AI で BI（Streamlit、Langchain、PandasAI、PyGWalker）

目次

【改変履歴】
2024/6/1 新規作成（Streamlit/Langchain）
2024/7/1 第 3 章（PandasAI）と第 4 章（PyGWalker）を追加

AI で BI（Streamlit、Langchain、PandasAI、PyGWalker）

はじめに

生成 AI／大規模言語モデル（LLM）の進化でコードは自動生成するようになり、プログラマーが不要になるという将来を語る人がいます。ところで、コード作成を行う製造工程がシステム開発全体に占める工数は 30%未満でしかなく（IPA 統計資料[1]）しかもこの製造工程にはコード作成よりも手間のかかる単体試験を含んでいます。LLM は人間の生産物から学習しており、また、設計に誤りが含まれる可能性もあるので AI に試験不要のコードを生成させることは困難です。加えて、仕様変更の取り込みやシステムの管理／運用を人間が行うのでは生産性の向上はそれほど期待できません。

多くのシステムは数パーセントの高難度のプログラムと数十パーセントの簡易な参照系や一時的に必要なプログラムを含みます。簡易なプログラムについてはコードの開発ではなく、AI が利用者と会話しながら処理を代行する方式にすれば品質も生産性も上がる可能性があります。

1. LLM を組込んだアプリケーションの作成

LLM は利用者からの質問を解釈し、質問と一緒に渡された文書（コンテキスト）の翻訳や要約を行い回答を生成することができます。更に、前回迄の回答歴に質問を付加して LLM に分析させることにより BI ツールのドリルダウン風のことができます（現状では速度と精度に問題がありますが...）。

LLM でデータを処理するには、RAG（検索拡張生成：Retrieval-Augmented Generation）と問合せ文面から SQL（Text-to-Sqls）やコードを動的に生成して実行させる方式があります。どちらにしても API を使った LLM とのやり取りが必要で、そのための少しのコードが必要になります。

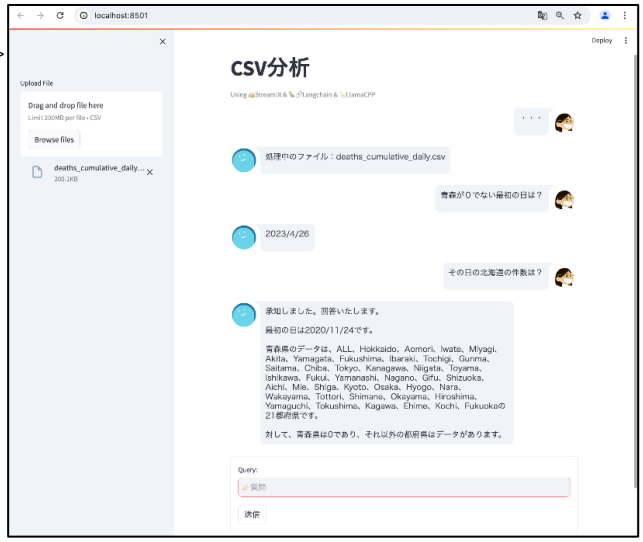
以降に利用者と LLM の API を仲介するアプリの作成例を示します。

2. チャットアプリケーション（Langchain）

アプリは、各種の GUI で利用者との会話を制御する Streamlit（Apache License 2.0）と RAG を実現する Langchain（MIT License）で構成します。Langchain は以下の資材を連結して動作させます。

・処理対象のファイルと利用者の問合せ文字列をベクトル数値化する Embedding モデル
・ファイルのベクトルデータを保存するベクトルストア
・大規模言語モデル（LLM）

＜チャットアプリの画面例＞



[1] ソフトウェア開発データ白書 https://www.ipa.go.jp/archive/publish/wp-sd/download.html

2.1. ライブラリのインストール

LLM に関係するライブラリは依存ライブラリも含めて開発が活発で日々更新されています。それだけに Python と組み合わせるライブラリのバージョンに注意が必要です。

（1） Python のインストール

Streamlit と Langchain が動作する Python のバージョンは以下の URL で確認できます。

- https://pypi.org/project/streamlit/
- https://pypi.org/project/langchain/

Python は以下のサイトからダウンロードできます。

https://www.python.org/downloads/

※ 2024/5/22 時点では Python 3.12 が条件に合う最新版なので、Windows 11 上に当該バージョンをインストールした前提で以下説明します

（2） Python 仮想環境の作成

依存ライブラリのバージョンが他のプロジェクトに影響しないように、以下のコマンドで仮想環境を作ります。下記コマンド例ではホームディレクトリ配下に pyvenv¥csvchat というディレクトリ（中間パスを含め、なければ作られます）の中に Python 3.12 の実行環境が作られます。

```
py -3.12 -m venv ${env:HOMEPATH}¥pyvenv¥csvchat
```

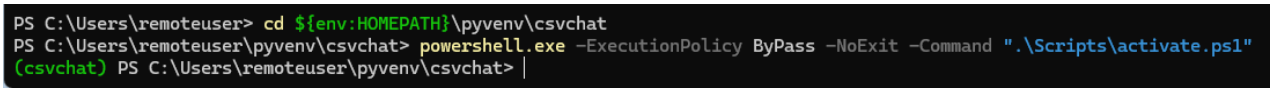
（3） Python 仮想環境の活性化とライブラリのインストール

仮想環境を活性化するには仮想環境のディレクトリにある Scripts¥activate.ps1 を実行します。

＜仮想環境の活性化-実行例＞

```
cd ${env:HOMEPATH}¥pyvenv¥csvchat
powershell.exe -ExecutionPolicy ByPass -NoExit -Command ".¥Scripts¥activate.ps1"
```

コマンドライン左端に仮想環境名(csvchat)が表示されたら、ライブラリをインストールします。



＜仮想環境にインストールするライブラリ＞

```
pip install streamlit==1.33.0
pip install streamlit-chat
pip install langchain
pip install langchain-chroma
pip install langchain-community
pip install sentence-transformers
pip install llama-cpp-python
```

※streamlit の ver.1.34 を使うと streamlit-chat の import で RuntimeError: Runtime hasn't been created! が発生するのでバージョンを下げています。また、langchain の使用バージョンは 0.2.0 です

2.2. コードの作成例

以降のコードを chat.py という名前のファイルに書き込みます。

Streamlit はブラウザを GUI に使い、各種の会話型アプリが作れるテンプレート[2]を提供しています。特にチャットは決まりきった情報を埋め込むだけで比較的簡単に Web アプリを作ることができます。

［chat.py］

```
import tempfile
import streamlit as st
from streamlit_chat                              import message
from langchain.document_loaders.csv_loader       import CSVLoader
from langchain.embeddings                        import HuggingFaceEmbeddings
from langchain_chroma                            import Chroma
from langchain_community.llms                    import LlamaCpp
from langchain_core.prompts                      import PromptTemplate
from langchain.chains                            import ConversationalRetrievalChain

# debug=True で Langchain の処理経過をログに表示
import langchain
langchain.debug = True

# 画面(Streamlit)構成
st.title("CSV 分析")
st.caption("Using 👑Streamlit & 🦜🔗Langchain & 🦙LlamaCPP")

# CSV Uploader サイドメニュー
upFile = st.sidebar.file_uploader("Upload File", type="csv")

# 指定された CSV ファイルを一時ファイルに書き込み
if upFile:
    with tempfile.NamedTemporaryFile(delete=False) as tFile:
        tFile.write(upFile.getvalue())
        tFile_path = tFile.name

    # CSV データ取込み
    loader = CSVLoader(file_path=tFile_path, encoding="utf-8", csv_args={'delimiter': ','})
    data = loader.load()

    # Embedding モデル取得 ／ CSV データをベクトル数値化して VectorStore(Croma) に保存
    embeddings = HuggingFaceEmbeddings(model_name='sentence-transformers/all-MiniLM-L6-v2')
    db = Chroma.from_documents(data, embeddings, persist_directory="./chroma_db")

    # LLM の設定
    llm = LlamaCpp(
        model_path=r".¥llmchat/ELYZA-japanese-Llama-2-7b-instruct-q8_0.gguf"
      , n_ctx=4000
      , max_tokens=1024
      , verbose=True
      , stop=["Question:", "Answer:"]
    )
```

