CIND820 - Final Project

> An Analysis of Factors Impacting the Stock Market Price

> Initial Results [Import Data/Data Cleaning/Visualization/Modelling]

Student: Stella Hao | Supervisor: Dr. Ashok Bhowmick

```
# import required library
!pip install pandas_ta
import pandas_ta as ta
import pandas as pd
import numpy as np
from pandas import datetime
import matplotlib.pyplot as plt

# import data
def parser(x):
    return datetime.strptime(x, '%Y-%m-%d')

cpi = pd.read_csv('cpi.csv',header=0, parse_dates=[0], index_col=0, squeeze=True, date_parser=parser)
gdi = pd.read_csv('gdi.csv',header=0, parse_dates=[0], index_col=0, squeeze=True, date_parser=parser)
housing = pd.read_csv('housing.csv',header=0, parse_dates=[0], index_col=0, squeeze=True, date_parser=parser)
prime = pd.read_csv('prime_rate.csv',header=0, parse_dates=[0], index_col=0, squeeze=True, date_parser=parser)
tsx = pd.read_csv('tsx_original.csv',header=0, parse_dates=[0], index_col=0, squeeze=True, date_parser=parser)
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pandas_ta in /usr/local/lib/python3.7/dist-packages (0.3.14b0)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from pandas_ta) (1.3.5)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas->pandas_ta) (2.8.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-packages (from pandas->pandas_ta) (1.21.6)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas->pandas_ta) (2022.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas->pandas_ta) (1.15.0)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: FutureWarning: The pandas.datetime class is deprecated and will be removed from panc
```

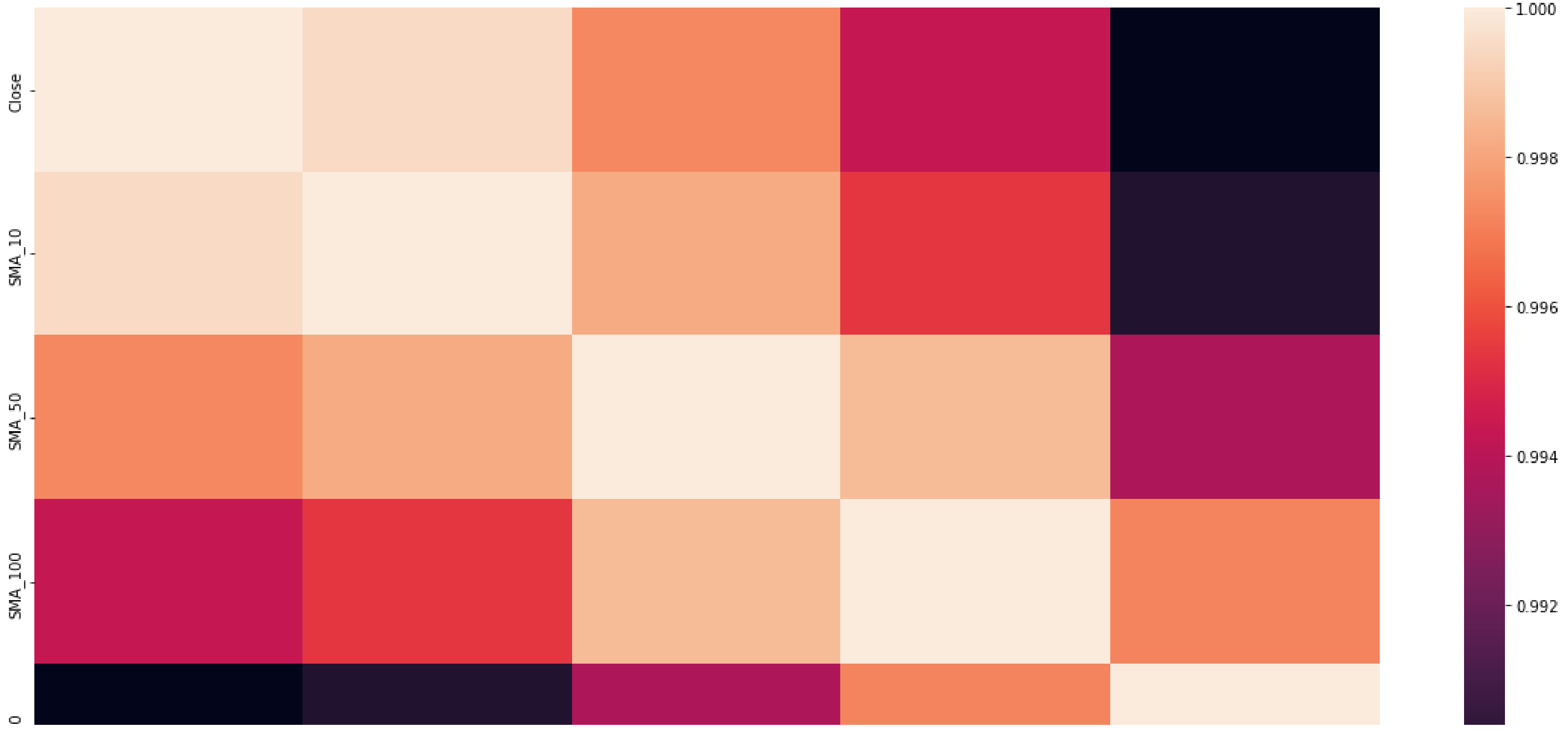
```
# For stock return data, drop records for null values
tsx_new = tsx.dropna()
# Add technical indicators (moving average 10, 50, 100, 200)
tsx_new.ta.sma(close='Close', length=10, append=True)
tsx_new.ta.sma(close='Close', length=50, append=True)
tsx_new.ta.sma(close='Close', length=100, append=True)
tsx_new.ta.sma(close='Close', length=200, append=True)
```

```
# drop null values after adding technical indicators
tsx_new1 = tsx_new.dropna()
# remove unrequired columns
tsx_new1 = tsx_new1.drop(columns=['Open','High','Low','Adj Close','Volume'])
```

```
/usr/local/lib/python3.7/dist-packages/pandas_ta/core.py:426: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[ind_name] = result
/usr/local/lib/python3.7/dist-packages/pandas_ta/core.py:426: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[ind_name] = result
/usr/local/lib/python3.7/dist-packages/pandas_ta/core.py:426: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[ind_name] = result
/usr/local/lib/python3.7/dist-packages/pandas_ta/core.py:426: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[ind_name] = result
```

```
#import library
import seaborn as sns

# calculate the correlation matrix
corr = tsx_new1.corr()

# plot the heatmap
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns)

#Based on the result all values are highly correlated as expected.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6cc093dd90>
```

