Public Finances in Modern History

🧭 Overview

This notebook pulls selected macro-fiscal indicators for Indonesia and its peer countries from the **IMF DataMapper API**, restructures the data into both **long** and **wide** formats, and saves the outputs with date-stamped filenames. Additionally, it generates:

1. 📄 A **correlation matrix PDF report** for cross-country comparisons.
2. 📊 A **clean heatmap correlation PDF** for visual presentation.
3. 📈 A **trend chart PDF** with per-country markers and a well-positioned legend.

🔧 Step 1: Import Libraries and Set Parameters

```
import requests
import pandas as pd
from tqdm import tqdm
from datetime import datetime
import time
```

```
# === Settings ===
BASE_URL = "<https://www.imf.org/external/datamapper/api/v1>"

YEAR_RANGE = list(range(2005, 2024))  # Inclusive range: 2005–2023

INDICATORS = {
    "rev": "Government revenue (% of GDP)",
    "exp": "Government expenditure (% of GDP)",
    "prim_exp": "Primary expenditure (% of GDP)",
    "ie": "Interest on public debt (% of GDP)",
    "pb": "Primary balance (% of GDP)",
    "d": "Gross public debt (% of GDP)",
    "rgc": "Real GDP growth rate (%)",
    "rltir": "Real long-term bond yield (%)"
}

PEER_COUNTRIES = {
    "IDN": "Indonesia",
    "PHL": "Philippines",
    "IND": "India",
    "VNM": "Vietnam",
    "COL": "Colombia",
    "MEX": "Mexico",
    "PER": "Peru",
    "ZAF": "South Africa",
    "ROU": "Romania",
    "HUN": "Hungary",
    "THA": "Thailand"
}
```

🌐 Step 2: Fetch and Process IMF Data

```
def fetch_and_process_data():
    """
    Fetches indicator data from IMF API for selected countries and years.

    Returns:
        df_long (DataFrame): Long format (country, year, indicator, value)
        df_wide (DataFrame): Wide format (country, year, [indicators])
    """
    print("Starting data processing...")
    all_data = []

    for code, label in tqdm(INDICATORS.items(), desc="Fetching indicators"):
        try:
            response = requests.get(f"{BASE_URL}/{code}", timeout=10)
            response.raise_for_status()
            values = response.json().get("values", {}).get(code, {})

            for country_code, year_dict in values.items():
                if country_code in PEER_COUNTRIES:
                    for year_str, value in year_dict.items():
                        try:
                            year = int(year_str)
                            if year in YEAR_RANGE:
                                all_data.append({
                                    "year": year,
                                    "country_code": country_code,
                                    "country": PEER_COUNTRIES[country_code],
                                    "indicator": label,
                                    "value": float(value) if value else None
                                })
                        except (ValueError, TypeError):
                            continue
        except Exception as e:
            print(f"⚠️ Skipping {code} due to error: {e}")
            continue

    df_long = pd.DataFrame(all_data)
    df_long = df_long.sort_values(["country", "year", "indicator"])

    df_wide = df_long.pivot_table(
        index=["country", "year"],
        columns="indicator",
        values="value"
    ).reset_index()

    return df_long, df_wide
```

💾 Step 3: Run and Export CSV Outputs

```
df_long, df_wide = fetch_and_process_data()

# Generate output filenames
today = datetime.now().strftime("%Y%m%d")
long_file = f"{today}_tfda_publicfinance_long.csv"
wide_file = f"{today}_tfda_publicfinance_wide.csv"

# Save CSV files
df_long.to_csv(long_file, index=False)
df_wide.to_csv(wide_file, index=False)

# Display summary
print(f"✅ Saved {len(df_long):,} rows (long format) → {long_file}")
print(f"✅ Saved {len(df_wide):,} rows (wide format) → {wide_file}")
df_wide.head()
```

