# Gaussian

July 26, 2024

[5]:
```
%pip install fredapi
```

```
Collecting fredapi
  Downloading fredapi-0.5.2-py3-none-any.whl.metadata (5.0 kB)
Collecting pandas (from fredapi)
  Downloading pandas-2.2.2-cp311-cp311-macosx_11_0_arm64.whl.metadata (19 kB)
Collecting numpy>=1.23.2 (from pandas->fredapi)
  Downloading numpy-2.0.1-cp311-cp311-macosx_14_0_arm64.whl.metadata (60 kB)
                           60.9/60.9 kB
4.5 MB/s eta 0:00:00
Requirement already satisfied: python-dateutil>=2.8.2 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
pandas->fredapi) (2.8.2)
Collecting pytz>=2020.1 (from pandas->fredapi)
  Using cached pytz-2024.1-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata>=2022.7 (from pandas->fredapi)
  Using cached tzdata-2024.1-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
python-dateutil>=2.8.2->pandas->fredapi) (1.16.0)
Downloading fredapi-0.5.2-py3-none-any.whl (11 kB)
Downloading pandas-2.2.2-cp311-cp311-macosx_11_0_arm64.whl (11.3 MB)
                         11.3/11.3 MB
4.7 MB/s eta 0:00:0000:0100:01
Downloading numpy-2.0.1-cp311-cp311-macosx_14_0_arm64.whl (5.3 MB)
                         5.3/5.3 MB
9.9 MB/s eta 0:00:00ta 0:00:01
Using cached pytz-2024.1-py2.py3-none-any.whl (505 kB)
Using cached tzdata-2024.1-py2.py3-none-any.whl (345 kB)
Installing collected packages: pytz, tzdata, numpy, pandas, fredapi
Successfully installed fredapi-0.5.2 numpy-2.0.1 pandas-2.2.2 pytz-2024.1
tzdata-2024.1
Note: you may need to restart the kernel to use updated packages.
```

[6]:
```
import pandas as pd
import fredapi
```

[7]:
```python
api_key = '6b7f972da2d1e8428579d50c4ba98047'
fred = fredapi.Fred(api_key = api_key)
```

[8]:
```python
def gen_df(category, series):
    gen_ser = fred.get_series(series, frequency = 'q')
    return pd.DataFrame({'Date': gen_ser.index, category + ' : Billions of␣
 ↪dollars': gen_ser.values})

def merge_dataframes(dataframes, on_column):
    merged_df = dataframes[0]
    for df in dataframes[1:]:
        merged_df = pd.merge(merged_df, df, on = on_column)
    return merged_df

dataframes_list = [
    gen_df('GDP', 'GDP'),
    gen_df('PCE', 'PCE'),
    gen_df('GPDI', 'GPDI'),
    gen_df('NETEXP', 'NETEXP'),
    gen_df('GovTotExp', 'W068RCQ027SBEA')
]

data = merge_dataframes(dataframes_list,'Date')
data
```

[8]:
```
           Date  GDP : Billions of dollars  PCE : Billions of dollars  \
0    1960-01-01                    542.648                      326.4
1    1960-04-01                    541.080                      332.2
2    1960-07-01                    545.604                      332.1
3    1960-10-01                    540.197                      334.0
4    1961-01-01                    545.018                      334.5
..          …                          …                          …
253  2023-04-01                  27063.012                    18419.0
254  2023-07-01                  27610.128                    18679.5
255  2023-10-01                  27956.998                    18914.5
256  2024-01-01                  28269.174                    19142.6
257  2024-04-01                  28629.153                    19377.7

     GPDI : Billions of dollars  NETEXP : Billions of dollars  \
0                        96.476                         2.858
1                        87.096                         3.395
2                        86.377                         4.682
3                        75.963                         5.880
4                        78.378                         5.902
..                          …                           …
253                    4780.290                      -806.093
254                    4915.033                      -779.231
```

```
255                    4954.426                     -783.734
256                    5020.538                     -834.896
257                    5144.567                     -894.362

     GovTotExp : Billions of dollars
0                           144.233
1                           147.417
2                           150.459
3                           153.780
4                           157.254
..                              …
253                        9422.404
254                       10007.677
255                        9700.808
256                        9925.034
257                       10067.640

[258 rows x 6 columns]
```

