In [1]:
```
from __future__ import annotations

import io
import logging
from typing import Iterable, Optional, Union

import pandas as pd
import requests
```

In [2]:
```
logger = logging.getLogger("fred_api")
logger.setLevel(logging.INFO)

# Remove existing handlers
for handler in logger.handlers[:]:
    logger.removeHandler(handler)

# File handler: logs all INFO+ to file
file_handler = logging.FileHandler("fred_logs.txt")
file_handler.setLevel(logging.INFO)
file_formatter = logging.Formatter(
    "%(asctime)s %(levelname)s [%(name)s]: %(message)s",
    datefmt="%Y-%m-%d %H:%M:%S"
)
file_handler.setFormatter(file_formatter)
logger.addHandler(file_handler)

# Console handler: logs only WARNING+ to console
console_handler = logging.StreamHandler()
console_handler.setLevel(logging.WARNING)
console_formatter = logging.Formatter(
    "%(levelname)s: %(message)s"
)
console_handler.setFormatter(console_formatter)
logger.addHandler(console_handler)

# Prevent propagation to root logger
logger.propagate = False

print("Hybrid logging enabled: Warnings/errors in console, all logs in fred_logs.txt")
```

```
Hybrid logging enabled: Warnings/errors in console, all logs in fred_logs.txt
```