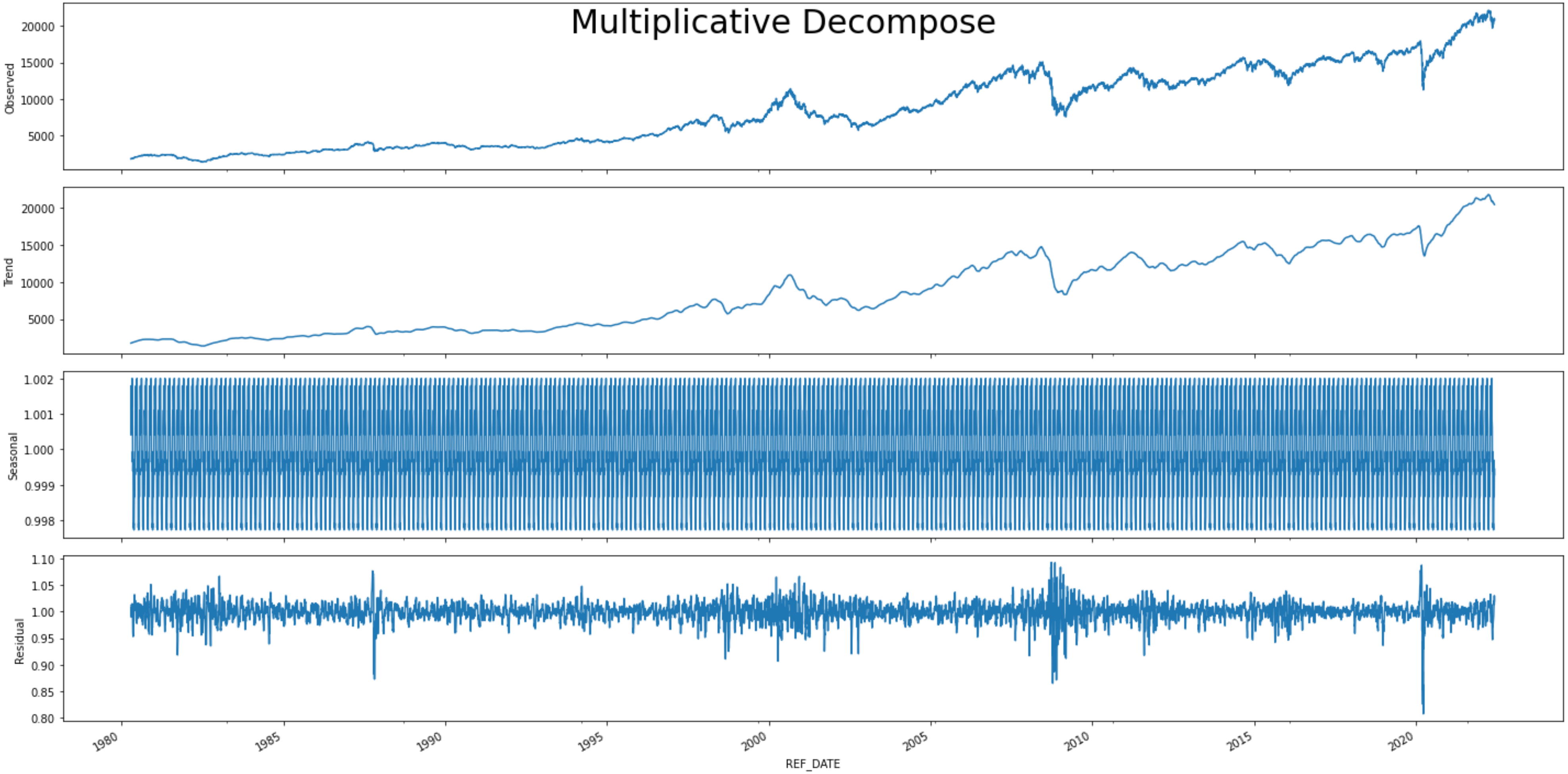



```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()

# Plot linear sequence, and set tick labels to the same color
ax.plot(tsx_new1['Close'], color='red',label='Close')
ax.plot(tsx_new1['SMA_10'], color='black',label='SMA_10')
ax.plot(tsx_new1['SMA_50'], color='blue',label='SMA_50')
ax.plot(tsx_new1['SMA_100'], color='orange',label='SMA_100')
ax.plot(tsx_new1['SMA_200'], color='yellow',label='SMA_200')
ax.tick_params(axis='y', labelcolor='black')

plt.xlabel("Date")
plt.ylabel("Value")
ax.legend(loc='upper left')
plt.show()

#Based on the graph, the close price, sma 10, sma 50, sma 100, and sma 200 closely tracks each other as expected.
#The close prices shows a large dip in 2020 whereas the SMA 10 shows a slightly less dip, followed by SMA 50, SMA 100 and SMA 200.
```

#This trend is due to the smoothing effect of the moving average which have smoothed out the dip across the relative moving average days.


```
#Seasonality Check code

from statsmodels.tsa.seasonal import seasonal_decompose
def decompose(df, column_name, frequency):
    """
    A function that returns the trend, seasonality and residual captured by applying both multiplicative and
    additive model.
    """
    result_mul = seasonal_decompose(df[column_name], model='multiplicative', extrapolate_trend = 'freq',freq=frequency)
    result_add = seasonal_decompose(df[column_name], model = 'additive', extrapolate_trend='freq',freq=frequency)

    plt.rcParams.update({'figure.figsize': (20, 10)})
```