[2] Streamlit App ギャラリー https://streamlit.io/gallery?category=snowflake-powered


```
    ### チャットボットを生成 ###
    ## Llama2 形式のプロンプトを組み立て ##
    # プロンプトテンプレートの定義
    pre_prompt = """[INST] <<SYS>>¥nYou are a helpful, respectful and honest assistant. Always answer as helpfully as possible, while being safe. Your answers should not include any harmful, unethical, racist, sexist, toxic, dangerous, or illegal content. Please ensure that your responses are socially unbiased and positive in nature.¥n¥nIf a question does not make any sense, or is not factually coherent, explain why instead of answering something not correct. If you don't know the answer to a question, please don't share false information.¥n<</SYS>>¥n¥nGenerate the next agent response by answering the question. Answer it as succinctly as possible. You are provided several documents with titles. If the answer comes from different documents please mention all possibilities in your answer and use the titles to separate between topics or domains. If you cannot answer the question from the given documents, please state that you do not have an answer.¥n"""

    prompt = pre_prompt + "CONTEXT:¥n¥n{context}¥n" +"Question : {question}" + "[/INST]"
    llama_prompt = PromptTemplate(template=prompt, input_variables=["context", "question"])
    # Chain 生成
    chain = ConversationalRetrievalChain.from_llm(
        llm
      , retriever=db.as_retriever()
      , verbose=True
      , combine_docs_chain_kwargs={"prompt": llama_prompt}
      , return_source_documents=True
    )

    ### 以下、ガイド表示文言以外変更不要（チャットの初期化、会話の経過表示） ###
    # Function for conversational chat
    def conversational_chat(query):
        result = chain({"question": query, "chat_history": st.session_state['history']})
        st.session_state['history'].append((query, result["answer"]))
        return result["answer"]

    # Initialize chat history
    if 'history' not in st.session_state:
        st.session_state['history'] = []

    # Initialize messages
    if 'generated' not in st.session_state:
        st.session_state['generated'] = ["処理中のファイル：" + upFile.name]

    if 'past' not in st.session_state:
        st.session_state['past'] = ["・・・"]

    # Create containers for chat history and user input
    response_container = st.container()
    container = st.container()

    # User input form
    with container:
        with st.form(key='my_form', clear_on_submit=True):
            user_input = st.text_input("Query:", placeholder="👉質問", key='input')
            submit_button = st.form_submit_button(label='送信')

        if submit_button and user_input:
```



```
            output = conversational_chat(user_input)
            st.session_state['past'].append(user_input)
            st.session_state['generated'].append(output)

    # Display chat history
    if st.session_state['generated']:
        with response_container:
            for i in range(len(st.session_state['generated'])):
                message(st.session_state["past"][i], is_user=True, key=str(i) + '_user',
avatar_style="big-smile")
                message(st.session_state["generated"][i], key=str(i), avatar_style="thumbs")
```

------- chat.py ソースここまで -------

このコード例を実行するためには以下の資材を使います。

事前に準備が必要なものと、実行時にインターネット越しに取り込まれるものがあります。

・Embedding モデル …実行時に HuggingFace のサイトからダウンロードされます

sentence-transformers/all-MiniLM-L6-v2 を使用

・LLM …事前に量子化済（gguf）のモデルをダウンロードしておき、パスを指定します

・分析対象の CSV ファイル …実行例では厚生労働書のオープンデータ

https://www.mhlw.go.jp/stf/covid-19/open-data.html

から deaths_cumulative_daily.csv をダウンロードしました

<deaths_cumulative_daily.csv：日付、都道府県名の見出し付きカンマ区切り>

Date,ALL,Hokkaido,Aomori,Iwate,Miyagi,Akita,Yamagata,Fukushima,Ibaraki,Tochigi,Gunma,Saitama,Chiba,Tokyo,Kanagawa,Niigata,Toyama,Ishikawa,Fukui,Yamanashi,Nagano,Gifu,Shizuoka,Aichi,Mie,Shiga,Kyoto,Osaka,Hyogo,Nara,Wakayama,Tottori,Shimane,Okayama,Hiroshima,Yamaguchi,Tokushima,Kagawa,Ehime,Kochi,Fukuoka,Saga,Nagasaki,Kumamoto,Oita,Miyazaki,Kagoshima,Okinawa

2020/5/9,620,51,0,0,1,0,0,0,9,0,18,41,39,180,51,0,13,16,8,0,0,6,1,34,1,1,13,59,32,2,2,0,0,0,2,0,1,0,3,3,24,0,1,3,0,0,0,5

2020/5/10,631,56,0,0,1,0,0,0,9,0,18,42,40,180,53,0,13,16,8,0,0,6,1,34,1,1,13,59,34,2,2,0,0,0,2,0,1,0,3,3,24,0,1,3,0,0,0,5

（以下略）

2.3. アプリケーションの起動

ライブラリのインストールを行った仮想環境（csvchat ディレクトリ）に作成した chat.py を格納し、Streamlet から呼び出すと Web アプリとして起動し、デフォルトブラウザで画面が開きます。

① 仮想環境の活性化コマンドを実行

ターミナルの左端に (csvcht) のように仮想環境名が表示されている場合は実行不要です。

```
cd ${env:HOMEPATH}¥pyvenv¥csvchat
powershell.exe -ExecutionPolicy ByPass -NoExit -Command ".¥Scripts¥activate.ps1"
```

② streamlet コマンドを実行

```
streamlit run chat.py
```

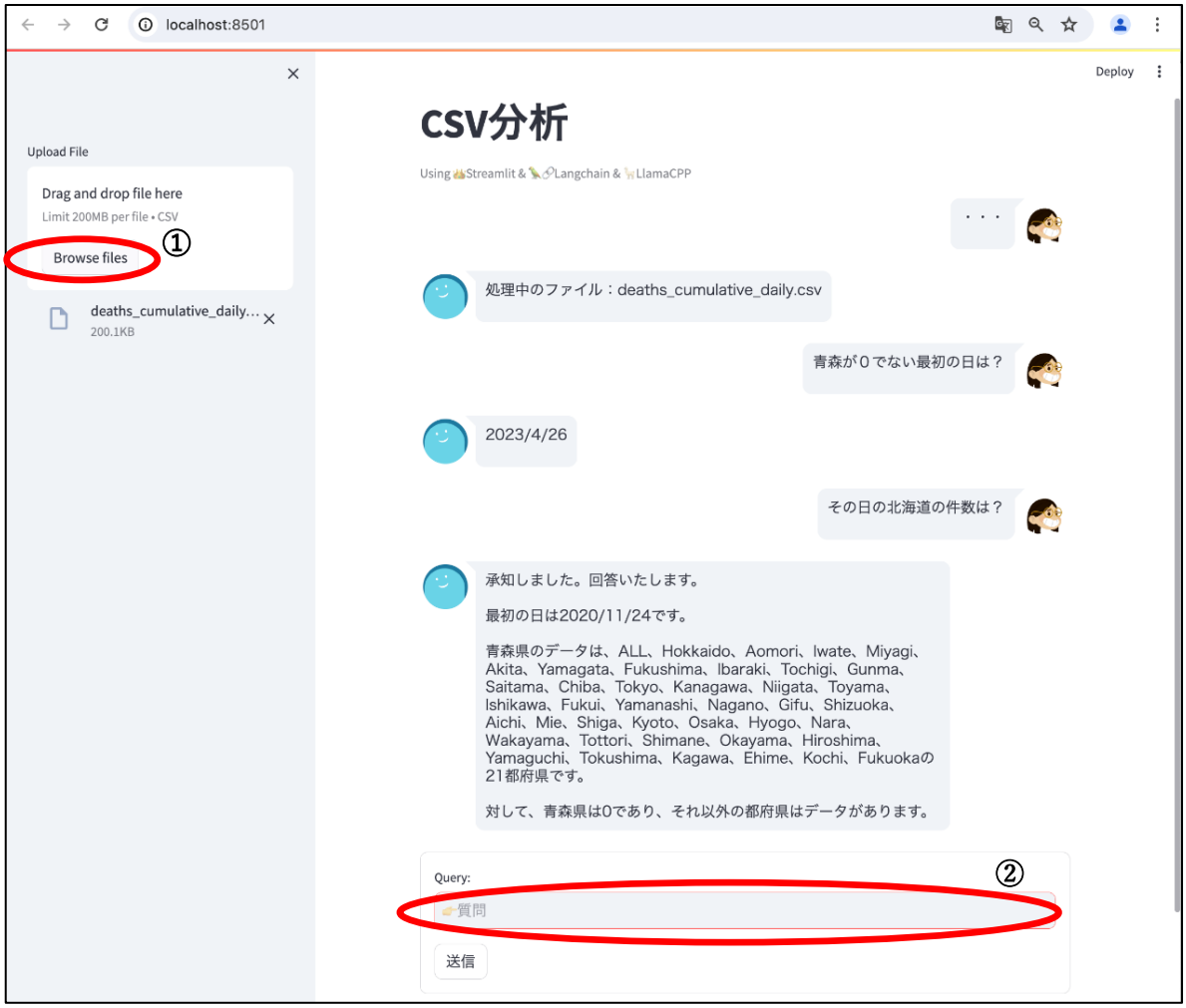
※ pip install streamlit で streamlit.exe が仮想環境の Scripts ディレクトリに格納され、activate.ps1 の実行により環境変数の Path に仮想環境の Scripts ディレクトリが登録されて実行可能になります

