JPN_CPI_var4

December 9, 2024

1 Import Libraries

[1]:
```
from deep_translator import GoogleTranslator
from scipy.stats import ttest_rel
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import adfuller
from textblob import TextBlob
import matplotlib.dates as mdates
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import re
import warnings

warnings.filterwarnings('ignore', category=UserWarning, module='openpyxl')
```

2 Global Variables

[2]:
```
# Directory Path
DIR_PATH_GOOGLE_TREND = '../data/google_trend'            # Directory that␣
 ↪contains data of google trends
DIR_PATH_RESULT = '../result'                             # Directory that␣
 ↪receives results

# File Paths
FILE_PATH_JPN_CPI = '../data/zmi2020s.csv'                # Monthly Time-Series␣
 ↪of CPI in Japan
FILE_PATH_JGB_10Y = '../data/JGB_RATE_10Y.csv'            # Monthly Time-Series␣
 ↪of 10Y JGB Rate
FILE_PATH_TOPIX = '../data/TOPIX.csv'                     # Monthly Time-Series␣
 ↪of TOPIX
FILE_PATH_USD_JPY = '../data/USD_JPY.csv'                 # Monthly Time-Series␣
 ↪of USD/JPY
```

```
FILE_PATH_UNEMPRT = '../data/UNEMPLOYMENT_RATE.xlsx'      # Monthly Time-Series␣
 ↪of Unemployment Rate in Japan
FILE_PATH_INCOME = '../data/INCOME.xls'                   # Monthly Time-Series␣
 ↪of Income in Japan
FILE_PATH_IIP_PAST = '../data/IIP_PAST.xlsx'              # Monthly Time-Series␣
 ↪of Index of Industry Production from Jan. 1978 to Dec. 2022 in Japan
FILE_PATH_IIP_LAST = '../data/IIP_LAST.xlsx'              # Monthly Time-Series␣
 ↪of Index of Industry Production after Jan. 2018 in Japan
FILE_PATH_CAI = '../data/CONSUMER_ACTIVITY_INDEX.xlsx'    # Monthly Time-Series␣
 ↪of Consumer Activity Index in Japan

# Data Period
YYYYMM_BGN = '201212' # Beginning point (Start of Abenomics)
YYYYMM_END = '202409' # Ending point (Most Recent Available)
```

3 Load Data

3.1 Macroeconomic Variables

[3]:
```
# A function that slices a data frame according to the analysis period
def slice_df(df):
    df['YYYYMM'] = pd.to_datetime(df['YYYYMM'], format='%Y%m')
    is_period = (YYYYMM_BGN<=df['YYYYMM'].dt.strftime('%Y%m')
                 ) & (df['YYYYMM'].dt.strftime('%Y%m')<=YYYYMM_END)
    df = df[is_period]
    df = df.copy()
    df['YYYYMM'] = df['YYYYMM'].dt.strftime('%Y%m')
    return df
```

[4]:
```
# Monthly Time-Series of CPI in Japan

## load data
df_JPN_CPI = pd.read_csv(FILE_PATH_JPN_CPI, encoding='shift-jis')
## Rename the column
df_JPN_CPI.columns = df_JPN_CPI.iloc[0,:]
df_JPN_CPI = df_JPN_CPI.rename(columns={'Group/Item': 'YYYYMM'})
## Delete the unnecessary rows
df_JPN_CPI = df_JPN_CPI.iloc[5:,:]
df_JPN_CPI = slice_df(df_JPN_CPI)
df_JPN_CPI = df_JPN_CPI.reset_index(drop=True)

df_JPN_CPI
```

[4]:
```
0    YYYYMM All items All items, less fresh food All items, less imputed rent  \
0    201212      94.1                        94.8                          92.9
1    201301      94.2                        94.5                          92.9
2    201302      94.0                        94.6                          92.7
```

```
3    201303      94.2                        94.9                          93.0
4    201304      94.5                        95.2                          93.3
..      …        …                           …                             …
137  202405     108.1                       107.5                         109.5
138  202406     108.2                       107.8                         109.6
139  202407     108.6                       108.3                         110.1
140  202408     109.1                       108.7                         110.8
141  202409     108.9                       108.2                         110.4

0   All items, less imputed rent & fresh food  \
0                                        93.6
1                                        93.3
2                                        93.4
3                                        93.8
4                                        94.1
..                                        …
137                                     108.8
138                                     109.2
139                                     109.8
140                                     110.3
141                                     109.7

0   All items, less fresh food and energy  \
0                                    94.5
1                                    94.2
2                                    94.2
3                                    94.5
4                                    94.8
..                                    …
137                                 106.6
138                                 106.6
139                                 106.9
140                                 107.4
141                                 107.5

0   All items, less food (less alcoholic beverages) and energy   Food  \
0                                                    96.2          87.8
1                                                    95.8          89.0
2                                                    95.8          88.1
3                                                    96.2          87.7
4                                                    96.6          87.7
..                                                    …            …
137                                                 103.6         116.8
138                                                 103.6         116.3
139                                                 103.8         116.4
140                                                 104.2         117.6
141                                                 104.2         119.0
```

```
0   Fresh food Food, less fresh food  … Miscellaneous  \
0         79.0                  89.6  …          94.9
1         86.4                  89.6  …          94.8
2         80.8                  89.6  …          94.9
3         78.6                  89.6  …          95.6
4         78.5                  89.5  …          95.7
..         …                    …  …           …
137      123.1                 115.7  …         104.6
138      118.5                 115.9  …         104.8
139      116.4                 116.3  …         104.8
140      120.8                 117.1  …         104.9
141      125.6                 117.8  …         105.1

0   Personal care services Toilet articles Personal effects Tobacco  \
0                     94.4            94.6             80.0    82.5
1                     94.4            94.7             79.5    82.5
2                     94.4            95.2             79.5    82.5
3                     94.4            96.1             83.3    82.5
4                     94.4            96.3             83.4    82.5
..                     …              …                …      …
137                  104.8           102.0            115.8   114.4
138                  104.9           102.0            117.1   114.4
139                  105.0           102.0            116.7   114.4
140                  105.4           101.8            117.0   114.4
141                  105.2           102.7            117.1   114.4

0   Other miscellaneous Energy Expenses for education  \
0                 104.7   98.5                  100.3
1                 104.7   98.7                  100.3
2                 104.7   99.9                  100.3
3                 104.7  100.1                  100.4
4                 104.7  100.4                  100.7
..                  …    …                     …
137               101.6  118.4                  101.3
138               101.7  121.8                  101.3
139               101.8  125.2                  101.4
140               102.0  124.5                  101.3
141               101.8  116.3                  101.3

0   Expenses for culture & recreation Expenses for information & communication
0                                91.6                                    104.7
1                                90.9                                    104.7
2                                90.8                                    104.6
3                                91.7                                    104.6
4                                92.1                                    104.7
..                                …                                      …
```

```
137                          112.1                          73.1
138                          111.1                          73.1
139                          112.3                          73.1
140                          114.8                          73.1
141                          112.5                          73.1

[142 rows x 79 columns]
```

[5]:
```
# Monthly Time-Series of 10Y JGB Rate

## load data
df_JGB_10Y = pd.read_csv(FILE_PATH_JGB_10Y)
## modify the data type
df_JGB_10Y['Date'] = pd.to_datetime(df_JGB_10Y['Date'], format='%m/%d/%Y')
df_JGB_10Y['Date'] = df_JGB_10Y['Date'].dt.strftime('%Y%m')
## rename the column
df_JGB_10Y = df_JGB_10Y.rename(columns={'Date': 'YYYYMM'})
df_JGB_10Y = df_JGB_10Y.sort_values(by='YYYYMM')
df_JGB_10Y = slice_df(df_JGB_10Y)
df_JGB_10Y = df_JGB_10Y.reset_index(drop=True)

df_JGB_10Y
```

[5]:
```
     YYYYMM  Price   Open   High    Low Change %
0    201212  0.802  0.707  0.809  0.690   12.64%
1    201301  0.754  0.797  0.848  0.726   -5.99%
2    201302  0.665  0.753  0.809  0.663  -11.80%
3    201303  0.556  0.664  0.697  0.511  -16.39%
4    201304  0.611  0.567  0.655  0.320    9.89%
..      …    …    …    …    …      …
137  202405  1.075  0.886  1.103  0.855   23.28%
138  202406  1.046  1.067  1.089  0.898   -2.70%
139  202407  1.061  1.073  1.106  1.000    1.43%
140  202408  0.891  1.040  1.058  0.733  -16.02%
141  202409  0.864  0.914  0.935  0.797   -3.03%

[142 rows x 6 columns]
```