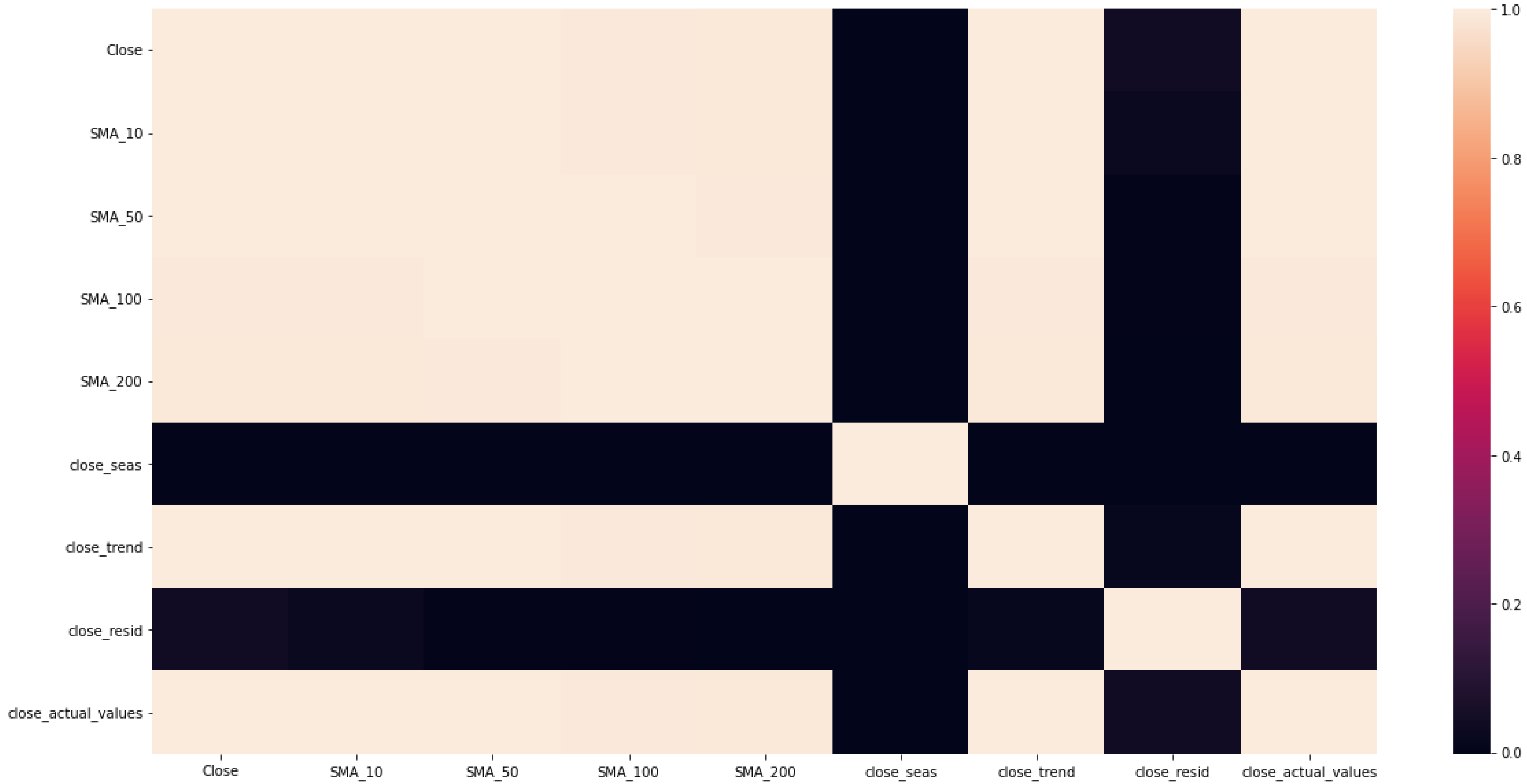
```
    result_mul.plot().suptitle('Multiplicative Decompose', fontsize=30)
    result_add.plot().suptitle('Additive Decompose', fontsize=30)
    plt.show()
    return result_mul, result_add

#Seasonality check for Close price
result_mul, result_add = decompose(tsx_new1, 'Close',36)
"""
As we can see, setting the time series frequency as 36, the trend was well captured.
Also, if we look at the residuals plot, we can see no well-defined pattern for the multiplicative decompose.
The additive depose shows a flat residual at the earlier years with an increasing trend to the recent years.
Therefore, we will use multiplicative decompose and we can say our time series was well decomposed into its components.
The residuals are also interesting, showing periods of high variability during the rapid falls and rise in the series.
"""
```




```
#add seasonality data in the dataset
df_reconstructed = pd.concat([result_mul.seasonal, result_mul.trend, result_mul.resid, result_mul.observed], axis = 1)
df_reconstructed.columns = ['close_seas', 'close_trend', 'close_resid', 'close_actual_values']
tsx_new_season = pd.merge(tsx_new1, df_reconstructed, on="REF_DATE")
```

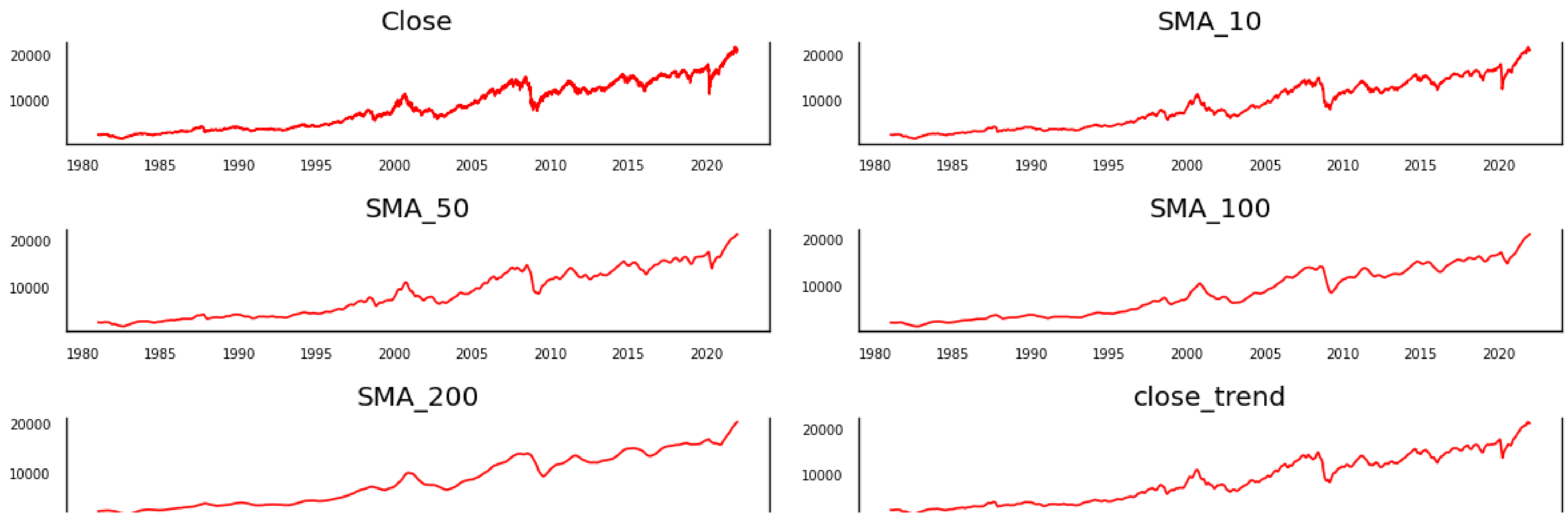
```
# calculate the correlation matrix
corr = tsx_new_season.corr()

# plot the heatmap
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6cc1531710>
```