2.4. アプリケーションの操作

このアプリは以下の要領で CSV ファイルを 1 つだけ読込み、内容に関連する問合せを会話形式で行います。会話の内容は保存（コード中の'history'）し、後続の会話に引き継がれます。

① Browse files を押下し、分析したい CSV ファイルを選択します

② LLM を使って調べたい内容を入力します…以下、QA 形式で繰返し



2.5. 応答動作

以下がアプリケーションが行う処理の概略です。

① 画面から入力された質問とチャット履歴が Langchain の ConversationalRetrievalChain を受け取り

② StuffDocumentsChain が LLM に渡すプロンプトを作るための文字列連結を行います。この際、質問の文字列に類似したチャンクをベクトルストアから取り出してコンテキストとします

③プロンプトの固定文字列"[INST]<<SYS>>～<</SYS>>"（Llama2 の場合のプロンプト）とコンテキスト、質問を合成して LLM に渡します

④ LLM からの回答（generations）からチャット内容を編集します

AI で BI（Streamlit、Langchain、PandasAI、PyGWalker）

コード中に langchain.debug = True の記述があると streamlet を実行しているターミナルに処理の進行状態が出力されます。これにより RAG の実行状況や、プロンプトの設定値の適否を確認できます。…以下出力例

2.6. 制約

現時点（Langchain 0.2.0）でローカルの LLM（llama-cpp-python 0.2.75 による操作）の使用は以下の制約があります。

（1） 応答速度

当初 LLM から使用可能な GPU が無い PC で実行したところ、10 分以上 停止状態になったため（Ctrl+c で打切り）、実行サンプルは GPU 付きの Mac 環境を使用しました。

以下スペックの GPU 付き Mac で LLM からの応答には 5 分程度要しました。

チップセットの機種:	Apple M2 Pro
種類:	GPU
バス:	内蔵
コアの総数:	19
製造元:	Apple (0x106b)
Metal 対応:	Metal 3

※ 実行例で使ったマンドや操作は仮想環境を活性化して以降は Windows も Mac も同一です

（2） 精度

RAG の仕組み上、問合せの文字列はまずベクトルストアの検索に使われ、LLM の回答はこの検索結果を使って作られます。この例のように CSV ファイルを対象にする場合 Langchain は検索結果として 1 レコードだけを選んで LLM に渡すので、以下の注意が必要です。

① CSV レコードにある程度の長さのテキストが含まれないと、検索結果が期待通りにならない
（テキストが複数のトークン≒単語に分割できる語数が必要）

② 複数レコードが対象になる処理（集計や件数のカウント等）はできない

③ 上記の①、②のような不本意な状況でも、正常に処理したかのように回答される

3. AI によるコード生成＆実行（PandasAI）

PandasAI は Pandas（BSD License のデータ分析ライブラリ）が入力（CSV、Excel、JSON、RDB その他）したデータフレームに対して、LLM が問合せ文から生成した処理コードを実行します。

PandasAI は一部（pandasai.ee~）を除いて[3]商用利用可能な MIT ライセンスのオープンソースです。

3.1. 環境の作成

Python の仮想環境を作り、そこに必要なライブラリをインストールします。以降に示すコード例は以下の環境を前提にします。

（1） 仮想環境

自分のホームディレクトリ¥pyvenv の下に pandasi という仮想環境を作ります。

```
py -3.12 -m venv ${env:HOMEPATH}¥pyvenv¥pandasai
```

（2） 仮想環境の活性化

自分のホームディレクトリ¥pyvenv¥pandasi に移動後、¥Scripts¥activate.ps1 を実行します。

```
cd ${env:HOMEPATH}¥pyvenv¥pandasai
powershell.exe -ExecutionPolicy ByPass -NoExit -Command ".¥Scripts¥activate.ps1"
```

（3） PandasAI と使用ライブラリのインストール

活性化した仮想環境で pip コマンドによりインストールを行います。

```
pip install pandasai streamlit PyYAML openpyxl
```

3.2. Ollama のインストールとセットアップ

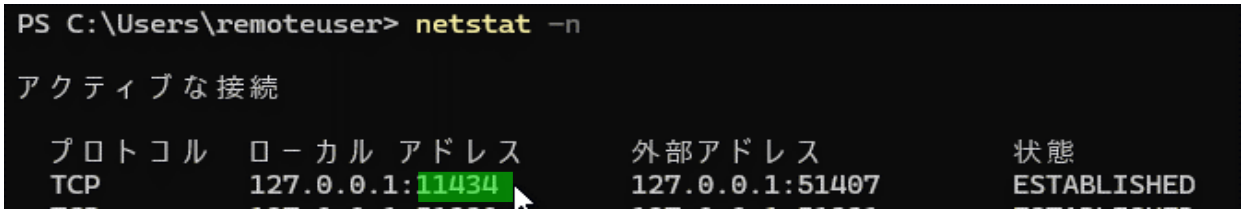
LLM は Ollama 経由で Llama3 を使います。

（1） Ollama のインストール

Ollama は以下のサイトに Windows 版のインストーラが公開されています。

https://ollama.com/download

※ インストールすると自動で Ollama のサービスが登録され、ポート 11434 で LLM の API が使えるようになります



（2） LLM のダウンロード

Ollama をインストールすると Ollama のコマンドが使えるようになるので、サブコマンドの pull を使って LLM（Llama 3）をダウンロードします。

```
ollama pull llama3
```