📊 Step 4: Generate Correlation Tables and Export to PDF

```
import pandas as pd
from datetime import datetime
from fpdf import FPDF
import os

class PDF(FPDF):
    def __init__(self):
        super().__init__(orientation='L')  # Landscape orientation
        self.set_auto_page_break(auto=True, margin=15)
        self.set_margins(15, 15, 15)  # Left, Top, Right margins

    def header(self):
        self.set_font('Arial', 'B', 12)
        self.cell(0, 10, '', 0, 1, 'C')

    def footer(self):
        self.set_y(-15)
        self.set_font('Arial', 'I', 8)
        self.cell(0, 10, f'Page {self.page_no()}', 0, 0, 'C')
```

```
def format_correlation_table(df, indicator):
    """Create a properly formatted correlation table with 2 decimal places"""
    # Calculate correlations
    corr_matrix = df.pivot_table(
        index="year",
        columns="country",
        values=indicator
    ).corr().round(2)

    # Convert to string with exactly 2 decimal places
    formatted_df = corr_matrix.apply(lambda x: x.map("{:.2f}".format))
    formatted_df.index.name = 'Country'
    formatted_df.reset_index(inplace=True)

    return formatted_df

def generate_correlation_report():
    """Generate landscape-format correlation tables PDF"""
    today = datetime.now().strftime("%Y%m%d")
    input_file = f"{today}_tfda_publicfinance_wide.csv"
    pdf_file = f"{today}_tfda_correlation_tables.pdf"

    try:
        df = pd.read_csv(input_file)
        print(f"✓ Successfully loaded data from {input_file}")

        # Verify required columns exist
        if 'country' not in df.columns:
            raise ValueError("'country' column not found in the data")
    except Exception as e:
        print(f"❌ Error loading data: {str(e)}")
        return

    INDICATORS = [
        "Government revenue (% of GDP)",
        "Government expenditure (% of GDP)",
        "Primary expenditure (% of GDP)",
        "Interest on public debt (% of GDP)",
        "Primary balance (% of GDP)",
        "Gross public debt (% of GDP)",
        "Real GDP growth rate (%)",
        "Real long-term bond yield (%)"
    ]

    # Initialize PDF
    pdf = PDF()
    pdf.set_title("")
    pdf.set_author("IMF DataMapper")
```

```
for indicator in INDICATORS:
    if indicator not in df.columns:
        print(f"⚠️ Skipping missing indicator: {indicator}")
        continue

    try:
        print(f"🔄 Processing {indicator.split('(')[0].strip()}...")

        # Get formatted correlation table
        table_df = format_correlation_table(df, indicator)

        # Add new PDF page
        pdf.add_page()

        # Add title
        pdf.set_font("Arial", 'B', 14)
        title = indicator.split('(')[0].strip()
        pdf.cell(0, 10, f"Correlation: {title}", 0, 1, 'C')
        pdf.ln(5)

        # Add metadata
        pdf.set_font("Arial", 'I', 10)
        pdf.cell(0, 6, "Period: 2005-2023 | Data Source: IMF", 0, 1, 'C')
        pdf.ln(8)

        # Create table
        col_widths = [40] + [20] * (len(table_df.columns) - 1)  # Wider first column

        # Headers
        pdf.set_font("Arial", 'B', 8)
        for i, col in enumerate(table_df.columns):
            pdf.cell(col_widths[i], 8, str(col), border=1, align='C')
        pdf.ln()

        # Data rows
        pdf.set_font("Arial", size=9)
        for _, row in table_df.iterrows():
            # Country name (left-aligned)
            pdf.cell(col_widths[0], 6, str(row['Country']), border=1, align='L')

            # Correlation values (right-aligned)
            for i, val in enumerate(row[1:], 1):
                pdf.cell(col_widths[i], 6, str(val), border=1, align='R')

            pdf.ln()

        print(f"✅ Completed {indicator.split('(')[0].strip()}")
        except Exception as e:
            print(f"❌ Failed to process {indicator}: {str(e)}")
            continue

    try:
        pdf.output(pdf_file)
        print(f"\n🎉 Successfully saved report to:\n{os.path.abspath(pdf_file)}")
    except Exception as e:
        print(f"\n❌ Failed to save PDF: {str(e)}")

if __name__ == "__main__":
    print("\n" + "="*60)
    print(" TFDA Peer Country Correlation Report ".center(60, '='))
    print("="*60 + "\n")

    generate_correlation_report()
    print("\n" + "="*60)
    print(" Process Completed ".center(60, '='))
    print("="*60 + "\n")
```

🔥 Step 5: Generate Clean Correlation Heatmaps

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from datetime import datetime
import os

def create_correlation_pdf():
    """Generate PDF with clean correlation heatmaps (no legend or axis labels)"""
    today = datetime.now().strftime("%Y%m%d")
    input_file = f"{today}_tfda_publicfinance_wide.csv"
    pdf_path = f"{today}_tfda_correlation_heatmaps.pdf"

