Lanyue Zhang (606086099)
Juntao Li (906092273)
Wenxuan Du (506084251)
Steven Li (705494229)

Final Project

June 9, 2023

[1]:
```
import pandas as pd
import numpy as np
import itertools

#Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns
import altair as alt

#statistics libraries
import statsmodels.api as sm
import statsmodels.formula.api as smf
import scipy
from scipy.stats import anderson
from statsmodels.tools.eval_measures import rmse
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import month_plot, seasonal_plot, plot_acf,␣
↪plot_pacf, quarter_plot
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.holtwinters import ExponentialSmoothing, SimpleExpSmoothing
from statsmodels.stats.diagnostic import acorr_ljungbox as ljung
from statsmodels.tsa.statespace.tools import diff as diff
import pmdarima as pm
from pmdarima import ARIMA, auto_arima
from scipy import signal
from scipy.stats import shapiro
from scipy.stats import boxcox
from sklearn.preprocessing import StandardScaler
from scipy.stats import pearsonr
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import grangercausalitytests

import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
gold = pd.read_excel('gold.xls')
gold
```

[2]:

```
           Date       Gold
0    1994-01-01  137.31491
1    1994-02-01  136.52517
2    1994-03-01  133.95854
3    1994-04-01  134.74827
4    1994-05-01  133.46496
..          …          …
347  2022-12-01  617.17670
348  2023-01-01  638.79566
349  2023-02-01  662.68509
350  2023-03-01  634.15597
351  2023-04-01  693.48470

[352 rows x 2 columns]
```

[3]:

```
gold['Date'] = pd.to_datetime(gold['Date'])
gold.set_index('Date', inplace=True)
```

[4]:

```
gold.head()
```

[4]:

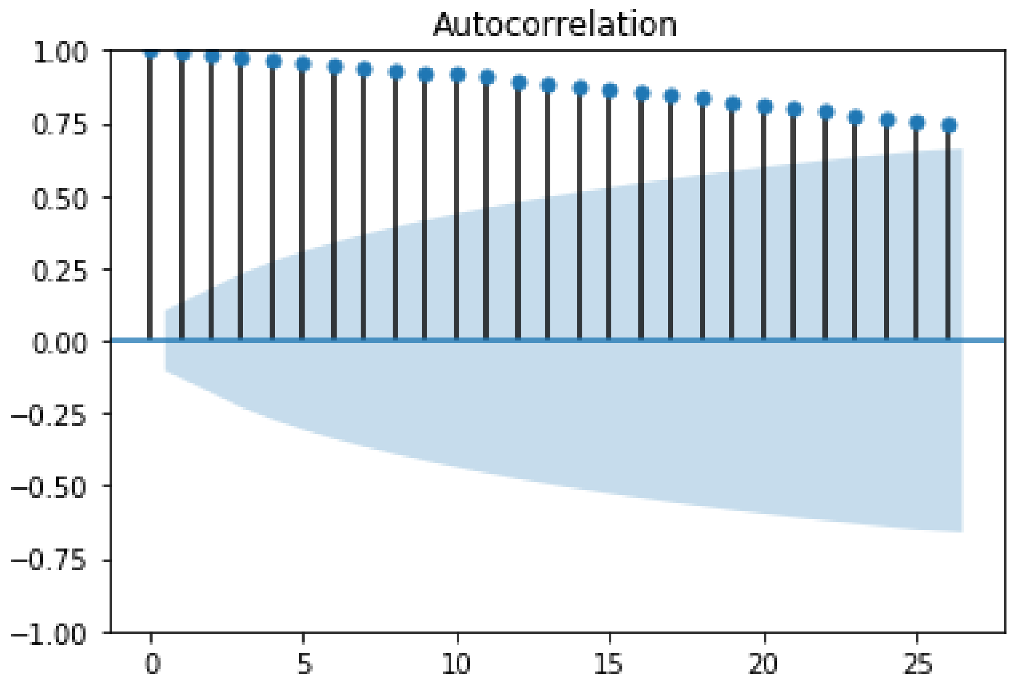
```
                 Gold
Date
1994-01-01  137.31491
1994-02-01  136.52517
1994-03-01  133.95854
1994-04-01  134.74827
1994-05-01  133.46496
```

1 Time Series Plot

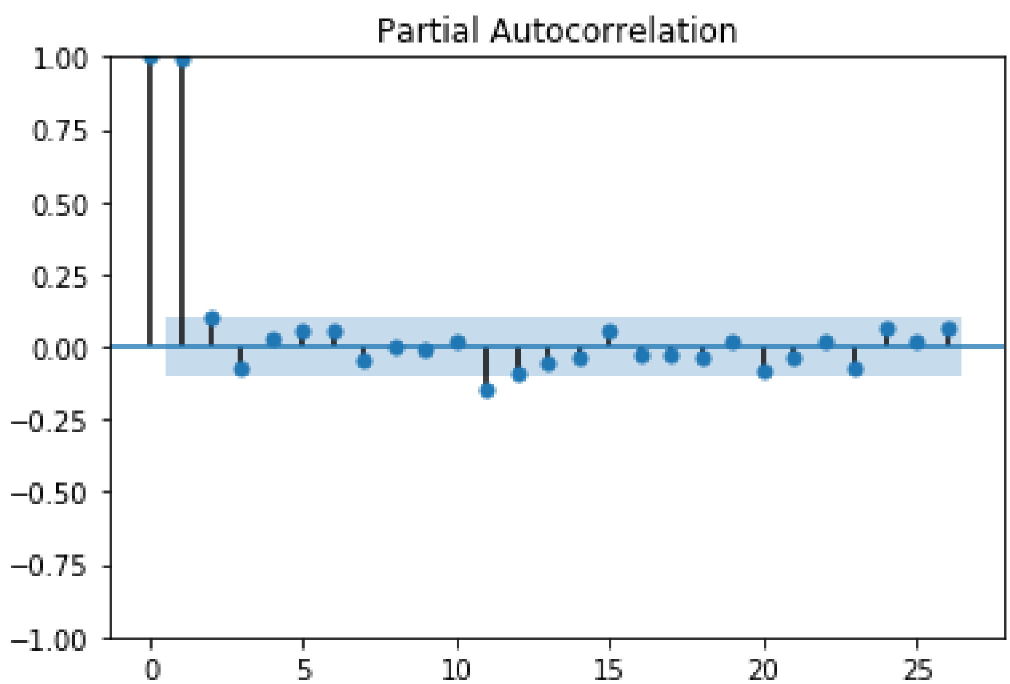
[64]:

```
plt.figure(figsize=(8,6))
plt.plot(gold['Gold'])
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Export Gold Index over Time')
plt.show()
```

[6]:
```
plot_acf(gold['Gold']);
```



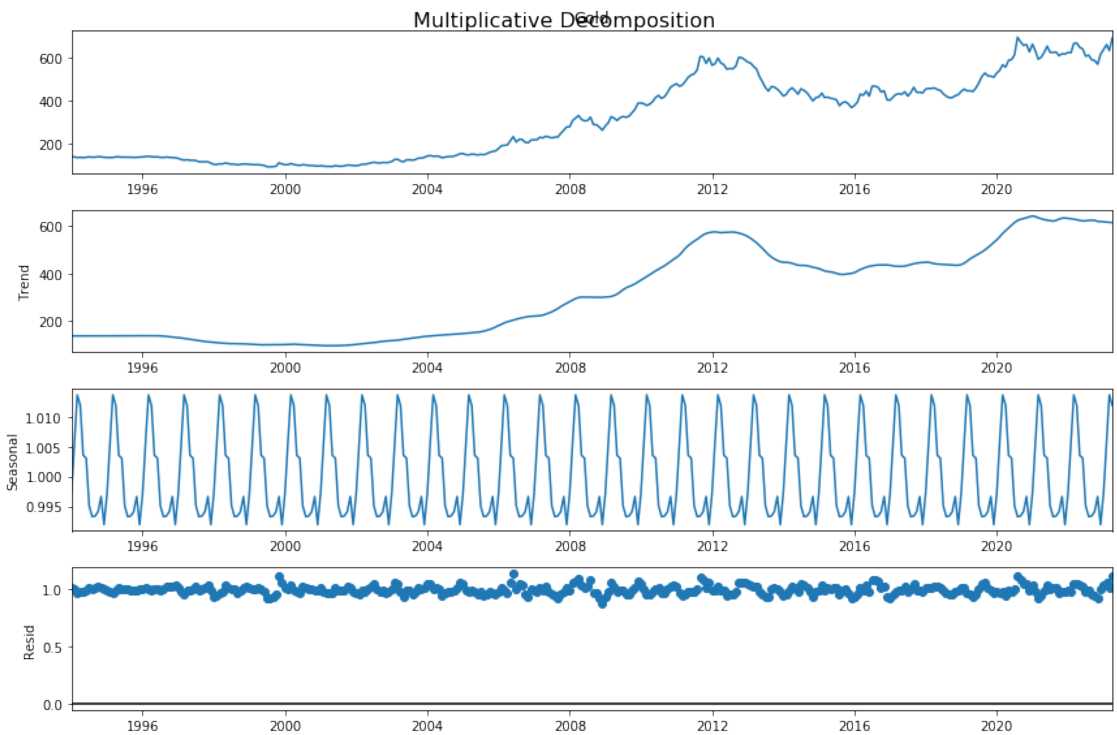
[7]:
```
plot_pacf(gold['Gold']);
```

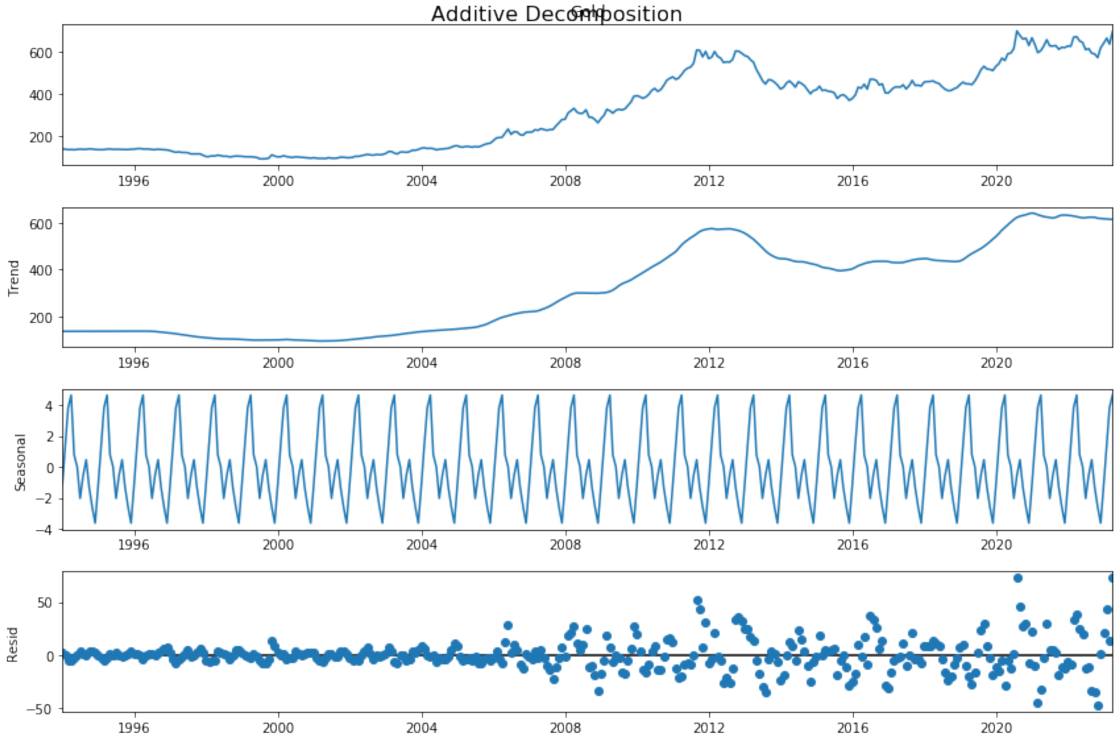


2 STL decomposition

[114]:
```python
# Multiplicative Decomposition
decomposeM = seasonal_decompose(gold["Gold"],model='multiplicative',
 ↪extrapolate_trend='freq')
plt.rcParams['figure.figsize'] = (12, 8);
#decomposeM.plot();
decomposeM.plot().suptitle('Multiplicative Decomposition', fontsize=16)

# Additive Decomposition
decomposeA = seasonal_decompose(gold["Gold"],model='additive',
 ↪extrapolate_trend='freq')
plt.rcParams['figure.figsize'] = (12, 8);
#decomposeA.plot();
decomposeA.plot().suptitle('Additive Decomposition', fontsize=16)
```