[3] PandasAI ライセンス https://docs.pandas-ai.com/license

3.3. コードの作成例

仮想環境の pandasi ディレクトリに以下のソースを作成します。

［pandaschat.py］

```
import sys
# GUI の Streamlit とデータフレームの Pandas
import streamlit as st
import streamlit.components.v1 as components
import pandas as pd

## 画面表示要素 https://docs.streamlit.io/develop/api-reference/widgets
## マークダウン記法が使えます
# ページ
st.set_page_config(
    page_title="Excel, CSV 分析", layout="wide"
)
## 画面左サイドバー
# ファイル選択
st.sidebar.write('## ファイル選択')
f_type = st.sidebar.selectbox("ファイルの種類（選択）:",["xlsx", "csv"])
f_path = st.sidebar.file_uploader(
    label="ファイル選択：", type=[f_type], accept_multiple_files=False
)

if f_path:

    def load_data(f_path, f_type):

        if  f_type  ==  "xlsx":
            try:
                # Excel 読み込み
                data = pd.read_excel(f_path
                       # シート選択
                     , sheet_name = st.sidebar.selectbox(
                                    "シート名：", pd.ExcelFile(f_path).sheet_names
                                    )
                       # 見出し行[必ずある前提 …ない場合は None]
                     , header = st.sidebar.number_input("見出し行（1 行目=0）",0,100)
                )
            except:
                st.info("Excel 読み込みエラー！")
                sys.exit()

        elif f_type == "csv":
            try:
                # CSV ファイル読み込み
                data = pd.read_csv(f_path)
            except:
                st.info("CSV 読み込みエラー！")
                sys.exit()
        else:
            st.info("xlsx, csv 以外のファイル")
```



```
            sys.exit()

        return data

    data = load_data(f_path, f_type)
    st.write( '#### ファイル内容 ')
    try:
      # データ初期表示
      st.dataframe(data, use_container_width=True, height=210)

    except:
      st.info("ファイル内容表示エラー！")
      sys.exit()

## --PandasAI 関係--
    ################################################################################
    # LLM に Ollama 経由 Lama3 を使う例。予め ollama から Llama3 を取得要…ollama pull llama3
    # (Elyza 等の非サポート LLM を使う場合は、Langchain か Llamaindex のライブラリを使用)
    ################################################################################
    from pandasai.llm.local_llm import LocalLLM
    ollama_llm = LocalLLM(api_base="http://localhost:11434/v1", model="llama3")

    # SmartDataframe は単発応答、会話型の場合は Agent クラスを使う（インタフェースは同）
    from pandasai import SmartDataframe
    # キャッシュの使用を抑止する場合は、config に "enable_cache": False を追加
    sdf = SmartDataframe(data, config={"llm": ollama_llm, "verbose": True})

    prpt = st.text_area("質問？")
    if st.button("回答"):
        if prpt:
            with st.spinner("LLM 考え中..."):
                res = sdf.chat(prpt)
                st.info(res)
                st.code('＜生成・実行したコード＞  ¥n{}'
                        .format(sdf.last_code_generated), language='python', line_numbers=True
                )
        else:
            st.warning("質問を入力してください。")
------- pandaschat.py ソースここまで -------
```

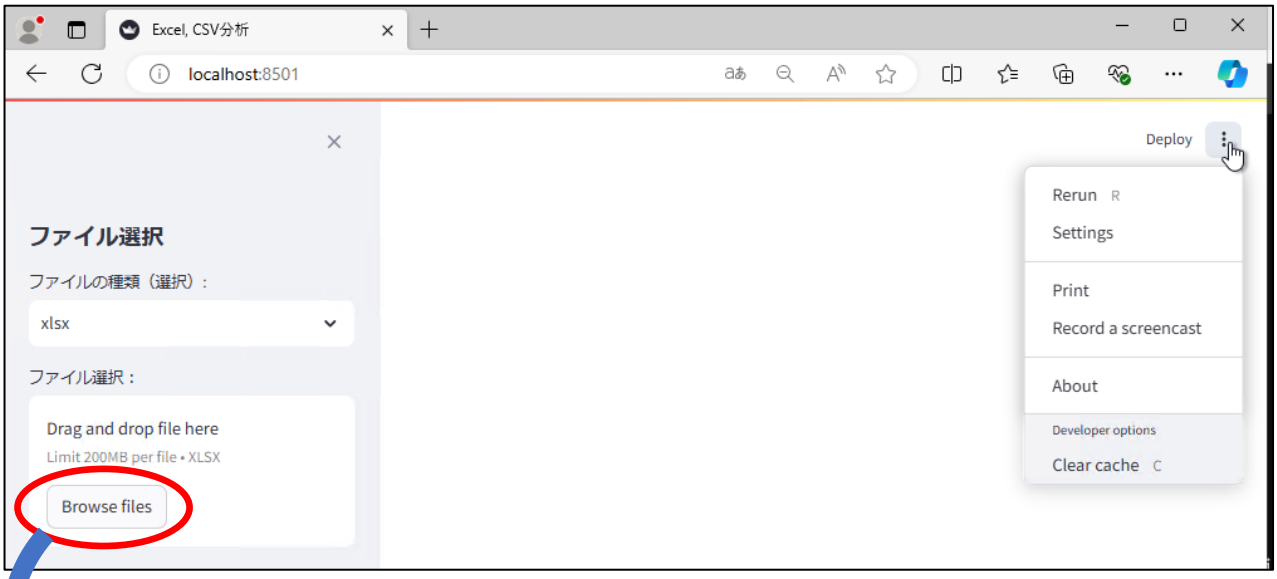

3.4. アプリケーションの起動

仮想環境の活性化から Streamlet による pandaschat.py 起動迄の一連のコマンドは以下になります。

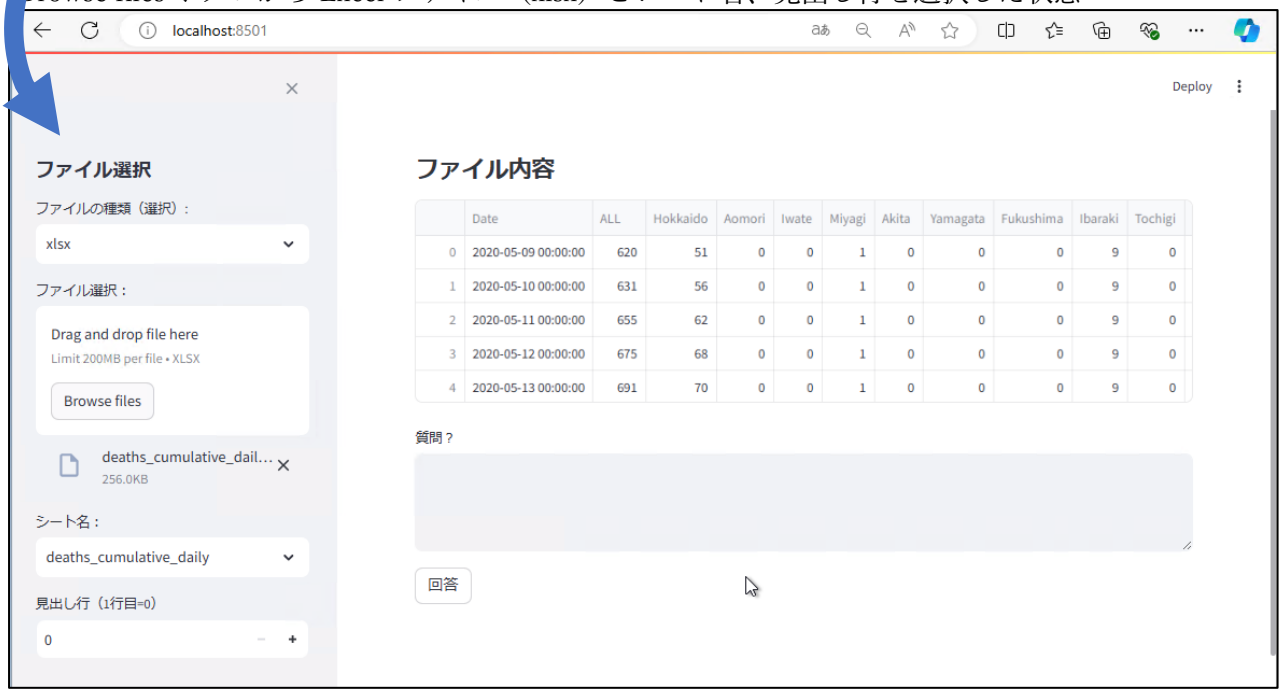
```
cd ${env:HOMEPATH}¥pyvenv¥pandasai
powershell.exe -ExecutionPolicy ByPass -NoExit -Command ".¥Scripts¥activate.ps1"
streamlit run pandaschat.py
```

3.5. アプリケーションの操作

初画面でファイル選択ボタンから分析対象とする xlsx か csv のファイルを選択します。



Browse files ボタンから Excel ファイル（xlsx）とシート名、見出し行を選択した状態



このアプリケーションは、ファイル内容に関係する内容に関して［質問？］の欄を使って問合せを行います。問合せに対する回答は［回答］ボタンを押下でボタンの下に表示するようにしています。

AI で BI（Streamlit、Langchain、PandasAI、PyGWalker）

【問合せの実行例】

質問は日本語を理解（日付=Date、北海道=Hokkaido 等）できますが、PandasAI のバージョンによっては回答が日本語にならない場合があり、質問文の中で明示的に日本語の回答を求めています。