In [3]:
```python
FRED_BASE = "https://fred.stlouisfed.org/graph/fredgraph.csv?id="

# Default logger
_default_logger = logging.getLogger("fred_api")
if not _default_logger.handlers:
    handler = logging.StreamHandler()
    formatter = logging.Formatter(
        "%(asctime)s %(levelname)s [%(name)s]: %(message)s", datefmt="%Y-%m-%d %H:%M:%S"
    )
    handler.setFormatter(formatter)
    _default_logger.addHandler(handler)
_default_logger.setLevel(logging.INFO)


# FRED loader
def get_fred_series(series_id: str) -> pd.DataFrame:
    """
    Fetch a FRED series using the fredgraph.csv endpoint.
    """
    url = f"https://fred.stlouisfed.org/graph/fredgraph.csv?id={series_id}"

    try:
        df = pd.read_csv(url)
    except Exception as exc:
        raise ConnectionError(f"Failed to download FRED series '{series_id}'.") from exc

    # Convert columns
    if "observation_date" not in df or series_id not in df:
        raise ValueError(f"Unexpected CSV format received for series '{series_id}'.")

    df["observation_date"] = pd.to_datetime(df["observation_date"], errors="raise")
    df[series_id] = pd.to_numeric(df[series_id], errors="coerce")

    df = df.rename(columns={"observation_date": "date", series_id: "value"})
    df = df.sort_values("date").reset_index(drop=True)

    return df


# Main API function
```

```python
def FredAPI(
    series_id: str,
    dates: Optional[Iterable[Union[str, pd.Timestamp]]] = None,
    start_date: Optional[str] = None,
    end_date: Optional[str] = None,
    timeout: int = 15,
    session: Optional[requests.Session] = None,
    logger: Optional[logging.Logger] = None,
) -> pd.DataFrame:

    log = logger or _default_logger

    # Helpers
    def _validate_input_modes(_dates, _start_date, _end_date):
        if _dates is not None and (_start_date is not None or _end_date is not None):
            raise ValueError("Pass either 'dates' OR ('start_date'/'end_date'), not both.")

    def _validate_types(_series_id, _dates, _start_date, _end_date):
        if not isinstance(_series_id, str) or not _series_id.strip():
            raise TypeError("series_id must be a non-empty string.")
        if _dates is not None and not isinstance(_dates, Iterable):
            raise TypeError("'dates' must be an iterable.")
        if _start_date is not None and not isinstance(_start_date, str):
            raise TypeError("'start_date' must be a string YYYY-MM-DD.")
        if _end_date is not None and not isinstance(_end_date, str):
            raise TypeError("'end_date' must be a string YYYY-MM-DD.")

    def _parse_dates_iterable(_dates):
        try:
            parsed = pd.to_datetime(list(_dates), errors="raise")
        except Exception as exc:
            raise ValueError("One or more items in 'dates' could not be parsed.") from exc
        return pd.DatetimeIndex(parsed.normalize())

    def _parse_single_date(date_str):
        try:
            ts = pd.to_datetime(date_str, format="%Y-%m-%d", errors="raise").normalize()
        except Exception as exc:
            raise ValueError(f"Date '{date_str}' must be YYYY-MM-DD.") from exc
        return ts
```

```
    # Validate inputs
    _validate_input_modes(dates, start_date, end_date)
    _validate_types(series_id, dates, start_date, end_date)

    # download data
    df = get_fred_series(series_id)

    if df.empty:
        raise ValueError(f"Series '{series_id}' returned no rows.")
    if df["value"].isna().all():
        raise ValueError(f"All values for series '{series_id}' are NaN.")

    # Routing
    if dates is None and start_date is None and end_date is None:
        return df

    min_date = df["date"].min()
    max_date = df["date"].max()

    # Handle specific dates
    if dates is not None:
        requested = _parse_dates_iterable(dates)

        too_early = requested[requested < min_date]
        if len(too_early) > 0:
            raise ValueError(
                f"Requested {len(too_early)} date(s) before series inception ({min_date.date()})."
            )

        out = (
            df.set_index("date")
            .reindex(requested)
            .ffill()
            .reset_index()
            .rename(columns={"index": "date"})
        )

        missing_count = (requested.difference(df["date"])).size
        if missing_count > 0:
            log.warning(
                f"{missing_count} requested date(s) missing in source; forward-filled."
            )
```

```
        return out

    # Handle ranges
    start_ts = _parse_single_date(start_date) if start_date else min_date
    end_ts = _parse_single_date(end_date) if end_date else max_date

    if start_ts > end_ts:
        raise ValueError(f"start_date {start_ts.date()} > end_date {end_ts.date()}.")
        log.warning(f"ValueError:(start_date {start_ts.date()} > end_date {end_ts.date()}.")

    if start_ts < min_date:
        raise ValueError(
            f"start_date {start_ts.date()} is before series inception ({min_date.date()})."
        )
        log.warning(f"start_date {start_ts.date()} is before series inception ({min_date.date()}).")

    daily = pd.date_range(start_ts, end_ts, freq="D")

    out = (
        df.set_index("date")
        .reindex(daily)
        .ffill()
        .reset_index()
        .rename(columns={"index": "date"})
    )

    missing_count = (daily.difference(df["date"])).size
    if missing_count > 0:
        log.warning(f"{missing_count} day(s) missing in source; forward-filled.")

    return out
```

In [4]:
```
df = FredAPI("DTB3")
print(df.head())
```

```
        date  value
0 1954-01-04   1.33
1 1954-01-05   1.28
2 1954-01-06   1.28
3 1954-01-07   1.31
4 1954-01-08   1.31
```

In [5]:
```
df = FredAPI("DTB3", start_date="1954-01-01")
df
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[5], line 1
----> 1 df = FredAPI("DTB3", start_date="1954-01-01")
      2 df

Cell In[3], line 140, in FredAPI(series_id, dates, start_date, end_date, timeout, session, logger)
    137     log.warning(f"ValueError:(start_date {start_ts.date()} > end_date {end_ts.date()}.")
    139 if start_ts < min_date:
--> 140     raise ValueError(
    141         f"start_date {start_ts.date()} is before series inception ({min_date.date()})."
    142     )
    143     log.warning(f"start_date {start_ts.date()} is before series inception ({min_date.date()}).")
    145 daily = pd.date_range(start_ts, end_ts, freq="D")

ValueError: start_date 1954-01-01 is before series inception (1954-01-04).
```

In [6]:
```
df = FredAPI("DTB3", end_date="2025-12-01")
df
```

```
WARNING: 7510 day(s) missing in source; forward-filled.
```