[11]:

```
%pip install matplotlib
```

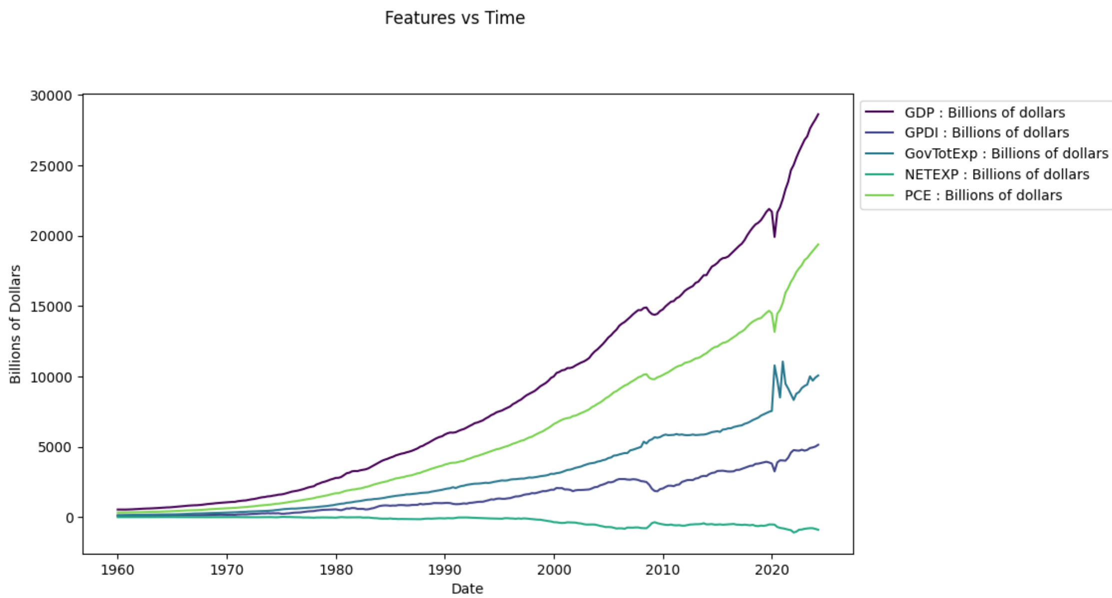
```
Collecting matplotlib
  Downloading matplotlib-3.9.1-cp311-cp311-macosx_11_0_arm64.whl.metadata (11
kB)
Collecting contourpy>=1.0.1 (from matplotlib)
  Using cached contourpy-1.2.1-cp311-cp311-macosx_11_0_arm64.whl.metadata (5.8
kB)
Collecting cycler>=0.10 (from matplotlib)
  Using cached cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.53.1-cp311-cp311-macosx_11_0_arm64.whl.metadata (162
kB)
                           162.6/162.6

kB 5.9 MB/s eta 0:00:00
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Using cached kiwisolver-1.4.5-cp311-cp311-macosx_11_0_arm64.whl.metadata (6.4
kB)
Requirement already satisfied: numpy>=1.23 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
matplotlib) (2.0.1)
Requirement already satisfied: packaging>=20.0 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
matplotlib) (23.2)
Collecting pillow>=8 (from matplotlib)
  Downloading pillow-10.4.0-cp311-cp311-macosx_11_0_arm64.whl.metadata (9.2 kB)
Collecting pyparsing>=2.3.1 (from matplotlib)
  Using cached pyparsing-3.1.2-py3-none-any.whl.metadata (5.1 kB)
```

```
Requirement already satisfied: python-dateutil>=2.7 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
python-dateutil>=2.7->matplotlib) (1.16.0)
Downloading matplotlib-3.9.1-cp311-cp311-macosx_11_0_arm64.whl (7.8 MB)
                         7.8/7.8 MB
5.5 MB/s eta 0:00:0000:0100:01m
Using cached contourpy-1.2.1-cp311-cp311-macosx_11_0_arm64.whl (245 kB)
Using cached cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.53.1-cp311-cp311-macosx_11_0_arm64.whl (2.2 MB)
                         2.2/2.2 MB
8.5 MB/s eta 0:00:00a 0:00:01
Using cached kiwisolver-1.4.5-cp311-cp311-macosx_11_0_arm64.whl (66 kB)
Downloading pillow-10.4.0-cp311-cp311-macosx_11_0_arm64.whl (3.4 MB)
                         3.4/3.4 MB
9.7 MB/s eta 0:00:00ta 0:00:01
Using cached pyparsing-3.1.2-py3-none-any.whl (103 kB)
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools, cycler,
contourpy, matplotlib
Successfully installed contourpy-1.2.1 cycler-0.12.1 fonttools-4.53.1
kiwisolver-1.4.5 matplotlib-3.9.1 pillow-10.4.0 pyparsing-3.1.2
Note: you may need to restart the kernel to use updated packages.
```