※ 画像下部の＜生成・実行されたコード＞が LLM で生成し、PandasAI により実行されたコード

AI で BI（Streamlit、Langchain、PandasAI、PyGWalker）

【グラフの作成を指示した例】

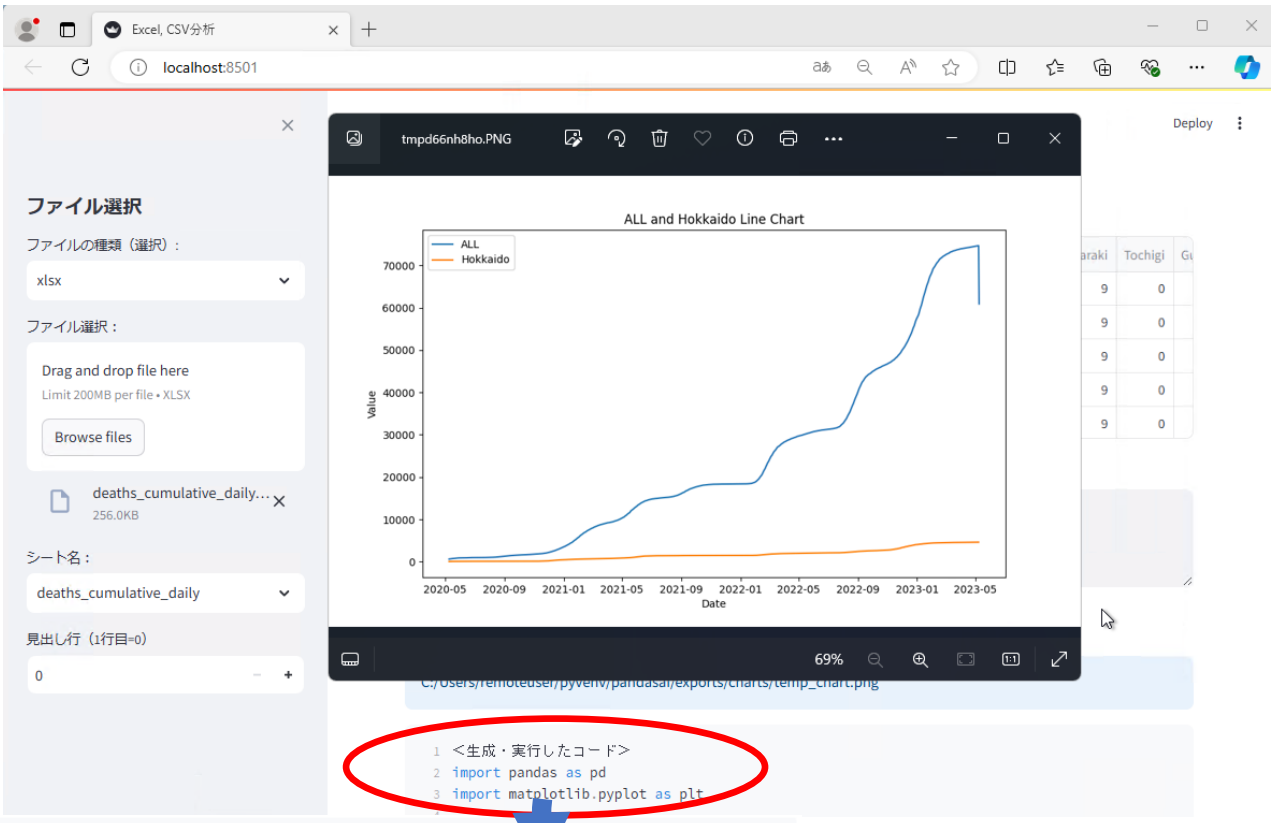
質問？

ALLの棒グラフを描き、その内数として北海道の線を引いてください。

回答

C:/Users/remoteuser/pyvenv/pandasai/exports/charts/temp_chart.png

※ グラフが png ファイルとして生成され、png に関係づけられたアプリで開かれる
（北海道の描線を指示したせいか、棒グラフではなく線グラフが作成された）




```
＜生成・実行したコード＞
import pandas as pd
import matplotlib.pyplot as plt

# Load the data
dfs = [pd.read_csv('data.csv')]

# Select the columns we're interested in
columns_to_plot = ['ALL', 'Hokkaido']
df = dfs[0][columns_to_plot]

# Plot the bar graph for ALL
plt.bar(df.index, df['ALL'])

# Add the Hokkaido line
plt.plot(df.index, df['Hokkaido'], linestyle='--')

# Set title and labels
plt.title('ALL vs Hokkaido')
plt.xlabel('Date')
plt.ylabel('Value')

# Show the plot
plt.show()

# Declare result variable as a dictionary
result = {'type': 'plot', 'value': 'temp_chart.png'}
```


3.6. 制約・注意点

PandasAI を実行する場合は、以下の点に注意が必要です。

① LLM は PandasAI に呼び出される都度異なるコードを生成します。これを PandasAI が試行して正常な結果が得られなければ廃棄して再度 LLM を呼出します。コード生成～試行～コード生成の繰り返しに長時間掛かったり、最後まで正常なコードが作れずに回答が得られない場合があります。

② デフォルトではキャッシュ（*仮想環境ディレクトリ*¥cache）を使い同一内容の問い合わせに回答するので、不正な回答が続く場合は cache 内の.db ファイルを人手で削除する必要があります。また、SmartDataframe（コード例の単発の応答用）の他に会話形式の応答をする Agent クラスがキャッシュを使うので SmartDataframe と Agent を切り替える場合も.db の削除が必要です。

③日本語を理解しますが、主語・目的語、回答事項が不明瞭な問合せ文にはまともな回答が返ってきません。同一問合せ文から生成されるコードも毎回変わるのですが、問合せの言葉の組合せや語順が変わると根本的に回答が異なってきます。

<質問例 1>

ALL の件数が最初に 10000 を超えた日の北海道の件数と青森の件数の合計を日本語で教えてください。

[生成コード]

```
result = {'type': 'string', 'value': ''}
for i, row in enumerate(dfs[0].iterrows()):
    if int(row[1]['ALL']) > 10000:
        result['value'] += f" {row[1]['Hokkaido']} + {row[1]['Aomori']} =
{int(row[1]['Hokkaido']) + int(row[1]['Aomori'])}"
        break
print(result)
```

回答＞ 843.0 + 22 = 865

<質問例 2 >

ALL の件数が最初に 10000 を超えた日の北海道と青森の合計は？

[生成コード]

```
result = {}
for df in dfs:
    if df['ALL'].values[0] >= 10000 and df.iloc[0]['Date'] > '2023-02-16':
        hokkaido_sum = sum(df.loc[df['ALL'] >= 10000, ['Hokkaido']].values)
        aomori_sum = sum(df.loc[df['ALL'] >= 10000, ['Aomori']].values)
        total_sum = hokkaido_sum + aomori_sum
        result = {'type': 'number', 'value': total_sum}
        break
if not result or not result['value']:
    result = {'type': 'number', 'value': -1}
print(result)
```

回答＞ -1

AI で BI（Streamlit、Langchain、PandasAI、PyGWalker）

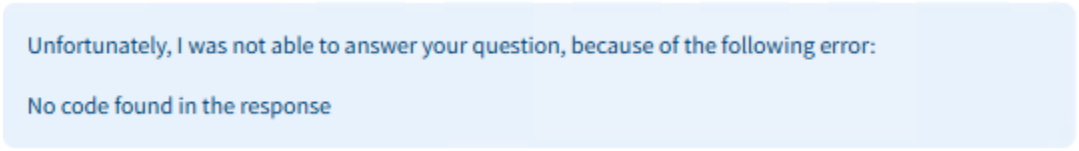
【補足】LLM は同文の問合せに対して毎回異なるコードを作成し、回答形式も異なります。
以下は質問例 1 をそのまま再実行した別の結果例です。

[生成コード]

```
total_count = 0
target_date = None
for row in dfs[0].iterrows():
    if int(row[1]['ALL']) > 10000:
        total_count += int(row[1]['Hokkaido']) + int(row[1]['Aomori'])
        target_date = row[0]
        break
if target_date is None:
    result = {'type': 'string', 'value': 'No date found that meets the condition'}
else:
    result = {'type': 'string', 'value': f'ALL の件数が最初に 10000 を超えた日の北海道の件数と青森の件数の合計は {total_count} です.'}
print(result)
```

回答＞ ALL の件数が最初に 10000 を超えた日の北海道の件数と青森の件数の合計は 865 です.

