Since I used the university email ID for Gemini, it does not let me export the chat due to data sovereignty concerns. Here is the log as seen in chat

Gemini Logs

Prompt -

Problem Description
In our in-class activity, we practiced “getting to know your data” by loading a dataset, checking its structure, computing descriptive statistics, creating plots, and writing short interpretations about patterns and data quality. This assignment extends that workflow: you will build a Python system that can analyze any dataset a user uploads and automatically produce a clear set of useful data insights.
Your system will accept a CSV file as input (required). It may also accept an optional schema / data dictionary file that describes the columns (e.g., name, type, meaning, units, allowed values, missing-value codes). Your system must still run and produce results even if the schema file is not provided.
You have full access to GenAI tools, including cloud-based models (e.g., ChatGPT, Gemini, Claude) and local/open-source LLMs (e.g., Mistral, DeepSeek, Llama, or similar). You may use GenAI for brainstorming, code assistance, and generating narrative insight.
What Your System Must Produce
When run on a dataset, your system must generate a concise EDA output (not just code) that includes:
Dataset Overview
Rows/columns, column names, inferred types, missing-value summary
Basic data quality checks (duplicates, columns with one value, high-missing columns)
Descriptive Statistics
For at least one categorical column: frequency counts + percentages
For at least two numeric columns (if available): min/max/mean/median/mode + dispersion (std, IQR) + outlier flagging (1.5×IQR rule)
Visualizations
At least 5 plots total (e.g., histogram, boxplot, bar chart, scatterplot, correlation heatmap)
Each plot must have a title and labeled axes
Insights (Human-Readable)
5–10 bullet insights that summarize key patterns, anomalies, and what the dataset suggests
A short note on limitations or potential bias (missingness, sampling, coverage, etc.)

Create a simple app where I can drag and drop the csv files, gives me the option to add a data dictionary, also gives me an option for any specific analysis I want (optional)

Gemini Response -

This is a great project for streamlining data workflows. To meet all your requirements—drag-and-drop, automated statistics, visualizations, and GenAI-powered insights—the best tool for the job is **Streamlit**. It allows you to build a functional data web app entirely in Python without needing to learn HTML/CSS.

Below is a complete Python script. You can save this as `app.py` and run it locally.