[114]:
```
Text(0.5, 0.98, 'Additive Decomposition')
```





[9]:
```
ljung_p = np.mean(ljung(x=decomposeA.resid.dropna())['lb_pvalue'])
ljung_p = round(ljung_p, 3)
print("Ljung Box (A), p value:", ljung_p, ", Residuals are uncorrelated" if␣
 ↪ljung_p>0.05 else ", Residuals are correlated")
```

```
Ljung Box (A), p value: 0.0 , Residuals are correlated
```

[10]:
```
ljung_p = np.mean(ljung(x=decomposeM.resid.dropna())['lb_pvalue'])
ljung_p = round(ljung_p, 3)
print("Ljung Box (A), p value:", ljung_p, ", Residuals are uncorrelated" if␣
 ↪ljung_p>0.05 else ", Residuals are correlated")
```

```
Ljung Box (A), p value: 0.0 , Residuals are correlated
```

```
[1]: import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import grangercausalitytests
from scipy.stats import pearsonr
from sklearn.metrics import mean_squared_error
```

3 VAR Model

```
[2]: df = pd.read_excel("G&O.xls",index_col=0)
df.head()
```

```
[2]:                 Oil       Gold
Date
1994-01-01  55.11754  137.31491
1994-02-01  52.44123  136.52517
1994-03-01  53.45389  133.95854
1994-04-01  61.19349  134.74827
1994-05-01  66.18445  133.46496
```

```
[3]: plt.figure(figsize=(12,6))
gold, = plt.plot(df['Gold'])
oil, = plt.plot(df['Oil'], color='red')

for year in range(1994, 2025):
    plt.axvline(datetime(year,1,1), linestyle='--', color='k', alpha=0.1)

plt.xlabel('Time')
plt.ylabel('Price Index')
plt.legend(['Gold', 'Oil'], fontsize=16)
plt.show()
```

3.1 ADF Test

[4]:
```
adf_test_oil = adfuller(df['Oil'])
adf_statistic_oil = adf_test_oil[0]
adf_pvalue_oil = adf_test_oil[1]
adf_critical_values_oil = adf_test_oil[4]

print('ADF Test for Oil:')
print(f'ADF Statistic: {adf_statistic_oil}')
print(f'p-value: {adf_pvalue_oil}')
```

```
ADF Test for Oil:
ADF Statistic: -2.3964936369495327
p-value: 0.14272850042174443
```

[5]:
```
adf_test_gold = adfuller(df['Gold'])
adf_statistic_gold = adf_test_gold[0]
adf_pvalue_gold = adf_test_gold[1]
adf_critical_values_gold = adf_test_gold[4]

print('ADF Test for Gold:')
print(f'ADF Statistic: {adf_statistic_gold}')
print(f'p-value: {adf_pvalue_gold}')
```

```
ADF Test for Gold:
ADF Statistic: 0.3405591027490211
p-value: 0.9791306790222112
```

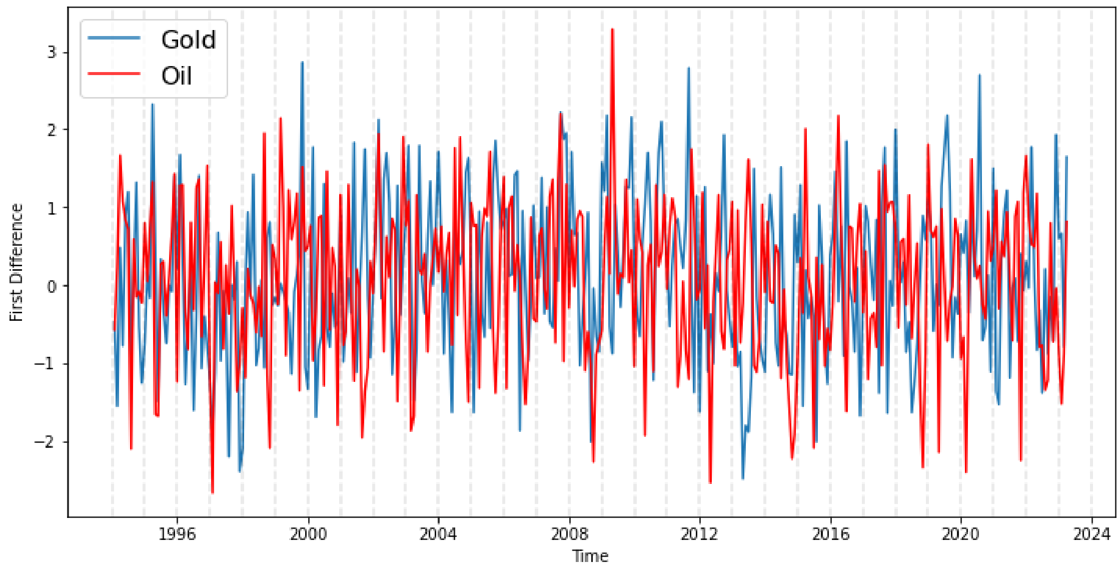
3.2 Normalized

[6]:
```
# Normalized
avg = df.mean()
dev = df.std()

for col in df:
    df[col] = (df[col]-avg.loc[col]) / dev.loc[col]


plt.figure(figsize=(12,6))
gold, = plt.plot(df['Gold'])
oil, = plt.plot(df['Oil'], color='red')

for year in range(1994, 2025):
    plt.axvline(datetime(year,1,1), linestyle='--', color='k', alpha=0.1)

plt.xlabel('Time')
plt.legend(['Gold', 'Oil'], fontsize=16)
plt.show()
```

[7]:
```
# Remove Trend
df = df.diff().dropna()

plt.figure(figsize=(12,6))
gold, = plt.plot(df['Gold'])
oil, = plt.plot(df['Oil'], color='red')
```

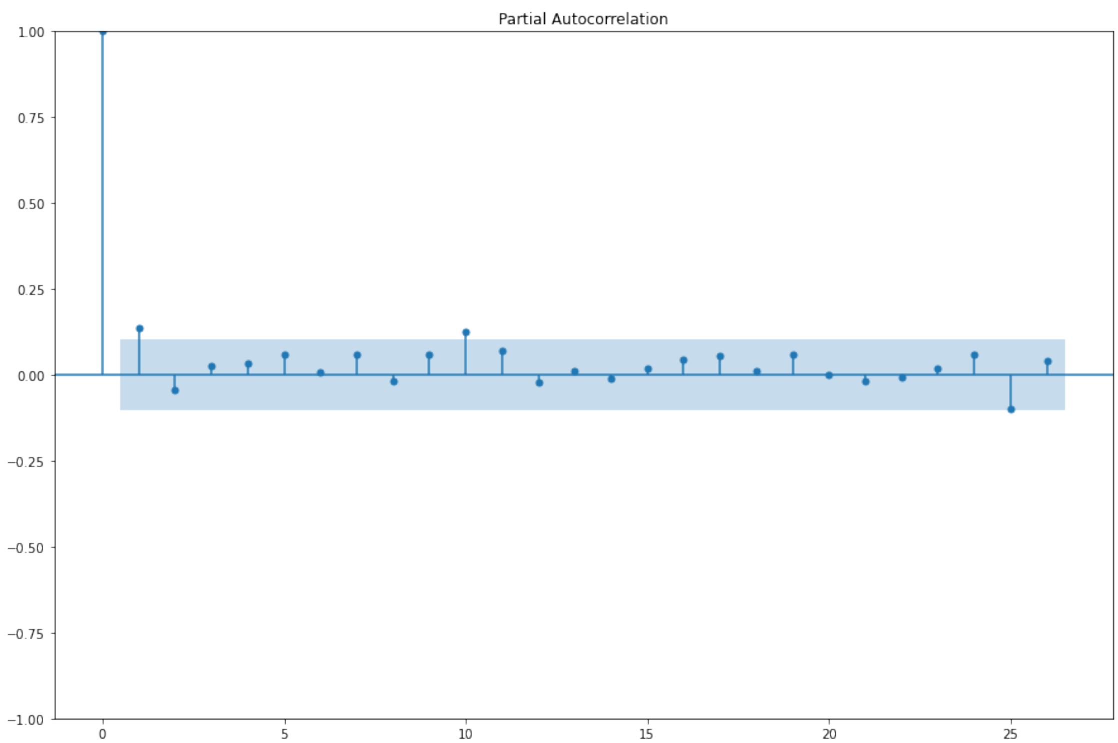
```python
for year in range(1994, 2025):
    plt.axvline(datetime(year,1,1), linestyle='--', color='k', alpha=0.1)

plt.xlabel('Time')
plt.ylabel('First Difference')
plt.legend(['Gold', 'Oil'], fontsize=16)
plt.show()
```

[8]:
```python
# Remove Increasing Volatility
vol = df.groupby(df.index.year).std()

df['gold_vol'] = df.index.map(lambda d: vol.loc[d.year, 'Gold'])
df['oil_vol'] = df.index.map(lambda d: vol.loc[d.year, 'Oil'])

df['Gold'] = df['Gold']/df['gold_vol']
df['Oil'] = df['Oil']/df['oil_vol']

plt.figure(figsize=(12,6))
gold, = plt.plot(df['Gold'])
oil, = plt.plot(df['Oil'], color='red')

for year in range(1994, 2025):
    plt.axvline(datetime(year,1,1), linestyle='--', color='k', alpha=0.1)

plt.xlabel('Time')
plt.ylabel('First Difference')
plt.legend(['Gold', 'Oil'], fontsize=16)
plt.show()
```



[9]:
```
fig, ax = plt.subplots(figsize=(15, 10))
plot_pacf(df["Gold"], method='ywm', ax=ax)
plt.show
```

[9]: `<function matplotlib.pyplot.show(close=None, block=None)>`



3.3 VAR Model

[10]:
```
var_df = df[["Gold","Oil"]]
var_df.index = pd.DatetimeIndex(var_df.index, freq='MS')

train_data = var_df[:-12]
test_data = var_df[-12:]
```