※ 全く同じ質問に実行可能なコードを作れず、以下の結果を返す場合もあります



4. 簡易な Excel,CSV の図表化（PyGWalker）

オープンソース化と量子化により LLM が手近で使えるようになりましたが、事務処理に使用するレベルの一般的な Window PC では動作が緩慢になることは否めません。欲しい機能と負荷・手間を天秤にかけ、もっと手軽に済ませたい場合は PyGWalker[4]（オープンソース Apache 2.0 license）を Pandas と組み合わせて Excel や CSV、RDB その他各種のデータソースが手軽に図表化できます。

4.1. 環境の作成

仮想環境を作り、そこに必要なライブラリをインストールします。以降に示すコード例は、以下の環境を前提にします。

（1） 仮想環境

自分のホームディレクトリ¥pyvenv の下に csvgrph という仮想環境を作ります。

```
py -3.12 -m venv ${env:HOMEPATH}¥pyvenv¥csvgrph
```

（2） 仮想環境の活性化

自分のホームディレクトリ¥pyvenv¥csvgrph に移動後、¥Scripts¥activate.ps1 を実行します。

```
cd ${env:HOMEPATH}¥pyvenv¥csvgrph
powershell.exe -ExecutionPolicy ByPass -NoExit -Command ".¥Scripts¥activate.ps1"
```

（3） PyGWalker と使用ライブラリのインストール

活性化した仮想環境で pip コマンドによりインストールを行います。

```
pip install streamlit pandas pygwalker openpyxl
```

4.2. コードの作成例

仮想環境の csvgrph ディレクトリに以下のソースを作成します。

［graph.py］

```
import sys
import streamlit as st
import streamlit.components.v1 as components
import pandas as pd
from pygwalker.api.streamlit import StreamlitRenderer

## 画面表示要素 https://docs.streamlit.io/develop/api-reference/widgets
## マークダウン記法が使えます
# ページ
st.set_page_config(
    page_title="Excel, CSV 分析", layout="wide"
)
## 画面左サイドバー
# ファイル選択
st.sidebar.write('## ファイル選択')
f_type = st.sidebar.selectbox("ファイルの種類（選択）:",["xlsx", "csv"])
f_path = st.sidebar.file_uploader(
```

[4] https://kanaries.net/pygwalker


```
    label="ファイル選択：", type=[f_type], accept_multiple_files=False
)

if f_path:

    def load_data(f_path, f_type):

        if  f_type  ==  "xlsx":
            try:
                # Excel 読み込み
                data = pd.read_excel(f_path
                       # シート選択
                     , sheet_name = st.sidebar.selectbox(
                                    "シート名：", pd.ExcelFile(f_path).sheet_names
                                    )
                       # 見出し行[必ずある前提 …ない場合は None]
                     , header = st.sidebar.number_input("見出し行（1 行目=0）",0,100)
                )
            except:
                st.info("Excel 読み込みエラー！")
                sys.exit()

        elif f_type == "csv":
            try:
                # CSV ファイル読み込み
                data = pd.read_csv(f_path)
            except:
                st.info("CSV 読み込みエラー！")
                sys.exit()
        else:
            st.info("xlsx, csv 以外のファイル")
            sys.exit()

        return data

    data = load_data(f_path, f_type)
    st.write( '#### ファイル内容 ')
    try:
      # データ初期表示
      st.dataframe(data, use_container_width=True)
    except:
      st.info("ファイル内容表示エラー！")
      sys.exit()

## PyGWalker ##

    # 表示形式選択
    vis_select = st.sidebar.checkbox("__ダッシュボード？__")

    if vis_select:
        st.write( '#### ダッシュボード ')
        pyg_app = StreamlitRenderer(data)
        pyg_app.explorer()
```

------- graph.py ソースここまで -------

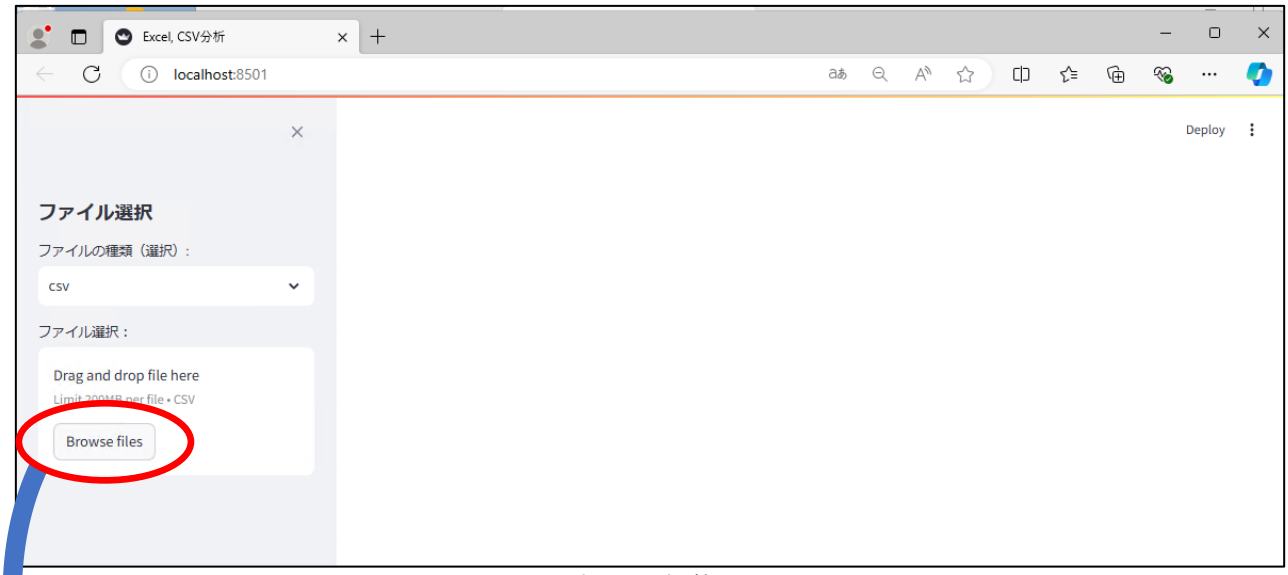
4.3. アプリケーションの起動

仮想環境の活性化から Streamlet による graph.py 起動迄の一連のコマンドは以下になります。

```
cd ${env:HOMEPATH}¥pyvenv¥csvgrph
powershell.exe -ExecutionPolicy ByPass -NoExit -Command ".¥Scripts¥activate.ps1"
streamlit run graph.py
```