    # Set Seaborn style
    sns.set_style("white", {
        'axes.grid': False,
        'font.family': 'sans-serif'
    })

    try:
        df = pd.read_csv(input_file)
        print(f"✓ Successfully loaded data from {input_file}")
    except FileNotFoundError:
        raise FileNotFoundError(f"Input file not found. Please run data collection script first.")
```

```
indicators = [
    "Government revenue (% of GDP)",
    "Government expenditure (% of GDP)",
    "Primary expenditure (% of GDP)",
    "Interest on public debt (% of GDP)",
    "Primary balance (% of GDP)",
    "Gross public debt (% of GDP)",
    "Real GDP growth rate (%)",
    "Real long-term bond yield (%)"
]

with PdfPages(pdf_path) as pdf:
    for indicator in indicators:
        if indicator not in df.columns:
            print(f"× Skipping missing indicator: {indicator}")
            continue

        try:
            # Prepare correlation matrix
            corr_matrix = df.pivot_table(
                index="year",
                columns="country",
                values=indicator
            ).corr().round(2)

            # Create figure
            plt.figure(figsize=(12, 10))

            # Create clean heatmap
            ax = sns.heatmap(
                corr_matrix,
                annot=True,
                fmt=".2f",
                cmap="viridis",
                vmin=-1,
                vmax=1,
                center=0,
                linewidths=0.5,
                cbar=False,  # No colorbar
                annot_kws={"size": 9}
            )

            # Remove axis labels
            ax.set(xlabel=None, ylabel=None)  # No x/y axis labels
            ax.tick_params(left=False, bottom=False)  # No tick marks

            # Formatting
```

```
                plt.title(
                    f"Correlation of {indicator.split('(')[0].strip()}\n(2005-2023)",
                    pad=20,
                    fontsize=14,
                    fontweight='bold'
                )
                plt.tight_layout()

                # Add to PDF
                pdf.savefig()
                plt.close()
                print(f"✓ Created clean heatmap for {indicator.split('(')[0].strip()}")

            except Exception as e:
                print(f"× Failed to create heatmap for {indicator}: {str(e)}")
                continue

    print(f"\n✅ Clean correlation heatmaps saved to {os.path.abspath(pdf_path)}")

if __name__ == "__main__":
    print("\n" + "="*60)
    print("Generating Clean Correlation Heatmaps".center(60))
    print("="*60 + "\n")

    try:
        create_correlation_pdf()
    except Exception as e:
        print(f"\n❌ Error: {e}")
```

📈 Step 6: Generate Trend Charts

```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
from matplotlib.backends.backend_pdf import PdfPages

def create_trends_pdf():
    """Generate PDF with perfectly positioned legend between title and plot"""
    today = datetime.now().strftime("%Y%m%d")
    input_file = f"{today}_tfda_publicfinance_wide.csv"
    output_pdf = f"{today}_tfda_indicator_trends.pdf"

    # Configure plot style
    sns.set_theme(
    style="whitegrid",
    context="notebook",
    rc={
        "axes.edgecolor": "#333333",
        "axes.linewidth": 0.8,
        "axes.labelsize": 11,
        "axes.titlesize": 14,
        "axes.titleweight": "bold",
        "grid.color": "#E0E0E0",
        "grid.linestyle": "--",
        "legend.frameon": False,
        "xtick.labelsize": 10,
        "ytick.labelsize": 10,
        "font.family": "sans-serif",
        "font.sans-serif": ["Arial", "DejaVu Sans", "Liberation Sans"]
    }
)

    plt.rcParams.update({
        'font.family': 'sans-serif',
        'axes.titlesize': 14,
        'axes.titleweight': 'bold',
        'legend.fontsize': 9,
        'legend.markerscale': 1.2,  # Scales marker size in legend
    })

    try:
        df = pd.read_csv(input_file)
        df['year'] = df['year'].astype(int)
        print(f"✓ Data loaded from {input_file}")
    except FileNotFoundError:
        raise FileNotFoundError("❌ Data file not found. Please run the main data script first.")