[6]:
```
# Monthly Time-Series of TOPIX

## load data
df_TOPIX = pd.read_csv(FILE_PATH_TOPIX)
## modify the data type
df_TOPIX['Date'] = pd.to_datetime(df_TOPIX['Date'], format='%m/%d/%Y')
df_TOPIX['Date'] = df_TOPIX['Date'].dt.strftime('%Y%m')
## rename the column
df_TOPIX = df_TOPIX.rename(columns={'Date': 'YYYYMM'})
```

```
df_TOPIX = df_TOPIX.sort_values(by='YYYYMM')
df_TOPIX = slice_df(df_TOPIX)
df_TOPIX = df_TOPIX.reset_index(drop=True)
## remove comma
df_TOPIX = df_TOPIX.replace({',': '', ' ': ''}, regex=True)

df_TOPIX
```

[6]:

	YYYYMM	Price	Open	High	Low	Vol.	Change %
0	201212	859.80	785.48	861.57	776.83	50.07B	10.02%
1	201301	940.25	876.97	942.08	862.62	67.36B	9.36%
2	201302	975.66	945.54	981.80	930.04	71.40B	3.77%
3	201303	1034.71	971.99	1061.75	971.22	64.70B	6.05%
4	201304	1165.13	1031.75	1176.36	971.33	90.58B	12.60%
..	…	…	…	…	…	…	…
137	202405	2772.49	2727.92	2785.68	2696.05	40.80B	1.07%
138	202406	2809.63	2791.68	2821.86	2692.52	34.34B	1.34%
139	202407	2794.26	2831.63	2946.60	2695.45	41.63B	-0.55%
140	202408	2712.63	2767.44	2768.22	2206.73	47.91B	-2.92%
141	202409	2645.94	2734.04	2743.75	2508.20	37.87B	-2.46%

[142 rows x 7 columns]

[7]:
```
# Monthly Time-Series of USD/JPY

## load data
df_USD_JPY = pd.read_csv(FILE_PATH_USD_JPY)
## modify the data type
df_USD_JPY['Date'] = pd.to_datetime(df_USD_JPY['Date'], format='%m/%d/%Y')
df_USD_JPY['Date'] = df_USD_JPY['Date'].dt.strftime('%Y%m')
## rename the column
df_USD_JPY = df_USD_JPY.rename(columns={'Date': 'YYYYMM'})
df_USD_JPY = df_USD_JPY.sort_values(by='YYYYMM')
df_USD_JPY = slice_df(df_USD_JPY)
df_USD_JPY = df_USD_JPY.reset_index(drop=True)

df_USD_JPY
```

[7]:

	YYYYMM	Price	Open	High	Low	Vol.	Change %
0	201212	86.74	82.37	86.80	81.71	NaN	5.20%
1	201301	91.72	86.75	91.79	86.53	NaN	5.74%
2	201302	92.53	91.71	94.78	90.94	NaN	0.88%
3	201303	94.19	92.56	96.72	92.43	NaN	1.79%
4	201304	97.41	94.21	99.96	92.56	NaN	3.42%
..	…	…	…	…	…	…	…
137	202405	157.31	157.74	158.04	151.87	NaN	-0.31%
138	202406	160.86	157.33	161.30	154.56	NaN	2.26%

```
139  202407  149.98  160.88  162.01  149.60   NaN   -6.76%
140  202408  146.16  149.97  150.89  141.66   NaN   -2.55%
141  202409  143.62  146.18  147.20  139.56   NaN   -1.74%

[142 rows x 7 columns]
```

[8]:
```
# Monthly Time-Series of Unemployment Rate in Japan

## load data
start_date = f"{YYYYMM_BGN[:4]}-{YYYYMM_BGN[4:]}-01"
end_date = f"{YYYYMM_END[:4]}-{YYYYMM_END[4:]}-01"
YYYYMM_list = pd.date_range(start=start_date, end=end_date, freq='MS').
 ↪strftime('%Y%m').tolist()
UNEMPT_list = pd.read_excel(FILE_PATH_UNEMPRT, usecols=[19], skiprows=728).
 ↪iloc[:142,0]
df_UNEMPRT = pd.DataFrame({
    'YYYYMM': YYYYMM_list,
    'Unemployment Rate': UNEMPT_list
})

df_UNEMPRT
```

[8]:
```
     YYYYMM  Unemployment Rate
0    201212                4.3
1    201301                4.2
2    201302                4.3
3    201303                4.1
4    201304                4.1
..      …                   …
137  202405                2.6
138  202406                2.5
139  202407                2.7
140  202408                2.5
141  202409                2.4

[142 rows x 2 columns]
```

[9]:
```
# Monthly Time-Series of Income in Japan

## load data
df_INCOME = pd.read_excel(FILE_PATH_INCOME, header=8)
df_INCOME = df_INCOME.iloc[:35,:]
## adjust format
df_INCOME = df_INCOME.drop(df_INCOME.columns[1:5], axis=1)
df_INCOME.columns = ['YYYY', '01', '02', '03', '04', '05', '06', '07', '08',␣
 ↪'09', '10', '11', '12']
df_INCOME = df_INCOME.melt(id_vars=['YYYY'], var_name='MM', value_name='Income')
```

```
df_INCOME['YYYYMM'] = df_INCOME['YYYY'].astype(str) + df_INCOME['MM'].
 ↪astype(str)
df_INCOME = df_INCOME[['YYYYMM', 'Income']]
df_INCOME = df_INCOME.sort_values(by='YYYYMM')
df_INCOME = slice_df(df_INCOME)
df_INCOME = df_INCOME.reset_index(drop=True)

df_INCOME
```

[9]:

	YYYYMM	Income
0	201212	97.7
1	201301	99
2	201302	98.8
3	201303	98.5
4	201304	98.8
..	…	…
137	202405	106.9
138	202406	109.2
139	202407	106.7
140	202408	106
141	202409	105.9

[142 rows x 2 columns]

[10]:
```
# Monthly Time-Series of Index of Industry Production in Japan

## Load data
### Past Series
df_IIP_PAST = pd.read_excel(FILE_PATH_IIP_PAST, header=2)
df_IIP_PAST = df_IIP_PAST.iloc[1:,1:3]
df_IIP_PAST.columns = ['YYYYMM', 'Indices of Industrial Production']
df_IIP_PAST = df_IIP_PAST.reset_index(drop=True)
### LAST Series
df_IIP_LAST = pd.read_excel(FILE_PATH_IIP_LAST, sheet_name=' ', header=2)
YYYYMM_list = df_IIP_LAST.columns[3:]
IIP_list = df_IIP_LAST.iloc[0, 3:]
df_IIP_LAST = pd.DataFrame({
    'YYYYMM': YYYYMM_list,
    'Indices of Industrial Production': IIP_list
})
df_IIP_LAST = df_IIP_LAST.reset_index(drop=True)

## merge the above two dataframes
df_IIP = pd.concat([df_IIP_PAST, df_IIP_LAST], axis=0)
df_IIP = df_IIP.drop_duplicates()
df_IIP = slice_df(df_IIP)
df_IIP = df_IIP.reset_index(drop=True)
```

```
df_IIP
```

[10]:
```
     YYYYMM Indices of Industrial Production
0    201212                            106.6
1    201301                            104.8
2    201302                            106.7
3    201303                            108.0
4    201304                            108.0
..      …                                …
137  202405                            104.4
138  202406                            100.0
139  202407                            103.1
140  202408                             99.7
141  202409                            101.3

[142 rows x 2 columns]
```

[11]:
```
# Monthly Time-Series of Consumer Activity Index in Japan

## load data
df_CAI = pd.read_excel(FILE_PATH_CAI, header=3)
df_CAI = df_CAI.iloc[1:,:]
# adjust format
df_CAI['Monthly'] = df_CAI['Monthly'].dt.strftime('%Y%m')
df_CAI = df_CAI.rename(columns={'Monthly': 'YYYYMM'})
df_CAI = slice_df(df_CAI)
df_CAI = df_CAI.reset_index(drop=True)

df_CAI
```

[11]:
```
     YYYYMM Nominal Consumption Activity Index  \
0    201212                           95.346165
1    201301                           95.899889
2    201302                           96.317619
3    201303                           96.571185
4    201304                           96.416837
..      …                                  …
137  202405                          108.657057
138  202406                          110.141225
139  202407                          110.411059
140  202408                          110.388744
141  202409                          109.967813

    Real Consumption Activity Index  \
0                         99.438676
1                        100.405992
2                        100.673723
```

```
3                        100.924339
4                        100.861526
..                              …
137                       98.199333
138                       98.958123
139                       99.347237
140                       98.919465
141                       98.810817

    Nominal Consumption Activity Index (travel balance adjusted)  \
0                                                    96.172959
1                                                    96.741508
2                                                    97.126255
3                                                    97.339891
4                                                    97.144409
..                                                         …
137                                                 106.376507
138                                                 107.772747
139                                                 108.428021
140                                                 108.665879
141                                                 108.185596

    Real Consumption Activity Index (travel balance adjusted)  \
0                                                   100.300797
1                                                   101.286993
2                                                   101.518766
3                                                   101.727531
4                                                   101.622474
..                                                         …
137                                                  96.138119
138                                                   96.82997
139                                                  97.562753
140                                                  97.375446
141                                                   97.20926

    Real Consumption Activity Index Plus Real Durable Goods Index  \
0                                 99.52268                   95.4098
1                                100.464707                101.029408
2                                100.719826                100.839247
3                                101.018245                100.085517
4                                100.922659                 103.35731
..                                      …                         …
137                               98.675243                104.207951
138                               99.433161                107.551951
139                               99.912951                109.939429
140                               99.464886                105.925415
141                               99.333336                108.636816
```

```
    Real Non-Durable Goods Index Real Services Index
0                     102.928572           97.734098
1                     102.553599           98.699957
2                     102.692007           99.153121
3                     103.681643           99.085065
4                     102.689734           98.998196
..                           …                   …
137                    92.522973          101.687291
138                    93.032216          102.193784
139                    93.041138          102.537605
140                    93.659744          101.900037
141                    91.822356          102.680612

[142 rows x 9 columns]
```

3.2 Google Trend

- I asked chatGPT to list words related to Japan's CPI, and then obtained Google Trends for each word. In previous research, I found a method of using the Google search suggestion function to create a group of words that are highly correlated with the target word.