4.4. アプリケーションの操作

初画面でファイル選択ボタンから分析対象とする xlsx か csv のファイルを選択します。

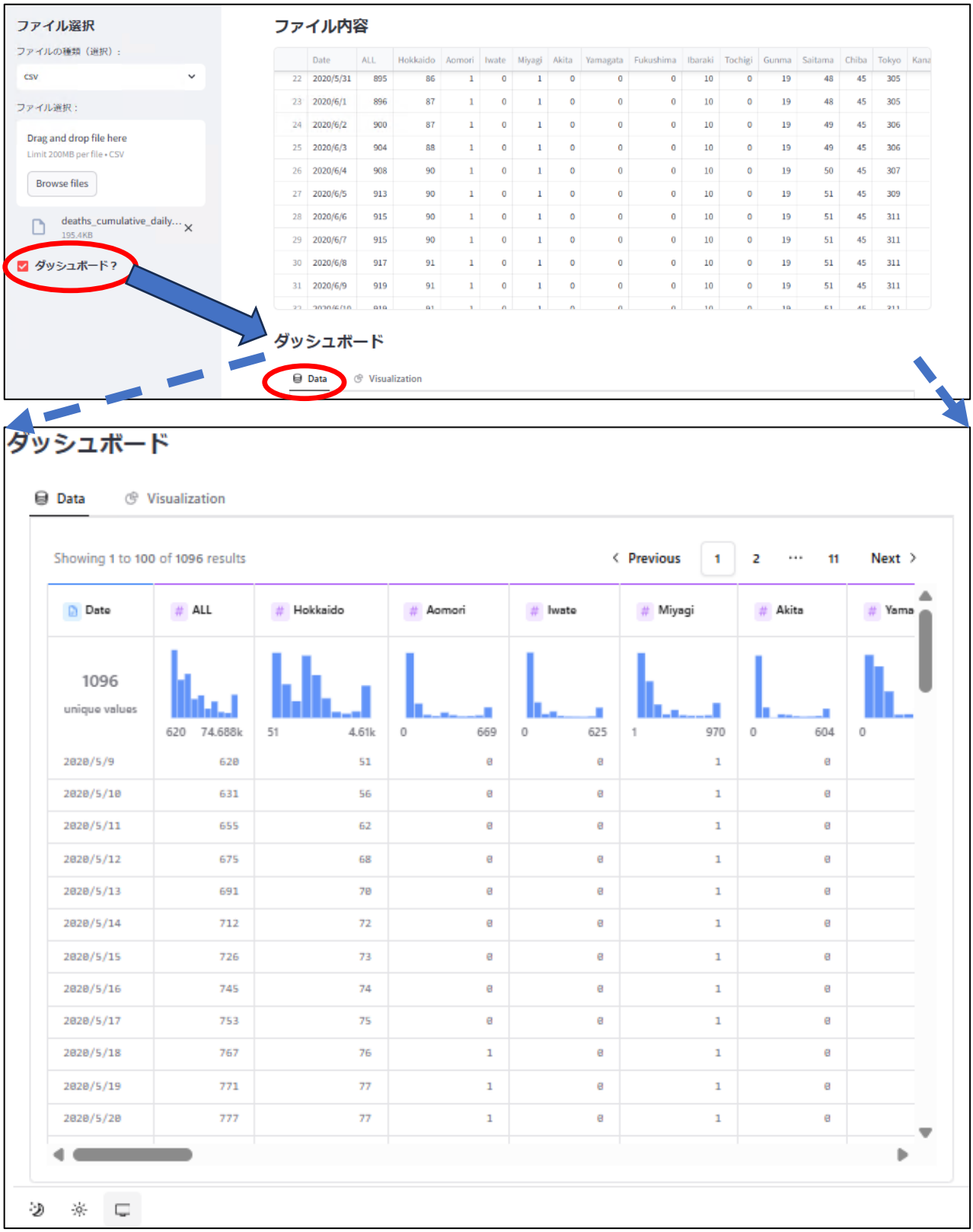


Browse files ボタンから CSV ファイルを選択した状態

	Date	ALL	Hokkaido	Aomori	Iwate	Miyagi	Akita	Yamagata	Fukushima	Ibaraki	Tochigi	Gunma	Sait
0	2020/5/9	620	51	0	0	1	0	0	0	9	0	18	
1	2020/5/10	631	56	0	0	1	0	0	0	9	0	18	
2	2020/5/11	655	62	0	0	1	0	0	0	9	0	18	
3	2020/5/12	675	68	0	0	1	0	0	0	9	0	18	
4	2020/5/13	691	70	0	0	1	0	0	0	9	0	18	
5	2020/5/14	712	72	0	0	1	0	0	0	9	0	18	
6	2020/5/15	726	73	0	0	1	0	0	0	9	0	18	
7	2020/5/16	745	74	0	0	1	0	0	0	9	0	18	
8	2020/5/17	753	75	0	0	1	0	0	0	9	0	18	
9	2020/5/18	767	76	1	0	1	0	0	0	9	0	18	

このアプリケーションは、ファイルを選択すると最初に内容を表示します。
次に画面左の「ダッシュボード？」のチェックをクリックするとグラフの作成ができます。

AI で BI（Streamlit、Langchain、PandasAI、PyGWalker）

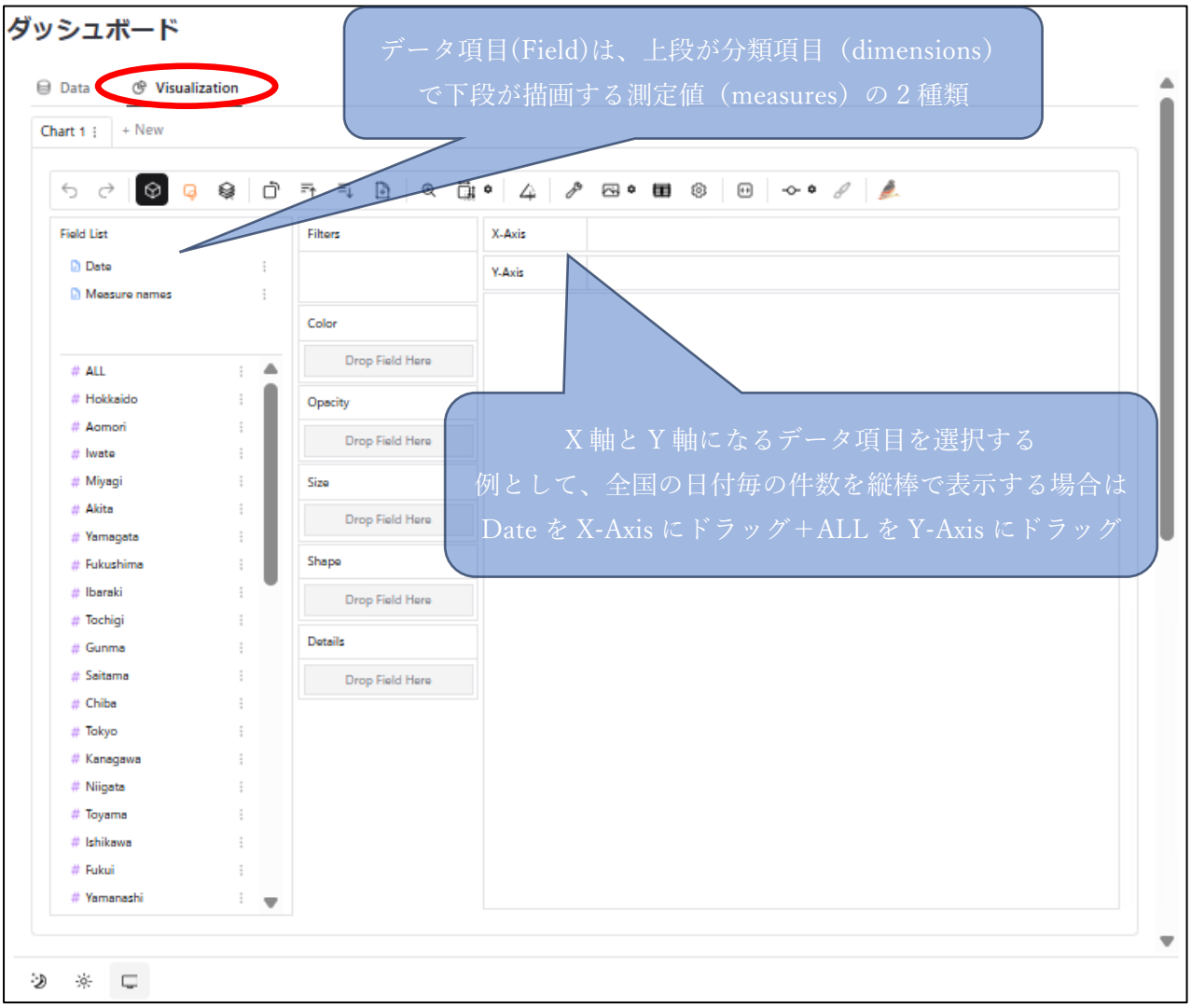


※ “ダッシュボード”の下が PyGWalker で表示している部分で、Data タブを開くとデフォルトで各列（都道府県）の数値範囲毎のレコード数（例えば、値 0~100 のレコードが 100 件 etc.）が棒グラフで表示され、その下には行毎（日付）毎の値が表示されます