Out[6]:

	date	value
0	1954-01-04	1.33
1	1954-01-05	1.28
2	1954-01-06	1.28
3	1954-01-07	1.31
4	1954-01-08	1.31
...	...	...
26260	2025-11-27	3.75
26261	2025-11-28	3.75
26262	2025-11-29	3.75
26263	2025-11-30	3.75
26264	2025-12-01	3.75

26265 rows × 2 columns

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26265 entries, 0 to 26264
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   date    26265 non-null  datetime64[ns]
 1   value   26265 non-null  float64       
dtypes: datetime64[ns](1), float64(1)
memory usage: 410.5 KB
```

In [ ]:
```
# 1) Full series (e.g., 3-Month Treasury Bill: DTB3)
df_full = FredAPI("DTB3")
print(df_full.head())
```

In [8]:
```
# 2) Specific dates
df_dates = FredAPI("DGS2", dates=["2024-12-30", "2024-12-31", "2025-01-01"])
print(df_dates)
```

```
        date  value
0 2024-12-30   4.24
1 2024-12-31   4.25
2 2025-01-01   4.25
```

In [9]:
```
# 3) Start/end range (daily; forward-filled)
df_range = FredAPI("DGS10", start_date="2025-01-01", end_date="2025-02-15")
print(df_range.tail())
```

```
WARNING: 13 day(s) missing in source; forward-filled.
         date  value
41 2025-02-11   4.54
42 2025-02-12   4.62
43 2025-02-13   4.52
44 2025-02-14   4.47
45 2025-02-15   4.47
```

In [10]:
```
# 4) Error: unknown series
try:
    FredAPI("XXXX")
except ValueError as e:
    print("Caught expected error:", e)
```

```
---------------------------------------------------------------------------
HTTPError                                 Traceback (most recent call last)
Cell In[3], line 25, in get_fred_series(series_id)
     24 try:
---> 25     df = pd.read_csv(url)
     26 except Exception as exc:

File ~\without_conda\Lib\site-packages\pandas\io\parsers\readers.py:1026, in read_csv(filepath_or_buffer, sep, delimi
ter, header, names, index_col, usecols, dtype, engine, converters, true_values, false_values, skipinitialspace, skipr
ows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime
_format, keep_date_col, date_parser, date_format, dayfirst, cache_dates, iterator, chunksize, compression, thousands,
decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, encoding_errors, dialect, on
_bad_lines, delim_whitespace, low_memory, memory_map, float_precision, storage_options, dtype_backend)
   1024 kwds.update(kwds_defaults)
-> 1026 return _read(filepath_or_buffer, kwds)

File ~\without_conda\Lib\site-packages\pandas\io\parsers\readers.py:620, in _read(filepath_or_buffer, kwds)
    619 # Create the parser.
--> 620 parser = TextFileReader(filepath_or_buffer, **kwds)
    622 if chunksize or iterator:

File ~\without_conda\Lib\site-packages\pandas\io\parsers\readers.py:1620, in TextFileReader.__init__(self, f, engine,
**kwds)
   1619 self.handles: IOHandles | None = None
-> 1620 self._engine = self._make_engine(f, self.engine)

File ~\without_conda\Lib\site-packages\pandas\io\parsers\readers.py:1880, in TextFileReader._make_engine(self, f, eng
ine)
   1879         mode += "b"
-> 1880 self.handles = get_handle(
   1881     f,
   1882     mode,
   1883     encoding=self.options.get("encoding", None),
   1884     compression=self.options.get("compression", None),
   1885     memory_map=self.options.get("memory_map", False),
   1886     is_text=is_text,
   1887     errors=self.options.get("encoding_errors", "strict"),
   1888     storage_options=self.options.get("storage_options", None),
   1889 )
   1890 assert self.handles is not None

File ~\without_conda\Lib\site-packages\pandas\io\common.py:728, in get_handle(path_or_buf, mode, encoding, compressio
```