[12]:
```python
# Initialize a dictionary to store DataFrames with file names as keys
dataframes = {}

# List of subfolders to skip
skip_folders = ['old']

# Walk through the folder and find all CSV files
for root, dirs, files in os.walk(DIR_PATH_GOOGLE_TREND):
    # Exclude specified folders
    dirs[:] = [d for d in dirs if d not in skip_folders]

    for file in files:
        if file.endswith('.csv'):
            file_path = os.path.join(root, file)
            file_name = os.path.basename(file).replace('.csv', '')  # Use the␣
 ↪file name without extension
            try:
                # Read the CSV file, assuming the first column is the key
                df = pd.read_csv(file_path, index_col=0, skiprows=2)
                df.columns = [file_name]  # Rename columns to the file name
                dataframes[file_name] = df
                print(f"Loaded: {file_name}")
            except Exception as e:
                print(f"Error loading {file_name}: {e}")
```

```python
# Merge all DataFrames on their index (first column as the key)
df_google_trends = None
for name, df in dataframes.items():
    if df_google_trends is None:
        df_google_trends = df
    else:
        df_google_trends = df_google_trends.join(df, how='outer')

# reset index
df_google_trends = df_google_trends.reset_index()
df_google_trends.insert(0, 'YYYYMM', df_google_trends[' '].str.replace('-', '',␣
 ↪regex=False))
df_google_trends = df_google_trends.drop(columns=[' '])

# Display the merged DataFrame
print(df_google_trends.head())
```

```
Loaded: Consumption Tax
Loaded: Cost of Living
Loaded: Daily Goods Prices
Loaded: Disposable Income
Loaded: Household Expenditure
Loaded: Inflation Rate
Loaded: Purchasing Power
Loaded: Real Income
Loaded: Service Prices
Loaded: Consumer Price Index
Loaded: Core Consumer Price Index
Loaded: Deflation
Loaded: Inflation
Loaded: Monetary Base
Loaded: Monetary Policy
Loaded: Nominal GDP
Loaded: Price Fluctuation
Loaded: Producer Price Index
Loaded: Real GDP
Loaded: Energy Prices
Loaded: Essential Goods
Loaded: Exchange Rate
Loaded: Import Prices
Loaded: Oil Prices
Loaded: Raw Material Prices
Loaded: Real Estate Prices
Loaded: Wage Trends
Loaded: Yen Appreciation
Loaded: Yen Depreciation
Loaded: Balance of Payments
Loaded: Global Competitiveness
```

```
Loaded: Global Economy
Loaded: Trade Balance
Loaded: Bank of Japan
Loaded: Economic Trend Index
Loaded: Fiscal Policy
Loaded: Government Expenditure
Loaded: Interest Rates
Loaded: Liquidity Provision
Loaded: Long-term Interest Rates
Loaded: Quantitative Easing
Loaded: Short-term Interest Rates
Loaded: Aging Society
Loaded: Change in Consumer Behavior
Loaded: Declining Birthrate
Loaded: Immigration Policy
Loaded: Labor Market
Loaded: Real Index
Loaded: Seasonally Adjusted Values
Loaded: Statistical Data
Loaded: Statistics Bureau of Japan
Loaded: Time Series Analysis
   YYYYMM  Consumption Tax  Cost of Living  Daily Goods Prices  \
0  201212              14               0                   0
1  201301              16               0                   0
2  201302              18               0                   0
3  201303              20               0                   0
4  201304              19               0                   0

   Disposable Income  Household Expenditure  Inflation Rate  Purchasing Power  \
0                 44                     60              39                50
1                 49                     86              76               100
2                 38                     64              33                70
3                 41                     61              28                50
4                 35                     68              39                61

   Real Income  Service Prices  …  Aging Society  \
0            0              90  …             53
1           43              94  …             82
2            0              93  …             64
3            0              93  …             39
4            0              89  …             56

   Change in Consumer Behavior  Declining Birthrate  Immigration Policy  \
0                            0                   26                  21
1                            0                   33                  22
2                            0                   18                  21
3                            0                   13                  17
4                            0                   21                  12
```

```
   Labor Market  Real Index  Seasonally Adjusted Values  Statistical Data  \
0            51           0                           0                65
1            90          50                           0                74
2            37           0                           0                60
3            38           0                           0                48
4            44           0                           0                52

   Statistics Bureau of Japan  Time Series Analysis
0                          74                    61
1                          78                    38
2                          61                    25
3                          47                    41
4                          71                    34

[5 rows x 53 columns]
```

[13]:
```python
# create dataframe to analyze

## merge CPI and JGB rate
df = pd.merge(
    df_JPN_CPI[['YYYYMM', 'All items', 'All items, less fresh food', 'All␣
 ↪items, less fresh food and energy']],
    df_JGB_10Y[['YYYYMM', 'Price']],
    on='YYYYMM', how='left'
)
## change columns names
df = df.rename(columns={
    'All items': 'CPI',
    'All items, less fresh food': 'Core CPI',
    'All items, less fresh food and energy': 'Core Core CPI',
    'Price': '10Y JGB Rate'
})

## merge df and TOPIX
df = pd.merge(df, df_TOPIX[['YYYYMM', 'Price']], on='YYYYMM', how='left')
## change columns names
df = df.rename(columns={'Price': 'TOPIX'})

## merge df and USD/JPY
df = pd.merge(df, df_USD_JPY[['YYYYMM', 'Price']], on='YYYYMM', how='left')
## change columns names
df = df.rename(columns={'Price': 'USD/JPY'})

## merge df and Unemployment Rate
df = pd.merge(df, df_UNEMPRT, on='YYYYMM', how='left')
```

```
## merge df and Income
df = pd.merge(df, df_INCOME, on='YYYYMM', how='left')

## merge df and IIP
df = pd.merge(df, df_IIP, on='YYYYMM', how='left')

## merge df and CAI
df = pd.merge(df, df_CAI, on='YYYYMM', how='left')

## merge df and Google Trends
df = pd.merge(df, df_google_trends, on='YYYYMM', how='left')

## convert any data type into numeric
df = df.apply(pd.to_numeric, errors='coerce')

## set index
df.set_index('YYYYMM', inplace=True)

df.head()
```

[13]:

YYYYMM	CPI	Core CPI	Core Core CPI	10Y JGB Rate	TOPIX	USD/JPY	\
201212	94.1	94.8	94.5	0.802	859.80	86.74	
201301	94.2	94.5	94.2	0.754	940.25	91.72	
201302	94.0	94.6	94.2	0.665	975.66	92.53	
201303	94.2	94.9	94.5	0.556	1034.71	94.19	
201304	94.5	95.2	94.8	0.611	1165.13	97.41	

YYYYMM	Unemployment Rate	Income	Indices of Industrial Production	\
201212	4.3	97.7	106.6	
201301	4.2	99.0	104.8	
201302	4.3	98.8	106.7	
201303	4.1	98.5	108.0	
201304	4.1	98.8	108.0	

YYYYMM	Nominal Consumption Activity Index	…	Aging Society	\
		…		
201212	95.346165	…	53	
201301	95.899889	…	82	
201302	96.317619	…	64	
201303	96.571185	…	39	
201304	96.416837	…	56	

YYYYMM	Change in Consumer Behavior	Declining Birthrate	Immigration Policy	\
201212	0	26	21	

```
201301                          0                  33                  22
201302                          0                  18                  21
201303                          0                  13                  17
201304                          0                  21                  12

        Labor Market  Real Index  Seasonally Adjusted Values  \
YYYYMM
201212           51           0                           0
201301           90          50                           0
201302           37           0                           0
201303           38           0                           0
201304           44           0                           0

        Statistical Data  Statistics Bureau of Japan  Time Series Analysis
YYYYMM
201212                65                          74                    61
201301                74                          78                    38
201302                60                          61                    25
201303                48                          47                    41
201304                52                          71                    34

[5 rows x 69 columns]
```