[11]:
```
var_mod = VAR(train_data)
var_mod_fit = var_mod.fit(maxlags=12)
var_mod_fit.summary()
```

[11]:
```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Thu, 08, Jun, 2023
Time:                     18:30:28
--------------------------------------------------------------------
No. of Equations:         2.00000    BIC:                   0.767051
Nobs:                     327.000    HQIC:                  0.418778
Log likelihood:          -908.650    FPE:                    1.20701
AIC:                     0.187547    Det(Omega_mle):         1.04165
--------------------------------------------------------------------
Results for equation Gold
===========================================================================
              coefficient       std. error           t-stat            prob
---------------------------------------------------------------------------
const            0.063003         0.059742            1.055           0.292
L1.Gold          0.135864         0.057532            2.362           0.018
L1.Oil           0.021502         0.058715            0.366           0.714
L2.Gold         -0.053441         0.057868           -0.924           0.356
L2.Oil           0.140323         0.058286            2.407           0.016
L3.Gold          0.030404         0.057641            0.527           0.598
L3.Oil          -0.036469         0.059061           -0.617           0.537
L4.Gold         -0.007928         0.057692           -0.137           0.891
L4.Oil          -0.071068         0.059262           -1.199           0.230
L5.Gold          0.057933         0.057798            1.002           0.316
L5.Oil          -0.040417         0.059400           -0.680           0.496
L6.Gold         -0.009281         0.057661           -0.161           0.872
L6.Oil          -0.014732         0.059161           -0.249           0.803
L7.Gold          0.103544         0.057954            1.787           0.074
L7.Oil           0.001673         0.059025            0.028           0.977
L8.Gold         -0.022509         0.058064           -0.388           0.698
L8.Oil          -0.047345         0.059137           -0.801           0.423
L9.Gold          0.044247         0.058091            0.762           0.446
```

```
L9.Oil          -0.013884         0.058929          -0.236           0.814
L10.Gold         0.123153         0.058298           2.112           0.035
L10.Oil         -0.014806         0.058412          -0.253           0.800
L11.Gold         0.071997         0.057829           1.245           0.213
L11.Oil          0.066070         0.058104           1.137           0.255
L12.Gold        -0.026091         0.057605          -0.453           0.651
L12.Oil          0.006586         0.058581           0.112           0.910
===========================================================================

Results for equation Oil
===========================================================================
              coefficient       std. error          t-stat            prob
---------------------------------------------------------------------------
const            0.061039         0.058229           1.048           0.295
L1.Gold          0.029253         0.056075           0.522           0.602
L1.Oil           0.045530         0.057228           0.796           0.426
L2.Gold          0.094658         0.056403           1.678           0.093
L2.Oil          -0.075694         0.056810          -1.332           0.183
L3.Gold          0.108027         0.056181           1.923           0.055
L3.Oil           0.007438         0.057565           0.129           0.897
L4.Gold          0.068013         0.056231           1.210           0.226
L4.Oil          -0.068223         0.057761          -1.181           0.238
L5.Gold          0.017901         0.056334           0.318           0.751
L5.Oil          -0.036593         0.057896          -0.632           0.527
L6.Gold          0.082861         0.056201           1.474           0.140
L6.Oil           0.039106         0.057663           0.678           0.498
L7.Gold         -0.014894         0.056486          -0.264           0.792
L7.Oil          -0.087961         0.057530          -1.529           0.126
L8.Gold         -0.069778         0.056594          -1.233           0.218
L8.Oil          -0.021524         0.057640          -0.373           0.709
L9.Gold          0.020237         0.056620           0.357           0.721
L9.Oil           0.021802         0.057436           0.380           0.704
L10.Gold         0.035720         0.056822           0.629           0.530
L10.Oil          0.043842         0.056933           0.770           0.441
L11.Gold        -0.041313         0.056365          -0.733           0.464
L11.Oil          0.122108         0.056632           2.156           0.031
L12.Gold        -0.063838         0.056146          -1.137           0.256
L12.Oil         -0.077052         0.057098          -1.349           0.177
===========================================================================

Correlation matrix of residuals
            Gold       Oil
Gold    1.000000  0.000252
Oil     0.000252  1.000000
```

Comment:

Therefore our final model will be:

$$Gold = 0.136Gold_{t-1} + 0.14Oil_{t-2}$$

3.4 Granger-Causality Test

[12]:
```
# Perform Granger-Causality test
max_lag = 5  # Maximum lag order
test_results = grangercausalitytests(var_df, maxlag=max_lag)

# Interpret the results
for lag in range(1, max_lag+1):
    print(f'Granger-Causality test (lag={lag}):')
    result = test_results[lag][0]
    p_value = result['ssr_chi2test'][1]
    if p_value < 0.05:
        print(f'  There is evidence of Granger causality (p-value={p_value:.
↪4f})')
    else:
        print(f'  No significant evidence of Granger causality␣
↪(p-value={p_value:.4f})')
    print()
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=1.1795  , p=0.2782  , df_denom=347, df_num=1
ssr based chi2 test:   chi2=1.1897  , p=0.2754  , df=1
likelihood ratio test: chi2=1.1877  , p=0.2758  , df=1
parameter F test:         F=1.1795  , p=0.2782  , df_denom=347, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=3.3004  , p=0.0380  , df_denom=344, df_num=2
ssr based chi2 test:   chi2=6.6968  , p=0.0351  , df=2
likelihood ratio test: chi2=6.6333  , p=0.0363  , df=2
parameter F test:         F=3.3004  , p=0.0380  , df_denom=344, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=2.2737  , p=0.0798  , df_denom=341, df_num=3
ssr based chi2 test:   chi2=6.9611  , p=0.0731  , df=3
likelihood ratio test: chi2=6.8923  , p=0.0754  , df=3
parameter F test:         F=2.2737  , p=0.0798  , df_denom=341, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=2.0311  , p=0.0897  , df_denom=338, df_num=4
```

```
ssr based chi2 test:   chi2=8.3407  , p=0.0799  , df=4
likelihood ratio test: chi2=8.2420  , p=0.0831  , df=4
parameter F test:         F=2.0311  , p=0.0897  , df_denom=338, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=1.5063  , p=0.1872  , df_denom=335, df_num=5
ssr based chi2 test:   chi2=7.7789  , p=0.1688  , df=5
likelihood ratio test: chi2=7.6928  , p=0.1740  , df=5
parameter F test:         F=1.5063  , p=0.1872  , df_denom=335, df_num=5
Granger-Causality test (lag=1):
  No significant evidence of Granger causality (p-value=0.2754)

Granger-Causality test (lag=2):
  There is evidence of Granger causality (p-value=0.0351)

Granger-Causality test (lag=3):
  No significant evidence of Granger causality (p-value=0.0731)

Granger-Causality test (lag=4):
  No significant evidence of Granger causality (p-value=0.0799)

Granger-Causality test (lag=5):
  No significant evidence of Granger causality (p-value=0.1688)
```

[13]:
```
gc_mod_fit = var_mod.fit(maxlags=12,ic="aic")
```

[14]:
```
gc_mod_fit.test_causality("Gold","Oil",kind='f').summary()
```

[14]: <class 'statsmodels.iolib.table.SimpleTable'>

[15]:
```
gc_mod_fit.test_causality("Oil","Gold",kind='f').summary()
```

[15]: <class 'statsmodels.iolib.table.SimpleTable'>

3.5 Forecast

[16]:
```
org_df = pd.read_excel("G&O.xls",index_col=0)
org_df.tail()
```

[16]:

	Oil	Gold
Date		
2022-12-01	289.90958	617.17670
2023-01-01	285.53345	638.79566
2023-02-01	278.04702	662.68509
2023-03-01	273.70705	634.15597
2023-04-01	277.68535	693.48470

[17]:
```
# Forecast 12 steps ahead
forecast = var_mod_fit.forecast(var_mod_fit.endog, steps=12)

forecast_gold = forecast[:, 0]

# Print forecasted values
print("Forecasted Gold Price:", forecast_gold)
```

```
Forecasted Gold Price: [-0.23615863 -0.10639113 -0.06046261  0.1164994
0.06634034  0.0424161
  0.04570182  0.16772837  0.39596913  0.3107158   0.05819648 -0.00217214]
```

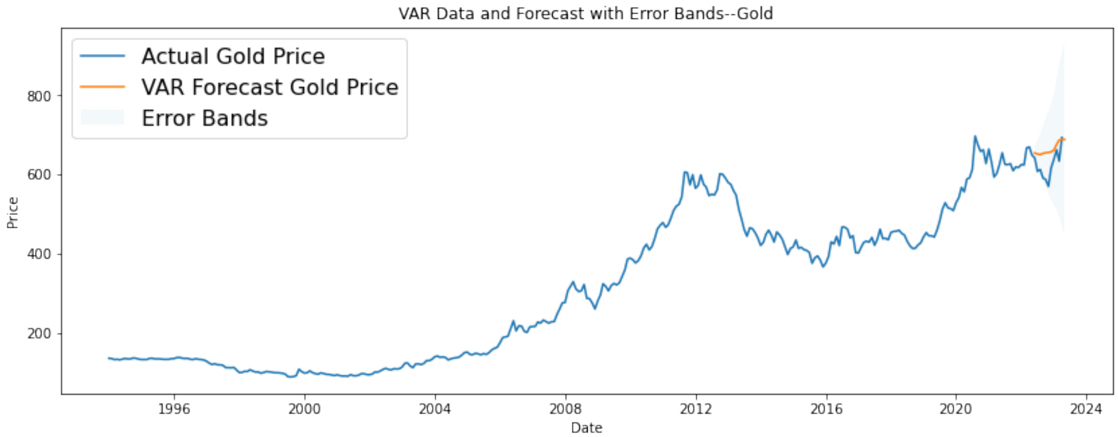
[18]:
```
# Transform Back
forecast_gold = forecast_gold * np.array(df['gold_vol'][-12:])
forecast_gold = pd.DataFrame(forecast_gold).cumsum()
forecast_gold = forecast_gold*dev[1]+2.1*avg[1]
forecast_gold = np.array(forecast_gold).flatten()
forecast_gold
```

[18]:
```
array([654.71496905, 652.1338232 , 650.666945  , 653.49332665,
       655.10280372, 656.13185712, 657.24062491, 661.3098683 ,
       675.61629665, 686.84250823, 688.94515646, 688.8666768 ])
```

[19]:
```
# Calculate confidence intervals for the forecasted values
alpha = 0.5 # significance level
forecast_ci = var_mod_fit.forecast_interval(var_mod_fit.endog[-var_mod_fit.k_ar:
 ↪], steps=12, alpha=alpha)

# Access confidence intervals for Gold and Transform Back
ci_lower_gold = pd.DataFrame(forecast_ci[1][:, 0]*np.array(df['gold_vol'][-12:
 ↪])).cumsum()*dev[1]+2.1*avg[1]
ci_lower_gold = np.array(ci_lower_gold).flatten()
ci_upper_gold = pd.DataFrame(forecast_ci[2][:, 0]*np.array(df['gold_vol'][-12:
 ↪])).cumsum()*dev[1]+2.1*avg[1]
ci_upper_gold = np.array(ci_upper_gold).flatten()
```

