- **Data Analytic Tool**

Documentation Report:

1) Importing Libraries:

- **Code:**

```
import streamlit as st
import pandas as pd
from core import file_handler, preprocessing, visualization
```

- **streamlit:** Library for creating interactive web apps easily for data analysis, without HTML/CSS.
- **pandas:** Powerful library to handle data (CSV, Excel, JSON), clean it, and perform analysis.
- **core module:** A folder or module containing three files (file_handler.py, preprocessing.py, visualization.py):
  - **file_handler:** reading files of various types.
  - **preprocessing:** cleaning data, type conversions, handling missing values.
  - **visualization:** creating charts and visualizations.

Importance: Sets up the tools needed for data handling and UI.

2) Page Setup:

- Code:

```
st.set_page_config(page_title="Data Cleaning Tool", layout="wide")
st.title("🧼 Internal Data Cleaning & Analysis Tool")
st.markdown("Upload a dataset to clean, visualize, and export it with zero code.")
```

- **st.set_page_config:** Page settings like title and layout (wide expands content across full screen).
- **st.title:** Main title of the app.
- **st.markdown:** Explanatory text showing the purpose of the tool.

Importance: Provides a clear and user-friendly interface.

---

3) File Upload:

- Code:

```
uploaded_file = st.file_uploader("📁 Upload your data file", type=["csv", "xlsx", "json", "db"])
df = None
cleaned_df = None
```

- **st.file_uploader:** UI component to upload files; accepts csv, xlsx, json, or db.
- **df and cleaned_df:** Variables to store original and cleaned datasets.

Importance: Allows the user to input their dataset.

---

4) Read the File by Type:

- Code:

```
if uploaded_file:
    ext = uploaded_file.name.split('.')[-1]
    try:
        if ext == "csv":
            df = file_handler.read_csv(uploaded_file)
        elif ext == "xlsx":
            df = file_handler.read_excel(uploaded_file)
        elif ext == "json":
            df = file_handler.read_json(uploaded_file)
        elif ext == "db":
            table_name = st.text_input("Enter table name from SQLite DB")
            if table_name:
                df = file_handler.read_sqlite(uploaded_file, table_name)
    except Exception as e:
        st.error(f"❌ Failed to read file: {e}")
        st.stop()
```

- **Determine file type** using its extension.
- **Read file** using file_handler functions for each type.
- **SQLite DB:** prompts for a table name if a database is uploaded.
- **try/except:** catches read errors and displays messages to the user.

Importance: Ensures data is loaded correctly and handles errors gracefully.

---

5) Preview the Data:

- Code:

```
st.subheader("📊 Data Preview")
st.dataframe(df.head(30), use_container_width=True)
```

- **st.subheader:** Section subtitle.
- **st.dataframe:** Shows the first 30 rows of the dataset in a table.

Importance: Lets the user see a preview of their data before cleaning.

6) Descriptive Statistics:

- Code:

```
st.subheader("📈 Descriptive Statistics")
with st.expander("👉 Show summary"):
  st.markdown("### 📐 Numeric Columns:")
  st.dataframe(df.describe().T)

  st.markdown("### 📋 Top Categorical Values:")
  for col in df.select_dtypes(include="object").columns:
    st.markdown(f"**{col}**")
    st.dataframe(df[col].value_counts().head(5).to_frame("Count"))
```

- **df.describe():** Provides statistics for numeric columns (mean, std, min, max, etc.).
- **Top values for categorical columns:** Uses value_counts() to show most common values.
- **st.expander:** Hides the details until user expands it.

Importance: Gives a quick overview of the dataset distribution.

---

7) Missing Values Summary:

- Code:

```
st.subheader("🕳️ Missing Value Summary")
st.dataframe(preprocessing.check_missing_values(df,
percent=True).to_frame("Missing (%)"))

with st.expander("🗺️ Missing Values Heatmap"):
  visualization.plot_missing_heatmap(df)
```

- **check_missing_values:** Computes missing values and their percentages.
- **plot_missing_heatmap:** Creates a heatmap showing locations of missing values.

Importance: Helps users identify which columns require cleaning.

---

8) Convert Column Data Type:

- Code:

```
st.subheader("🔁 Convert Column Type")
col = st.selectbox("Select column to convert", df.columns)
new_type = st.selectbox("Convert to", ["str", "float", "int", "bool"])

try:
    df = preprocessing.convert_data_type(df, col, new_type)
    st.success(f"Column '{col}' converted to {new_type}")
except Exception as e:
    st.error(f"Conversion failed: {e}")
```

- **Select column** and **new type**.
- **convert_data_type**: Converts column type safely, handling errors.

Importance: Corrects data types before analysis or cleaning.

---

9) Time-Series Cleaning:

- Code:

```
st.subheader("⏳ Time-Series Cleaning")
time_col = st.selectbox("Select date column for forward fill", ["None"] +
df.columns.tolist())

if time_col != "None":
    with st.expander("⚙️ Optional: Set date format"):
        fmt = st.text_input("Format (e.g. %Y-%m-%d)", "")
    if fmt:
        df[time_col] = pd.to_datetime(df[time_col], format=fmt, errors="coerce")
    else:
        df[time_col] = pd.to_datetime(df[time_col], errors="coerce")
    df = df.sort_values(by=time_col)
    df = df.ffill()
    st.success("Forward fill applied.")
```

- **Convert column to datetime** using pd.to_datetime.
- **sort_values + ffill():** Sort data chronologically and fill missing values with previous row values.

Importance: Prepares time-series data for analysis.

10) Handling Missing Values:

- Code:

```
df_before = df.copy()
st.header("🧼 Clean Missing Values")
strategy = st.selectbox("Choose strategy", ["drop", "mean", "median", "mode",
"constant"])
const_val = None
if strategy == "constant":
    val_input = st.text_input("Constant fill value:")
    if val_input.replace('.', '', 1).isdigit():
        const_val = float(val_input) if '.' in val_input else int(val_input)
    else:
        const_val = val_input
```

- **Copy the dataset before cleaning** to compare results.
- **Choose missing value strategy:**
  - **drop:** remove rows with missing values.
  - **mean/median/mode:** fill missing values with average, median, or mode.
  - **constant:** fill with user-defined value.

Importance: Gives flexibility to handle missing values appropriately.

11) Execute Cleaning & Show Results:

- Code:

```
if st.button("🚀 Clean Now"):
    cleaned_df = preprocessing.process_missing_values(df, strategy=strategy,
fill_value=const_val)

    st.success("✅ Cleaning completed")

    col1, col2 = st.columns(2)
    with col1:
        st.markdown("🔍 **Before**")
        st.dataframe(df_before.head(30))
    with col2:
        st.markdown("✅ **After**")
        st.dataframe(cleaned_df.head(30))
```

- **Button triggers the cleaning process.**
- **Compare before and after cleaning** side by side using columns.

Importance: Lets the user see the effect of the cleaning immediately.

---

12) Download Cleaned Data:

- Code:

```
st.subheader("💾 Download")
csv = cleaned_df.to_csv(index=False).encode("utf-8")
st.download_button("⬇ Download CSV", data=csv, file_name="cleaned_data.csv",
mime="text/csv")
```

- **Convert cleaned data to CSV.**
- **st.download_button:** Allows user to download cleaned data locally.

Importance: Enables the user to use the cleaned dataset outside the app.