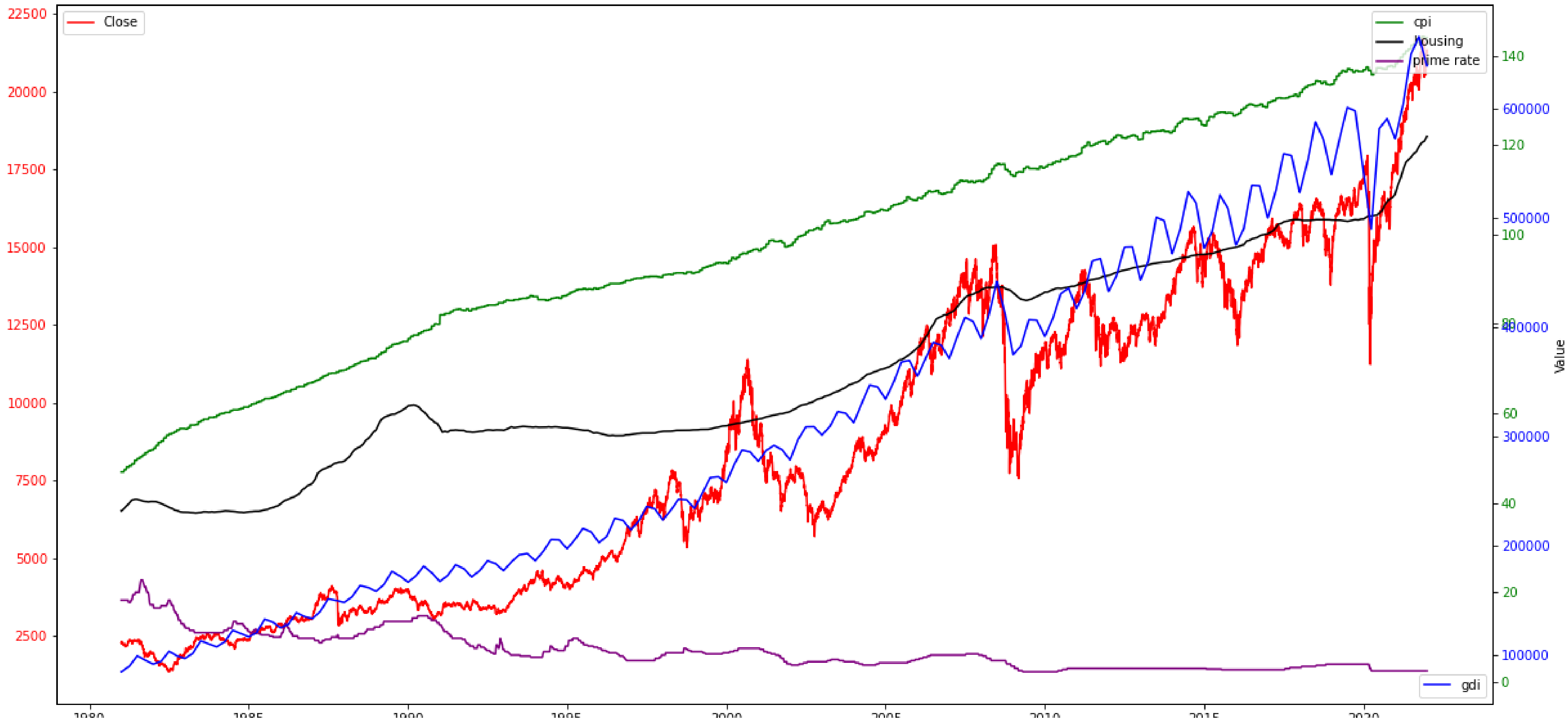

```
#Drop seasonality and residual as these do not show correlation
#drop duplicate value close_actual_values
tsx_new_season = tsx_new_season.drop(columns=['close_actual_values','close_seas','close_resid'])


#Convert cpi, gdi, housing, and prime rate into daily data
cpi_fill=cpi.resample('D').ffill()
cpi_time = cpi_fill.interpolate(method='time')

gdi_fill = gdi.resample('D')
gdi_time = gdi_fill.interpolate(method='time')
```

```
housing_fill = housing.resample('D')
housing_time = housing_fill.interpolate(method='time')

prime_fill = prime.resample('D')
prime_time = prime_fill.interpolate(method='time')

#Merge data set into one dataframe

final_dt = pd.merge(tsx_new_season, cpi_time, on="REF_DATE")
final_dt = pd.merge(final_dt, gdi_time, on="REF_DATE")
final_dt = pd.merge(final_dt, housing_time, on="REF_DATE")
final_dt = pd.merge(final_dt, prime_time, on="REF_DATE")


#Visualize the final dataset
# Plot
fig, axes = plt.subplots(nrows=5, ncols=2, dpi=120, figsize=(10,6))
for i, ax in enumerate(axes.flatten()):
    data = final_dt[final_dt.columns[i]]
    ax.plot(data, color='red', linewidth=1)
    # Decorations
    ax.set_title(final_dt.columns[i])
    ax.xaxis.set_ticks_position('none')
    ax.yaxis.set_ticks_position('none')
    ax.spines["top"].set_alpha(0)
    ax.tick_params(labelsize=6)

plt.tight_layout();
#all variable shows upward trend except for prime rate which is showing a downward trend
```




```
#Graph Close price, cpi, gdi, housing, prime rate on the same graph
fig, ax = plt.subplots()

# Plot linear sequence, and set tick labels to the same color
ax.plot(final_dt['Close'], color='red',label='Close')
ax.tick_params(axis='y', labelcolor='red')

# Generate a new Axes instance, on the twin-X axes (same position)
ax2 = ax.twinx()
ax3 = ax.twinx()

# Plot other variables and change tick color
ax2.plot(final_dt['cpi'], color='green',label='cpi')
ax2.tick_params(axis='y', labelcolor='green')
ax2.plot(final_dt['housing'], color='black',label='housing')
ax2.plot(final_dt['prime_rate'], color='purple',label='prime rate')

ax3.plot(final_dt['gross_income'], color='blue',label='gdi')
ax3.tick_params(axis='y', labelcolor='blue')

plt.xlabel("Date")
plt.ylabel("Value")
ax.legend(loc='upper left')
ax2.legend(loc='upper right')
ax3.legend(loc='lower right')
plt.show()
```

```
#Close price y-scale in color red
#cpi, housing, prime rate y-scale in color green
#gross domestic income (GDI) y-scale in color blue
#all data shows upward trend execpt prime rate
```




```
#Testing Causation using Granger's Causality Test
from statsmodels.tsa.stattools import grangercausalitytests
maxlag=16
test = 'ssr_chi2test'
def grangers_causation_matrix(data, variables, test='ssr_chi2test', verbose=False):
#Check Granger Causality of all possible combinations of the Time series at the 0.05 significance level.
#Null hypothesis is: X does not cause Y.
    df = pd.DataFrame(np.zeros((len(variables), len(variables))), columns=variables, index=variables)
    for c in df.columns:
        for r in df.index:
```

```
            test_result = grangercausalitytests(data[[r, c]], maxlag=maxlag, verbose=False)
            p_values = [round(test_result[i+1][0][test][1],4) for i in range(maxlag)]
            min_p_value = np.min(p_values)
            df.loc[r, c] = min_p_value
    df.columns = [var + '_x' for var in variables]
    df.index = [var + '_y' for var in variables]
    return df

#Call grangers_causation_matrix to produce visualized table
grangers_causation_matrix(final_dt, variables = final_dt.columns)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:1752: ValueWarning: covariance of constraints does not have full rank. The number
  'rank is %d' % (J, J_), ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:1752: ValueWarning: covariance of constraints does not have full rank. The number
  'rank is %d' % (J, J_), ValueWarning)
```