[29]:
```
# Plot forecast with error bands
plt.figure(figsize=(14, 5))
plt.plot(org_df['Gold'], label='Actual Gold Price')
forecast_dates = pd.date_range(start=org_df.index[-12] + pd.DateOffset(days=1),␣
 ↪periods=12, freq='M')
plt.plot(forecast_dates, forecast_gold, label='VAR Forecast Gold Price')
plt.fill_between(forecast_dates,
                 ci_lower_gold,
                 ci_upper_gold,
                 alpha=0.05, label='Error Bands')
plt.legend(fontsize=16, loc='upper left')
```

```
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('VAR Data and Forecast with Error Bands--Gold')
plt.show()
```



3.6 Model Performance

[24]:
```
# Calculate RMSE
rmse = np.sqrt(mean_squared_error(org_df["Gold"][-12:], forecast_gold))
print("RMSE:", rmse)
```

```
RMSE: 47.183848206899654
```

[25]:
```
# Calculate MAPE
mape = np.mean(np.abs((org_df["Gold"][-12:] - forecast_gold) /
 ↪org_df["Gold"][-12:])) * 100
print("MAPE:", mape)
```

```
MAPE: 6.62067276982332
```

In [1]:
```
########## Working with time series data in R
library(forecast)
library(ggplot2)
library(readxl)
data = read_excel('G&O.xls')
```

```
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
```

In [2]:
```
library(readxl)
```

In [3]:
```
head(data)

data[1,1]
```

A tibble: 6 × 3

Date	Oil	Gold
<dttm>	<dbl>	<dbl>
2008-02-01	369.5281	307.5025
2008-03-01	402.4522	318.6574
2008-04-01	429.2582	330.2073
2008-05-01	483.2078	312.3396
2008-06-01	517.9212	305.5281
2008-07-01	521.1387	307.0089

A tibble: 1 ×
1

Date
<dttm>
2008-02-01

In [4]:
```r
# convert data to a monthly time series, starting 1/2004
my_ts <- ts(data$Gold, start=1994, freq=12)
my_ts
```

A Time Series: 16 × 12

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
1994	307.5025	318.6574	330.2073	312.3396	305.5281	307.0089	323.0997	288.2527	287.5617
1995	297.2359	325.1728	318.3613	307.1076	320.3356	325.8638	322.2113	327.4432	343.6328
1996	385.0938	377.4926	383.5143	395.4590	415.1037	424.5804	410.4640	419.0523	438.6969
1997	467.4235	473.2478	490.6219	509.5755	520.5331	525.3702	544.1264	606.5153	605.2320
1998	573.0503	599.4077	576.2093	568.4107	547.2853	550.5429	548.8648	561.9941	602.3692
1999	575.3208	560.6120	548.5686	513.5242	488.1540	461.5992	444.9161	465.9427	463.7710
2000	429.8124	449.2596	460.1185	447.6802	430.3060	455.6762	448.0750	437.1175	418.0652
2001	435.1431	414.3139	416.8806	411.1550	409.0819	403.6525	376.7029	390.4245	395.1629
2002	393.5834	430.5035	425.1728	444.4225	421.3228	468.1145	467.7196	462.3889	440.6713
2003	417.1767	428.9240	432.5765	429.8124	441.8559	421.9151	437.1175	462.4877	438.8944
2004	457.0582	457.3544	460.1185	452.1224	447.6802	432.4778	420.8292	413.8203	414.6101
2005	454.0967	446.0020	446.1007	442.5469	459.8223	483.1194	512.9319	529.3188	516.5844
2006	542.2507	567.8184	556.8608	588.8450	592.3001	613.8203	697.1372	675.2221	658.5390
2007	631.3919	594.1757	603.3564	625.5676	655.0839	626.1599	625.3702	627.5420	609.9704
2008	624.5805	667.4235	669.7927	649.5558	641.6584	608.0948	613.0306	591.6091	587.8578
2009	662.6851	634.1560	693.4847						

In [5]:

```
plot(my_ts,col = 'red', main = 'Gold', ylab = 'Index')

autoplot(my_ts) +
  ggtitle('Gold Price Trends') +
  ylab('Index') +
  xlab('Date')

# stats
mean(my_ts)

sd(my_ts)

median(my_ts)

summary(my_ts)
```

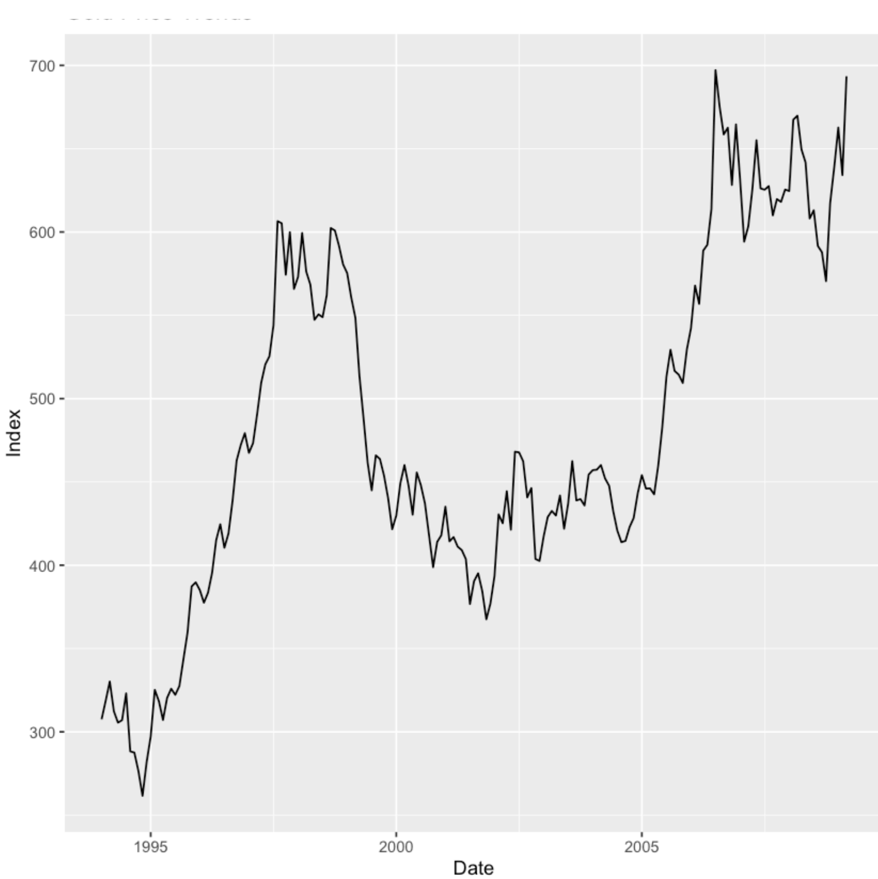
479.892004535519

104.875175410121

454.39289

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  261.6   414.9   454.4   479.9   573.7   697.1 
```

Gold Price Trends



In [6]:
```
# assignment
tsmean = mean(my_ts)
print(tsmean)

tsmean <- mean(my_ts)
print(tsmean)

mean(my_ts) -> tsmean
print(tsmean)
```

```
[1] 479.892
[1] 479.892
[1] 479.892
```

4. MAPA

In [7]:
```
install.packages("MAPA")
```

```
The downloaded binary packages are in
        /var/folders/81/jjv33vws4cs7b2lsyrkgf1q00000gn/T//Rtmp8wh7OB/
downloaded_packages
```

In [8]:
```
install.packages("thief")
```

```
The downloaded binary packages are in
        /var/folders/81/jjv33vws4cs7b2lsyrkgf1q00000gn/T//Rtmp8wh7OB/
downloaded_packages
```

In [9]:
```
################ ARIMA ###################################
library(forecast)
# arima example
model1 = auto.arima(my_ts)
model1

# make a 12 step forecast
fcast = forecast(model1, 12)
fcast

# get the forecasted values
fcast$mean
```

```
Series: my_ts
ARIMA(0,1,0) with drift

Coefficients:
       drift
      2.1208
s.e.  1.4659

sigma^2 = 393.2:  log likelihood = -801.42
AIC=1606.83   AICc=1606.9   BIC=1613.24

         Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
Apr 2009       695.6055 670.1921 721.0189 656.7390 734.4720
May 2009       697.7263 661.7862 733.6663 642.7608 752.6918
Jun 2009       699.8470 655.8297 743.8644 632.5283 767.1658
Jul 2009       701.9678 651.1410 752.7947 624.2349 779.7008
Aug 2009       704.0886 647.2625 760.9148 617.1805 790.9967
Sep 2009       706.2094 643.9595 768.4593 611.0063 801.4124
Oct 2009       708.3302 641.0926 775.5678 605.4991 811.1612
Nov 2009       710.4510 638.5709 782.3310 600.5199 820.3820
Dec 2009       712.5717 636.3314 788.8120 595.9723 829.1712
Jan 2010       714.6925 634.3282 795.0568 591.7859 837.5991
Feb 2010       716.8133 632.5265 801.1001 587.9078 845.7188
Mar 2010       718.9341 630.8994 806.9688 584.2966 853.5715
```

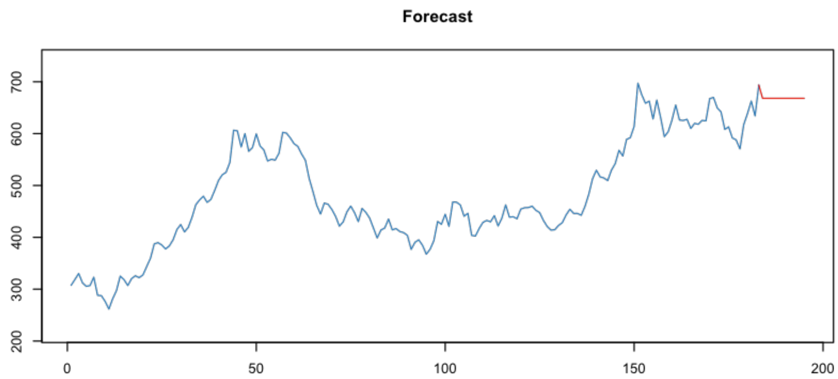
A Time Series: 2 × 12

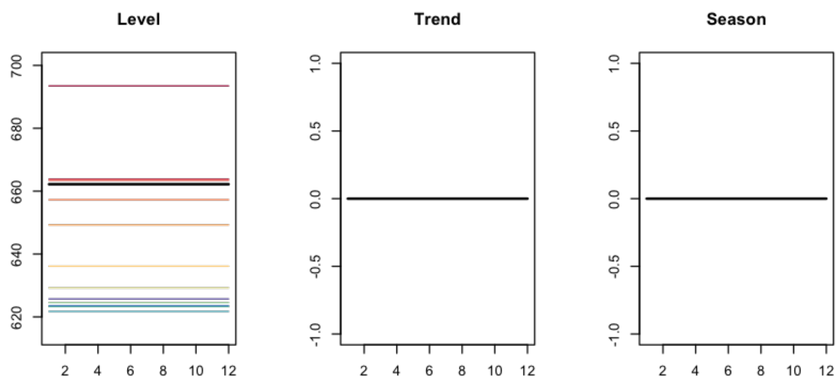
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
2009				695.6055	697.7263	699.8470	701.9678	704.0886	706.2094
2010	714.6925	716.8133	718.9341						

In [10]:

```
######### MAPA example ################################################
library(MAPA)

# Detailed view of the data at each temporal aggregation level (daily,
# Note: paral = 2 --> Run in parallel cluster mode with e.g., in my ca
mapasimple(my_ts,outplot=2,paral=2)