[12]:

```python
# Visualization
import matplotlib.pyplot as plt
#separating date column from feature columns
date_column = 'Date'
feature_columns = data.columns.difference([date_column])
#set the plot
fig, ax = plt.subplots(figsize = (10,6))
fig.suptitle('Features vs Time', y = 1.02)

#graphing features onto plot
for i, feature in enumerate(feature_columns):
    ax.plot(data[date_column], data[feature], label = feature, color = plt.cm.
 ↪viridis(i / len(feature_columns)))

#label axis
ax.set_xlabel('Date')
ax.set_ylabel('Billions of Dollars')
ax.legend(loc='upper left', bbox_to_anchor = (1,1))
#display plot
plt.show()
```

```
Matplotlib is building the font cache; this may take a moment.
```




[16]:
```
%pip install statsmodels
```

```
Collecting statsmodels
  Downloading statsmodels-0.14.2-cp311-cp311-macosx_11_0_arm64.whl.metadata (9.2
kB)
Requirement already satisfied: numpy>=1.22.3 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
statsmodels) (2.0.1)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
statsmodels) (1.14.0)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
statsmodels) (2.2.2)
Collecting patsy>=0.5.6 (from statsmodels)
  Downloading patsy-0.5.6-py2.py3-none-any.whl.metadata (3.5 kB)
Requirement already satisfied: packaging>=21.3 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
statsmodels) (23.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
pandas!=2.1.0,>=1.4->statsmodels) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
```

```
Requirement already satisfied: six in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
patsy>=0.5.6->statsmodels) (1.16.0)
Downloading statsmodels-0.14.2-cp311-cp311-macosx_11_0_arm64.whl (10.1 MB)
                         10.1/10.1 MB
15.4 MB/s eta 0:00:00 0:00:01
Downloading patsy-0.5.6-py2.py3-none-any.whl (233 kB)
                         233.9/233.9 kB
17.2 MB/s eta 0:00:00
Installing collected packages: patsy, statsmodels
Successfully installed patsy-0.5.6 statsmodels-0.14.2
Note: you may need to restart the kernel to use updated packages.
```