	Close_x	SMA_10_x	SMA_50_x	SMA_100_x	SMA_200_x	close_trend_x	cpi_x	gross_income_x	housing_x	prime_rate_x
Close_y	1.0000	0.0000	0.0012	0.4432	0.0000	0.0000	0.0000	0.0000	0.0048	0.0085
SMA_10_y	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0026
SMA_50_y	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0002	0.0000
SMA_100_y	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
SMA_200_y	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0237
close_trend_y	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000
cpi_y	0.0000	0.0000	0.0000	0.0002	0.0017	0.0000	1.0000	0.0000	0.0000	0.0056
gross_income_y	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0009
housing_y	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0011
prime_rate_y	0.1124	0.2435	0.2576	0.0333	0.0105	0.2688	0.0285	0.2561	0.0001	1.0000

```
"""
Looking at the result where Close_y is the dependend variable, the p value for all the x values are significant at 0.05 value except
for SMA_100. Therefore, we will remove it from the dataset.
"""
final_dt = final_dt.drop(columns=['SMA_100'])


#test for cointegration

from statsmodels.tsa.vector_ar.vecm import coint_johansen
"""
    Johansen cointegration test of the cointegration rank of a VECM
```

```
    Parameters
    ----------
    endog : array_like (nobs_tot x neqs)
        Data to test
    det_order : int
        * -1 - no deterministic terms
        * 0 - constant term
        * 1 - linear trend
    k_ar_diff : int, nonnegative
        Number of lagged differences in the model.
"""

def cointegration_test(final_dt, alpha=0.05):
    out = coint_johansen(final_dt,1,9)
    d = {'0.90':0, '0.95':1, '0.99':2}
    traces = out.lr1
    cvts = out.cvt[:, d[str(1-alpha)]]
    def adjust(val, length= 6): return str(val).ljust(length)

    # Summary
    print('Name   ::  Test Stat > C(95%)    =>   Signif  \n', '--'*20)
    for col, trace, cvt in zip(final_dt.columns, traces, cvts):
        print(adjust(col), ':: ', adjust(round(trace,2), 9), ">", adjust(cvt, 8), ' =>  ' , trace > cvt)

cointegration_test(final_dt)
"""
CPI, gross_income, housing and prime rate shows no cointegration whereas the other attributes are cointegrated.
"""
```

```
Name   ::  Test Stat > C(95%)    =>   Signif
 ----------------------------------------
Close  ::  2622.04   > 215.1268  =>   True
SMA_10 ::  1266.78   > 175.1584  =>   True
SMA_50 ::  442.35    > 139.278   =>   True
SMA_200 ::  196.85    > 107.3429  =>   True
close_trend ::  93.86     > 79.3422   =>   True
cpi    ::  50.84     > 55.2459   =>   False
gross_income ::  22.1      > 35.0116   =>   False
housing ::  3.87      > 18.3985   =>   False
prime_rate ::  -0.0      > 3.8415    =>   False
'\nCPI, gross_income, housing and prime rate shows no cointegration whereas the other attributes are cointegrated.\n'
```

```
#training and testing dataset. Setting the testing dataset to be the last 14 days.
nobs = 14 #14 days
df_train, df_test = final_dt[0:-nobs], final_dt[-nobs:]
```

```
# Check size
print(df_train.shape)
print(df_test.shape)
```

```
(10284, 9)
(14, 9)
```

```
#stationary test
def adfuller_test(series, signif=0.05, name='', verbose=False):
#Perform ADFuller to test for Stationarity of given dataset and print report
    r = adfuller(series, autolag='AIC')
    output = {'test_statistic':round(r[0], 4), 'pvalue':round(r[1], 4), 'n_lags':round(r[2], 4), 'n_obs':r[3]}
    p_value = output['pvalue']
    def adjust(val, length= 6): return str(val).ljust(length)

    # Print Summary
    print(f'    Augmented Dickey-Fuller Test on "{name}"', "\n   ", '-'*47)
    print(f' Null Hypothesis: Data has unit root. Non-Stationary.')
    print(f' Significance Level    = {signif}')
    print(f' Test Statistic        = {output["test_statistic"]}')
    print(f' No. Lags Chosen       = {output["n_lags"]}')

    for key,val in r[4].items():
        print(f' Critical value {adjust(key)} = {round(val, 3)}')

    if p_value <= signif:
        print(f" => P-Value = {p_value}. Rejecting Null Hypothesis.")
        print(f" => Series is Stationary.")
    else:
        print(f" => P-Value = {p_value}. Weak evidence to reject the Null Hypothesis.")
        print(f" => Series is Non-Stationary.")
```

```
#import stats model
from statsmodels.tsa.stattools import adfuller
# ADF Test on each column of the training dataset
for name, column in df_train.iteritems():
    adfuller_test(column, name=column.name)
    print('\n')

#Data shows non-stationary for all the variables
```

```
 Critical value 10%    = -2.567
 => P-Value = 0.9178. Weak evidence to reject the Null Hypothesis.
 => Series is Non-Stationary.


    Augmented Dickey-Fuller Test on "cpi"
```

```
   Augmented Dickey-Fuller Test on "cpi"
   -----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -1.2385
No. Lags Chosen       = 23
Critical value 1%     = -3.431
Critical value 5%     = -2.862
Critical value 10%    = -2.567
=> P-Value = 0.6568. Weak evidence to reject the Null Hypothesis.
=> Series is Non-Stationary.


   Augmented Dickey-Fuller Test on "gross_income"
   -----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -0.0916
No. Lags Chosen       = 39
Critical value 1%     = -3.431
Critical value 5%     = -2.862
Critical value 10%    = -2.567
=> P-Value = 0.9503. Weak evidence to reject the Null Hypothesis.
=> Series is Non-Stationary.


   Augmented Dickey-Fuller Test on "housing"
   -----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = 1.4879
No. Lags Chosen       = 39
Critical value 1%     = -3.431
Critical value 5%     = -2.862
Critical value 10%    = -2.567
=> P-Value = 0.9975. Weak evidence to reject the Null Hypothesis.
=> Series is Non-Stationary.


   Augmented Dickey-Fuller Test on "prime_rate"
   -----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.

Significance Level    = 0.05
Test Statistic        = -2.6729
No. Lags Chosen       = 36
Critical value 1%     = -3.431
Critical value 5%     = -2.862
Critical value 10%    = -2.567
=> P-Value = 0.0789. Weak evidence to reject the Null Hypothesis.
=> Series is Non-Stationary.
```

```
# 1st difference
final_differenced = df_train.diff().dropna()


# ADF Test on each column of 1st Differences Dataframe
for name, column in final_differenced.iteritems():
    adfuller_test(column, name=column.name)
    print('\n')

#all variable is stationary after 1st difference
```

```
      Critical value 10%    = -2.567
      => P-Value = 0.0. Rejecting Null Hypothesis.
      => Series is Stationary.


         Augmented Dickey-Fuller Test on "cpi"
         -----------------------------------------------
      Null Hypothesis: Data has unit root. Non-Stationary.
      Significance Level    = 0.05
      Test Statistic        = -17.1547
      No. Lags Chosen       = 22
      Critical value 1%     = -3.431
      Critical value 5%     = -2.862
      Critical value 10%    = -2.567
      => P-Value = 0.0. Rejecting Null Hypothesis.
      => Series is Stationary.


         Augmented Dickey-Fuller Test on "gross_income"
         -----------------------------------------------
      Null Hypothesis: Data has unit root. Non-Stationary.
      Significance Level    = 0.05
      Test Statistic        = -10.4513
      No. Lags Chosen       = 38
      Critical value 1%     = -3.431
      Critical value 5%     = -2.862
      Critical value 10%    = -2.567
      => P-Value = 0.0. Rejecting Null Hypothesis.

      => Series is Stationary.


         Augmented Dickey-Fuller Test on "housing"
         -----------------------------------------------
      Null Hypothesis: Data has unit root. Non-Stationary.
      Significance Level    = 0.05
      Test Statistic        = -5.5907
      No. Lags Chosen       = 38
```

```
Critical value 1%     = -3.431
Critical value 5%     = -2.862
Critical value 10%    = -2.567
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


   Augmented Dickey-Fuller Test on "prime_rate"
   -----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -12.5921
No. Lags Chosen       = 35
Critical value 1%     = -3.431
Critical value 5%     = -2.862
Critical value 10%    = -2.567
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.
```

```
#Import Stats model
from statsmodels.tsa.api import VAR
#Select the Order (P) of VAR model based on 1st difference
model = VAR(final_differenced)
for i in range(15):
    result = model.fit(i)
    print('Lag Order =', i)
    print('AIC : ', result.aic)
    print('BIC : ', result.bic)
    print('FPE : ', result.fpe)
    print('HQIC: ', result.hqic, '\n')
#Order 9 has AIC at the lowest AIC :  -31.86451275759908
```

```
FPE :  34.21334550556829
HQIC:  3.6311647743634703

Lag Order = 6
AIC :  3.3815156684183396
BIC :  3.730123580793003
FPE :  29.415348269930295
HQIC:  3.4993570728962013

Lag Order = 7
AIC :  3.1916868512442873
BIC :  3.5973737160107673
FPE :  24.329468248961557
HQIC:  3.3288235578646814

Lag Order = 8
```

```
AIC :  2.832909334451681
BIC :  3.295684727041699
FPE :  16.99486919053567
HQIC:  2.9893447666776547

Lag Order = 9
AIC :  -31.86451275759908
BIC :  -31.344639259181903
FPE :  1.4501711659608295e-14
HQIC:  -31.68877517535861

Lag Order = 10
AIC :  -9.372142554107956
BIC :  -8.795161369287152
FPE :  8.506130012510381e-05

HQIC:  -9.177099396497846

Lag Order = 11
AIC :  -8.720862296644693
BIC :  -8.086763842270008
FPE :  0.00016314735282614877
HQIC:  -8.506510137363223

Lag Order = 12
AIC :  -8.793704098295851
BIC :  -8.102478788642298
FPE :  0.00015168615724970907
HQIC:  -8.560039510094368

Lag Order = 13
AIC :  -8.582311051594685
BIC :  -7.833949298361597
FPE :  0.0001873931109690033
HQIC:  -8.329330606277251

Lag Order = 14
AIC :  -8.643570407605447
BIC :  -7.83806261991554
FPE :  0.00017625847064460528
HQIC:  -8.371270676028484
```

```
#Train the VAR Model of Selected Order(9)
model_fitted = model.fit(9)
model_fitted.summary()
```

```
L5.SMA_50              -0.000051         0.000202           -0.252           0.801
L5.SMA_200              0.000258         0.000816            0.317           0.751
L5.close_trend         -0.000032         0.000398           -0.079           0.937
L5.cpi                  0.004749         0.003171            1.498           0.134
```