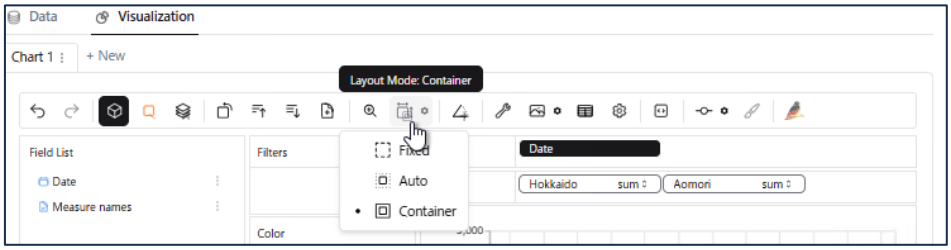
AI で BI（Streamlit、Langchain、PandasAI、PyGWalker）

Visualization（見える化）タブ

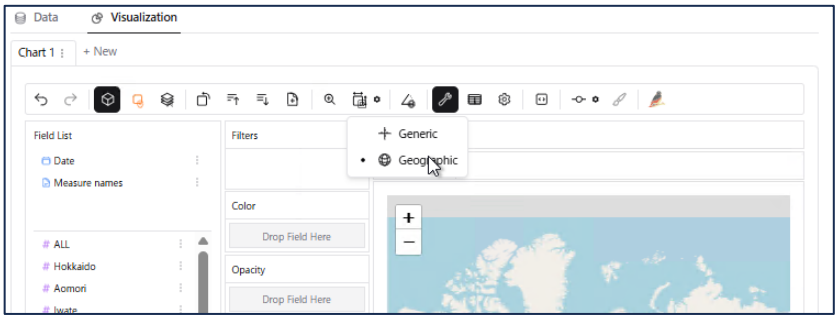
簡単な操作で簡単なグラフを描いたり地図情報のプロット等ができます。



グラフの表示サイズ … Fix 全てのグラフを一つの領域に表示、Auto グラフを並べる、
Container ブラウザ上の表示領域に合わせる

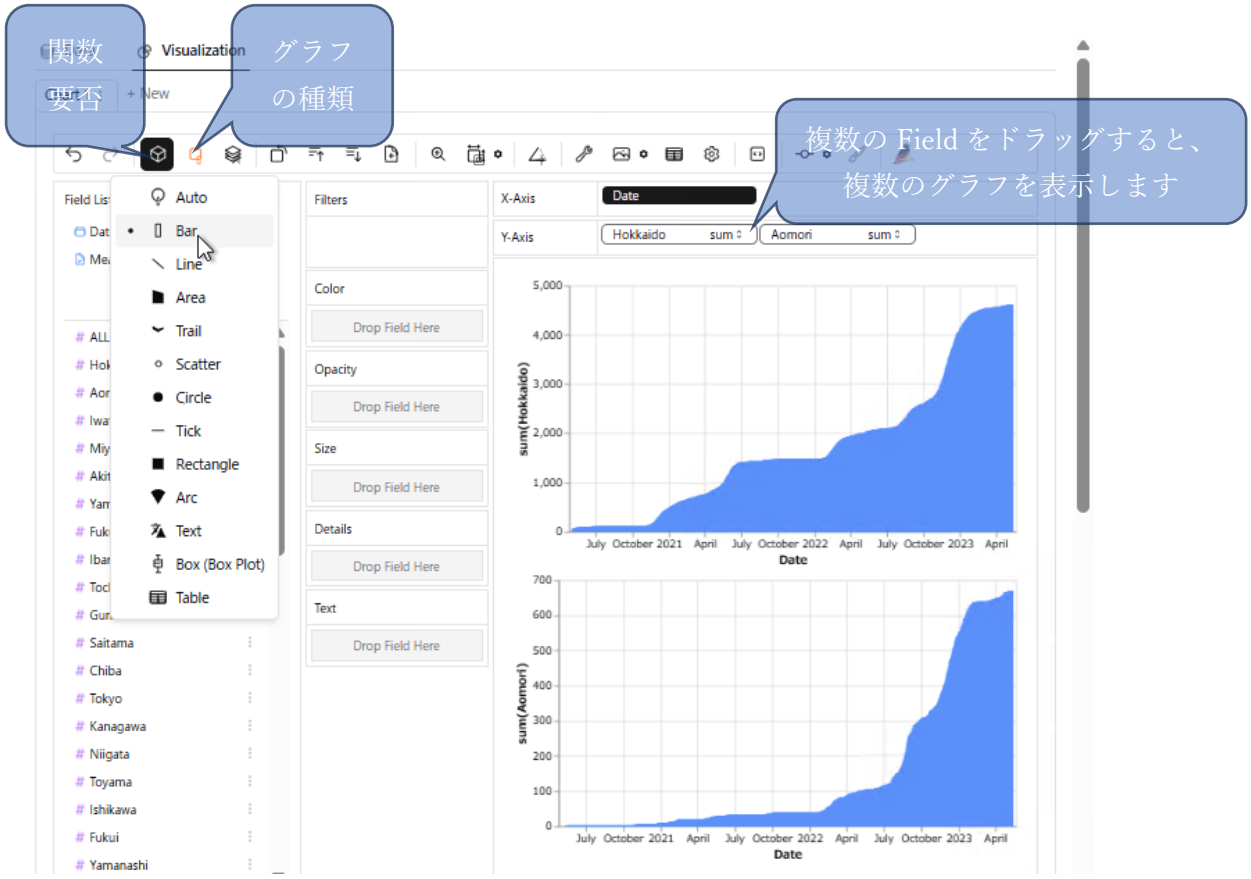


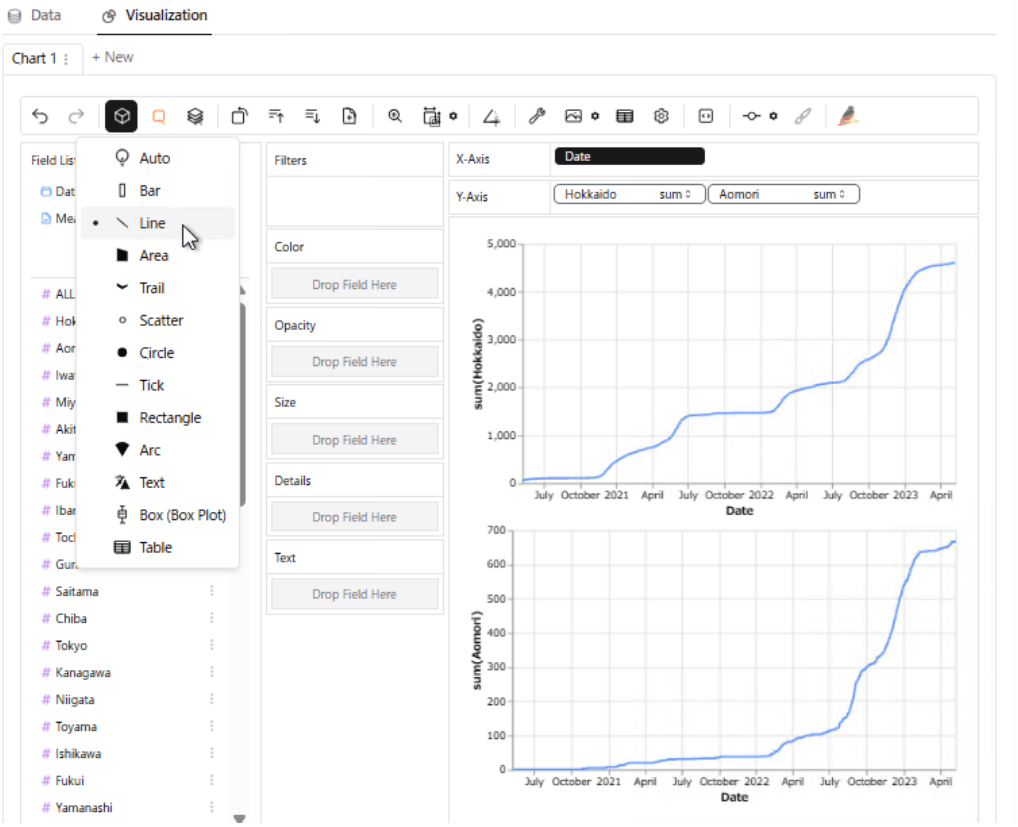
地図表示



AI で BI（Streamlit、Langchain、PandasAI、PyGWalker）

その他の機能ボタン





以上