[17]:

```python
from statsmodels.tsa.stattools import adfuller
#iterating through each feature
for column in data.columns:
    if column!= 'Date':
        result = adfuller(data[column])
        print(f"ADF statistic for {column}: {result[0]}")
        print(f"P-value for {column}: {result[1]}")
        print("Critical Values:")
        for key, value in result[4].items():
            print(f"{key}: {value}")
# creating separation line between each feature
        print("\n" + "=" * 40 + "\n")
```

```
ADF statistic for GDP : Billions of dollars: 7.435656436752694
P-value for GDP : Billions of dollars: 1.0
Critical Values:
1%: -3.4561550092339512
5%: -2.8728972266578676
10%: -2.5728222369384763

========================================

ADF statistic for PCE : Billions of dollars: 6.8778373692584145
P-value for PCE : Billions of dollars: 1.0
Critical Values:
1%: -3.4561550092339512
5%: -2.8728972266578676
10%: -2.5728222369384763

========================================

ADF statistic for GPDI : Billions of dollars: 2.992625554816609
P-value for GPDI : Billions of dollars: 1.0
Critical Values:
1%: -3.4560535712549925
```

```
5%: -2.8728527662442334
10%: -2.5727985212493754

========================================

ADF statistic for NETEXP : Billions of dollars: 0.08039679949897652
P-value for NETEXP : Billions of dollars: 0.9646969244646914
Critical Values:
1%: -3.4564641849494113
5%: -2.873032730098417
10%: -2.572894516864816

========================================

ADF statistic for GovTotExp : Billions of dollars: 2.7872131183908557
P-value for GovTotExp : Billions of dollars: 1.0
Critical Values:
1%: -3.4577787098622674
5%: -2.873608704758507
10%: -2.573201765981991

========================================
```

[26]:
```
# differencing and storing original dataset
data_diff = data.drop('Date', axis = 1).diff().dropna()
#printing ADF test for new dataset
for column in data_diff.columns:
    result = adfuller(data_diff[column])
    print(f"ADF Statistic for {column}: {result[0]}")
    print(f"P-value for {column}: {result[1]}")
    print("Critical Values:")
    for key, value in result[4].items():
        print(f" {key}: {value}")

    print("\n" + "=" * 40 + "\n")
```

```
ADF Statistic for GDP : Billions of dollars: -3.886345364333152
P-value for GDP : Billions of dollars: 0.002137667738212588
Critical Values:
 1%: -3.4565688966099373
 5%: -2.8730786194395455
 10%: -2.5729189953388762

========================================

ADF Statistic for PCE : Billions of dollars: -6.039053627599207
P-value for PCE : Billions of dollars: 1.3598172420763878e-07
```

```
Critical Values:
 1%: -3.456360306409983
 5%: -2.8729872043802356
 10%: -2.572870232500465

==========================================

ADF Statistic for GPDI : Billions of dollars: -14.608670267780347
P-value for GPDI : Billions of dollars: 4.0558964179422356e-27
Critical Values:
 1%: -3.4561550092339512
 5%: -2.8728972266578676
 10%: -2.5728222369384763

==========================================

ADF Statistic for NETEXP : Billions of dollars: -8.804306346787321
P-value for NETEXP : Billions of dollars: 2.071671900401648e-14
Critical Values:
 1%: -3.4564641849494113
 5%: -2.873032730098417
 10%: -2.572894516864816

==========================================

ADF Statistic for GovTotExp : Billions of dollars: -3.108588974313343
P-value for GovTotExp : Billions of dollars: 0.025924060990244243
Critical Values:
 1%: -3.457664132155201
 5%: -2.8735585105960224
 10%: -2.5731749894132916

==========================================
```