L5.gross_income	0.000001	0.000001	0.426	0.670
L5.housing	0.033003	0.023640	1.396	0.163
L5.prime_rate	0.266822	0.012345	21.614	0.000
L6.Close	-0.000010	0.000007	-1.507	0.132
L6.SMA_10	0.000035	0.000041	0.873	0.383
L6.SMA_50	0.000089	0.000202	0.442	0.659
L6.SMA_200	-0.000153	0.000816	-0.187	0.852
L6.close_trend	-0.000092	0.000387	-0.239	0.811
L6.cpi	-0.000296	0.003166	-0.093	0.926
L6.gross_income	-0.000000	0.000002	-0.007	0.995
L6.housing	0.013013	0.032467	0.401	0.689
L6.prime_rate	-0.114865	0.012631	-9.094	0.000
L7.Close	-0.000009	0.000007	-1.305	0.192
L7.SMA_10	-0.000004	0.000041	-0.088	0.930
L7.SMA_50	-0.000129	0.000202	-0.635	0.525
L7.SMA_200	-0.000043	0.000817	-0.053	0.958
L7.close_trend	0.000152	0.000352	0.431	0.666
L7.cpi	0.000406	0.003164	0.128	0.898
L7.gross_income	0.000000	0.000002	0.016	0.987
L7.housing	0.016214	0.031772	0.510	0.610
L7.prime_rate	-0.080818	0.011957	-6.759	0.000
L8.Close	0.000003	0.000007	0.469	0.639
L8.SMA_10	-0.000037	0.000041	-0.910	0.363
L8.SMA_50	-0.000032	0.000203	-0.157	0.875
L8.SMA_200	-0.000535	0.000819	-0.654	0.513
L8.close_trend	0.000081	0.000288	0.281	0.779
L8.cpi	0.001550	0.003163	0.490	0.624
L8.gross_income	0.000000	0.000002	0.152	0.879
L8.housing	-0.022054	0.031768	-0.694	0.488
L8.prime_rate	0.086435	0.011256	7.679	0.000
L9.Close	-0.000007	0.000004	-1.610	0.107
L9.SMA_10	0.000019	0.000030	0.615	0.539
L9.SMA_50	0.000024	0.000147	0.165	0.869
L9.SMA_200	0.000606	0.000589	1.029	0.303
L9.close_trend	-0.000138	0.000143	-0.964	0.335
L9.cpi	-0.000837	0.003162	-0.265	0.791
L9.gross_income	-0.000000	0.000002	-0.098	0.922
L9.housing	0.036006	0.032245	1.117	0.264
L9.prime_rate	0.007628	0.009917	0.769	0.442