#Dynamic Fit to the data
mapafit <- mapaest(my_ts,outplot=2, paral=2)
plot(mapafit)
mapafit
mapafor(my_ts,mapafit,ifh=12,fh=0)


# Best fit MAPA model by temporal decomposition + Forecast
# N=None, A=Additive, M=Multiplicative, d=Damped
# Exmaple: MAM = Holt-Winters
mapafit <- mapaest(my_ts,paral=2)
mapafor(my_ts,mapafit)

# Forecast with Error Bands
mapa(my_ts,conf.lvl=c(0.8,0.9,0.95,0.99),outplot=1)
```

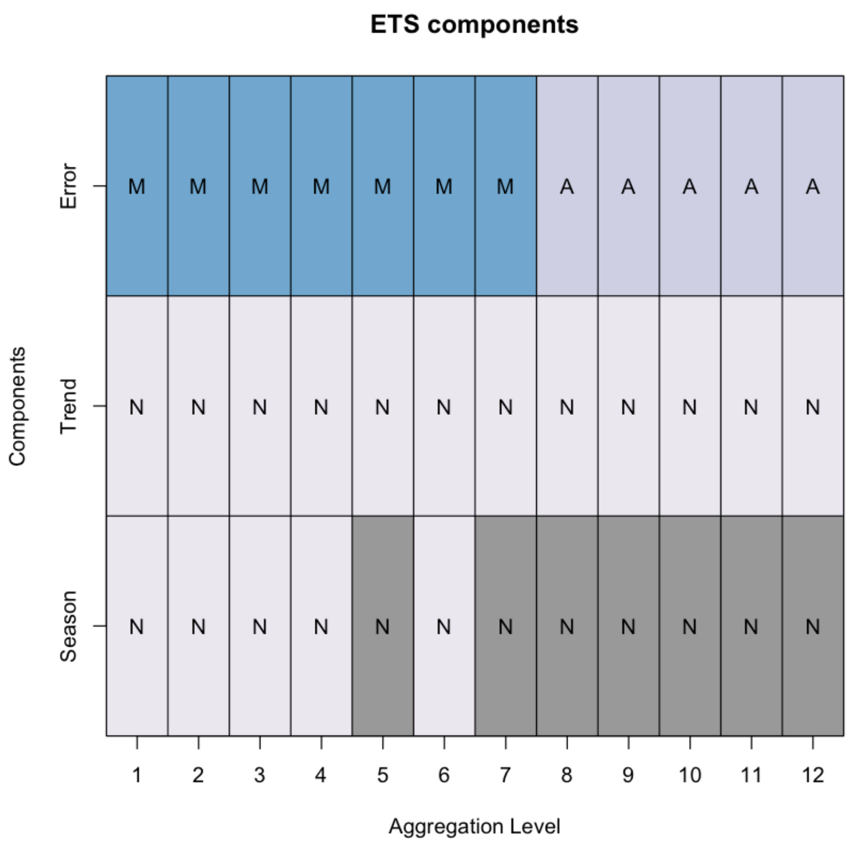
```
Loading required package: parallel

Loading required package: RColorBrewer

Loading required package: smooth

Loading required package: greybox

Package "greybox", v1.0.8 loaded.


This is package "smooth", v3.2.1



Running with 8 cores

     t+1      t+2      t+3      t+4      t+5      t+6      t+7      t
+8
668.0484 668.0484 668.0484 668.0484 668.0484 668.0484 668.0484 668.04
84
     t+9     t+10     t+11     t+12
668.0484 668.0484 668.0484 668.0484

Running with 8 cores
```





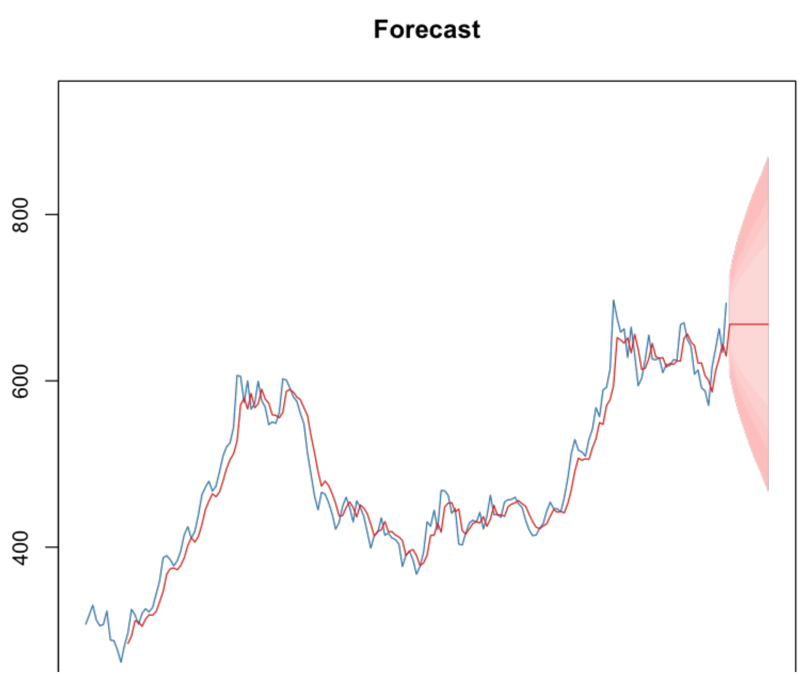

```
MAPA fitted using ets   Original frequency: 12
Aggregation level: 1    Method: ETS(MNN)
Aggregation level: 2    Method: ETS(MNN)
Aggregation level: 3    Method: ETS(MNN)
Aggregation level: 4    Method: ETS(MNN)
Aggregation level: 5    Method: ETS(MNN)
Aggregation level: 6    Method: ETS(MNN)
Aggregation level: 7    Method: ETS(MNN)
Aggregation level: 8    Method: ETS(ANN)
Aggregation level: 9    Method: ETS(ANN)
Aggregation level: 10   Method: ETS(ANN)
Aggregation level: 11   Method: ETS(ANN)
Aggregation level: 12   Method: ETS(ANN)

MAPA fit MSE: 560.89, MAE: 18.06
 MAPA fit MSE: 987.74, MAE: 24.14

 MAPA fit MSE: 1445.63, MAE: 30.06
 MAPA fit MSE: 1898.27, MAE: 34.8
 MAPA fit MSE: 2352.04, MAE: 38.85
 MAPA fit MSE: 2843.24, MAE: 43.16
 MAPA fit MSE: 3367.11, MAE: 45.98
 MAPA fit MSE: 3880.34, MAE: 48.85
 MAPA fit MSE: 4387.12, MAE: 51.75
 MAPA fit MSE: 4897.57, MAE: 54.7
 MAPA fit MSE: 5522.04, MAE: 58.64
```

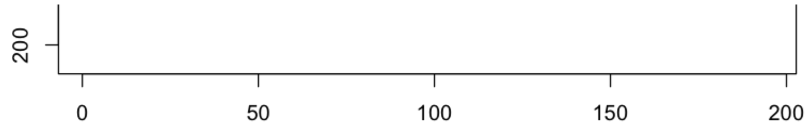
```
 MAPA fit MSE: 6158.1, MAE: 61.96
Out-of-samplpe forecasts:
NULL

Running with 8 cores

MAPA fit MSE: 560.89, MAE: 18.06
Out-of-samplpe forecasts:
     t+1      t+2      t+3      t+4      t+5      t+6      t+7      t
+8
668.0484 668.0484 668.0484 668.0484 668.0484 668.0484 668.0484 668.04
84
     t+9     t+10     t+11     t+12
668.0484 668.0484 668.0484 668.0484
```




```
MAPA fit MSE: 560.89, MAE: 18.06
Out-of-samplpe forecasts:

     Lower 0.80 Lower 0.90 Lower 0.95 Lower 0.99 Forecast Upper 0.99
Upper 0.95
t+1    637.6973   629.0931   621.6304   607.0447 668.0484   729.0520
714.4664
t+2    627.7713   616.3533   606.4499   587.0942 668.0484   749.0026
729.6469
t+3    619.3219   605.5086   593.5277   570.1116 668.0484   765.9852
742.5691
```

```
t+4    612.2123   596.3835   582.6544   555.8217 668.0484   780.2751
753.4423
t+5    605.8960   588.2766   572.9944   543.1263 668.0484   792.9705
763.1023
t+6    599.7134   580.3414   563.5391   530.6999 668.0484   805.3969
772.5577
t+7    593.6841   572.6029   554.3180   518.5814 668.0484   817.5154
781.7788
t+8    588.2175   565.5866   545.9576   507.5939 668.0484   828.5029
790.1392
t+9    583.1644   559.1010   538.2296   497.4375 668.0484   838.6593
797.8672
t+10   578.3621   552.9372   530.8850   487.7851 668.0484   848.3117
805.2118
t+11   572.8158   545.8186   522.4026   476.6374 668.0484   859.4594
813.6942
t+12   567.4805   538.9709   514.2431   465.9139 668.0484   870.1829
821.8537
      Upper 0.90 Upper 0.80
t+1    707.0036   698.3995
t+2    719.7435   708.3255
t+3    730.5881   716.7749
t+4    739.7133   723.8845
t+5    747.8202   730.2008
t+6    755.7554   736.3834
t+7    763.4939   742.4127
t+8    770.5102   747.8793
t+9    776.9958   752.9324
t+10   783.1596   757.7347
t+11   790.2782   763.2810
t+12   797.1259   768.6163
```





In [11]:
```
mapafit <- mapaest(my_ts,paral=2)
mapafor(my_ts,mapafit)
```

