Documentation: Combine Multiple Excel Sheets into One

Overview

This program reads multiple sheets from an Excel file, merges them into a single dataset, and saves the result as a new Excel file. It is useful for consolidating data spread across different sheets into one master file for further analysis.

Steps Performed

1. Import Libraries
   - pandas is used for handling Excel files and DataFrames.

2. Load the Excel File
   - The file path of the source Excel file is defined.
   - Individual sheets are read into separate DataFrames using pd.read_excel.

3. Combine Sheets
   - The two DataFrames are concatenated using pd.concat.
   - ignore_index=True ensures the final DataFrame has a continuous index.

4. Save to Excel
   - The combined DataFrame is exported to a new Excel file using to_excel.
   - The index is set to False to avoid writing row numbers into the file.

5. Confirmation Message
   - A message is printed to confirm the successful save.

Code

```
import pandas as pd  # Import the pandas library for data handling

# Step 1: Define the file path of the input Excel file

file_path = "ConsolidatedIndianFinancialSystemCode.xlsx"



# Step 2: Read sheets into DataFrames

# sheet_name=0 means the first sheet, sheet_name=1 means the second sheet

sheet1 = pd.read_excel(file_path, sheet_name=0)

sheet2 = pd.read_excel(file_path, sheet_name=1)



# Step 3: Combine both sheets into a single DataFrame

combined_df = pd.concat([sheet1, sheet2], ignore_index=True)



# Step 4: Save the combined DataFrame into a new Excel file

output_file = "Combined_IFSC1.xlsx"

combined_df.to_excel(output_file, index=False)
```

```
# Step 5: Print success message

print(f"✅ Combined file saved as {output_file}")
```

Output

The combined file will be saved in the working directory as:

Combined_IFSC1.xlsx

Bank IFSC Data Cleaning & Standardization – Documentation

Overview

This project focuses on cleaning and standardizing Indian Financial System Code (IFSC) data extracted from a consolidated dataset (Combined_IFSC1.csv). The dataset includes information such as bank names, IFSC codes, branch names, addresses, city details, states, STD codes, and phone numbers.

The pipeline applies systematic data validation, deduplication, and enrichment, ensuring data quality before storage or further analysis.

Requirements

- Python 3.x
- Libraries:
  - pandas – Data manipulation
  - numpy – Handling missing values
  - sqlalchemy – Database connections
  - openpyxl – Excel handling
  - re – Regex for IFSC validation

Step-by-Step Data Processing

1. Load CSV Data

```
import pandas as pd

import re


from sqlalchemy import create_engine


import os
```

```
file_path = "Combined_IFSC1.csv"

df = pd.read_csv(file_path)
```

2. Standardize Column Names

```
df.columns = df.columns.str.strip().str.lower()

df = df.rename(columns={

    "bank": "bank",

    "ifsc": "ifsc",

    "branch": "branch",

    "address": "address",

    "city1": "city1",

    "city2": "city2",

    "state": "state",

    "std code": "stdCode",

    "phone": "phone"

})
```

3. Clean Text Fields

- Strip extra spaces
- Replace blanks with NaN

```
import numpy as np

for col in ["bank", "branch", "address", "city1", "city2", "state"]:

    df[col] = df[col].astype(str).str.strip().replace('', np.nan)
```

4. IFSC Code Validation

- Remove duplicates
- Validate IFSC with regex (^[A-Z]{4}0[A-Z0-9]{6}$)

```
pattern = re.compile(r"^[A-Z]{4}0[A-Z0-9]{6}$")

df = df.drop_duplicates(subset=["ifsc"])

df = df[df["ifsc"].notna()]

df = df[df["ifsc"].apply(lambda x: bool(pattern.match(str(x))))]
```

5. Bank-wise Analysis

Count number of accounts and branches per bank.

```
bank_counts = df.groupby('bank').size().reset_index(name='num_accounts')

highest_bank = bank_counts.loc[bank_counts['num_accounts'].idxmax()]

lowest_bank = bank_counts.loc[bank_counts['num_accounts'].idxmin()]



branch_counts = df.groupby('bank')['branch'].nunique().reset_index(name='num_branches')

most_branches = branch_counts.loc[branch_counts['num_branches'].idxmax()]

fewest_branches = branch_counts.loc[branch_counts['num_branches'].idxmin()]
```

6. State Standardization

Correct common spelling mistakes and abbreviations.

```
state_corrections = {

    "MADHY PRADESH": "MADHYA PRADESH",

    "UTTA PRADES": "UTTAR PRADESH",

    "ODHISA": "ODISHA",

    "ORISSA": "ODISHA",

    "CHHATISGARH": "CHHATTISGARH",
    "KARANATAKA": "KARNATAKA",

    "TN": "TAMIL NADU",

    "TELENGANA": "TELANGANA",

    "GUJRAT": "GUJARAT",

    "MH": "MAHARASHTRA",

    "UP": "UTTAR PRADESH",

    ...

}

df['state'] = df['state'].str.strip().replace(state_corrections).replace('', np.nan)
```

7. STD Code Cleaning & Enrichment

- Validate STD codes
- Map missing codes from city names

```
city_std_map = {

    "NEW DELHI": "011",

    "MUMBAI": "022",

    "KOLKATA": "033",
    "CHENNAI": "044",

    "BANGALORE": "080",

    "PUNE": "020",

    "HYDERABAD": "040",

    ...

}

df['city1'] = df['city1'].astype(str).str.upper().str.strip()

df['stdCode'] = df['city1'].map(city_std_map)

df['stdCode'] = df['stdCode'].apply(

    lambda x: str(int(float(x))) if pd.notna(x) and str(x).strip() not in ["", "nan"] else None

)
```

8. Phone Number Cleaning

- Remove invalid numbers (`9999999999`, `1234567890`, etc.)
- Ensure only numeric
- Standardize with `NaN` where missing

```
df['phone'] = df['phone'].astype(str).str.strip()

df['phone'] = df['phone'].apply(
    lambda x: str(int(float(x))) if pd.notna(x) and str(x).strip() not in ["", "nan"] else None
).replace({'': np.nan, 'nan': np.nan})
```

Understood 👍 Let's **add the Bank Master Extraction section** directly to your documentation. You can copy-paste this into your existing Word file (`Untitled document.docx`) so it becomes part of the same document.

9. Standardize Column Names

```
df.columns = df.columns.str.strip().str.upper()
```

Converts all column names to **uppercase** and removes extra spaces.

10. Extract Unique Bank Names

```
unique_banks = df["BANK"].dropna().unique()
```

- Selects the `BANK` column
- Drops `NaN` values
- Extracts unique bank names

11. Create Bank Master Table

```
bank_df = pd.DataFrame({
    "bank_id": range(1, len(unique_banks) + 1),
    "bank_name": unique_banks
```

})

- Creates a new DataFrame `bank_df`
- `bank_id`: auto-increment IDs starting from 1
- `bank_name`: unique list of banks

12. Reset Index

bank_df = bank_df.reset_index(drop=True)

Resets the DataFrame index and removes the old index column.

13. Print Without Index

print(bank_df.to_string(index=False))

Displays the DataFrame **without the left index column** for a clean tabular format.

14. Save Cleaned Data

df.to_csv("cleaned_bankifsc_data.csv", index=False)

Conclusion

The pipeline successfully:

- Standardized IFSC codes
- Cleaned text fields (bank, branch, city, state)
- Corrected state spellings and abbreviations
- Validated and mapped STD codes
- Removed invalid phone numbers
- Produced a final clean dataset (cleaned_bankifsc_data.csv) ready for database insertion or analysis.