[30]:

```
from statsmodels.tsa.stattools import grangercausalitytests
lags = [6,9,1,1]

for column, lag in zip(data_diff.columns, lags):
    df_new = data_diff[['GDP : Billions of dollars', column]]
    print(f'For: {column}')
    gc_res = grangercausalitytests(df_new, lag)
    print("\n" + "=" * 40 + "\n")
```

```
For: GDP : Billions of dollars

Granger Causality
number of lags (no zero) 1
```

```
ssr based F test:         F=-0.0000 , p=1.0000  , df_denom=254, df_num=1
ssr based chi2 test:   chi2=-0.0000 , p=1.0000  , df=1
likelihood ratio test: chi2=-0.0000 , p=1.0000  , df=1
parameter F test:         F=2.1900  , p=0.1401  , df_denom=254, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.0000  , p=1.0000  , df_denom=252, df_num=2
ssr based chi2 test:   chi2=0.0000  , p=1.0000  , df=2
likelihood ratio test: chi2=-0.0000 , p=1.0000  , df=2
parameter F test:         F=7.8050  , p=0.0005  , df_denom=252, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=-0.0000 , p=1.0000  , df_denom=250, df_num=3
ssr based chi2 test:   chi2=-0.0000 , p=1.0000  , df=3
likelihood ratio test: chi2=-0.0000 , p=1.0000  , df=3
parameter F test:         F=8.7149  , p=0.0000  , df_denom=250, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=0.0000  , p=1.0000  , df_denom=248, df_num=4
ssr based chi2 test:   chi2=0.0000  , p=1.0000  , df=4
likelihood ratio test: chi2=-0.0000 , p=1.0000  , df=4
parameter F test:         F=7.2269  , p=0.0000  , df_denom=248, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=0.0000  , p=1.0000  , df_denom=246, df_num=5
ssr based chi2 test:   chi2=0.0000  , p=1.0000  , df=5
likelihood ratio test: chi2=-0.0000 , p=1.0000  , df=5
parameter F test:         F=7.2337  , p=0.0000  , df_denom=246, df_num=5

Granger Causality
number of lags (no zero) 6
ssr based F test:         F=0.0000  , p=1.0000  , df_denom=244, df_num=6
ssr based chi2 test:   chi2=0.0000  , p=1.0000  , df=6
likelihood ratio test: chi2=-0.0000 , p=1.0000  , df=6
parameter F test:         F=5.9489  , p=0.0000  , df_denom=244, df_num=6

========================================

For: PCE : Billions of dollars

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.9211  , p=0.3381  , df_denom=253, df_num=1
ssr based chi2 test:   chi2=0.9320  , p=0.3343  , df=1
```

```
likelihood ratio test: chi2=0.9303  , p=0.3348  , df=1
parameter F test:         F=0.9211  , p=0.3381  , df_denom=253, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.7615  , p=0.4680  , df_denom=250, df_num=2
ssr based chi2 test:   chi2=1.5534  , p=0.4599  , df=2
likelihood ratio test: chi2=1.5487  , p=0.4610  , df=2
parameter F test:         F=0.7615  , p=0.4680  , df_denom=250, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=0.8279  , p=0.4796  , df_denom=247, df_num=3
ssr based chi2 test:   chi2=2.5542  , p=0.4656  , df=3
likelihood ratio test: chi2=2.5414  , p=0.4679  , df=3
parameter F test:         F=0.8279  , p=0.4796  , df_denom=247, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=0.8084  , p=0.5208  , df_denom=244, df_num=4
ssr based chi2 test:   chi2=3.3530  , p=0.5006  , df=4
likelihood ratio test: chi2=3.3310  , p=0.5040  , df=4
parameter F test:         F=0.8084  , p=0.5208  , df_denom=244, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=1.7989  , p=0.1137  , df_denom=241, df_num=5
ssr based chi2 test:   chi2=9.4049  , p=0.0940  , df=5
likelihood ratio test: chi2=9.2337  , p=0.1001  , df=5
parameter F test:         F=1.7989  , p=0.1137  , df_denom=241, df_num=5

Granger Causality
number of lags (no zero) 6
ssr based F test:         F=2.0568  , p=0.0591  , df_denom=238, df_num=6
ssr based chi2 test:   chi2=13.0150 , p=0.0428  , df=6
likelihood ratio test: chi2=12.6888 , p=0.0483  , df=6
parameter F test:         F=2.0568  , p=0.0591  , df_denom=238, df_num=6

Granger Causality
number of lags (no zero) 7
ssr based F test:         F=1.8111  , p=0.0859  , df_denom=235, df_num=7
ssr based chi2 test:   chi2=13.4870 , p=0.0611  , df=7
likelihood ratio test: chi2=13.1358 , p=0.0689  , df=7
parameter F test:         F=1.8111  , p=0.0859  , df_denom=235, df_num=7

Granger Causality
number of lags (no zero) 8
ssr based F test:         F=1.6846  , p=0.1030  , df_denom=232, df_num=8
```

```
ssr based chi2 test:   chi2=14.4640 , p=0.0704  , df=8
likelihood ratio test: chi2=14.0595 , p=0.0802  , df=8
parameter F test:         F=1.6846  , p=0.1030  , df_denom=232, df_num=8

Granger Causality
number of lags (no zero) 9
ssr based F test:         F=1.5375  , p=0.1358  , df_denom=229, df_num=9
ssr based chi2 test:   chi2=14.9857 , p=0.0913  , df=9
likelihood ratio test: chi2=14.5504 , p=0.1040  , df=9
parameter F test:         F=1.5375  , p=0.1358  , df_denom=229, df_num=9

========================================

For: GPDI : Billions of dollars

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.4007  , p=0.5273  , df_denom=253, df_num=1
ssr based chi2 test:   chi2=0.4055  , p=0.5243  , df=1
likelihood ratio test: chi2=0.4052  , p=0.5244  , df=1
parameter F test:         F=0.4007  , p=0.5273  , df_denom=253, df_num=1

========================================

For: NETEXP : Billions of dollars

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=9.6776  , p=0.0021  , df_denom=253, df_num=1
ssr based chi2 test:   chi2=9.7923  , p=0.0018  , df=1
likelihood ratio test: chi2=9.6097  , p=0.0019  , df=1
parameter F test:         F=9.6776  , p=0.0021  , df_denom=253, df_num=1

========================================
```

