```
# Helper functions
def json_gpt(input: str):
    GPT_MODEL = "gpt-3.5-turbo"
    openai.api_key = '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'
    completion = openai.ChatCompletion.create(
        model=GPT_MODEL,
        messages=[
            {"role": "system", "content": "Output only valid JSON"},
            {"role": "user", "content": input},
        ],
        temperature=0.5,
    )

    text = completion.choices[0].message.content
    parsed = json.loads(text)

    return parsed


def embeddings(input: list[str]) -> list[list[str]]:
    response = openai.Embedding.create(model="text-embedding-ada-002", input=input)
    return [data.embedding for data in response.data]


# This function will generate the response for us.

def generateResponse(companyName, date):
  USER_QUESTION = "What caused the changes in the" + companyName + "stock prices?"
  QUERIES_INPUT = f"""
  You have access to a search API that returns recent news articles.
  Generate an array of search queries that are relevant to this question.
  Use a variation of related keywords for the queries, trying to be as general as possible.
  Include as many queries as you can think of, including and excluding terms.
  For example, include queries like ['keyword_1 keyword_2', 'keyword_1', 'keyword_2'].
  Be creative. The more queries you include, the more likely you are to find relevant results.

  User question: {USER_QUESTION}

  Format: {{"queries": ["query_1", "query_2", "query_3"]}}
  """

  queries = json_gpt(QUERIES_INPUT)["queries"]

  # Let's include the original question as well for good measure
  queries.append(USER_QUESTION)

  dateObj = datetime.strptime(date, '%Y-%m-%d')
  dateObj = dateObj - timedelta(days=20)
  dateObj = datetime.strftime(dateObj, '%Y-%m-%d')
  date = date + ',' + dateObj
  tags = companyName
  keywords = companyName + ', shares, stocks, finance, trading, market, stock price'
  url = "https://api.apilayer.com/financelayer/news?keywords="+keywords+"&date="+date+"&limit=2"

  payload = {}
  headers= {
    "apikey": "1Dt3h1V1vmwBOjcKIip6kAJtrMbzYLaN"
  }

  apiResponse = requests.request("GET", url, headers=headers, data = payload)

  if apiResponse != null:
    status_code = apiResponse.status_code
    result = apiResponse.text
    jsonObj = json.loads(result)
    title = jsonObj['data'][0]['title']
    description = jsonObj['data'][0]['description']
    publishingDate = jsonObj['data'][0]['published_at']
    articles = []

    for query in tqdm(queries):
      articles = articles + jsonObj["data"]

    # remove duplicates
    articles = list({article["url"]: article for article in articles}.values())

    HA_INPUT = f"""
    Generate a hypothetical answer to the user's question. This answer will be used to rank search results.
    Pretend you have all the information you need to answer, but don't use any actual facts. Instead, use placeholders
    like NAME did something, or NAME said something at PLACE.

    User question: {USER_QUESTION}
```

```
    user question: {USER_QUESTION}

    Format: {{"hypotheticalAnswer": "hypothetical answer text"}}
    """

    hypothetical_answer = json_gpt(HA_INPUT)["hypotheticalAnswer"]

    hypothetical_answer_embedding = embeddings(hypothetical_answer)[0]

    article_embeddings = embeddings(
      [
          f"{article['title']} {article['description']}"
          for article in articles
      ]
    )

    # Calculate cosine similarity
    cosine_similarities = []
    for article_embedding in article_embeddings:
      cosine_similarities.append(dot(hypothetical_answer_embedding, article_embedding))

    scored_articles = zip(articles, cosine_similarities)

    # Sort articles by cosine similarity
    sorted_articles = sorted(scored_articles, key=lambda x: x[1], reverse=True)

    formatted_top_results = [
      {
          "title": article["title"],
          "description": article["description"],
          "url": article["url"],
      }
    ]

    ANSWER_INPUT = f"""
    Generate an answer to the user's question based on the given search results.
    TOP_RESULTS: {formatted_top_results}
    USER_QUESTION: {USER_QUESTION}

    Include as much information as possible in the answer. Reference the relevant search result urls as markdown links.
    """

    completion = openai.ChatCompletion.create(
        model=GPT_MODEL,
        messages=[{"role": "user", "content": ANSWER_INPUT}],
        temperature=0.5,
        stream=True,
    )

    text = ""
    for chunk in completion:
        text += chunk.choices[0].delta.get("content", "")

    response = 'As per our investigations, we detected the presence of a financial fraud after analyzing the price related information th
  return response + text


# This function will take the query of the user as the input and will return the query response to the user.

def processQuery(query):
  answer = ''
  tokens = query.split(" ")
  if query.find('Identify from the following data if there is any possibility of insider trade') :
    companyName = tokens[15]
    date = tokens[17]
    # highPrice = tokens[19]
    # lowPrice = tokens[21]
    # openPrice = tokens[23]
    # closePrice = tokens[25]
    # volumeOfSharesTraded = tokens[27]

    # Deep Learning model code which will detect any price fluctuations or anomalies are there in the data or not.
    modelName = companyName + 'model.pkl'
    model = pickle.load(open(modelName, 'rb'))

    # Make prediction
    data = pd.DataFrame({'ds': [date], 'y': [closePrice]})
    forecast = model.predict(data)
    data.ds = data.ds.astype('datetime64[ns]')
    performance = pd.merge(data, forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']], on='ds')
    data = performance[performance['ds']== date]
    anomaly = 'yes' if (data['y']< data['yhat_lower']).bool() | (data['y']> data['yhat_upper']).bool() else 'no'
```

```
    if anomaly == 'yes':
      answer = generateResponse(companyName, date)

      # This code is to fetch the trading data from the data set which is to be displayed to the user in the form of a table
      df = pd.read_csv("insider_dataset.csv")
      df_filtered = df[df['Trade Date'] == date]
      df_final = df_filtered[df_filtered['Company'] == companyName]
    else:
      answer = 'As per our investigations and calculations, the provided trade does not indicate any drastic price fluctuations in the st
    return answer

  else:
    openai.api_key = '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'
    messages = [{"role": "user", "content": query}]
    response = openai.ChatCompletion.create(
    model=model,
    messages=messages,
    temperature=0,
  )

  answer = response.choices[0].message["content"]
  return answer
```