```
n, memory_map, is_text, errors, storage_options)
    727 # open URLs
--> 728 ioargs = _get_filepath_or_buffer(
    729     path_or_buf,
    730     encoding=encoding,
    731     compression=compression,
    732     mode=mode,
    733     storage_options=storage_options,
    734 )
    736 handle = ioargs.filepath_or_buffer

File ~\without_conda\Lib\site-packages\pandas\io\common.py:384, in _get_filepath_or_buffer(filepath_or_buffer, encodi
ng, compression, mode, storage_options)
    383 req_info = urllib.request.Request(filepath_or_buffer, headers=storage_options)
--> 384 with urlopen(req_info) as req:
    385     content_encoding = req.headers.get("Content-Encoding", None)

File ~\without_conda\Lib\site-packages\pandas\io\common.py:289, in urlopen(*args, **kwargs)
    287 import urllib.request
--> 289 return urllib.request.urlopen(*args, **kwargs)

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64__qbz5n2kfra8p0\Lib\urllib\requ
est.py:216, in urlopen(url, data, timeout, cafile, capath, cadefault, context)
    215     opener = _opener
--> 216 return opener.open(url, data, timeout)

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64__qbz5n2kfra8p0\Lib\urllib\requ
est.py:525, in OpenerDirector.open(self, fullurl, data, timeout)
    524     meth = getattr(processor, meth_name)
--> 525     response = meth(req, response)
    527 return response

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64__qbz5n2kfra8p0\Lib\urllib\requ
est.py:634, in HTTPErrorProcessor.http_response(self, request, response)
    633 if not (200 <= code < 300):
--> 634     response = self.parent.error(
    635         'http', request, response, code, msg, hdrs)
    637 return response

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64__qbz5n2kfra8p0\Lib\urllib\requ
est.py:563, in OpenerDirector.error(self, proto, *args)
    562 args = (dict, 'default', 'http_error_default') + orig_args
```

```
--> 563 return self._call_chain(*args)

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64__qbz5n2kfra8p0\Lib\urllib\requ
est.py:496, in OpenerDirector._call_chain(self, chain, kind, meth_name, *args)
    495 func = getattr(handler, meth_name)
--> 496 result = func(*args)
    497 if result is not None:

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64__qbz5n2kfra8p0\Lib\urllib\requ
est.py:643, in HTTPDefaultErrorHandler.http_error_default(self, req, fp, code, msg, hdrs)
    642 def http_error_default(self, req, fp, code, msg, hdrs):
--> 643     raise HTTPError(req.full_url, code, msg, hdrs, fp)

HTTPError: HTTP Error 404: Not Found

The above exception was the direct cause of the following exception:

ConnectionError                           Traceback (most recent call last)
Cell In[10], line 3
      1 # 4) Error: unknown series
      2 try:
----> 3     FredAPI("XXXX")
      4 except ValueError as e:
      5     print("Caught expected error:", e)

Cell In[3], line 91, in FredAPI(series_id, dates, start_date, end_date, timeout, session, logger)
     88 _validate_types(series_id, dates, start_date, end_date)
     90 # ---------- NEW: download using your function ----------
---> 91 df = get_fred_series(series_id)
     93 if df.empty:
     94     raise ValueError(f"Series '{series_id}' returned no rows.")

Cell In[3], line 27, in get_fred_series(series_id)
     25     df = pd.read_csv(url)
     26 except Exception as exc:
---> 27     raise ConnectionError(f"Failed to download FRED series '{series_id}'.") from exc
     29 # Convert columns
     30 if "observation_date" not in df or series_id not in df:

ConnectionError: Failed to download FRED series 'XXXX'.
```

In [11]:
```python
# 5) Error: requested dates before inception
try:
    FredAPI("UNRATE", ["1900-01-01"])
except ValueError as e:
    print("Caught expected error:", e)
```

```
Caught expected error: Requested 1 date(s) before series inception (1948-01-01).
```

In [ ]:
```python

```