[33]:

```
# Ensure the column name is correct and consistent
split_index = int(len(data_diff) * 0.90)
train_data = data_diff.iloc[:split_index]
test_data = data_diff.iloc[split_index:]

# Assigning GDP column to target variable
X_train = train_data.drop('GDP : Billions of dollars', axis=1)
y_train = train_data['GDP : Billions of dollars']
X_test = test_data.drop('GDP : Billions of dollars', axis=1)
y_test = test_data['GDP : Billions of dollars']
```

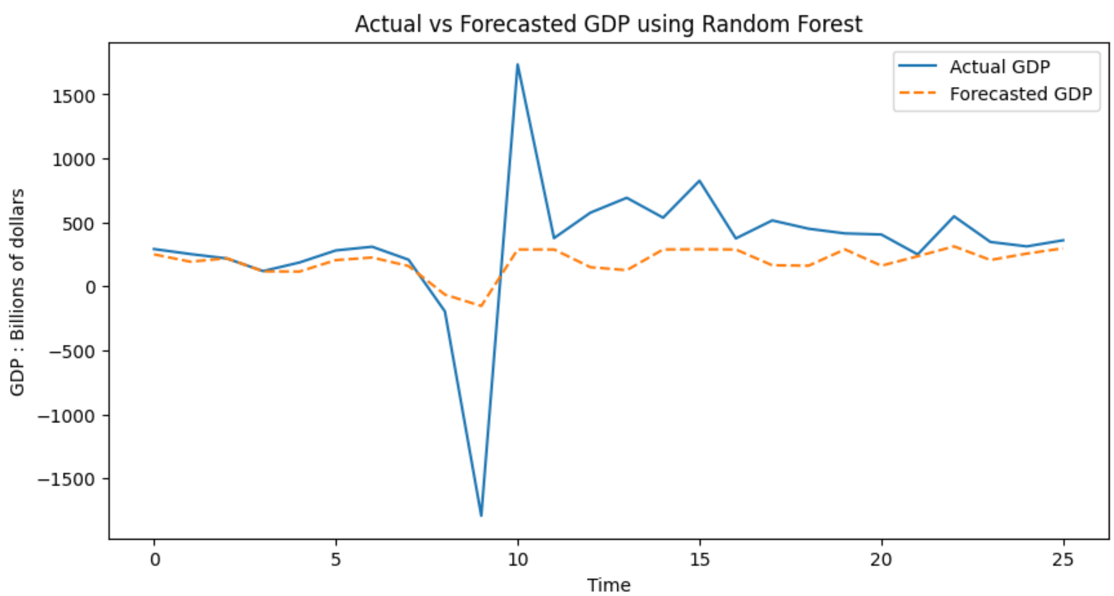
[36]:
```
%pip install scikit-learn
```

```
Collecting scikit-learn
  Downloading scikit_learn-1.5.1-cp311-cp311-macosx_12_0_arm64.whl.metadata (12
kB)
Requirement already satisfied: numpy>=1.19.5 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
scikit-learn) (2.0.1)
Requirement already satisfied: scipy>=1.6.0 in
/Users/neerajnamani/miniconda3/envs/py311/lib/python3.11/site-packages (from
scikit-learn) (1.14.0)
Collecting joblib>=1.2.0 (from scikit-learn)
  Using cached joblib-1.4.2-py3-none-any.whl.metadata (5.4 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn)
  Using cached threadpoolctl-3.5.0-py3-none-any.whl.metadata (13 kB)
Downloading scikit_learn-1.5.1-cp311-cp311-macosx_12_0_arm64.whl (11.0 MB)
                         11.0/11.0 MB
7.6 MB/s eta 0:00:0000:0100:01
Using cached joblib-1.4.2-py3-none-any.whl (301 kB)
Using cached threadpoolctl-3.5.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, joblib, scikit-learn
Successfully installed joblib-1.4.2 scikit-learn-1.5.1 threadpoolctl-3.5.0
Note: you may need to restart the kernel to use updated packages.
```