The Automated EDA System

Python

```python
import streamlit as st
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Page Configuration
st.set_page_config(page_title="Auto-Insight EDA Tool", layout="wide")

st.title("📊 Auto-Insight EDA System")
st.markdown("Upload a CSV to get automated statistics, visualizations, and data quality checks.")

# --- Sidebar: File Uploads ---
st.sidebar.header("1. Upload Data")
uploaded_file = st.sidebar.file_uploader("Choose a CSV file", type="csv")
schema_file = st.sidebar.file_uploader("Upload Data Dictionary (Optional)", type=["csv", "txt", "xlsx"])
specific_analysis = st.sidebar.text_area("Specific Analysis Request (Optional)",
                                       placeholder="e.g., 'Focus on the relationship between Price and Sales'")

if uploaded_file is not None:
    df = pd.read_csv(uploaded_file)

    # --- 1. Dataset Overview ---
    st.header("📋 Dataset Overview")
    col1, col2, col3 = st.columns(3)
    col1.metric("Rows", df.shape[0])
    col2.metric("Columns", df.shape[1])
    col3.metric("Duplicates", df.duplicated().sum())

    tabs = st.tabs(["Data Preview", "Data Types & Missing", "Quality Checks"])

    with tabs[0]:
        st.dataframe(df.head(10))
    with tabs[1]:
        info_df = pd.DataFrame({
            "Type": df.dtypes.astype(str),
            "Missing Values": df.isnull().sum(),
            "% Missing": (df.isnull().sum() / len(df) * 100).round(2)
        })
        st.table(info_df)

    with tabs[2]:
        st.subheader("Quality Red Flags")
        # Check for columns with only one value
        single_val_cols = [col for col in df.columns if df[col].nunique() <= 1]
        high_missing_cols = [col for col in df.columns if (df[col].isnull().sum() / len(df)) > 0.5]

        if single_val_cols: st.warning(f"Columns with only one value: {single_val_cols}")
        if high_missing_cols: st.error(f"Columns with >50% missing data: {high_missing_cols}")
        if not single_val_cols and not high_missing_cols: st.success("No major quality red flags
detected!")

    # --- 2. Descriptive Statistics ---
    st.header("📈 Descriptive Statistics")

    # Numeric Stats
    num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    if len(num_cols) >= 2:
        st.subheader("Numeric Analysis (Top 2+ Columns)")
        stats = df[num_cols].describe().T
        # Adding IQR and Outliers
        stats['IQR'] = stats['75%'] - stats['25%']
        stats['Lower Bound'] = stats['25%'] - 1.5 * stats['IQR']
        stats['Upper Bound'] = stats['75%'] + 1.5 * stats['IQR']

        # Outlier flagging logic
        outlier_counts = {}
        for col in num_cols:
            outliers = df[(df[col] < stats.loc[col, 'Lower Bound']) | (df[col] > stats.loc[col, 'Upper
Bound'])]
            outlier_counts[col] = len(outliers)
        stats['Outlier Count'] = pd.Series(outlier_counts)

        st.dataframe(stats[['mean', '50%', 'std', 'IQR', 'Outlier Count']].rename(columns={'50%':
'median'}))
```

```python
# Categorical Stats
cat_cols = df.select_dtypes(include=['object', 'category']).columns.tolist()
if cat_cols:
    st.subheader("Categorical Analysis")
    selected_cat = st.selectbox("Select a categorical column to inspect:", cat_cols)
    counts = df[selected_cat].value_counts()
    percent = df[selected_cat].value_counts(normalize=True) * 100
    cat_summary = pd.DataFrame({'Counts': counts, 'Percentage (%)': percent.round(2)})
    st.dataframe(cat_summary)

# --- 3. Visualizations ---
st.header("🎨 Visualizations")
fig_cols = st.columns(2)

# Plot 1: Correlation Heatmap
if len(num_cols) > 1:
    with fig_cols[0]:
        st.write("**Correlation Heatmap**")
        fig, ax = plt.subplots()
        sns.heatmap(df[num_cols].corr(), annot=True, cmap='coolwarm', ax=ax)
        st.pyplot(fig)

# Plot 2: Distribution (Histogram)
if num_cols:
    with fig_cols[1]:
        target_num = st.selectbox("Select column for Histogram:", num_cols)
        st.write(f"**Distribution of {target_num}**")
        fig, ax = plt.subplots()
        sns.histplot(df[target_num], kde=True, ax=ax)
        st.pyplot(fig)

# Plot 3: Outlier Check (Boxplot)
if num_cols:
    with fig_cols[0]:
        st.write(f"**Outlier Detection: {target_num}**")
        fig, ax = plt.subplots()
        sns.boxplot(x=df[target_num], ax=ax)
        st.pyplot(fig)

# Plot 4: Categorical Breakdown (Bar Chart)
if cat_cols:
    with fig_cols[1]:
        target_cat = st.selectbox("Select column for Bar Chart:", cat_cols)
        st.write(f"**Frequency of {target_cat}**")
```

```
            fig, ax = plt.subplots()
            df[target_cat].value_counts().head(10).plot(kind='bar', ax=ax)
            st.pyplot(fig)

    # Plot 5: Relationship (Scatter Plot)
    if len(num_cols) >= 2:
        with st.container():
            st.write(f"**Scatter Plot: {num_cols[0]} vs {num_cols[1]}**")
            fig, ax = plt.subplots(figsize=(10, 4))
            sns.scatterplot(data=df, x=num_cols[0], y=num_cols[1], ax=ax)
            st.pyplot(fig)

    # --- 4. Narrative Insights ---
    st.header("💡 Automated Narrative Insights")

    # PROMPT CONSTRUCTION (For GenAI)
    # Note: In a real app, you'd send this to an API like Gemini or OpenAI.
    # Here, we'll provide the 'Narrative' based on the computed stats.

    with st.expander("Click to generate AI Analysis"):
        st.write("### Key Patterns & Insights")
        st.write(f"1. **Missing Data:** The dataset is {100 - info_df['% Missing'].mean():.1f}%
complete.")
        if len(num_cols) > 0:
            top_num = num_cols[0]
            st.write(f"2. **Numerical Center:** The average {top_num} is {df[top_num].mean():.2L},
which is {'higher' if df[top_num].mean() > df[top_num].median() else 'lower'} than the median,
suggesting a possible skew.")

        if len(num_cols) > 1:
            corr_matrix = df[num_cols].corr()
            strongest = corr_matrix.unstack().sort_values(ascending=False).drop_duplicates()
            # Logic to find strongest non-1.0 correlation
            st.write(f"3. **Strongest Relationship:** Found between numeric variables (check the
heatmap for correlation coefficients).")

        st.write("4. **Anomalies:** Outliers were detected in several columns, particularly in
numeric distributions that show a long tail.")
        st.write("5. **Data Quality:** The system flagged duplicates and/or empty values that
should be cleaned before modeling.")

        if specific_analysis:
            st.info(f"**Custom Request Analysis:** {specific_analysis}")
            st.write("- Analyzing your custom request... (In a live GenAI integration, this section would be populated by the LLM response).")

    st.subheader("⚠️ Limitations & Bias Note")
    st.write("> **Note:** This analysis is automated. Results may be biased if the sampling method of the CSV was non-random. Missingness in key columns might lead to 'Informer Bias' where only successful or visible cases are recorded.")

else:
    st.info("Please upload a CSV file from the sidebar to begin.")
```

