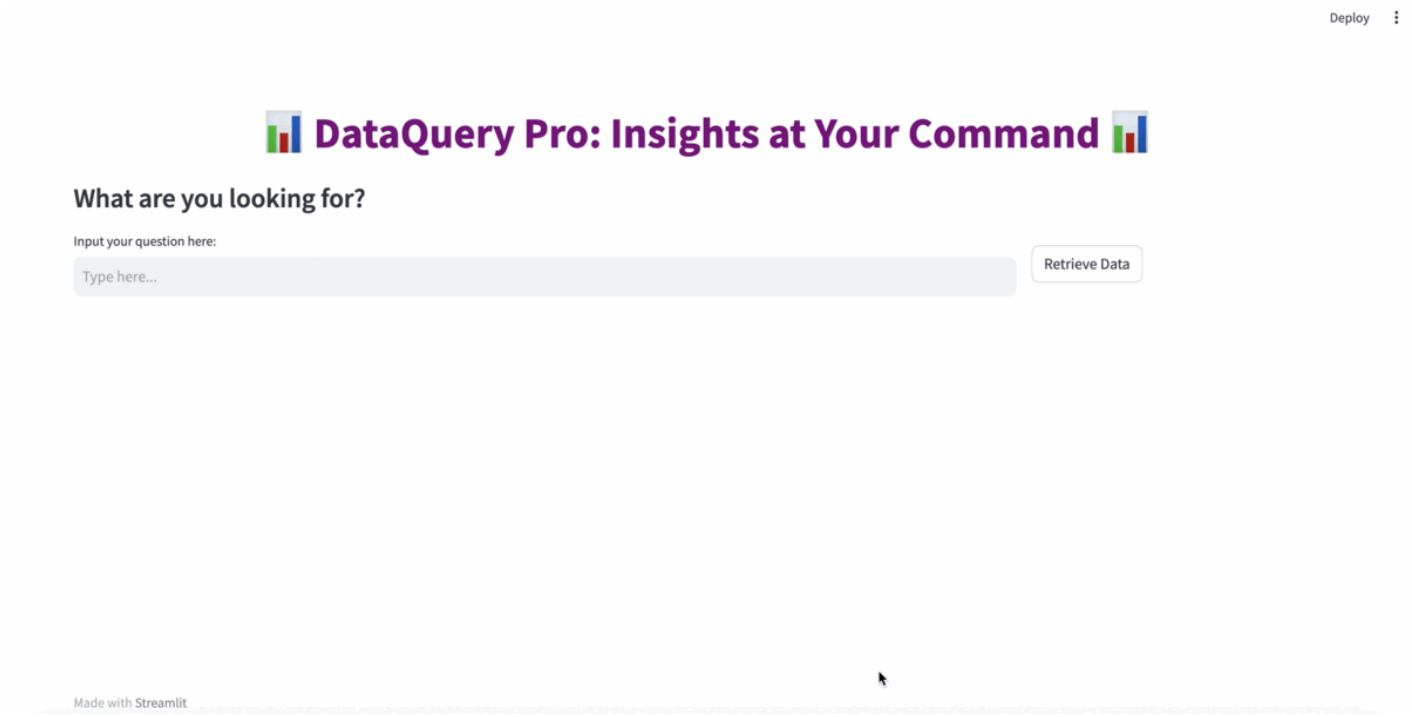
Text-to-Data Retrieval and Insightful Chart Generation

This Jupyter Notebook guides you through the implementation of an application designed to convert natural language questions into SQL queries, retrieve data from a database, and automatically generate insightful charts based on the retrieved data. This tool is aimed at making data analysis accessible to both technical and non-technical users, streamlining the process of data retrieval and visualization. ¶



Setup and Configuration

In [ ]:
```
pip install streamlit google-generativeai-sdk langchain sqlite3 pandas plotly
```

Now, let's set up our environment:

Replace 'YOUR_API_KEY_HERE' with your actual Google API key and /path/to/your/xx.db with the path to your SQLite database file.

In [ ]:
```
import streamlit as st
import google.generativeai as genai
import langchain
import sqlite3
import pandas as pd
import plotly.express as px
```

```
# Google API Configuration
GOOGLE_API_KEY = 'YOUR_API_KEY_HERE'
genai.configure(api_key=GOOGLE_API_KEY)

# Database Path Configuration
database_path = '/path/to/your/xx.db'
```

Query Generation Function

This function uses Google's generative AI to convert English prompts into SQL queries.

