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

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

# ---------- SET YOUR API KEY HERE ----------
api_key = ""
# -------------------------------------------

# Fixed parameters (exam conditions)
stat_code    = "631Y001"   # internet banking
freq         = "A"         # annual (fixed)
start_period = "2000"      # YYYY
end_period   = "2023"      # YYYY
item_code1   = "H2200"     # transfer amount (unit: 100 million
KRW)

# Build URL (annual)
base = "https://ecos.bok.or.kr/api/StatisticSearch"
url =
f"{base}/{api_key}/json/kr/1/9999/{stat_code}/{freq}/{start_pe
riod}/{end_period}/{item_code1}"

resp = requests.get(url, timeout=30)
if resp.status_code != 200:
    raise RuntimeError(f"HTTP {resp.status_code} returned.
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 for the given
parameters. Response snippet: " + json.dumps(data)[:800])

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

# Basic preview
print("Preview of raw data:")
display(df.head(10))
```

```
print("Columns:", df.columns.tolist())

# Preprocessing: TIME (YYYY) -> datetime, numeric conversion
if 'TIME' not in df.columns:
    raise RuntimeError("No TIME column in response. Check API
output.")

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:
    # fallback: first numeric-looking column (exclude TIME)
    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("Could not find value column in
response. Columns: " + str(df.columns.tolist()))

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

# index and sorting
df = df.set_index('date').sort_index()

# missing values: linear interpolate
missing_before = df['value'].isna().sum()
df['value_interp'] = df['value'].interpolate(method='linear',
limit_direction='both')
missing_after = df['value_interp'].isna().sum()
print(f"Missing before: {missing_before}, after interpolation:
{missing_after}")

display(df[['year', 'value', 'value_interp']].head(12))
```

```
# Analysis: moving average and comparisons (annual)
df['ma_3'] = df['value_interp'].rolling(window=3,
min_periods=1).mean()
df['pct_yoy'] = df['value_interp'].pct_change(periods=1) *
100  # annual change in percent

mean_recent_3 = df.loc['2021-01-01':'2023-12-31',
'value_interp'].mean()
mean_prev_3   = df.loc['2018-01-01':'2020-12-31',
'value_interp'].mean()
change_rate_3y = (mean_recent_3 - mean_prev_3) / mean_prev_3 *
100 if mean_prev_3 != 0 else np.nan

latest_date = df.index.max()
latest_val = df.loc[latest_date, 'value_interp']
one_year_ago = latest_date - pd.DateOffset(years=1)
val_one_year_ago = df['value_interp'].get(one_year_ago,
np.nan)
pct_yoy_latest = (latest_val - val_one_year_ago) /
val_one_year_ago * 100 if pd.notna(val_one_year_ago) and
val_one_year_ago != 0 else np.nan

print(f"\nLatest observation: {latest_date.year} =
{latest_val:.3f} (unit: 100 million KRW)")
print(f"One year before: {one_year_ago.year} =
{val_one_year_ago:.3f}")
print(f"YoY % (latest) = {pct_yoy_latest:.2f}%")
print(f"\nRecent 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 (English labels)
# 1) Time series + 3-year moving average
plt.figure(figsize=(12,5))
plt.plot(df.index.year, df['value_interp'], marker='o',
label='Original (interpolated)')
plt.plot(df.index.year, df['ma_3'], marker='o', label='3-year
moving average', linewidth=2)
plt.title('Internet Banking - Transfer Amount (Annual)',
fontsize=14)
plt.xlabel('Year')
plt.ylabel('Amount (100 million KRW)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
# 2) Annual mean bar plot (use 'YE' to avoid deprecation)
annual = df['value_interp'].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 Amount by Year')
plt.xlabel('Year')
plt.ylabel('Average Amount (100 million KRW)')
plt.tight_layout()
plt.show()

# Save cleaned 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)
```