```
Running with 8 cores

MAPA fit MSE: 560.89, MAE: 18.06
Out-of-samplpe forecasts:
     t+1      t+2      t+3      t+4      t+5      t+6      t+7      t
+8
668.0484 668.0484 668.0484 668.0484 668.0484 668.0484 668.0484 668.04
84
     t+9     t+10     t+11     t+12
668.0484 668.0484 668.0484 668.0484
```

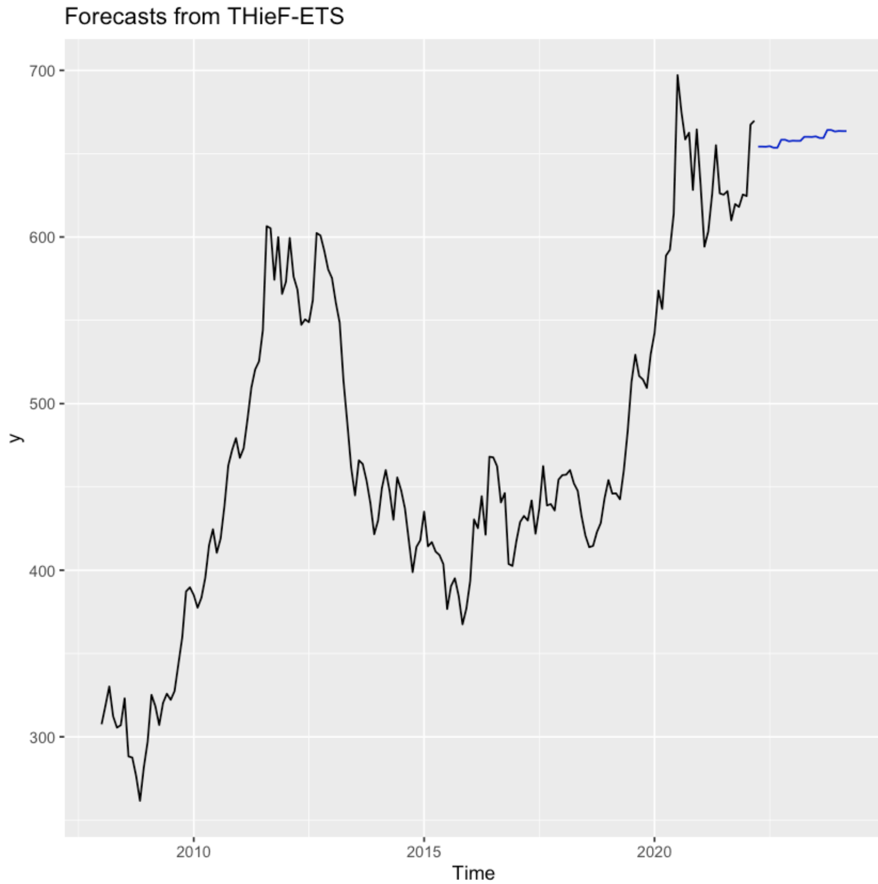
In [12]:
```
ts_data <- ts(data$Gold, start=2008, freq=12)
train_data <- window(ts_data, end = c(2022, 3))
test_data <- window(ts_data, start = c(2022, 4))
```

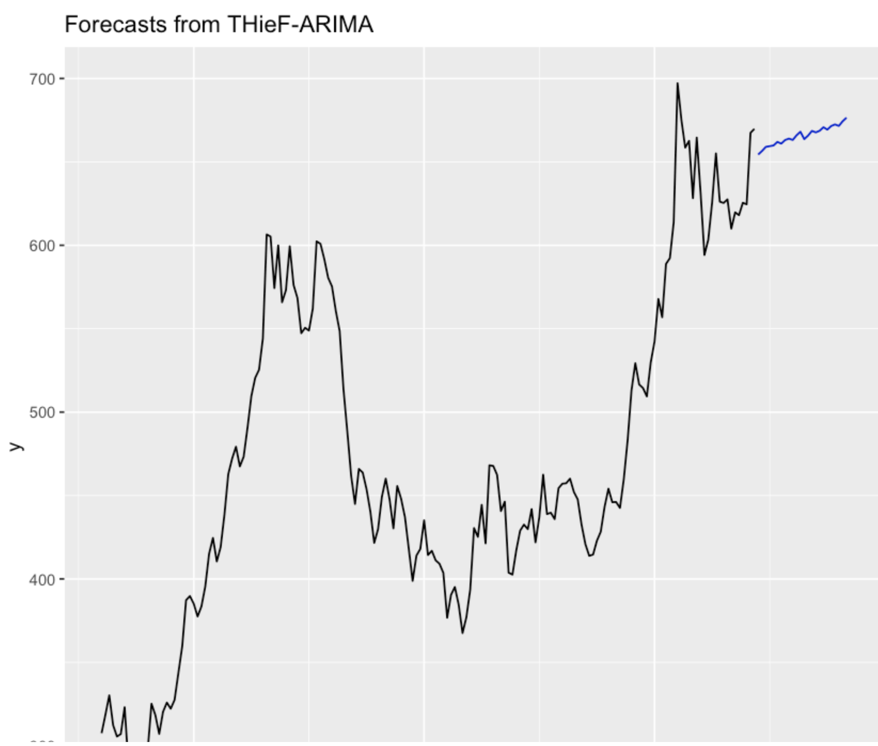
5. THieF

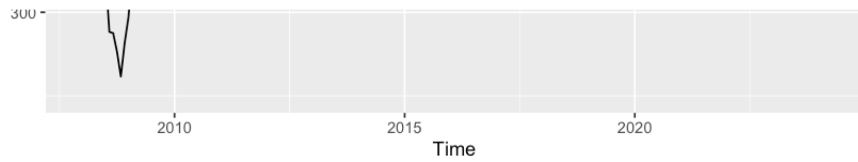
In [13]:
```
######### Thief example ################################################
# install.packages('thief', dependencies = TRUE)
library(thief)
library(ggplot2)

fc <- thief(train_data)
autoplot(fc)

fc <- thief(train_data, usemodel = 'arima')
fc
autoplot(fc)
```

```
          Jan      Feb      Mar      Apr      May      Jun      Jul
Aug
2022                             654.3989 656.5300 659.0074 659.4236 6
59.8415
2023 663.1512 665.9324 668.0635 663.6346 665.7657 668.5731 667.6691 6
68.6513
2024 671.5462 674.3548 676.4860
```

```
          Sep      Oct      Nov      Dec
2022 661.9726 660.9455 663.0766 663.9502
2023 670.7824 669.3399 671.4710 672.4550
2024
```







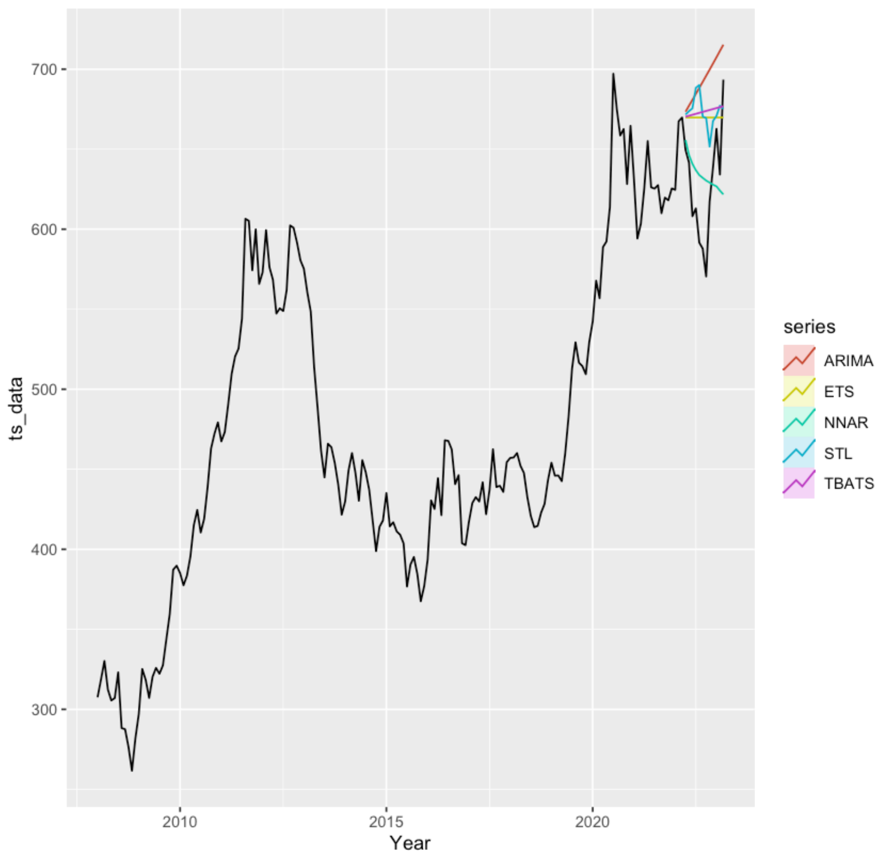
In [14]:
```
print(mapafit)
```