In [ ]:
```
def get_gemini_response(question, prompt):
    model = genai.GenerativeModel('gemini-pro')
    response = model.generate_content([prompt[0], question])
    return response.text
```

SQL Query Execution Function

Here's how we execute the SQL query and return the results as a pandas DataFrame.

In [ ]:
```
def read_sql_query(sql, db):
    conn = sqlite3.connect(db)
    df = pd.read_sql_query(sql, conn)
    conn.close()
    return df
```

Extracting SQL Queries from Responses

This function parses the AI model's response to extract the SQL query.

In [ ]:
```
def get_sql_query_from_response(response):
    try:
        query_start = response.index('SELECT')
        query_end = response.index(';') + 1
        return response[query_start:query_end]
    except ValueError:
        st.error("Could not extract SQL query from the response.")
        return None
```

Determining Chart Types

Based on the structure and data type of the DataFrame, this function decides the most suitable type of chart to generate.

In [ ]:
```
def determine_chart_type(df):
    if len(df.columns) == 2:
        if df.dtypes[1] in ['int64', 'float64'] and len(df) > 1:
            return 'bar'
        elif df.dtypes[1] in ['int64', 'float64'] and len(df) <= 10:
            return 'pie'
    elif len(df.columns) >= 3 and df.dtypes[1] in ['int64', 'float64']:
        return 'line'
    return None
```

Generating Charts

This function generates and displays a chart based on the determined chart type and the data provided.

In [ ]:
```
def generate_chart(df, chart_type):
    if chart_type == 'bar':
        fig = px.bar(df, x=df.columns[0], y=df.columns[1], title=f"{df.colu
mns[0]} vs. {df.columns[1]}",
                     template="plotly_white", color=df.columns[0])
    elif chart_type == 'pie':
        fig = px.pie(df, names=df.columns[0], values=df.columns[1], title=
f"Distribution of {df.columns[0]}",
                     template="plotly_white")
    elif chart_type == 'line':
        fig = px.line(df, x=df.columns[0], y=df.columns[1], title=f"{df.col
umns[1]} Over {df.columns[0]}",
                      template="plotly_white", markers=True)
    else:
        st.write("No suitable chart type determined for this data.")
        return

    fig.update_layout(plot_bgcolor="rgba(0,0,0,0)")
    st.plotly_chart(fig, use_container_width=True)
```

Streamlit Web Interface

We'll set up the Streamlit web interface to allow users to input their questions and see the results.

In [ ]:
```
st.set_page_config(page_title="SQL Query Retrieval App", layout="wide")
st.markdown("""
    <h1 style="color: purple; text-align: center;">
        📊 DataQuery Pro: Insights at Your Command 📊
    </h1>
    """, unsafe_allow_html=True)

with st.container():
    st.subheader("What are you looking for?")
    col1, col2 = st.columns([3, 1], gap="small")
    with col1:
        question = st.text_input("Input your question here:", key="input",
placeholder="Type here...")
    with col2:
        submit = st.button("Retrieve Data", help="Click to submit your ques
tion.")
```

Handling User Input and Displaying Results

Upon receiving a question from the user, the app generates and executes the SQL query, then visualizes the results.

In [ ]:
```
if submit and question:
    response = get_gemini_response(question, prompt)
    sql_query = get_sql_query_from_response(response)
    if sql_query:
        st.code(sql_query, language='sql')
        df = read_sql_query(sql_query, database_path)
        if not df.empty:
            col_data, col_chart = st.columns(2)
            with col_data:
                st.subheader("Query Results:")
                st.dataframe(df)
            chart_type = determine_chart_type(df)
            if chart_type:
                with col_chart:
                    st.subheader("Visualization:")
                    generate_chart(df, chart_type)
        else:
            st.write("No results found for the given query.")
```

Conclusion

This notebook guides you through creating an application that simplifies data retrieval and visualization from the Job_Postings.db database. By leveraging advanced AI models and interactive web and visualization technologies, users can gain insights without deep SQL knowledge. Experiment with different questions and explore various datasets to extend the application's capabilities.