update the code for support of .txt, .json and excel files

```
import os, sys
project_root = os.path.abspath(os.path.join(__file__,
"../../.."))
sys.path.insert(0, project_root)  # add repo entrypoint to
python path
import pandas as pd
import sqlite3
import re
import unicodedata
import json
from pathlib import Path
import src.utils.data_cleaning as dc
import src.configs.config as cfg

# Config
DEFAULT_DB_PATH = Path("data/ingested.db")
DEFAULT_TABLE_NAME = "data_table"

def persist_to_sqlite(
    df: pd.DataFrame,
    meta_df: pd.DataFrame,
    db_path: Path,
    table_name: str
    ):
    """ Save pandas dataframe to a sqlite database along
with its metadata.

    Args:
        df (pd.DataFrame): Dataframe to be saved.
        meta_df (pd.DataFrame): Metadata dataframe to be
saved.
        db_path (Path): Path to the sqlite database file.
        table_name (str): Name of the table to save the
dataframe.
    """
    conn = sqlite3.connect(db_path) # Create a connection
to the SQLite database
    df.to_sql(table_name, conn, if_exists="replace",
index=False) # Save main dataframe
    meta_df.to_sql(f"{table_name}__meta", conn,
if_exists="replace", index=False)   # Save metadata
dataframe
    conn.commit()   # Commit the changes
    conn.close()    # Close the connection

# Entry point
def ingest_csv_sqlite(
    csv_path: str,
```

```
    db_path: str = str(DEFAULT_DB_PATH),
    table_name: str = DEFAULT_TABLE_NAME,
    ):
    """ Ingest a CSV file into a SQLite database with data
cleaning and metadata generation.

    Args:
        csv_path (str): Path of the CSV file to ingest.
        db_path (str, optional): Path of database to save.
Defaults to str(DEFAULT_DB_PATH).
        table_name (str, optional): Name of table in DB.
Defaults to DEFAULT_TABLE_NAME.

    Returns:
        tuple: A tuple containing two pandas DataFrames:
        - clean_df (pd.DataFrame): The cleaned version of the
original CSV data
        - meta_df (pd.DataFrame): Metadata about the
cleaned data including column names, data types, null
ratios, and sample values
    """
    csv_path = Path(csv_path)
    df = pd.read_csv(csv_path)

    clean_df = dc.clean_dataframe(df)
    meta_df = dc.generate_metadata(clean_df)

    persist_to_sqlite(
        clean_df,
        meta_df,
        Path(db_path),
        table_name
    )

    print(f"[OK] Ingested {csv_path.name}")
    print(f"Table: {table_name}")
    print(f"Rows: {len(clean_df)}")
    print("Columns:", list(clean_df.columns))

    return clean_df, meta_df


if __name__ == "__main__":

    ingest_csv_sqlite(
        csv_path="data/in/financials.csv",
        db_path="data/ingested.db",
        table_name="financials"
    )
```

Below is a **clean, production-ready update** that adds support for:

- `.csv`
- `.txt` (delimited text)
- `.json`
- **Excel** (`.xls`, `.xlsx`)

Key ideas:

- Use **file extension–based dispatch**
- Keep **one ingestion entry point**
- Preserve your existing **cleaning + metadata + SQLite persistence pipeline**

Updated Code (Drop-in Replacement)

```
import os, sys from pathlib import Path import pandas as pd import sqlite3 import json project_root = os.path.abspath(os.path.join(__file__, "../../..")) sys.path.insert(0, project_root) import src.utils.data_cleaning as dc import src.configs.config as cfg # ==================== # Config # ==================== DEFAULT_DB_PATH = Path("data/ingested.db") DEFAULT_TABLE_NAME = "data_table" SUPPORTED_EXTENSIONS = {".csv", ".txt", ".json", ".xls", ".xlsx"} # ==================== # SQLite Persistence # ==================== def persist_to_sqlite( df: pd.DataFrame, meta_df: pd.DataFrame, db_path: Path, table_name: str ): """Save pandas dataframe and metadata to SQLite.""" conn = sqlite3.connect(db_path) df.to_sql(table_name, conn, if_exists="replace", index=False) meta_df.to_sql(f"{table_name}__meta", conn, if_exists="replace", index=False) conn.commit() conn.close() # ==================== # File Readers # ==================== def read_csv_or_txt(path: Path) -> pd.DataFrame: """ Read CSV or TXT file. TXT is assumed to be delimiter-based (comma, tab, or pipe auto-detected). """ return pd.read_csv(path, sep=None, engine="python") def read_json(path: Path) -> pd.DataFrame: """ Read JSON file. Supports: - list of dicts - dict of lists """ with open(path, "r", encoding="utf-8") as f: data = json.load(f) return pd.DataFrame(data) def read_excel(path: Path) -> pd.DataFrame: """ Read Excel file. Defaults to first sheet. """ return pd.read_excel(path) def load_dataframe(file_path: Path) -> pd.DataFrame: """ Dispatch loader based on file extension. """ ext = file_path.suffix.lower() if ext == ".csv": return read_csv_or_txt(file_path) elif ext == ".txt": return read_csv_or_txt(file_path) elif ext == ".json": return read_json(file_path) elif ext in {".xls", ".xlsx"}: return read_excel(file_path) else: raise ValueError( f"Unsupported file type: {ext}. " f"Supported types: {sorted(SUPPORTED_EXTENSIONS)}" ) # ==================== # Entry Point # ==================== def ingest_file_sqlite( file_path: str, db_path: str = str(DEFAULT_DB_PATH), table_name: str = DEFAULT_TABLE_NAME, ): """ Ingest CSV, TXT, JSON, or Excel file into SQLite with cleaning + metadata. Args: file_path (str): Path to input file db_path (str): SQLite DB path table_name (str): SQLite table name Returns: clean_df, meta_df """ file_path = Path(file_path) if not file_path.exists(): raise FileNotFoundError(f"File not found: {file_path}") if file_path.suffix.lower() not in SUPPORTED_EXTENSIONS: raise ValueError( f"Unsupported file type: {file_path.suffix}. " f"Supported types: {sorted(SUPPORTED_EXTENSIONS)}" ) # Load df = load_dataframe(file_path) # Clean + metadata clean_df = dc.clean_dataframe(df) meta_df = dc.generate_metadata(clean_df) # Persist persist_to_sqlite( clean_df, meta_df, Path(db_path), table_name ) print(f"[OK] Ingested {file_path.name}") print(f"Table: {table_name}") print(f"Rows: {len(clean_df)}") print("Columns:", list(clean_df.columns)) return clean_df, meta_df #
```

```
==================== # CLI / Script Usage # ==================== if __name__
== "__main__": ingest_file_sqlite( file_path="data/in/financials.xlsx", # csv
| txt | json | xls | xlsx db_path="data/ingested.db", table_name="financials"
)
```

What This Gives You

- ✅ **Single ingestion API** for all file types
- ✅ Automatic delimiter detection for `.txt`
- ✅ Robust JSON loading (list or dict formats)
- ✅ Excel support without extra config
- ✅ Backward-compatible with your cleaning + metadata logic