```
import requests, json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
plt.rcParams['figure.figsize'] = (12,5)
sns.set_style('whitegrid')

# ---------- Put your API key here ----------
api_key = "Q5M1FTDZTGXSQ7D55I3J"
# -------------------------------------------

# Fixed parameters
stat_code    = "501Y001"
freq         = "A"
start_period = "2009"
end_period   = "2023"
item_code1   = "ZZZ00"

# Request
base = "https://ecos.bok.or.kr/api/StatisticSearch"
url =
f"{base}/{api_key}/json/kr/1/9999/{stat_code}/{freq}/{start_period}/{en
d_period}/{item_code1}"
resp = requests.get(url, timeout=30)
if resp.status_code != 200:
    raise RuntimeError(f"HTTP {resp.status_code}. Response:
{resp.text[:800]}")
data = resp.json()
if 'StatisticSearch' not in data or
int(data['StatisticSearch'].get('list_total_count', 0)) == 0:
    raise RuntimeError("No data returned. Response snippet: " +
json.dumps(data)[:800])

rows = data['StatisticSearch'].get('row', [])
rows = rows if isinstance(rows, list) else [rows]
df = pd.DataFrame(rows)

# Basic checks and preprocessing
if 'TIME' not in df.columns:
    raise RuntimeError("TIME column missing. Columns: " +
str(df.columns.tolist()))

df['TIME'] = df['TIME'].astype(str)
df['year'] = df['TIME'].str.slice(0,4).astype(int)
df['date'] = pd.to_datetime(df['year'].astype(str) + '-01-01')
```

```
# find value column
val_col = None
for c in df.columns:
    if 'DATA' in c.upper() and 'VALUE' in c.upper():
        val_col = c
        break
if val_col is None:
    for c in df.columns:
        if c == 'TIME': continue
        try:
            pd.to_numeric(df[c].astype(str).str.replace(',', ''),
errors='raise')
            val_col = c
            break
        except:
            continue
if val_col is None:
    raise RuntimeError("Value column not found. Columns: " +
str(df.columns.tolist()))

df['value'] = pd.to_numeric(df[val_col].astype(str).str.replace(',',
'').replace('', np.nan), errors='coerce')
df = df.set_index('date').sort_index()

# Fill missing with column mean (educational)
missing_before = df['value'].isna().sum()
mean_val = df['value'].mean(skipna=True)
df['value_filled'] = df['value'].fillna(mean_val)
missing_after = df['value_filled'].isna().sum()
print(f"Missing before: {missing_before}, after filling with mean
({mean_val:.3f}): {missing_after}")

# Moving average and YoY
df['ma_3'] = df['value_filled'].rolling(window=3, min_periods=1).mean()
df['pct_yoy'] = df['value_filled'].pct_change(periods=1) * 100

# Recent/previous 3-year means
mean_recent_3 = df.loc['2021-01-01':'2023-12-31',
'value_filled'].mean()
mean_prev_3   = df.loc['2018-01-01':'2020-12-31',
'value_filled'].mean()
change_rate_3y = (mean_recent_3 - mean_prev_3) / mean_prev_3 * 100 if
(mean_prev_3 and not np.isnan(mean_prev_3)) else np.nan

# Latest observation and safe previous-year retrieval (by year)
latest_date = df.index.max()
latest_val = df.loc[latest_date, 'value_filled']
```

```
prev_year = latest_date.year - 1
prev_vals = df[df['year'] == prev_year]['value_filled']
if not prev_vals.empty:
    # if multiple rows for prev_year, take mean (but for annual data
typically one)
    val_one_year_ago = float(prev_vals.mean())
else:
    val_one_year_ago = np.nan

# Now compute YoY safely
if pd.notna(val_one_year_ago) and val_one_year_ago != 0:
    pct_yoy_latest = (latest_val - val_one_year_ago) / val_one_year_ago
* 100
else:
    pct_yoy_latest = np.nan

print(f"\nLatest year: {latest_date.year}, value = {latest_val:.3f}")
if pd.notna(val_one_year_ago):
    print(f"1 year before: {prev_year}, value =
{val_one_year_ago:.3f}")
else:
    print(f"1 year before: {prev_year}, value = NaN (not available in
data)")
print(f"YoY % (latest) = {pct_yoy_latest if not
np.isnan(pct_yoy_latest) else 'NaN'}")
print(f"Recent 3-year mean (2021-2023) = {mean_recent_3:.3f}")
print(f"Previous 3-year mean (2018-2020) = {mean_prev_3:.3f}")
print(f"3-year change rate (%) = {change_rate_3y:.3f}")

# Visualization (and save)
plt.figure(figsize=(12,5))
plt.plot(df.index.year, df['value_filled'], marker='o', label='Value
(filled)')
plt.plot(df.index.year, df['ma_3'], marker='o', label='3-year moving
average', linewidth=2)
plt.title('Balance Sheet - All Industries (Annual)')
plt.xlabel('Year')
plt.ylabel('Amount (unit as provided by API)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("timeseries_501Y001_ZZZ00.png", dpi=150)
plt.show()

annual = df['value_filled'].resample('YE').mean()
annual.index = annual.index.year
plt.figure(figsize=(12,4))
```

```
sns.barplot(x=annual.index.astype(str), y=annual.values)
plt.xticks(rotation=45)
plt.title('Annual Average: All Industries')
plt.xlabel('Year')
plt.ylabel('Average Amount')
plt.tight_layout()
plt.savefig("annual_bar_501Y001_ZZZ00.png", dpi=150)
plt.show()

# Save CSV
out_fname = f"ecos_{stat_code}_{item_code1}_annual_cleaned.csv"
df.to_csv(out_fname, encoding='utf-8-sig', index=True)
print("Saved:", out_fname)
```