[38]:
```
from sklearn.ensemble import RandomForestRegressor

rf_model = RandomForestRegressor(n_estimators = 100, random_state = 42)
rf_model.fit(X_train, y_train)

y_pred = rf_model.predict(X_test)

import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score

# Define the printevals function
def printevals(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    rmse = mse ** 0.5
    r2 = r2_score(y_true, y_pred)
    print(f"Mean Squared Error: {mse}")
    print(f"Root Mean Squared Error: {rmse}")
    print(f"R^2 Score: {r2}")

# Define the plotresults function
def plotresults(title):
    plt.figure(figsize=(10, 5))
    plt.plot(y_test.reset_index(drop=True), label='Actual GDP')
    plt.plot(y_pred, label='Forecasted GDP', linestyle='--')
```

```
    plt.title(title)
    plt.xlabel('Time')
    plt.ylabel('GDP : Billions of dollars')
    plt.legend()
    plt.show()

# Assuming you have already trained your model and made predictions
# rf_model = ...  # Your Random Forest model
# y_pred = rf_model.predict(X_test)

# Evaluate and plot results
printevals(y_test, y_pred)
plotresults('Actual vs Forecasted GDP using Random Forest')
```

```
Mean Squared Error: 232807.4460297813
Root Mean Squared Error: 482.50123940750797
R^2 Score: 0.18876026061487017
```




[39]:
```
from sklearn.neighbors import KNeighborsRegressor
# iterate over all k=1 to k=10
for i in range(1,10):
    knn_model = KNeighborsRegressor(n_neighbors=i)
    knn_model.fit(X_train, y_train)

    y_pred = knn_model.predict(X_test)
    print(f'for k = {i}')
    printevals(y_test,y_pred)
```

```python
    print("\n" + "=" * 40 + "\n")
```