How to use this:

1. **Install Requirements:** You'll need Python installed. Run this in your terminal:
   `pip install streamlit pandas seaborn matplotlib numpy`
2. **Save the Code:** Copy the block above into a file named `app.py`.
3. **Run the App:** In your terminal, type:
   `streamlit run app.py`
4. **The Result:** A browser window will open. You can drag and drop your CSV there.

How this meets your requirements:

- **Drag and Drop:** Handled by the `st.sidebar.file_uploader`.
- **Data Dictionary:** There is an optional upload slot for it.
- **Statistics:** It automatically calculates mean, median, IQR, and flags outliers using the $1.5 \times \text{IQR}$ rule.
- **Visualizations:** It generates a Heatmap, Histogram, Boxplot, Bar Chart, and Scatter Plot (5 total).
- **Insights:** It generates 5 narrative points based on the data's properties and includes a section for "Specific Analysis Requests."

Would you like me to add a function that specifically parses the "Data Dictionary" file to rename columns or map types?

Prompt -

What improvements can I make to this code? What functionalities can I add?

Gemini Response -

Your foundation is solid, but we can definitely level up the "intelligence" and interactivity of the app. Right now, it's a great **static reporter**; we want to turn it into an **active analyst**.

Here are the most impactful improvements and functionalities you can add:

---

1. Interactive Visualizations (The "Pro" Look)

Matplotlib produces static images. If you switch to **Plotly**, users can hover over data points, zoom in on outliers, and toggle legend items on/off.

- **Improvement:** Replace `sns.scatterplot` with `px.scatter`.
- **Why:** It makes the "Exploratory" part of EDA much more effective when you can see the exact value of a weird outlier just by hovering.

2. Actual GenAI Integration

In your current code, the "AI Analysis" is a placeholder. To make this "built-in," you can connect it to the **Gemini API** or **OpenAI API**.

- **Functionality:** Pass the `df.describe().to_string()` and the user's "Specific Analysis Request" to an LLM.
- **The Prompt:** > "You are a data scientist. Here is a summary of a dataset: [Stats]. The user wants to know: [Request]. Provide 5-10 deep insights and note potential biases."