==

Correlation matrix of residuals

	Close	SMA_10	SMA_50	SMA_200	close_trend	cpi	gross_income	housing	prime_rate
Close	1.000000	1.000000	0.697465	0.697692	-0.035888	0.034156	0.026285	0.019573	0.001368
SMA_10	1.000000	1.000000	0.697465	0.697692	-0.035888	0.034156	0.026285	0.019573	0.001368
SMA_50	0.697465	0.697465	1.000000	0.473742	-0.037517	0.020887	0.035707	0.004227	-0.001268
SMA_200	0.697692	0.697692	0.473742	1.000000	-0.021196	0.016749	0.017406	0.029785	0.002628
close_trend	-0.035888	-0.035888	-0.037517	-0.021196	1.000000	-0.017974	-0.004868	0.028386	-0.008971
cpi	0.034156	0.034156	0.020887	0.016749	-0.017974	1.000000	0.038095	0.037187	-0.004811
gross_income	0.026285	0.026285	0.035707	0.017406	-0.004868	0.038095	1.000000	0.128416	0.011955
housing	0.019573	0.019573	0.004227	0.029785	0.028386	0.037187	0.128416	1.000000	0.030527

```
prime_rate      0.001368  0.001368 -0.001268  0.002628    -0.008971 -0.004811      0.011955  0.030527    1.000000
```

```
#Check for Serial Correlation of Residuals (Errors) using Durbin Watson Statistic
from statsmodels.stats.stattools import durbin_watson
out = durbin_watson(model_fitted.resid)

for col, val in zip(final_differenced.columns, out):
    print(col, ':', round(val, 2))
#The value of this statistic can vary between 0 and 4. The closer it is to the value 2, then there is no significant serial correlation.
#The closer to 0, there is a positive serial correlation, and the closer it is to 4 implies negative serial correlation.
#All the values are closer to 2 which means no serial correlation
```

```
Close : 2.02
SMA_10 : 2.02
SMA_50 : 2.01
SMA_200 : 2.01
close_trend : 2.15
cpi : 2.0
gross_income : 2.01
housing : 1.99
prime_rate : 2.0
```

```
#Forecast VAR model using statsmodels
# Get the lag order
lag_order = model_fitted.k_ar
print(lag_order)  #> 9

# Input data for forecasting
forecast_input = final_differenced.values[-lag_order:]
forecast_input
```

```
9
array([[ 2.30996100e+01, -5.34099610e+01,  1.31719922e+01,
         1.37800000e+01, -2.02916125e+00,  0.00000000e+00,
        -8.79228261e+02,  3.00000000e-02,  0.00000000e+00],
       [-4.89000000e+02, -1.05719922e+02,  1.01100000e+01,
         1.09990039e+01, -2.20277236e+00,  0.00000000e+00,
        -2.93076087e+02,  1.00000000e-02,  0.00000000e+00],
       [-1.95400390e+02, -1.18840039e+02,  4.40597660e+00,
         9.86049805e+00, -3.10419375e+00, -2.00000000e-01,
        -2.93076087e+02,  1.00000000e-02,  0.00000000e+00],
       [ 2.97400390e+02, -8.75500000e+01,  7.21000000e+00,
         1.19359961e+01, -5.18058264e+00,  0.00000000e+00,
        -2.93076087e+02,  3.54838710e-02,  0.00000000e+00],
       [-1.28699220e+02, -9.21699220e+01,  3.42800780e+00,
         1.17960058e+01, -8.79999458e+00,  0.00000000e+00,
        -2.93076087e+02,  3.54838710e-02,  0.00000000e+00],
```

```
       [ 2.27798830e+02, -5.59701170e+01,  9.16800780e+00,
         1.23839942e+01, -8.29305014e+00,  0.00000000e+00,
        -8.79228261e+02,  1.06451613e-01,  0.00000000e+00],
       [ 3.01599610e+02, -2.91101560e+01,  1.39859766e+01,
         1.37300000e+01, -1.06625162e+01,  0.00000000e+00,
        -2.93076087e+02,  3.54838710e-02,  0.00000000e+00],
       [-8.52988300e+01, -4.71000000e+01,  1.80660156e+01,
         1.37365039e+01, -1.72027994e+01,  0.00000000e+00,
        -2.93076087e+02,  3.54838710e-02,  0.00000000e+00],
       [-1.51900390e+02, -6.87699220e+01,  1.53480078e+01,
         1.22050000e+01, -1.79847276e+01,  0.00000000e+00,
        -2.93076087e+02,  3.54838710e-02,  0.00000000e+00]])
```

```
fc = model_fitted.forecast(y=forecast_input, steps=nobs)
df_forecast = pd.DataFrame(fc, index=final_dt.index[-nobs:], columns=final_dt.columns + '_1d')

df_forecast
```