```
MAPA fitted using ets   Original frequency: 12
Aggregation level: 1    Method: ETS(MNN)
Aggregation level: 2    Method: ETS(MNN)
Aggregation level: 3    Method: ETS(MNN)
Aggregation level: 4    Method: ETS(MNN)
Aggregation level: 5    Method: ETS(MNN)
Aggregation level: 6    Method: ETS(MNN)
Aggregation level: 7    Method: ETS(MNN)
Aggregation level: 8    Method: ETS(ANN)
Aggregation level: 9    Method: ETS(ANN)
Aggregation level: 10   Method: ETS(ANN)
Aggregation level: 11   Method: ETS(ANN)
Aggregation level: 12   Method: ETS(ANN)
```

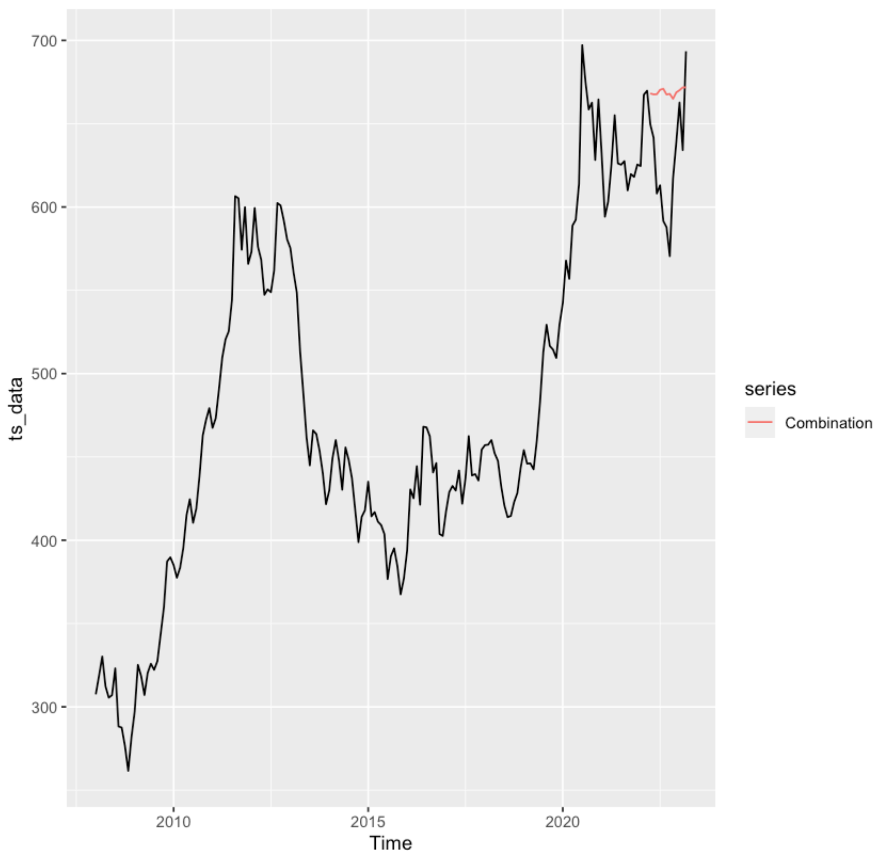
6. Combination

In [15]:
```
h <- nrow(data) - length(train_data)
ETS <- forecast(ets(train_data), h=h)
ARIMA <- forecast(auto.arima(train_data, lambda=0, biasadj=TRUE),
 h=h)
STL <- stlf(train_data, lambda=0, h=h, biasadj=TRUE)
NNAR <- forecast(nnetar(train_data), h=h)
TBATS <- forecast(tbats(train_data, biasadj=TRUE), h=h)
```

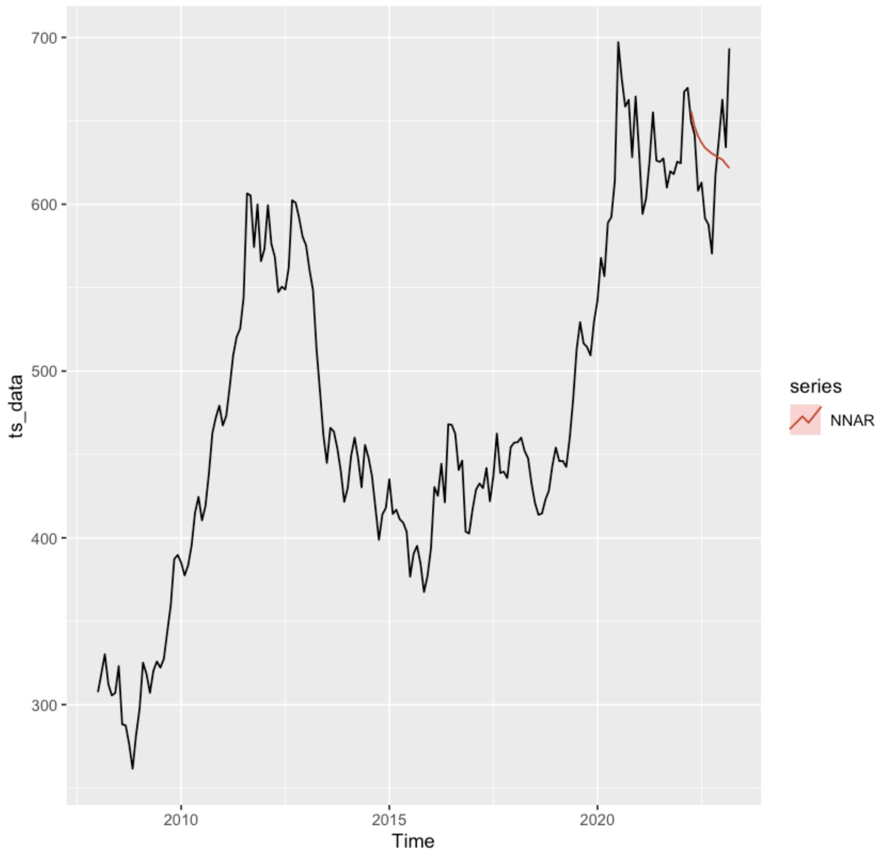
In [19]:
```
Combination <- (ETS[["mean"]] + ARIMA[["mean"]] +
 STL[["mean"]] + NNAR[["mean"]] + TBATS[["mean"]])/5
autoplot(ts_data) +
 autolayer(ETS, series="ETS", PI=FALSE) +
 autolayer(ARIMA, series="ARIMA", PI=FALSE) +
 autolayer(STL, series="STL", PI=FALSE) +
 autolayer(NNAR, series="NNAR", PI=FALSE) +
 autolayer(TBATS, series="TBATS", PI=FALSE) +
 xlab("Year")
```



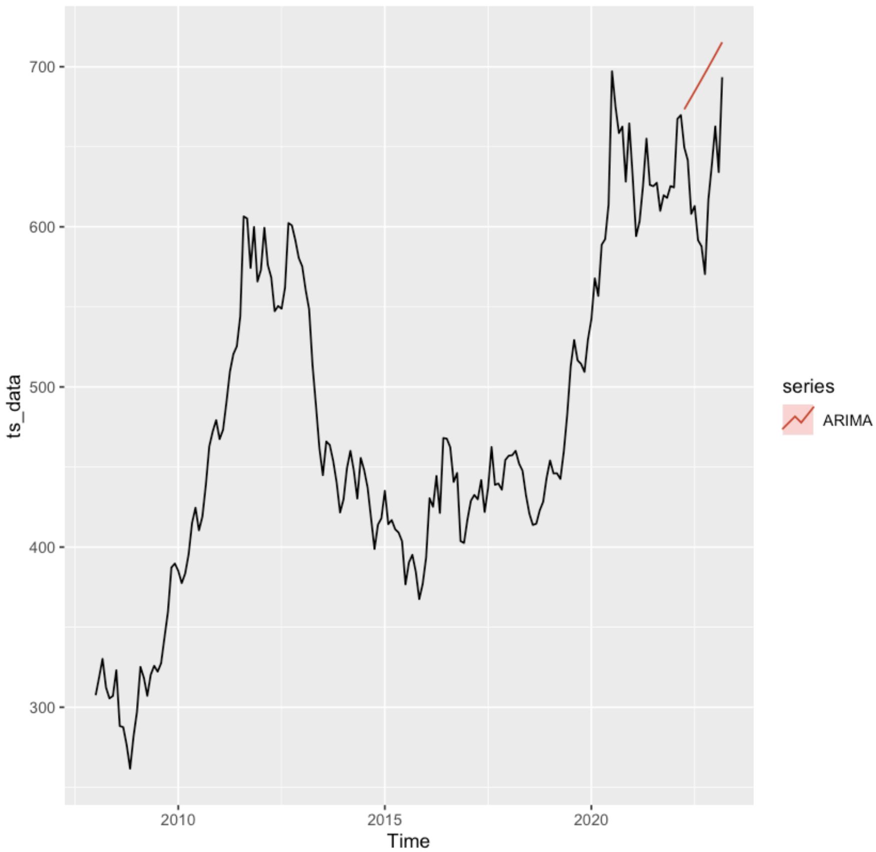
In [20]:
```
autoplot(ts_data) +
  autolayer(Combination, series="Combination")
```



In [21]:
```
autoplot(ts_data) +
 autolayer(NNAR, series="NNAR", PI=FALSE)
```



In [22]:
```
autoplot(ts_data) +
 autolayer(ARIMA, series="ARIMA", PI=FALSE)
```



7. ARIMA

In [23]:
```
ARIMA <- forecast(auto.arima(train_data, lambda=0, biasadj=TRUE), h=h)
# Forecast using the trained model
forecast <- forecast(ARIMA, h = 12)

# Extract the forecasted values
forecast_values <- forecast$mean

# Calculate the Mean Absolute Error (MAE)
mae <- mean(abs(forecast_values - test_data))

# Print the MAE
print(mae)
```

```
[1] 68.418
```

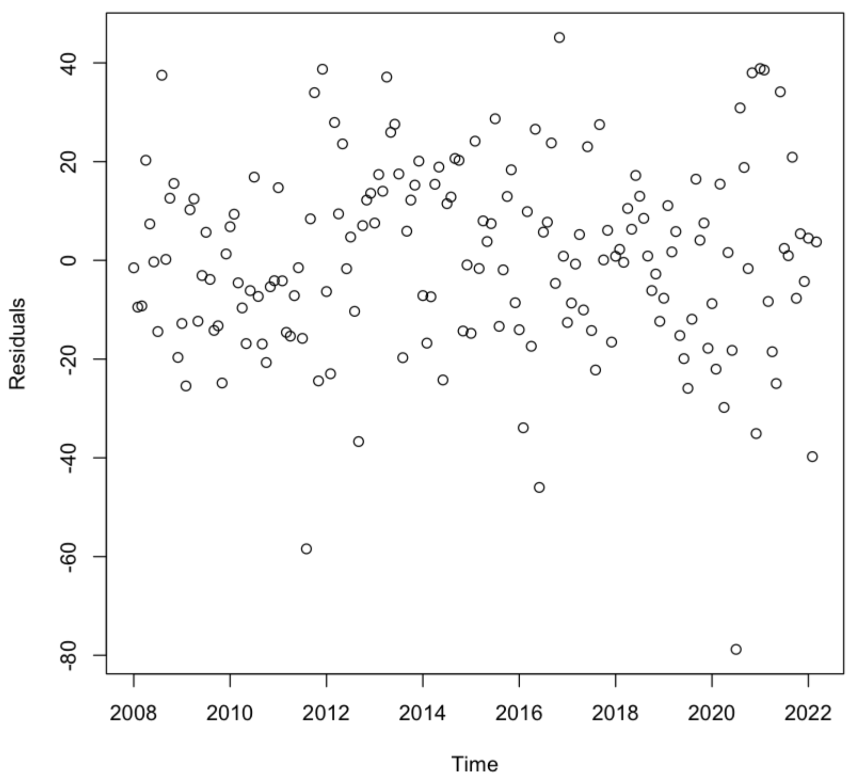
In [24]:
```
mse <- mean((forecast_values - test_data)^2)
mrse <- sqrt(mse)
print(mrse)
```

```
[1] 75.06919
```

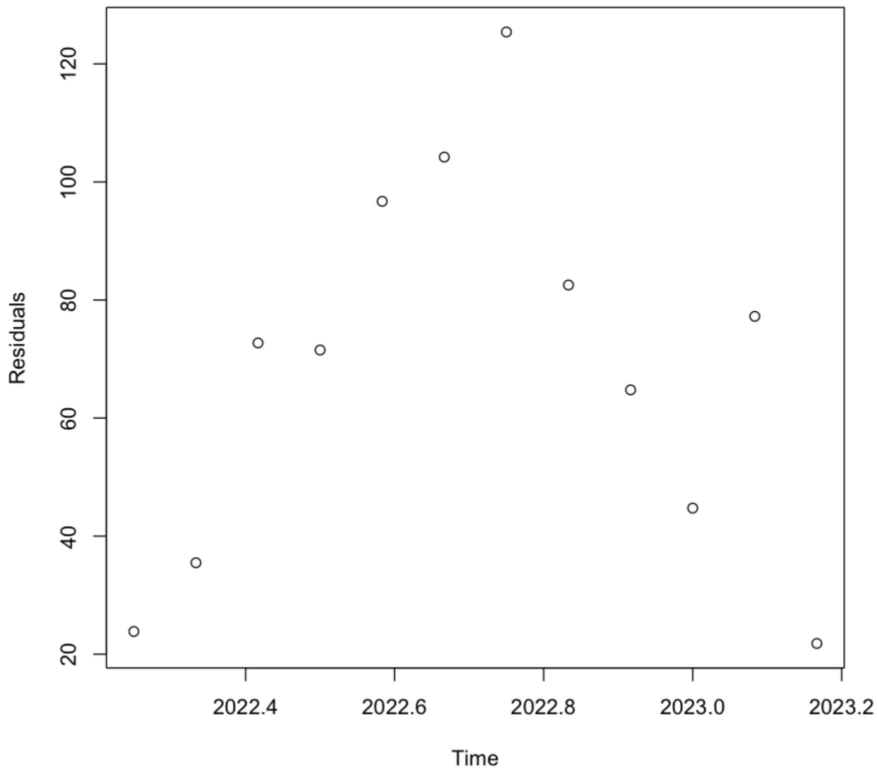
In [25]:
```
mape <- mean(abs((forecast_values - test_data) / test_data)) * 100

# Print the MAPE
print(mape)
```

```
[1] 11.2063
```

In [26]:
```
fitted_values_arima <- fitted(ARIMA)
plot(fitted_values_arima - train_data, type = "p", ylab = "Residuals")
```



In [27]:
```
plot(forecast_values - test_data, type = "p", ylab = "Residuals")
```



In [28]:
```
summary(ARIMA)
```

```
Forecast method: ARIMA(1,1,0)(1,0,0)[12] with drift

Model Information:
Series: train_data
ARIMA(1,1,0)(1,0,0)[12] with drift
Box Cox transformation: lambda= 0

Coefficients:
         ar1    sar1   drift
      0.0599  0.0016  0.0046
s.e.  0.0766  0.0811  0.0032

sigma^2 = 0.001591:  log likelihood = 307.98
AIC=-607.96   AICc=-607.72   BIC=-595.42