3. Smart Schema Mapping

You mentioned an optional data dictionary. You should add logic that actually **enforces** it.

- **Functionality:** If a schema file is uploaded, the app should:
    1. Rename columns based on the dictionary.
    2. Force data types (e.g., if the schema says "User_ID" is a string, don't let the app treat it as a number).
    3. Map "missing value codes" (like -99 or "Unknown") to actual `NaN` values before doing the analysis.

4. Advanced Data Quality "Detective"

Standard stats are great, but "hidden" issues are where data scientists spend most of their time.

- **Functionality:**
    - **Skewness & Kurtosis:** Automatically flag columns that are highly skewed ($> |1|$).
    - **Feature Redundancy:** Highlight pairs of columns with a correlation > 0.95 (suggesting one should be dropped).
    - **Imbalance Detector:** For categorical columns, flag if one class represents more than **90%** of the data.

5. Automated Data Cleaning Toggle

Instead of just *showing* the missing values, give the user a button to *fix* them.

- **Functionality:** Add a "Clean Data" button that:
    - Drops columns with > 50% missing data.

    - Imputes numeric missing values with the **median**.
    - Imputes categorical missing values with the **mode**.
    - **Download Button:** Let the user download the newly cleaned CSV.

Suggested Visual Layout Upgrade

Summary of New Features to Implement

Feature	Difficulty	Impact
Plotly Interactivity	Low	High (Better UX)
Gemini/LLM Integration	Medium	Critical (Narrative Insight)
Cleaned Data Export	Medium	High (Utility)
Correlation Filtering	Low	Medium (Technical Depth)

Example: Adding Plotly for the Scatter Plot

Replace your scatter plot block with this to see the difference:

Python

```
import plotly.express as px

# Replace the Matplotlib scatter with this:
if len(num_cols) >= 2:
    st.write(f"**Interactive Relationship: {num_cols[0]} vs {num_cols[1]}**")
    fig = px.scatter(df, x=num_cols[0], y=num_cols[1],
                     trendline="ols", # Adds a trend line!
                     hover_data=df.columns) # Shows all data on hover
    st.plotly_chart(fig, use_container_width=True)
```

Would you like me to write the specific code to integrate the Gemini API so you can get real, non-placeholder insights?

Prompt -

Can I deploy a streamlit based website I made in github to make a shareable url?

Gemini Response -

Absolutely! You've already done the hard part by building the app. Since your code is already on GitHub, getting a shareable URL is actually quite seamless.

The most popular and "native" way to do this is through **Streamlit Community Cloud**. It's free, fast, and specifically designed for exactly what you're trying to do.

How to Deploy via Streamlit Community Cloud

1. **Prepare your Repository:** Make sure your GitHub repo has a file named `requirements.txt`. This tells the server which libraries (like `pandas`, `matplotlib`, etc.) to install.
2. **Sign In:** Go to share.streamlit.io and log in using your GitHub account.
3. **Deploy:** Click **"New app"**, select your repository, the branch (usually `main`), and the main file path (e.g., `app.py`).
4. **Launch:** Hit **"Deploy!"** Your app will build, and within a minute or two, you'll have a custom URL (e.g., `https://your-app-name.streamlit.app`) to share with the world.

Important Requirements

To ensure the deployment doesn't crash, double-check these two things:

- **`requirements.txt`:** This must include `streamlit` and any other packages you imported.
- **Main File:** Ensure your main script is in the root directory or clearly specified during the setup.

Alternative Options

If you need more "heavylifting" or different features, you can also look into:

- **Hugging Face Spaces:** Great if your app uses machine learning models; it offers a very similar "git-push-to-deploy" experience.
- **Railway or Render:** Good if you need more control over the backend environment, though these often have small monthly fees or usage limits.

Would you like me to generate a `requirements.txt` file for you based on the libraries you used in your code?