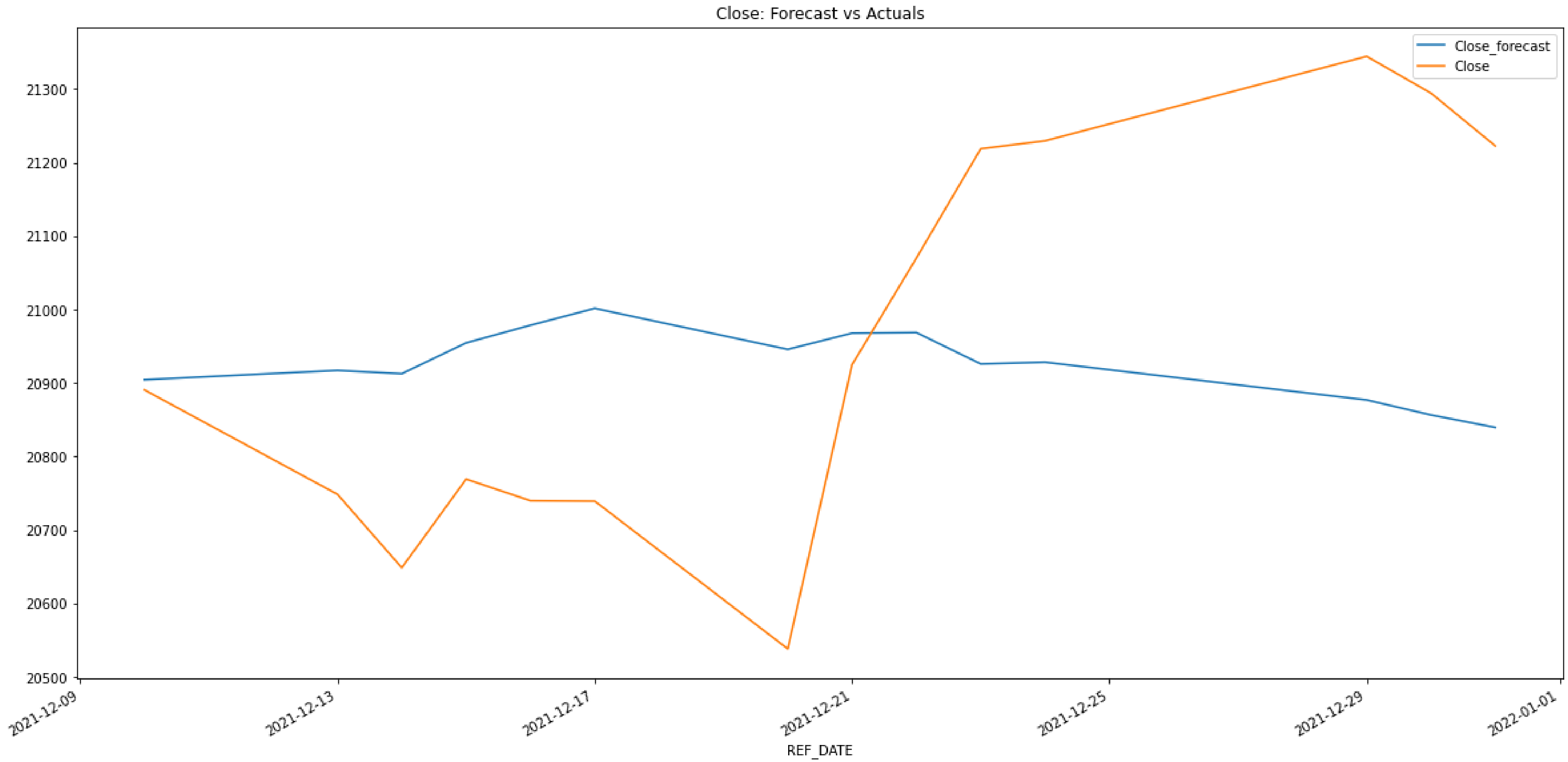
REF_DATE	Close_1d	SMA_10_1d	SMA_50_1d	SMA_200_1d	close_trend_1d	cpi_1d	gross_income_1d	housing_1d	prime_rate_1d
2021-12-10	-21.003561	-22.140395	13.830735	12.196729	-17.357416	-0.001565	-372.441106	0.058963	-0.002814
2021-12-13	12.691094	-23.181247	14.116314	12.128447	-16.947581	0.035272	-709.136170	0.090750	0.000743
2021-12-14	-4.478079	25.270945	13.144049	12.223679	-15.692596	-0.031565	-260.647097	0.034113	-0.001837
2021-12-15	41.943243	49.005309	13.563367	12.259026	-14.390127	0.017642	-241.650195	0.037683	-0.002725
2021-12-16	24.058832	21.671153	13.193638	12.270032	-10.595551	0.010226	-221.757567	0.038505	-0.000200
2021-12-17	22.925859	36.833661	13.163218	12.339594	-9.306800	-0.008402	-284.714115	0.062106	0.002198
2021-12-20	-55.779601	8.475818	11.356865	11.849795	-10.976443	0.000009	-497.679533	0.073948	0.004443
2021-12-21	22.070292	-19.477114	10.479235	12.016954	-12.138361	-0.016891	-138.661343	0.034271	0.004376
2021-12-22	0.802622	-10.866969	9.543743	11.872539	-6.382380	0.004611	-207.931244	0.036964	0.005332
2021-12-23	-42.724987	0.050571	7.828674	11.650661	-1.385174	-0.016006	-163.842908	0.040050	0.009440
2021-12-24	2.337762	2.384704	6.796203	11.508396	4.951559	-0.026764	-254.045107	0.059352	0.008028
2021-12-29	-51.360289	-4.020435	5.354386	11.176070	13.460337	-0.010805	-367.896691	0.060994	0.006263
2021-12-30	-20.434165	-5.616043	4.335692	11.090227	11.686269	-0.009831	-86.041718	0.032374	0.004318
2021-12-31	-16.965768	-11.506944	3.698901	10.906320	9.256311	0.005458	-137.078108	0.035301	0.001064

```
#Transform to real forecast
```

```
def invert_transformation(df_train, df_forecast, second_diff=False):
    """Revert back the differencing to get the forecast to original scale."""
    df_fc = df_forecast.copy()
    columns = df_train.columns
    for col in columns:
        # Roll back 1st Diff
        df_fc[str(col)+'_forecast'] = df_train[col].iloc[-1] + df_fc[str(col)+'_1d'].cumsum()
    return df_fc

df_results = invert_transformation(df_train, df_forecast, second_diff=False)
df_results.loc[:, ['Close_forecast', 'SMA_10_forecast', 'SMA_50_forecast', 'SMA_200_forecast','close_trend_forecast','cpi_forecast','gross_
```

REF_DATE	Close_forecast	SMA_10_forecast	SMA_50_forecast	SMA_200_forecast	close_trend_forecast	cpi_forecast	gross_income_forecast	housing_fore
2021-12-10	20904.496439	20860.009605	21077.204758	20093.498741	21080.764682	143.998435	645800.308894	121.1
2021-12-13	20917.187533	20836.828358	21091.321072	20105.627188	21063.817101	144.033707	645091.172724	121.2

```
#Plot forecast vs actual

df_results['Close_forecast'].plot(legend=True,title='Close: Forecast vs Actuals')
df_test['Close'][-nobs:].plot(legend=True);
plt.show()
```



```
#Evaluate Forecast
from statsmodels.tsa.stattools import acf
def forecast_accuracy(forecast, actual):
    mape = np.mean(np.abs(forecast - actual)/np.abs(actual))  # MAPE
    me = np.mean(forecast - actual)             # ME
    mae = np.mean(np.abs(forecast - actual))    # MAE
    mpe = np.mean((forecast - actual)/actual)   # MPE
    rmse = np.mean((forecast - actual)**2)**.5  # RMSE
    corr = np.corrcoef(forecast, actual)[0,1]   # corr
    return({'mape':mape, 'me':me, 'mae': mae,
            'mpe': mpe, 'rmse':rmse, 'corr':corr})

print('Forecast Accuracy of: Close')
accuracy_prod = forecast_accuracy(df_results['Close_forecast'].values, df_test['Close'])
for k, v in accuracy_prod.items():
    print(k, ': ', round(v,4))

"""
MAPE: Mean Absolute Percentage Error is 0.0121
ME: Mean Error is -28.6015
MAE: Mean Absolute Error is 254.9133
MPE: Mean Percentage Error: -0.0012
RMSE: Root Mean Squared Error is 289.2311
CORR: Correlation is -0.5816

Since the difference between MAE and RMSE is around 30, which means the variance in the individual errors
in the sample is relatively small.
The MAPE is low at 1.21%.
CORR represents the Pearson product-moment correlation coefficients. At -0.5818 there exist a medium/large association between
the actual value and the predicted value.

Coefficient, r
Strength of Association Positive  Negative
Small                  .1 to .3 or  -0.1 to -0.3
Medium                 .3 to .5 or  -0.3 to -0.5
Large                  .5 to 1.0 or -0.5 to -1.0
"""
```

```
Forecast Accuracy of: Close
mape :  0.0121
me :  -28.6015
mae :  254.9133
mpe :  -0.0012
rmse :  289.2311
corr :  -0.5816
'\nMAPE: Mean Absolute Percentage Error is 0.0121\nME: Mean Error is -28.6015\nMAE: Mean Absolute Error is 254.9133\nMPE: Mean Percentage Error: -
0.0012\nRMSE: Root Mean Squared Error is 289.2311\nCORR: Correlation is -0.5816\n\nSince the difference between MAE and RMSE is around 30, which me
ans the variance in the individual errors \nin the sample is relatively small.\nThe MAPE is low at 1.21%.\nCORR represents the Pearson product-mome
nt correlation coefficients. At -0.5818 there exist a medium/large association between \nthe actual value and the predicted value. \n\nCoefficient,
r\nStrength of Association\tPositive\tNegative\nSmall\t                 .1 to .3 or\t-0.1 to -0.3\nMedium\t               .3 to .5 or\t-0.3 to -0.5
\nLarge\t                 .5 to 1.0 or\t-0.5 to -1.0\n'
```

0s completed at 2:54 PM