4 Summary Statistics

4.0.1 Static Summary Statistics

[14]:
```
# Define categories and other series
categories = {
    "Consumption and Living": sorted([
        "Consumption Tax", "Cost of Living", "Daily Goods Prices", "Disposable␣
↪Income",
        "Household Expenditure", "Inflation Rate", "Purchasing Power", "Real␣
↪Income",
        "Service Prices"
    ]),
    "Economic Indicators": sorted([
        "Consumer Price Index", "Core Consumer Price Index", "Deflation",␣
↪"Inflation",
        "Monetary Base", "Monetary Policy", "Nominal GDP", "Price Fluctuation",
        "Producer Price Index", "Real GDP"
    ]),
    "Government and Monetary Policies": sorted([
        "Bank of Japan", "Economic Trend Index", "Fiscal Policy", "Government␣
↪Expenditure",
```

```
        "Interest Rates", "Liquidity Provision", "Long-term Interest Rates",␣
 ↪"Quantitative Easing",
        "Short-term Interest Rates"
    ]),
    "International Influences": sorted([
        "Balance of Payments", "Global Competitiveness", "Global Economy",␣
 ↪"Trade Balance"
    ]),
    "Price Influences": sorted([
        "Energy Prices", "Essential Goods", "Exchange Rate", "Import Prices",
        "Oil Prices", "Raw Material Prices", "Real Estate Prices", "Wage␣
 ↪Trends",
        "Yen Appreciation", "Yen Depreciation"
    ]),
    "Social Factors": sorted([
        "Aging Society", "Change in Consumer Behavior", "Declining Birthrate",
        "Immigration Policy", "Labor Market"
    ]),
    "Statistics Related": sorted([
        "Real Index", "Seasonally Adjusted Values", "Statistics Bureau of␣
 ↪Japan",
        "Statistical Data", "Time Series Analysis"
    ])
}

# Define other series
other_part_1 = [
    "CPI", "Core CPI", "Core Core CPI", "10Y JGB Rate", "TOPIX", "USD/JPY",
    "Income", "Indices of Industrial Production", "Unemployment Rate"
]
other_part_2 = sorted([
    series for series in df.columns if series not in set().union(*categories.
 ↪values()) and series not in other_part_1
])

# Add "Other" categories to the dictionary
categories["Macroeconomic Variables (Part 1)"] = other_part_1
categories["Macroeconomic Variables (Part 2)"] = other_part_2

# Define Units
units = {
    '10Y JGB Rate': '%',
    'Unemployment Rate': '%'
}
```

[15]:
```
# Function to compute and save static summary statistics for each category
def generate_static_sumstat(df, categories, output_dir=DIR_PATH_RESULT+"/
↪static_sumstat"):
    os.makedirs(output_dir, exist_ok=True)

    for category, series_list in categories.items():
        # Extract relevant columns
        category_data = df[series_list].dropna(axis=1, how='all')  # Drop␣
↪columns with all NaN

        if category_data.empty:
            print(f"No data available for category: {category}")
            continue

        # Calculate basic statistics and transpose
        stats = category_data.describe().transpose()

        # Ensure 'count' is formatted as an integer
        stats['count'] = stats['count'].astype(int)

        # Save as LaTeX
        output_file = os.path.join(output_dir, f"{category.replace(' ',␣
↪'_')}_statistics.tex")
        latex_table = stats.to_latex(
            index=True,
            float_format="%.1f",
            na_rep="N/A",
            caption=f"Basic Statistics for {category}".replace("%", r"\%"),
            label=f"tab:statistics_{category.replace(' ', '_')}".replace("%",␣
↪r"\%"),
            escape=False  # Avoid automatic escaping to allow custom escapes
        )

        # Replace '%' with '\%' in the entire LaTeX table
        latex_table = latex_table.replace("%", r"\%")

        # Create the complete LaTeX file content
        latex_content = f"""
        \\documentclass[a4paper,12pt]{{article}}
        \\usepackage[utf8]{{inputenc}}
        \\usepackage[table]{{xcolor}}
        \\usepackage{{booktabs}}
        \\usepackage{{lscape}}

        \\begin{{document}}

        \\begin{{landscape}}
```

```
        {latex_table}
        \\end{{landscape}}

        \\end{{document}}
        """

        # Write to file
        with open(output_file, "w") as f:
            f.write(latex_content)
        print(f"Statistics for {category} saved to {output_file}")

generate_static_sumstat(df, categories)
```

```
Statistics for Consumption and Living saved to
../result/static_sumstat\Consumption_and_Living_statistics.tex
Statistics for Economic Indicators saved to
../result/static_sumstat\Economic_Indicators_statistics.tex
Statistics for Government and Monetary Policies saved to
../result/static_sumstat\Government_and_Monetary_Policies_statistics.tex
Statistics for International Influences saved to
../result/static_sumstat\International_Influences_statistics.tex
Statistics for Price Influences saved to
../result/static_sumstat\Price_Influences_statistics.tex
Statistics for Social Factors saved to
../result/static_sumstat\Social_Factors_statistics.tex
Statistics for Statistics Related saved to
../result/static_sumstat\Statistics_Related_statistics.tex
Statistics for Macroeconomic Variables (Part 1) saved to
../result/static_sumstat\Macroeconomic_Variables_(Part_1)_statistics.tex
Statistics for Macroeconomic Variables (Part 2) saved to
../result/static_sumstat\Macroeconomic_Variables_(Part_2)_statistics.tex
```