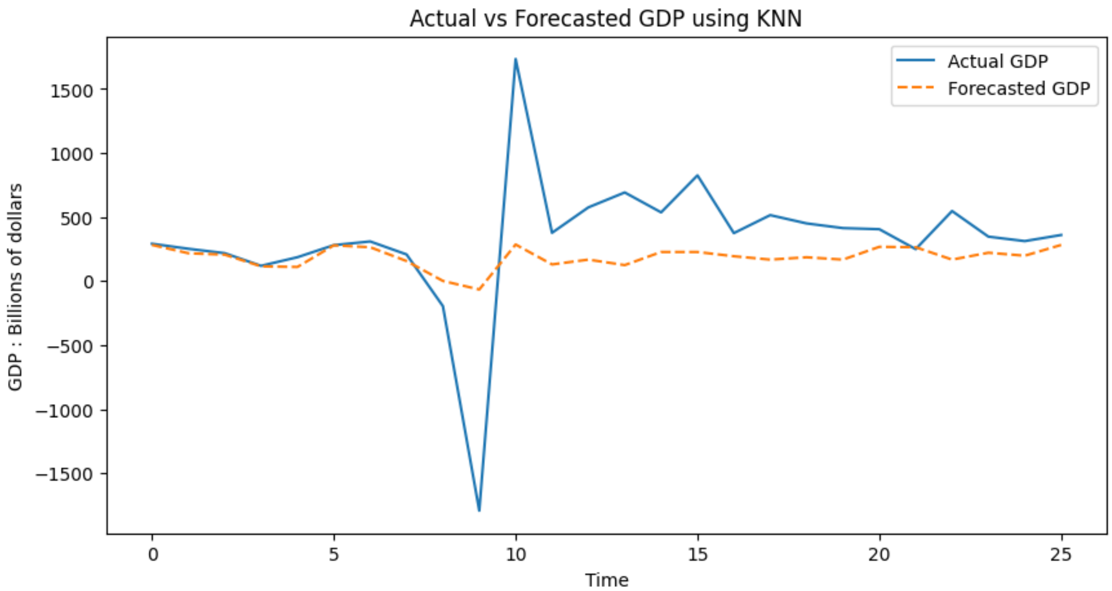
```
for k = 1
Mean Squared Error: 240281.69660884605
Root Mean Squared Error: 490.1853696397375
R^2 Score: 0.16271552194665762

========================================

for k = 2
Mean Squared Error: 254614.91029419223
Root Mean Squared Error: 504.59380722933196
R^2 Score: 0.1127700724649251

========================================

for k = 3
Mean Squared Error: 258879.7438215
Root Mean Squared Error: 508.80226397049375
R^2 Score: 0.0979088534695004

========================================

for k = 4
Mean Squared Error: 257326.62522003113
Root Mean Squared Error: 507.27371824295324
R^2 Score: 0.10332084329618574

========================================

for k = 5
Mean Squared Error: 266163.3469519337
Root Mean Squared Error: 515.9102121027782
R^2 Score: 0.07252844401059544

========================================

for k = 6
Mean Squared Error: 271011.1762016729
Root Mean Squared Error: 520.5873377269878
R^2 Score: 0.05563572084297408

========================================

for k = 7
Mean Squared Error: 270831.94846411614
Root Mean Squared Error: 520.4151693255262
```

```
R^2 Score: 0.056260256980394496

========================================

for k = 8
Mean Squared Error: 275173.34869414533
Root Mean Squared Error: 524.5696795413793
R^2 Score: 0.04113223401018018

========================================

for k = 9
Mean Squared Error: 275421.8648684799
Root Mean Squared Error: 524.8065023115471
R^2 Score: 0.04026625570949305

========================================
```

[40]:

```
#applying model with optimal k values
knn_model = KNeighborsRegressor(n_neighbors = 2)
knn_model.fit(X_train, y_train)

y_pred = knn_model.predict(X_test)

printevals(y_test, y_pred)

plotresults('Actual vs Forecasted GDP using KNN')
```

```
Mean Squared Error: 254614.91029419223
Root Mean Squared Error: 504.59380722933196
R^2 Score: 0.1127700724649251
```


Actual vs Forecasted GDP using KNN
Actual GDP
Forecasted GDP
GDP : Billions of dollars
1500
1000
500
0
−500
−1000
−1500
0
5
10
15
20
25
Time

[ ]: