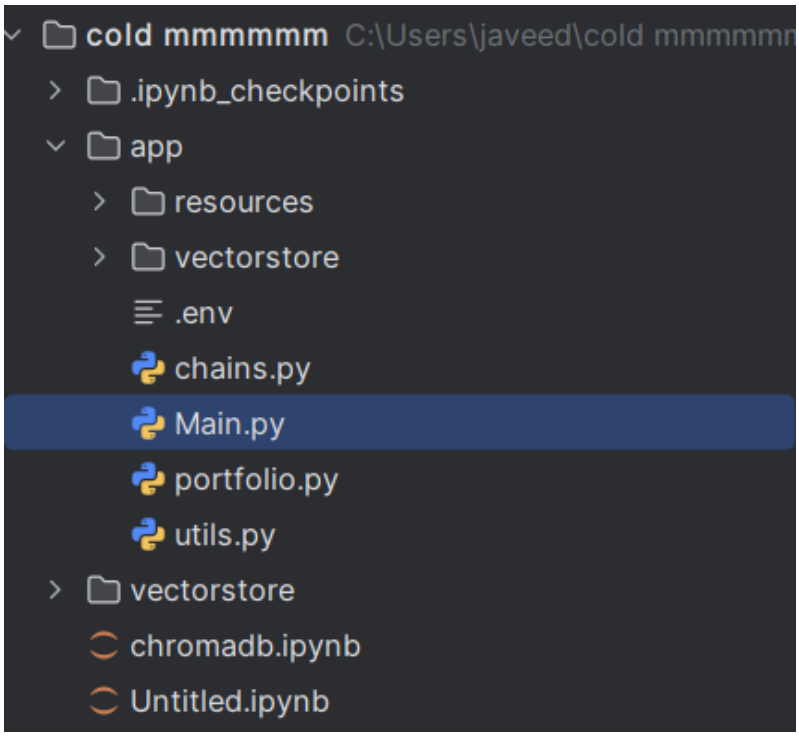
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.1.7:8501



dummy mail generator

now lets work on real generator



main.py file→

```
 import streamlit as st
from langchain_community.document_loaders import WebBaseLoader
from chains import Chain
from portfolio import Portfolio
from utils import clean_text

def create_streamlit_app(llm, portfolio, clean_text):
    st.title("Cold Mail Generator")
    url_input = st.text_input(
        "Enter a URL:",
        value="https://careers.nike.com/software-engineer-i-reliability-engineering-itc/job/R-68795"
    )
    submit_button = st.button("Submit")

    if submit_button:
        try:
            loader = WebBaseLoader([url_input])
            data = clean_text(loader.load().pop().page_content)
            portfolio.load_portfolio()
            jobs = llm.extract_jobs(data)

            for job in jobs:
                skills = job.get('skills', [])
                links = portfolio.query_links(skills)
                email = llm.write_mail(job, links)
                st.code(email, language='markdown')
        except Exception as e:
            st.error(f"An error occurred: {e}")

if __name__ == "__main__":
    chain = Chain()
    portfolio = Portfolio()
    st.set_page_config(layout="wide", page_title="Cold Email Generator", page_icon="📧")
    create_streamlit_app(chain, portfolio, clean_text)
```

chains.py→

```
 import os
from langchain_groq import ChatGroq
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import JsonOutputParser
from langchain_core.exceptions import OutputParserException
from dotenv import load_dotenv

load_dotenv()

class Chain:
    def __init__(self):
        self.llm = ChatGroq(
            temperature=0,
            groq_api_key=os.getenv("GROQ_API_KEY"),
```

```
            model_name="llama-3.1-8b-instant"
        )

    def extract_jobs(self, cleaned_text):
        prompt_extract = PromptTemplate.from_template(
            """
            ### SCRAPED TEXT FROM WEBSITE:
            {page_data}

            ### INSTRUCTION:
            The scraped text is from a career page of a website.
            Your job is to extract the job postings and return them in JSON format containing
            the following keys: 'role', 'experience', 'skills', and 'description'.
            Only return valid JSON.

            ### VALID JSON (NO PREAMBLE):
            """
        )
        chain_extract = prompt_extract | self.llm
        res = chain_extract.invoke(input={'page_data': cleaned_text})

        try:
            json_parser = JsonOutputParser()
            res = json_parser.parse(res.content)
        except OutputParserException:
            raise OutputParserException("Context too big. Unable to parse jobs.")

        return res if isinstance(res, list) else [res]

    def write_mail(self, job, links):
        prompt_
```

portfolio→

```
 import os
import pandas as pd
import chromadb
import uuid

class Portfolio:
    def __init__(self, file_path=None):
        # Dynamically find the correct path to the CSV
        if file_path is None:
            base_dir = os.path.dirname(__file__)
            file_path = os.path.join(base_dir, "resources", "my_portfolio.csv")

        print(f"📂 Loading portfolio from: {file_path}")  # Debug info (optional)

        self.file_path = file_path
        self.data = pd.read_csv(file_path)
        self.chroma_client = chromadb.PersistentClient('vectorstore')
        self.collection = self.chroma_client.get_or_create_collection(name="portfolio")

```

```
    def load_portfolio(self):
        if not self.collection.count():
            for _, row in self.data.iterrows():
                self.collection.add(
                    documents=row["Techstack"],
                    metadatas={"links": row["Links"]},
                    ids=[str(uuid.uuid4())]
                )

    def query_links(self, skills):
        return self.collection.query(query_texts=skills, n_
```

utils.py→

```
import re

def clean_text(text):
    # Remove HTML tags
    text = re.sub(r'&lt;[^&gt;]*?&gt;', '', text)

    # Remove URLs
    text = re.sub(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', '', text)

    # Remove special characters
    text = re.sub(r'[^a-zA-Z0-9 ]', '', text)

    # Replace multiple spaces with a single space
    text = re.sub(r'\s{2,}', ' ', text)

    # Trim leading and trailing whitespace
    text = text.strip()

    # Remove extra whitespace
    text = ' '.join(text.split())

    return text
```

.env →

```
GROQ_API_KEY = [redacted]
```

```
Error: Invalid value: File does not exist: app\Main.py
PS C:\Users\javeed> cd "C:\Users\javeed\cold mmmmmm"
PS C:\Users\javeed\cold mmmmmm> streamlit run app/Main.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8502
  Network URL: http://192.168.1.7:8502

USER_AGENT environment variable not set, consider setting it to identify your requests.
📁 Loading portfolio from: C:\Users\javeed\cold mmmmmm\app\resources\my_portfolio.csv
📁 Loading portfolio from: C:\Users\javeed\cold mmmmmm\app\resources\my_portfolio.csv
📁 Loading portfolio from: C:\Users\javeed\cold mmmmmm\app\resources\my_portfolio.csv
📁 Loading portfolio from: C:\Users\javeed\cold mmmmmm\app\resources\my_portfolio.csv
📁 Loading portfolio from: C:\Users\javeed\cold mmmmmm\app\resources\my_portfolio.csv
📁 Loading portfolio from: C:\Users\javeed\cold mmmmmm\app\resources\my_portfolio.csv
```