4.0.2 Dynamic Summary Statistics

[16]:
```
def sanitize_filename(filename):
    """
    Sanitize a filename by replacing invalid characters and spaces with␣
 ↪underscores.
    """
    # Replace spaces with underscores and sanitize invalid characters
    return re.sub(r'[\/\\\:\*\?\"\<\>\| ]', '_', filename)

def generate_dynamic_sumstat(df, categories, units=None, interval_months=12,␣
 ↪start_label_date=None, output_dir=DIR_PATH_RESULT+'/dynamic_sumstat'):
    """
    Plot time series for each column in the DataFrame, grouped by categories.
```

```
    Each category is stored in a separate folder with sanitized folder and file␣
↪names.
    Labels on the x-axis start from a specific date if provided.

    Parameters:
    - df: DataFrame containing time series data
    - categories: Dictionary where keys are category names and values are lists␣
↪of series (column names)
    - units: Dictionary where keys are series names and values are their␣
↪corresponding y-axis units (e.g., "%", "Billion Yen")
    - interval_months: Interval of months for x-axis tick marks
    - start_label_date: Specific date (`str` in 'YYYY-MM-DD' format) to start␣
↪showing x-axis labels
    - output_dir: Directory to save the plots
    """
    for category, series_list in categories.items():
        # Sanitize the folder name for the category
        sanitized_category = sanitize_filename(category)
        category_dir = os.path.join(output_dir, sanitized_category)
        os.makedirs(category_dir, exist_ok=True)

        for series in series_list:
            if series not in df.columns:
                print(f"Series '{series}' not found in DataFrame. Skipping.")
                continue

            # Extract data for the current series
            series_data = df[series].dropna().copy()  # Drop NaN values
            if series_data.empty:
                print(f"No data available for series: {series}")
                continue

            # Ensure the index is a datetime object
            series_data.index = pd.to_datetime(series_data.index, format='%Y%m')
            series_data.index = pd.date_range(start=series_data.index[0],␣
↪periods=len(df), freq='ME')

            # Create a plot
            plt.figure(figsize=(14, 6))
            plt.plot(series_data.index, series_data.values, label=series,␣
↪color="blue")
            plt.title(f"Time Series Trend: {series}", fontsize=16)

            # Set y-axis label with unit if available
            if units and series in units:
                y_label = f"Value ({units[series]})"
```

```
            else:
                y_label = "Value"
            plt.ylabel(y_label, fontsize=14)

            # Format x-axis
            plt.xlabel("Time (YYYY-MM)", fontsize=14)
            plt.grid(True)
            plt.legend(loc="best", fontsize=10)
            plt.xticks(rotation=45)

            # Determine start and end date for x-axis
            start_date = series_data.index.min()
            end_date = series_data.index.max()

            # If `start_label_date` is provided, override the x-axis starting␣
 ↪point for labels
            if start_label_date:
                start_label_date = pd.to_datetime(start_label_date)
                if start_label_date > start_date:
                    start_date = start_label_date

            # Format x-axis ticks as YYYY-MM
            ticks = pd.date_range(start=start_date, end=end_date,␣
 ↪freq=f'{interval_months}ME')
            plt.gca().set_xticks(ticks)  # Set custom ticks
            plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m')) ␣
 ↪# Format tick labels as YYYY-MM

            # Align x-axis labels to the adjusted date range
            plt.xlim(series_data.index.min(), end_date)

            # Sanitize the file name
            sanitized_series = sanitize_filename(series)

            # Save the plot to the category's folder
            output_file = os.path.join(category_dir, f"{sanitized_series}.pdf")
            plt.savefig(output_file, dpi=300, bbox_inches="tight")
            print(f"Plot saved for series: {series} -> {output_file}")

            # Close the plot to free memory
            plt.close()

generate_dynamic_sumstat(df, categories, units=units,␣
 ↪start_label_date="2013-01-1")
```

```
Plot saved for series: Consumption Tax ->
../result/dynamic_sumstat\Consumption_and_Living\Consumption_Tax.pdf
```

```
Plot saved for series: Cost of Living ->
../result/dynamic_sumstat\Consumption_and_Living\Cost_of_Living.pdf
Plot saved for series: Daily Goods Prices ->
../result/dynamic_sumstat\Consumption_and_Living\Daily_Goods_Prices.pdf
Plot saved for series: Disposable Income ->
../result/dynamic_sumstat\Consumption_and_Living\Disposable_Income.pdf
Plot saved for series: Household Expenditure ->
../result/dynamic_sumstat\Consumption_and_Living\Household_Expenditure.pdf
Plot saved for series: Inflation Rate ->
../result/dynamic_sumstat\Consumption_and_Living\Inflation_Rate.pdf
Plot saved for series: Purchasing Power ->
../result/dynamic_sumstat\Consumption_and_Living\Purchasing_Power.pdf
Plot saved for series: Real Income ->
../result/dynamic_sumstat\Consumption_and_Living\Real_Income.pdf
Plot saved for series: Service Prices ->
../result/dynamic_sumstat\Consumption_and_Living\Service_Prices.pdf
Plot saved for series: Consumer Price Index ->
../result/dynamic_sumstat\Economic_Indicators\Consumer_Price_Index.pdf
Plot saved for series: Core Consumer Price Index ->
../result/dynamic_sumstat\Economic_Indicators\Core_Consumer_Price_Index.pdf
Plot saved for series: Deflation ->
../result/dynamic_sumstat\Economic_Indicators\Deflation.pdf
Plot saved for series: Inflation ->
../result/dynamic_sumstat\Economic_Indicators\Inflation.pdf
Plot saved for series: Monetary Base ->
../result/dynamic_sumstat\Economic_Indicators\Monetary_Base.pdf
Plot saved for series: Monetary Policy ->
../result/dynamic_sumstat\Economic_Indicators\Monetary_Policy.pdf
Plot saved for series: Nominal GDP ->
../result/dynamic_sumstat\Economic_Indicators\Nominal_GDP.pdf
Plot saved for series: Price Fluctuation ->
../result/dynamic_sumstat\Economic_Indicators\Price_Fluctuation.pdf
Plot saved for series: Producer Price Index ->
../result/dynamic_sumstat\Economic_Indicators\Producer_Price_Index.pdf
Plot saved for series: Real GDP ->
../result/dynamic_sumstat\Economic_Indicators\Real_GDP.pdf
Plot saved for series: Bank of Japan ->
../result/dynamic_sumstat\Government_and_Monetary_Policies\Bank_of_Japan.pdf
Plot saved for series: Economic Trend Index -> ../result/dynamic_sumstat\Governm
ent_and_Monetary_Policies\Economic_Trend_Index.pdf
Plot saved for series: Fiscal Policy ->
../result/dynamic_sumstat\Government_and_Monetary_Policies\Fiscal_Policy.pdf
Plot saved for series: Government Expenditure -> ../result/dynamic_sumstat\Gover
nment_and_Monetary_Policies\Government_Expenditure.pdf
Plot saved for series: Interest Rates ->
../result/dynamic_sumstat\Government_and_Monetary_Policies\Interest_Rates.pdf
Plot saved for series: Liquidity Provision -> ../result/dynamic_sumstat\Governme
nt_and_Monetary_Policies\Liquidity_Provision.pdf
```

```
Plot saved for series: Long-term Interest Rates ->
../result/dynamic_sumstat\Government_and_Monetary_Policies\Long-
term_Interest_Rates.pdf
Plot saved for series: Quantitative Easing -> ../result/dynamic_sumstat\Governme
nt_and_Monetary_Policies\Quantitative_Easing.pdf
Plot saved for series: Short-term Interest Rates ->
../result/dynamic_sumstat\Government_and_Monetary_Policies\Short-
term_Interest_Rates.pdf
Plot saved for series: Balance of Payments ->
../result/dynamic_sumstat\International_Influences\Balance_of_Payments.pdf
Plot saved for series: Global Competitiveness ->
../result/dynamic_sumstat\International_Influences\Global_Competitiveness.pdf
Plot saved for series: Global Economy ->
../result/dynamic_sumstat\International_Influences\Global_Economy.pdf
Plot saved for series: Trade Balance ->
../result/dynamic_sumstat\International_Influences\Trade_Balance.pdf
Plot saved for series: Energy Prices ->
../result/dynamic_sumstat\Price_Influences\Energy_Prices.pdf
Plot saved for series: Essential Goods ->
../result/dynamic_sumstat\Price_Influences\Essential_Goods.pdf
Plot saved for series: Exchange Rate ->
../result/dynamic_sumstat\Price_Influences\Exchange_Rate.pdf
Plot saved for series: Import Prices ->
../result/dynamic_sumstat\Price_Influences\Import_Prices.pdf
Plot saved for series: Oil Prices ->
../result/dynamic_sumstat\Price_Influences\Oil_Prices.pdf
Plot saved for series: Raw Material Prices ->
../result/dynamic_sumstat\Price_Influences\Raw_Material_Prices.pdf
Plot saved for series: Real Estate Prices ->
../result/dynamic_sumstat\Price_Influences\Real_Estate_Prices.pdf
Plot saved for series: Wage Trends ->
../result/dynamic_sumstat\Price_Influences\Wage_Trends.pdf
Plot saved for series: Yen Appreciation ->
../result/dynamic_sumstat\Price_Influences\Yen_Appreciation.pdf
Plot saved for series: Yen Depreciation ->
../result/dynamic_sumstat\Price_Influences\Yen_Depreciation.pdf
Plot saved for series: Aging Society ->
../result/dynamic_sumstat\Social_Factors\Aging_Society.pdf
Plot saved for series: Change in Consumer Behavior ->
../result/dynamic_sumstat\Social_Factors\Change_in_Consumer_Behavior.pdf
Plot saved for series: Declining Birthrate ->
../result/dynamic_sumstat\Social_Factors\Declining_Birthrate.pdf
Plot saved for series: Immigration Policy ->
../result/dynamic_sumstat\Social_Factors\Immigration_Policy.pdf
Plot saved for series: Labor Market ->
../result/dynamic_sumstat\Social_Factors\Labor_Market.pdf
Plot saved for series: Real Index ->
../result/dynamic_sumstat\Statistics_Related\Real_Index.pdf
```

```
Plot saved for series: Seasonally Adjusted Values ->
../result/dynamic_sumstat\Statistics_Related\Seasonally_Adjusted_Values.pdf
Plot saved for series: Statistical Data ->
../result/dynamic_sumstat\Statistics_Related\Statistical_Data.pdf
Plot saved for series: Statistics Bureau of Japan ->
../result/dynamic_sumstat\Statistics_Related\Statistics_Bureau_of_Japan.pdf
Plot saved for series: Time Series Analysis ->
../result/dynamic_sumstat\Statistics_Related\Time_Series_Analysis.pdf
Plot saved for series: CPI ->
../result/dynamic_sumstat\Macroeconomic_Variables_(Part_1)\CPI.pdf
Plot saved for series: Core CPI ->
../result/dynamic_sumstat\Macroeconomic_Variables_(Part_1)\Core_CPI.pdf
Plot saved for series: Core Core CPI ->
../result/dynamic_sumstat\Macroeconomic_Variables_(Part_1)\Core_Core_CPI.pdf
Plot saved for series: 10Y JGB Rate ->
../result/dynamic_sumstat\Macroeconomic_Variables_(Part_1)\10Y_JGB_Rate.pdf
Plot saved for series: TOPIX ->
../result/dynamic_sumstat\Macroeconomic_Variables_(Part_1)\TOPIX.pdf
Plot saved for series: USD/JPY ->
../result/dynamic_sumstat\Macroeconomic_Variables_(Part_1)\USD_JPY.pdf
Plot saved for series: Income ->
../result/dynamic_sumstat\Macroeconomic_Variables_(Part_1)\Income.pdf
Plot saved for series: Indices of Industrial Production -> ../result/dynamic_sum
stat\Macroeconomic_Variables_(Part_1)\Indices_of_Industrial_Production.pdf
Plot saved for series: Unemployment Rate ->
../result/dynamic_sumstat\Macroeconomic_Variables_(Part_1)\Unemployment_Rate.pdf
Plot saved for series: Nominal Consumption Activity Index -> ../result/dynamic_s
umstat\Macroeconomic_Variables_(Part_2)\Nominal_Consumption_Activity_Index.pdf
Plot saved for series: Nominal Consumption Activity Index (travel balance
adjusted) -> ../result/dynamic_sumstat\Macroeconomic_Variables_(Part_2)\Nominal_
Consumption_Activity_Index_(travel_balance_adjusted).pdf
Plot saved for series: Real Consumption Activity Index -> ../result/dynamic_sums
tat\Macroeconomic_Variables_(Part_2)\Real_Consumption_Activity_Index.pdf
Plot saved for series: Real Consumption Activity Index (travel balance adjusted)
-> ../result/dynamic_sumstat\Macroeconomic_Variables_(Part_2)\Real_Consumption_A
ctivity_Index_(travel_balance_adjusted).pdf
Plot saved for series: Real Consumption Activity Index Plus -> ../result/dynamic
_sumstat\Macroeconomic_Variables_(Part_2)\Real_Consumption_Activity_Index_Plus.p
df
Plot saved for series: Real Durable Goods Index -> ../result/dynamic_sumstat\Mac
roeconomic_Variables_(Part_2)\Real_Durable_Goods_Index.pdf
Plot saved for series: Real Non-Durable Goods Index ->
../result/dynamic_sumstat\Macroeconomic_Variables_(Part_2)\Real_Non-
Durable_Goods_Index.pdf
Plot saved for series: Real Services Index -> ../result/dynamic_sumstat\Macroeco
nomic_Variables_(Part_2)\Real_Services_Index.pdf
```

5 Conduct VAR Analysis

[17]:
```
# normalize each columns
scaler = StandardScaler()
df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns, index=df.index)
```

[18]:
```
# conduct VAR

# Step 1: Fix index
df.index = pd.to_datetime(df.index.astype(str), format='%Y%m')
df.index = pd.date_range(start=df.index[0], periods=len(df), freq='ME')

# Step 2: Dynamic forecasting evaluation with and without specific columns
def dynamic_forecasting_evaluation(df, exclude_columns=None,␣
 ↪train_end_date='2021-01-31'):
    """
    Perform dynamic forecasting evaluation with or without specific columns.
    """
    if exclude_columns:
        # Drop specific columns if specified
        df_filtered = df.drop(columns=[col for col in exclude_columns if col in␣
 ↪df.columns])
    else:
        df_filtered = df.copy()

    # Split into training and testing sets
    train_size = df_filtered.index.get_loc(train_end_date)  # Use data before␣
 ↪train_end_date for training
    train_data = df_filtered.iloc[:train_size]
    test_data = df_filtered.iloc[train_size:]

    # Fit VAR model on training data
    model = VAR(train_data)
    results = model.fit(maxlags=2)

    # Perform dynamic forecasting
    forecast_values = []
    for i in range(len(test_data)):
        # Create a temporary dataset for iterative forecasting
        temp_data = pd.concat([train_data, test_data.iloc[:i]])  # Update with␣
 ↪observed data
        forecast = results.forecast(temp_data.values[-results.k_ar:], steps=1)
        forecast_values.append(forecast[0])  # Collect the first-step forecast

    # Convert forecast values into a DataFrame
    forecast_dynamic = pd.DataFrame(forecast_values, index=test_data.index,␣
 ↪columns=test_data.columns)
```

```
    # Evaluate forecast accuracy
    metrics = {}
    for col in test_data.columns:
        actual = test_data[col]
        predicted = forecast_dynamic[col]

        mae = mean_absolute_error(actual, predicted)
        mse = mean_squared_error(actual, predicted)
        rmse = np.sqrt(mse)

        metrics[col] = {
            'MAE': mae,
            'MSE': mse,
            'RMSE': rmse
        }

    return forecast_dynamic, test_data, metrics

# Perform evaluation with specific columns included
include_forecast, include_test, include_metrics =␣
 ↪dynamic_forecasting_evaluation(df)

# Perform evaluation with specific columns excluded
google_trend_columns = [col for col in df_google_trends.columns if col not in␣
 ↪'YYYYMM'] # google trend columns
exclude_columns = google_trend_columns
exclude_forecast, exclude_test, exclude_metrics =␣
 ↪dynamic_forecasting_evaluation(df, exclude_columns=exclude_columns)

# Step 3: Visualize the dynamic forecast for comparison
def plot_forecast(actual, forecast_include, forecast_exclude, col,␣
 ↪interval_months=6, save_dir=DIR_PATH_RESULT+'/dynamic_results'):
    """
    Plot and save the dynamic forecast comparison for a specific column with␣
 ↪x-axis labels aligned to month-end dates.

    Parameters:
    - actual: DataFrame of actual values
    - forecast_include: DataFrame of forecasted values (with specific columns)
    - forecast_exclude: DataFrame of forecasted values (without specific␣
 ↪columns)
    - col: Column name to plot
    - interval_months: Interval in months for the x-axis labels
    - save_dir: Directory to save the plots
    """
```

```
    os.makedirs(save_dir, exist_ok=True)  # Ensure the save directory exists

    plt.figure(figsize=(10, 8))

    # Plot Observed values
    plt.plot(actual.index, actual[col], label=f'Actual {col}', color='blue')

    # Plot Forecasted values (with selected columns)
    if col in forecast_include.columns:
        plt.plot(forecast_include.index, forecast_include[col],␣
↪label=f'Forecasted {col} (With Google Trends)', color='orange',␣
↪linestyle='dashed')

    # Plot Forecasted values (without selected columns)
    if col in forecast_exclude.columns:
        plt.plot(forecast_exclude.index, forecast_exclude[col],␣
↪label=f'Forecasted {col} (Without Google Trends)', color='green',␣
↪linestyle='dotted')

    # Customize the plot
    plt.title(f'Dynamic Forecast Comparison: {col}')
    plt.xlabel('Time (YYYY-MM)')
    plt.ylabel('Values')
    plt.legend()
    plt.grid()

    # Set x-axis labels to align with month-end dates
    ax = plt.gca()
    locator = mdates.MonthLocator(interval=interval_months)  # Set interval for␣
↪labels
    ax.xaxis.set_major_locator(locator)
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))  # Format␣
↪labels as 'YYYY-MM'

    # Align x-axis labels to start from the minimum date
    start_date = actual.index.min()  # Minimum date from actual data
    end_date = actual.index.max()  # Maximum date from actual data
    ax.set_xlim(start_date, end_date)  # Ensure x-axis aligns with actual data␣
↪range

    # Adjust x-axis ticks to start from the minimum date
    ticks = pd.date_range(start=start_date, end=end_date,␣
↪freq=f'{interval_months}ME')
    plt.xticks(ticks, rotation=45)  # Set custom ticks and rotate for␣
↪readability
```

```
    # Sanitize the file name
    sanitized_col = sanitize_filename(col)

    # Save the plot
    file_path = os.path.join(save_dir, f"{sanitized_col}_forecast_comparison.
 ↪pdf")
    plt.savefig(file_path)
    print(f"Plot saved for {col}: {file_path}")

    # Close the plot
    plt.close()

# Plot for specific columns with sanitized filenames
for col in ['CPI', 'Core CPI', 'Core Core CPI']:
    plot_forecast(
        include_test,
        include_forecast,
        exclude_forecast,
        col,
        interval_months=6  # Set label interval to 6 months
    )
```

```
Plot saved for CPI: ../result/dynamic_results\CPI_forecast_comparison.pdf
Plot saved for Core CPI:
../result/dynamic_results\Core_CPI_forecast_comparison.pdf
Plot saved for Core Core CPI:
../result/dynamic_results\Core_Core_CPI_forecast_comparison.pdf
```

[19]:
```
def generate_grouped_tex_tables_single_table_per_category(include_metrics,␣
 ↪exclude_metrics, output_file_prefix='metrics_comparison_grouped_'):
    """
    Generate a single LaTeX file for each economic category, including split␣
 ↪"Other".

    Parameters:
    - include_metrics: Dictionary of metrics with Google Trends
    - exclude_metrics: Dictionary of metrics without Google Trends
    - output_file_prefix: Prefix for the output LaTeX files
    """
    # Define the economic categories and their corresponding series
    categories = {
        "Consumption and Living": sorted([
            "Consumption Tax", "Cost of Living", "Daily Goods Prices",␣
 ↪"Disposable Income",
            "Household Expenditure", "Inflation Rate", "Purchasing Power",␣
 ↪"Real Income",
            "Service Prices"
```

```
        ]),
        "Economic Indicators": sorted([
            "Consumer Price Index", "Core Consumer Price Index", "Deflation",␣
↪"Inflation",
            "Monetary Base", "Monetary Policy", "Nominal GDP", "Price␣
↪Fluctuation",
            "Producer Price Index", "Real GDP"
        ]),
        "Government and Monetary Policies": sorted([
            "Bank of Japan", "Economic Trend Index", "Fiscal Policy",␣
↪"Government Expenditure",
            "Interest Rates", "Liquidity Provision", "Long-term Interest␣
↪Rates", "Quantitative Easing",
            "Short-term Interest Rates"
        ]),
        "International Influences": sorted([
            "Balance of Payments", "Global Competitiveness", "Global Economy",␣
↪"Trade Balance"
        ]),
        "Price Influences": sorted([
            "Energy Prices", "Essential Goods", "Exchange Rate", "Import␣
↪Prices",
            "Oil Prices", "Raw Material Prices", "Real Estate Prices", "Wage␣
↪Trends",
            "Yen Appreciation", "Yen Depreciation"
        ]),
        "Social Factors": sorted([
            "Aging Society", "Change in Consumer Behavior", "Declining␣
↪Birthrate",
            "Immigration Policy", "Labor Market"
        ]),
        "Statistics Related": sorted([
            "Real Index", "Seasonally Adjusted Values", "Statistics Bureau of␣
↪Japan",
            "Statistical Data", "Time Series Analysis"
        ])
    }

    # Combine all categorized series
    categorized_series = set().union(*categories.values())

    # Define the series for the "Other" category
    other_part_1 = [
        "CPI", "Core CPI", "Core Core CPI", "10Y JGB rate", "TOPIX", "USD/JPY",
        "Income", "Indices of Industrial Production", "Unemployment Rate"
    ]
```

```
    other_part_2 = sorted([series for series in include_metrics.keys() if␣
↪series not in categorized_series and series not in other_part_1])

    # Add "Other" parts to categories
    categories["Macroeconomic Variables (Part 1)"] = other_part_1
    categories["Macroeconomic Variables (Part 2)"] = other_part_2

    # Generate a LaTeX table for each category
    for category, series_list in categories.items():
        rows = []
        for series in series_list:
            include_mae = include_metrics.get(series, {}).get('MAE', 'N/A')
            include_mse = include_metrics.get(series, {}).get('MSE', 'N/A')
            include_rmse = include_metrics.get(series, {}).get('RMSE', 'N/A')

            exclude_mae = exclude_metrics.get(series, {}).get('MAE', 'N/A')
            exclude_mse = exclude_metrics.get(series, {}).get('MSE', 'N/A')
            exclude_rmse = exclude_metrics.get(series, {}).get('RMSE', 'N/A')

            # Append rows for MAE, MSE, and RMSE
            rows.append([f"{series}", 'MAE', include_mae, exclude_mae])
            rows.append(["", 'MSE', include_mse, exclude_mse])  # Series name␣
↪only on MAE row
            rows.append(["", 'RMSE', include_rmse, exclude_rmse])
            rows.append(["\\arrayrulecolor{black!30}\\midrule", "", "", ""])  #␣
↪Add a thin horizontal line after each series

        # Remove the last unnecessary line
        if rows[-1][0] == "\\arrayrulecolor{black!30}\\midrule":
            rows.pop()

        # Create a DataFrame for formatting the LaTeX table
        df = pd.DataFrame(rows, columns=['Series', 'Metric', 'With Google␣
↪Trends', 'Without Google Trends'])

        # Generate LaTeX table with centered columns
        latex_table = df.to_latex(
            index=False,
            float_format="%.3f",
            na_rep="N/A",
            escape=False,  # Keep LaTeX commands like \midrule
            caption=f"{category}",
            label=f"tab:metrics_comparison_{category.replace(' ', '_')}",
            column_format="llcc"  # Align first two columns to the left, last␣
↪two columns centered
        )
```

```
        # Create the complete LaTeX file content
        latex_content = f"""
\\documentclass[a4paper,12pt]{{article}}
\\usepackage[utf8]{{inputenc}}
\\usepackage[table]{{xcolor}}
\\usepackage{{booktabs}}
\\usepackage{{lscape}}

\\begin{{document}}

\\section*{{Comparison of Metrics with and without Google Trends}}

{latex_table}

\\end{{document}}
"""

        # Write the content to a LaTeX file
        output_file = f"{output_file_prefix}{category.replace(' ', '_')}.tex"
        os.makedirs(os.path.dirname(output_file), exist_ok=True)
        with open(output_file, 'w') as f:
            f.write(latex_content)
        print(f"LaTeX file saved for {category}: {output_file}")

output_file_prefix = os.path.join(DIR_PATH_RESULT+'/static_results',␣
 ↪'metrics_comparison_grouped_')
generate_grouped_tex_tables_single_table_per_category(include_metrics,␣
 ↪exclude_metrics, output_file_prefix=output_file_prefix)
```

```
LaTeX file saved for Consumption and Living:
../result/static_results\metrics_comparison_grouped_Consumption_and_Living.tex
LaTeX file saved for Economic Indicators:
../result/static_results\metrics_comparison_grouped_Economic_Indicators.tex
LaTeX file saved for Government and Monetary Policies: ../result/static_results\
metrics_comparison_grouped_Government_and_Monetary_Policies.tex
LaTeX file saved for International Influences:
../result/static_results\metrics_comparison_grouped_International_Influences.tex
LaTeX file saved for Price Influences:
../result/static_results\metrics_comparison_grouped_Price_Influences.tex
LaTeX file saved for Social Factors:
../result/static_results\metrics_comparison_grouped_Social_Factors.tex
LaTeX file saved for Statistics Related:
../result/static_results\metrics_comparison_grouped_Statistics_Related.tex
LaTeX file saved for Macroeconomic Variables (Part 1): ../result/static_results\
metrics_comparison_grouped_Macroeconomic_Variables_(Part_1).tex
LaTeX file saved for Macroeconomic Variables (Part 2): ../result/static_results\
metrics_comparison_grouped_Macroeconomic_Variables_(Part_2).tex
```

[20]:
```
# A function to compare model forecasts
def compare_model_forecasts(actual, forecast1, forecast2):
    """
    Compare the forecast accuracy of two models using statistical tests.
    """
    comparison_results = {}

    # Ensure common columns between all dataframes
    common_columns = set(actual.columns) & set(forecast1.columns) &␣
 ↪set(forecast2.columns)

    for col in common_columns:  # Only iterate over common columns
        actual_col = actual[col]
        pred1_col = forecast1[col]
        pred2_col = forecast2[col]

        # Calculate prediction errors
        error1 = actual_col - pred1_col
        error2 = actual_col - pred2_col

        # Calculate accuracy metrics for both models
        mae1 = mean_absolute_error(actual_col, pred1_col)
        mse1 = mean_squared_error(actual_col, pred1_col)
        rmse1 = np.sqrt(mse1)

        mae2 = mean_absolute_error(actual_col, pred2_col)
        mse2 = mean_squared_error(actual_col, pred2_col)
        rmse2 = np.sqrt(mse2)

        # Perform statistical tests on prediction errors
        t_stat, t_p_value = ttest_rel(error1, error2)  # Paired t-test

        # Save results
        comparison_results[col] = {
            'Model 1': {'MAE': mae1, 'MSE': mse1, 'RMSE': rmse1},
            'Model 2': {'MAE': mae2, 'MSE': mse2, 'RMSE': rmse2},
            't-test': {'t_stat': t_stat, 'p_value': t_p_value}
        }

    return comparison_results

# Example usage
comparison_results = compare_model_forecasts(include_test, include_forecast,␣
 ↪exclude_forecast)

# Display the results
for col, metrics in comparison_results.items():
```

```
    print(f"\nComparison for {col}:")
    print(f"Model 1 - MAE: {metrics['Model 1']['MAE']:.3f}, MSE:␣
↪{metrics['Model 1']['MSE']:.3f}, RMSE: {metrics['Model 1']['RMSE']:.3f}")
    print(f"Model 2 - MAE: {metrics['Model 2']['MAE']:.3f}, MSE:␣
↪{metrics['Model 2']['MSE']:.3f}, RMSE: {metrics['Model 2']['RMSE']:.3f}")
    print(f"t-test - t_stat: {metrics['t-test']['t_stat']}, p_value:␣
↪{metrics['t-test']['p_value']}")
```

```
Comparison for CPI:
Model 1 - MAE: 0.289, MSE: 0.139, RMSE: 0.373
Model 2 - MAE: 0.540, MSE: 0.424, RMSE: 0.651
t-test - t_stat: -8.42863540474261, p_value: 9.8599000277496e-11

Comparison for Nominal Consumption Activity Index (travel balance adjusted):
Model 1 - MAE: 0.871, MSE: 1.170, RMSE: 1.081
Model 2 - MAE: 1.372, MSE: 3.598, RMSE: 1.897
t-test - t_stat: -2.332389022460625, p_value: 0.0243189959736908

Comparison for TOPIX:
Model 1 - MAE: 0.440, MSE: 0.299, RMSE: 0.547
Model 2 - MAE: 0.762, MSE: 1.047, RMSE: 1.023
t-test - t_stat: -2.5636870392469033, p_value: 0.013849739641349834

Comparison for Real Consumption Activity Index:
Model 1 - MAE: 0.995, MSE: 1.509, RMSE: 1.229
Model 2 - MAE: 1.818, MSE: 6.338, RMSE: 2.518
t-test - t_stat: -2.2849444762306783, p_value: 0.027193130417528653

Comparison for Income:
Model 1 - MAE: 0.752, MSE: 0.807, RMSE: 0.898
Model 2 - MAE: 1.184, MSE: 2.502, RMSE: 1.582
t-test - t_stat: -4.530759757968656, p_value: 4.462872998436875e-05

Comparison for Indices of Industrial Production:
Model 1 - MAE: 0.705, MSE: 0.730, RMSE: 0.854
Model 2 - MAE: 2.499, MSE: 12.867, RMSE: 3.587
t-test - t_stat: -3.5730189543048123, p_value: 0.0008701364165546975

Comparison for Real Consumption Activity Index (travel balance adjusted):
Model 1 - MAE: 0.951, MSE: 1.390, RMSE: 1.179
Model 2 - MAE: 1.638, MSE: 5.177, RMSE: 2.275
t-test - t_stat: -2.164629438071008, p_value: 0.03588292081937106

Comparison for Real Consumption Activity Index Plus:
Model 1 - MAE: 1.052, MSE: 1.678, RMSE: 1.295
Model 2 - MAE: 1.956, MSE: 7.484, RMSE: 2.736
```

```
t-test - t_stat: -2.294603421665015, p_value: 0.026584533663275657

Comparison for Nominal Consumption Activity Index:
Model 1 - MAE: 0.817, MSE: 1.061, RMSE: 1.030
Model 2 - MAE: 1.387, MSE: 3.655, RMSE: 1.912
t-test - t_stat: -2.464365910455429, p_value: 0.01770233632569545

Comparison for Core Core CPI:
Model 1 - MAE: 0.431, MSE: 0.269, RMSE: 0.519
Model 2 - MAE: 0.467, MSE: 0.282, RMSE: 0.531
t-test - t_stat: -7.242983236735963, p_value: 5.042463489237791e-09

Comparison for Core CPI:
Model 1 - MAE: 0.270, MSE: 0.110, RMSE: 0.331
Model 2 - MAE: 0.411, MSE: 0.208, RMSE: 0.456
t-test - t_stat: -7.43810692129769, p_value: 2.619514124063843e-09

Comparison for USD/JPY:
Model 1 - MAE: 0.576, MSE: 0.577, RMSE: 0.760
Model 2 - MAE: 1.166, MSE: 2.674, RMSE: 1.635
t-test - t_stat: -4.562771683571068, p_value: 4.024976859181818e-05

Comparison for Real Durable Goods Index:
Model 1 - MAE: 2.145, MSE: 6.941, RMSE: 2.635
Model 2 - MAE: 2.027, MSE: 7.483, RMSE: 2.736
t-test - t_stat: 0.13259691862691014, p_value: 0.8951167125496784

Comparison for Real Non-Durable Goods Index:
Model 1 - MAE: 1.302, MSE: 2.431, RMSE: 1.559
Model 2 - MAE: 2.126, MSE: 8.111, RMSE: 2.848
t-test - t_stat: -3.984798942602014, p_value: 0.000250488354451701

Comparison for Real Services Index:
Model 1 - MAE: 0.776, MSE: 0.946, RMSE: 0.973
Model 2 - MAE: 0.980, MSE: 1.931, RMSE: 1.390
t-test - t_stat: -1.4008611456613738, p_value: 0.16826911042259368

Comparison for 10Y JGB Rate:
Model 1 - MAE: 0.752, MSE: 0.924, RMSE: 0.961
Model 2 - MAE: 0.801, MSE: 0.879, RMSE: 0.938
t-test - t_stat: 1.8461985406937425, p_value: 0.0715972329961246

Comparison for Unemployment Rate:
Model 1 - MAE: 0.695, MSE: 0.647, RMSE: 0.804
Model 2 - MAE: 0.922, MSE: 1.736, RMSE: 1.318
t-test - t_stat: -8.207998658975209, p_value: 2.031780550914765e-10
```

[21]:
```
def save_forecast_comparison_to_tex_split(
    comparison_results,
    output_file_prefix,
    specified_series,
):
    """
    Save the forecast comparison results to two separate LaTeX files:
    one for the specified series and one for the remaining sorted series.

    Parameters:
    - comparison_results: Dictionary of comparison results as returned by␣
 ↪compare_model_forecasts.
    - output_file_prefix: Prefix for the LaTeX file paths.
    - specified_series: List of series to include in the first table,␣
 ↪preserving their order.
    """
    # Separate specified series and other series
    specified_results = {series: comparison_results[series] for series in␣
 ↪specified_series if series in comparison_results}
    other_results = {
        series: metrics
        for series, metrics in comparison_results.items()
        if series not in specified_series
    }

    # Sort other series alphabetically
    other_results = dict(sorted(other_results.items()))

    # Helper function to create LaTeX table content
    def generate_table_content(results, title, label, ordered_series=None):
        rows = []
        series_list = ordered_series if ordered_series else results.keys()
        for series in series_list:
            metrics = results[series]
            model1 = metrics["Model 1"]
            model2 = metrics["Model 2"]
            t_test = metrics["t-test"]

            # Add rows for each series
            rows.append(
                [
                    f"{series}",
                    "Model 1",
                    f"MAE={model1['MAE']:.3f}, MSE={model1['MSE']:.3f},␣
 ↪RMSE={model1['RMSE']:.3f}",
                ]
            )
```

```
            rows.append(
                [
                    "",
                    "Model 2",
                    f"MAE={model2['MAE']:.3f}, MSE={model2['MSE']:.3f},␣
↪RMSE={model2['RMSE']:.3f}",
                ]
            )
            rows.append(
                [
                    "",
                    "t-test",
                    f"t-stat={t_test['t_stat']:.3f}, p-value={t_test['p_value']:
↪.3f}",
                ]
            )
            rows.append(["\\arrayrulecolor{black!30}\\midrule", "", ""])  #␣
↪Thin line

        # Remove the last unnecessary line
        if rows and rows[-1][0] == "\\arrayrulecolor{black!30}\\midrule":
            rows.pop()

        # Create a DataFrame for LaTeX export
        df = pd.DataFrame(
            rows, columns=["Series", "Details", "Metrics"]
        )

        # Generate LaTeX table
        return df.to_latex(
            index=False,
            escape=False,  # Keep LaTeX commands like \midrule
            na_rep="N/A",
            caption=title,
            label=label,
            column_format="llp{10cm}",  # Adjust column width for the Metrics␣
↪column
        )

    # Generate and save the first table
    specified_table = generate_table_content(
        specified_results,
        title="Forecast Comparison for Specified Series",
        label="tab:forecast_comparison_specified",
        ordered_series=specified_series,  # Preserve the order
    )
    specified_file = f"{output_file_prefix}_specified.tex"
```

```
    with open(specified_file, "w") as f:
        f.write(
            f"""
\\documentclass[a4paper,12pt]{{article}}
\\usepackage[utf8]{{inputenc}}
\\usepackage{{booktabs}}
\\usepackage[table]{{xcolor}}

\\begin{{document}}

\\section*{{Forecast Comparison Results}}

{specified_table}

\\end{{document}}
"""
        )
    print(f"LaTeX file saved for specified series: {specified_file}")

    # Generate and save the second table
    other_table = generate_table_content(
        other_results,
        title="Forecast Comparison for Other Series",
        label="tab:forecast_comparison_other",
    )
    other_file = f"{output_file_prefix}_other.tex"
    with open(other_file, "w") as f:
        f.write(
            f"""
\\documentclass[a4paper,12pt]{{article}}
\\usepackage[utf8]{{inputenc}}
\\usepackage{{booktabs}}
\\usepackage[table]{{xcolor}}

\\begin{{document}}

\\section*{{Forecast Comparison Results}}

{other_table}

\\end{{document}}
"""
        )
    print(f"LaTeX file saved for other series: {other_file}")


# Example usage
```

```
specified_series = [
    "CPI",
    "Core CPI",
    "Core Core CPI",
    "10Y JGB Rate",
    "TOPIX",
    "USD/JPY",
    "Income",
    "Indices of Industrial Production",
    "Unemployment Rate",
]
output_file_prefix = os.path.join(DIR_PATH_RESULT+'/static_results',␣
 ↪"forecast_comparison")
save_forecast_comparison_to_tex_split(comparison_results, output_file_prefix,␣
 ↪specified_series)
```

```
LaTeX file saved for specified series:
../result/static_results\forecast_comparison_specified.tex
LaTeX file saved for other series:
../result/static_results\forecast_comparison_other.tex
```