    indicators = [
        "Government revenue (% of GDP)",
        "Government expenditure (% of GDP)",
        "Primary expenditure (% of GDP)",
        "Interest on public debt (% of GDP)",
        "Primary balance (% of GDP)",
        "Gross public debt (% of GDP)",
        "Real GDP growth rate (%)",
        "Real long-term bond yield (%)"
    ]

    country_styles = {
        'Indonesia':    {'color': '#FF0000', 'linewidth': 2.5, 'alpha': 1.0, 'marker': 'o'},
        'Philippines':  {'color': '#7FB3D5', 'linewidth': 2.0, 'alpha': 0.8, 'marker': 's'},
        'India':       {'color': '#82C09A', 'linewidth': 2.0, 'alpha': 0.8, 'marker': 'D'},
        'Vietnam':       {'color': '#D4A5A5', 'linewidth': 2.0, 'alpha': 0.8, 'marker': '^'},
        'Colombia':      {'color': '#D4B483', 'linewidth': 2.0, 'alpha': 0.8, 'marker': 'v'},
        'Mexico':        {'color': '#A5C0D4', 'linewidth': 2.0, 'alpha': 0.8, 'marker': '<'},
        'Peru':          {'color': '#B5D4A5', 'linewidth': 2.0, 'alpha': 0.8, 'marker': '>'},
        'South Africa':  {'color': '#D4A5C0', 'linewidth': 2.0, 'alpha': 0.8, 'marker': 'p'},
        'Romania':       {'color': '#C0C0C0', 'linewidth': 2.0, 'alpha': 0.8, 'marker': '*'},
        'Hungary':       {'color': '#D4D4A5', 'linewidth': 2.0, 'alpha': 0.8, 'marker': 'X'},
        'Thailand':      {'color': '#A5A5D4', 'linewidth': 2.0, 'alpha': 0.8, 'marker': 'd'}
    }

    with PdfPages(output_pdf) as pdf:
        for indicator in indicators:
            if indicator not in df.columns:
                print(f"× Skipping missing indicator: {indicator}")
                continue

            try:
                # Create figure with adjusted proportions
                fig = plt.figure(figsize=(12, 8))
                ax = fig.add_subplot(111)

                # Define x-axis ticks
                years = sorted(df['year'].unique())
                xticks = years[::5] + ([years[-1]] if years[-1] not in years[::5] else [])

                # Plot each country
                for country in df['country'].unique():
                    data = df[df['country'] == country]
                    style = country_styles.get(country, {
                        'color': '#999999', 'linewidth': 1.8, 'marker': 'o'
                    })

                    sns.lineplot(
                        data=data,
                        x="year",
                        y=indicator,
                        ax=ax,
                        label=country,
                        color=style['color'],
                        linewidth=style['linewidth'],
                        marker=style['marker'],
                        markersize=7,
                        markeredgecolor='none'
                    )

                # Set title (moved up to make space)
                title = ax.set_title(
                    f"{indicator.split('(')[0].strip()} — Trend by Country",
                    pad=25,  # Increased padding
                    y=1.08   # Explicit vertical position
                )

                # Format axes
                ax.set_xlabel("")
                ax.set_ylabel(indicator)
                ax.set_xticks(xticks)

                # Calculate legend columns (2 rows worth)
                n_countries = len(df['country'].unique())
                ncol = (n_countries + 1) // 2

                # Create legend between title and plot
                legend = ax.legend(
                    loc='upper center',
                    bbox_to_anchor=(0.5, 1.1),  # Precise positioning
                    bbox_transform=ax.transAxes,
                    ncol=ncol,
                    frameon=False,
                    handletextpad=0.4,
                    columnspacing=1.2
                )

                # Adjust layout with exact spacing
                plt.subplots_adjust(top=0.75)  # Fine-tuned spacing
                plt.tight_layout()

                pdf.savefig(fig, bbox_inches='tight')
                plt.close()
                print(f"✓ Chart created: {indicator.split('(')[0].strip()}")

            except Exception as e:
                print(f"× Failed to process {indicator}: {e}")

    print(f"\n✅ All charts exported to PDF:\n{os.path.abspath(output_pdf)}")

if __name__ == "__main__":
    print("\n" + "="*60)
    print(" Generating Final Trend Charts ".center(60, '='))
    print("="*60 + "\n")

    try:
        create_trends_pdf()
    except Exception as e:
        print(f"\n❌ Error occurred: {e}")

    print("\n" + "="*60)
    print(" Process Completed ".center(60, '='))
    print("="*60 + "\n")
```

✅ Step 7: Final Output Recap

Once all steps are executed successfully, you should have:

- `{date}_tfda_publicfinance_long.csv`
- `{date}_tfda_publicfinance_wide.csv`
- `{date}_tfda_correlation_tables.pdf`
- `{date}_tfda_correlation_heatmaps.pdf`
- `{date}_tfda_indicator_trends.pdf`