```
import os
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

input_dir = "/Users/xxn/Desktop/Jacklyn's research/data3"
output_dir = "/Users/xxn/Desktop/Jacklyn's research/Trading_data_after_processing"
os.makedirs(output_dir, exist_ok=True)

for file_name in os.listdir(input_dir):
    if file_name.endswith(".xlsx"):
        country_name = file_name.replace(".xlsx", "")
        file_path = os.path.join(input_dir, file_name)
        print(f"Processing {country_name}...")

        try:
            xls = pd.ExcelFile(file_path)
            sheets = xls.sheet_names
        except Exception as e:
            print(f"Error reading {file_name}: {e}")
            continue

        valid_sheets = [sheet for sheet in sheets if not sheet.lower().startswith("wb")]
        if not valid_sheets:
            print(f"No valid sheets found for {country_name}. Skipping...")
            continue

        merged_data = None
        for sheet in valid_sheets:
            try:
                df = pd.read_excel(file_path, sheet_name=sheet)
                required_cols = ['DateTime', 'Value']
                if not all(col in df.columns for col in required_cols):
                    print(f"Sheet '{sheet}' in {country_name} missing required columns. Skipping sheet.")
                    continue
                df = df[['DateTime', 'Value']].rename(columns={'Value': sheet})
                df['DateTime'] = pd.to_datetime(df['DateTime'])
                df.sort_values('DateTime', inplace=True)

                if merged_data is None:
                    merged_data = df
                else:
                    merged_data = pd.merge(merged_data, df, on='DateTime', how='outer')
            except Exception as e:
                print(f"Error in sheet '{sheet}' for {country_name}: {e}")
                continue

        if merged_data is None or merged_data.empty:
            print(f"No valid data for {country_name}. Skipping...")
            continue

        merged_data.set_index('DateTime', inplace=True)
        numeric_cols = merged_data.select_dtypes(include=['number']).columns
        if merged_data.isnull().values.any():
            print(f"Imputing missing values for {country_name}.")
            merged_data[numeric_cols] = merged_data[numeric_cols].fillna(merged_data[numeric_cols].mean())

        if merged_data.shape[0] < 2:
            print(f"Insufficient data for PCA in {country_name}. Skipping...")
            continue

        scaler = StandardScaler()
        scaled_data = scaler.fit_transform(merged_data[numeric_cols])

        pca = PCA()
        pca.fit(scaled_data)
        eigenvalues = pca.explained_variance_
        selected_components = [i+1 for i, val in enumerate(eigenvalues) if val > 1]
        if not selected_components:
            print(f"No components retained for {country_name} (Kaiser Rule). Skipping...")
```

```
            continue

        pca = PCA(n_components=len(selected_components))
        scores = pca.fit_transform(scaled_data)
        loadings = pca.components_

        loadings_df = pd.DataFrame(loadings, columns=numeric_cols,
                                   index=[f"PC{i}" for i in range(1, len(selected_components)+1)])
        loadings_df = loadings_df.transpose()

        sorted_loadings_dict = {}
        for pc in loadings_df.columns:
            pc_series = loadings_df[pc].copy()
            pc_series = pc_series.reindex(pc_series.abs().sort_values(ascending=False).index)
            pc_series = pc_series.apply(lambda x: x if abs(x) >= 0.4 else 0)
            sorted_loadings_dict[pc] = pc_series

        sorted_loadings_df = pd.DataFrame(sorted_loadings_dict)

        pca_output_dir = os.path.join(output_dir, country_name)
        os.makedirs(pca_output_dir, exist_ok=True)

        sorted_loadings_df.to_csv(os.path.join(pca_output_dir, "pca_loadings_sorted_thresholded.csv"))

        scores_df = pd.DataFrame(scores,
                                 columns=[f"PC{i}" for i in range(1, len(selected_components)+1)],
                                 index=merged_data.index)
        scores_df.to_csv(os.path.join(pca_output_dir, "pca_scores.csv"))

        if len(selected_components) >= 2:
            plt.figure(figsize=(8, 6))
            plt.scatter(scores_df["PC1"], scores_df["PC2"], color='blue', alpha=0.5)
            for idx, row in scores_df.iterrows():
                plt.text(row["PC1"], row["PC2"], str(idx.date()), fontsize=8)
            plt.xlabel("PC1")
            plt.ylabel("PC2")
            plt.title(f"PCA Score Plot for {country_name}")
            plt.grid()
            plt.tight_layout()
            plt.savefig(os.path.join(pca_output_dir, "pca_score_plot.png"))
            plt.close()

        with open(os.path.join(pca_output_dir, "PCA_summary.txt"), "w", encoding="utf-8") as f:
            f.write(f"PCA Summary for {country_name}\n")
            f.write("="*40 + "\n\n")
            f.write("Eigenvalues:\n")
            for i, val in enumerate(eigenvalues, start=1):
                f.write(f"PC{i}: {val:.4f}\n")
            f.write("\nComponents retained (Kaiser Rule):\n")
            f.write(f"{selected_components}\n")
            f.write(f"Number of components retained: {len(selected_components)}\n\n")
            f.write("Explained Variance Ratio:\n")
            for i, ratio in enumerate(pca.explained_variance_ratio_, start=1):
                f.write(f"PC{i}: {ratio*100:.2f}%\n")

        print(f"PCA completed for {country_name}. Results saved in {pca_output_dir}\n")

print("All files processed successfully.")
```