Error measures:
                     ME     RMSE     MAE       MPE     MAPE      MASE
Training set -0.4102585 18.97984 14.7419 -0.154984 3.154251 0.2449689
                    ACF1
Training set -0.02497386

Forecasts:
         Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
Apr 2022       673.4007 639.3327 708.1569 622.2634 727.5824
May 2022       677.1454 627.4772 728.2826 603.2153 757.5747
Jun 2022       680.8178 619.2381 744.6603 589.7342 781.9150
Jul 2022       684.5578 612.9195 759.2616 579.1500 803.5333
Aug 2022       688.3231 607.7748 772.7479 570.3447 823.4611
Sep 2022       692.0733 603.4141 785.4282 562.7449 842.1907
Oct 2022       695.8927 599.7044 797.6045 556.1040 860.1393
Nov 2022       699.7117 596.4523 809.3310 550.1634 877.4253
Dec 2022       703.5676 593.5985 820.7416 544.8138 894.2339
Jan 2023       707.4290 591.0507 831.8660 539.9307 910.6260
Feb 2023       711.3881 588.8362 842.8619 535.5090 926.7959
Mar 2023       715.2971 586.7941 853.5931 531.3790 942.6104
```

In [29]:

```
model_a <- forecast(arima(train_data, order = c(1, 1, 0)), h=h)

# Print the model summary
summary(model_a)
```

```
Forecast method: ARIMA(1,1,0)

Model Information:

Call:
arima(x = train_data, order = c(1, 1, 0))

Coefficients:
         ar1
      0.0471
s.e.  0.0765

sigma^2 estimated as 364.2:  log likelihood = -742.51,  aic = 1489.03

Error measures:
                   ME     RMSE      MAE       MPE    MAPE      MASE
ACF1
Training set 2.021343 19.02692 14.71198 0.3600966 3.15007 0.2444716 -
0.01170269

Forecasts:
         Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
Apr 2022       669.9042 645.4486 694.3597 632.5026 707.3057
May 2022       669.9094 634.5010 705.3179 615.7569 724.0620
Jun 2022       669.9097 626.1808 713.6386 603.0321 736.7872
Jul 2022       669.9097 619.2066 720.6128 592.3660 747.4534
Aug 2022       669.9097 613.0819 726.7375 582.9990 756.8203
Sep 2022       669.9097 607.5559 732.2635 574.5478 765.2716
Oct 2022       669.9097 602.4812 737.3381 566.7868 773.0326
Nov 2022       669.9097 597.7627 742.0567 559.5704 780.2490
Dec 2022       669.9097 593.3343 746.4850 552.7978 787.0216
Jan 2023       669.9097 589.1484 750.6709 546.3960 793.4234
Feb 2023       669.9097 585.1690 754.6503 540.3100 799.5093
Mar 2023       669.9097 581.3683 758.4510 534.4973 805.3220
```

In [30]:
```
forecast <- forecast(model_a, h = 12)

# Extract the forecasted values
forecast_values_a <- forecast$mean

mse <- mean((forecast_values_a - test_data)^2)
mrse <- sqrt(mse)
print(mrse)
```

```
[1] 55.26651
```

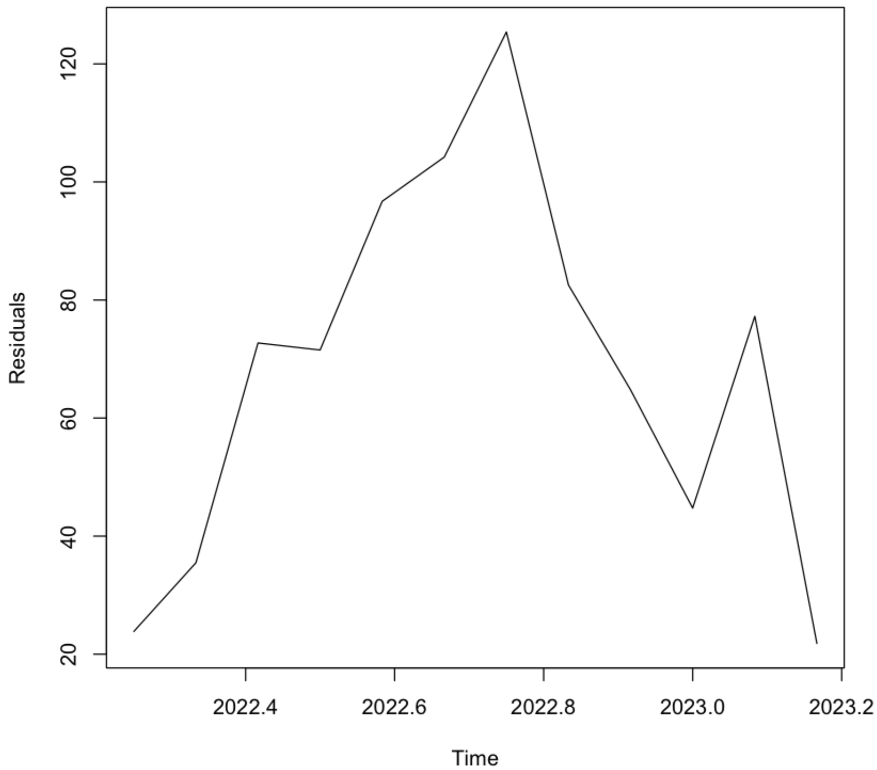
In [31]:
```
mape <- mean(abs((forecast_values_a - test_data) / test_data)) * 100

# Print the MAPE
print(mape)
```

```
[1] 7.928594
```

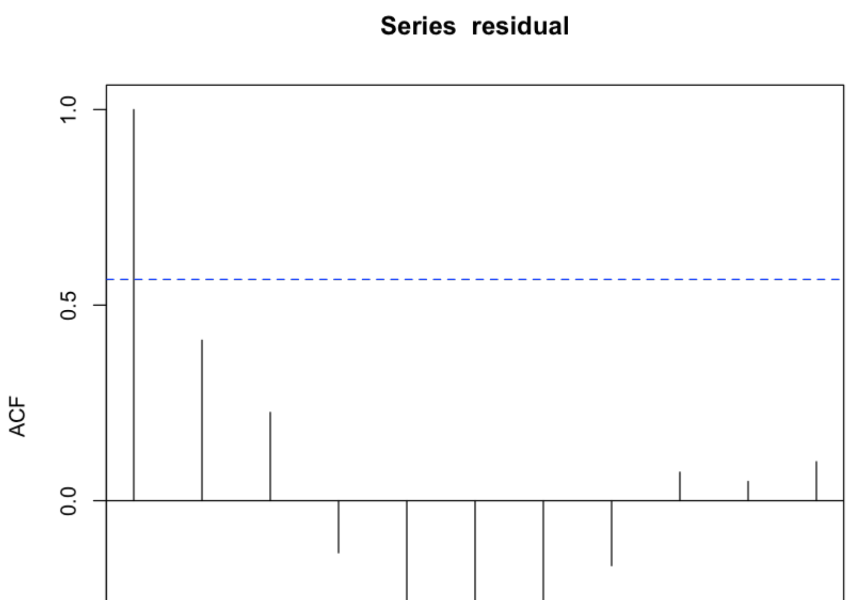
In [36]:
```
fitted_values <- ARIMA$mean
```

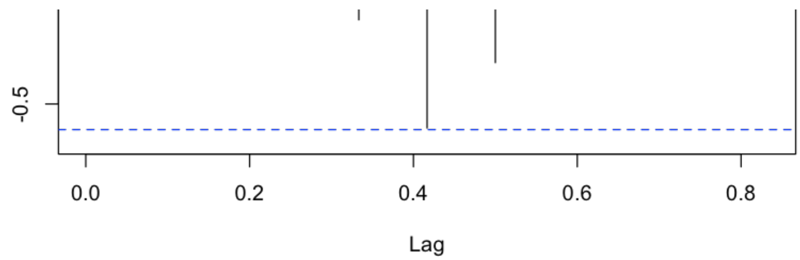
In [37]:
```
residual <- forecast_values - test_data
```

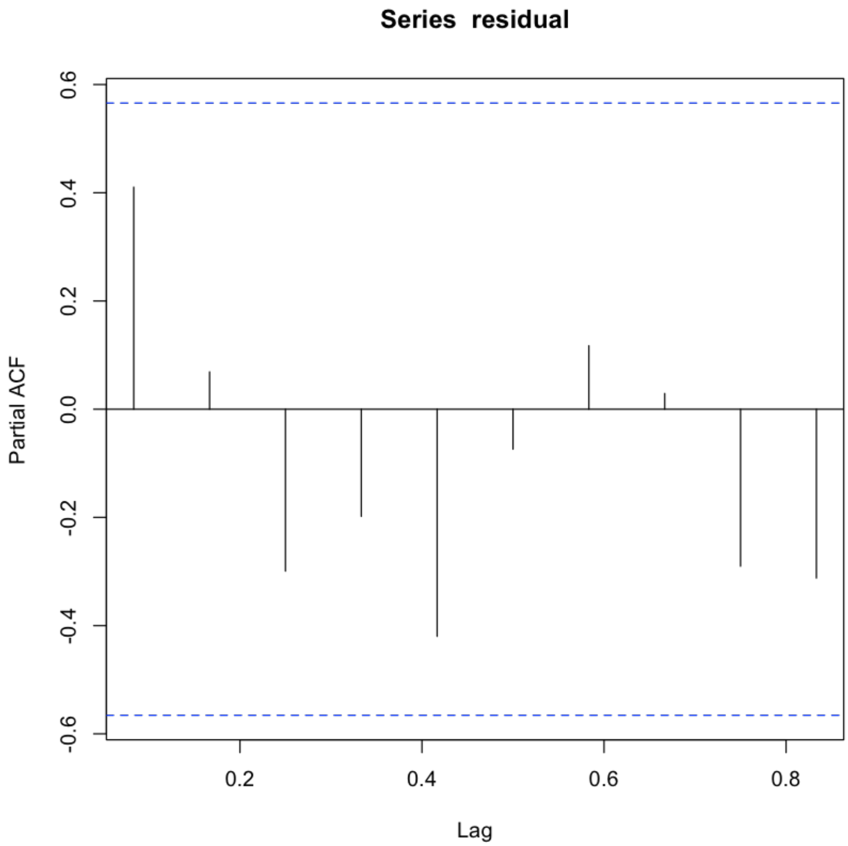
In [38]:
```
plot(residual, type = "l", ylab = "Residuals")
```



In [39]:
```
acf(residual)

# Plot the PACF
pacf(residual)
```







In [40]:
```
fitted_values_arima <- fitted(ARIMA)
```

In [42]:
```
mse <- mean((forecast_values - test_data)^2)
mrse <- sqrt(mse)
print(mrse)
```

```
[1] 75.06919
```

In [43]:
```
mape <- mean(abs((forecast_values - test_data) / test_data)) * 100

# Print the MAPE
print(mape)
```

```
[1] 11.2063
```

8. NNAR

In [44]:
```
library(forecast)

# Train the NNAR model
model <- nnetar(train_data)

# Forecast using the trained model
forecast <- forecast(model, h = 12)

# Extract the forecasted values
forecast_values_NNAR <- forecast$mean

# Calculate the Mean Absolute Error (MAE)
mae <- mean(abs(forecast_values_NNAR - test_data))

# Print the MAE
print(mae)
```

```
[1] 29.69858
```

In [45]:
```
fitted_values_nnar <- fitted(model)
```

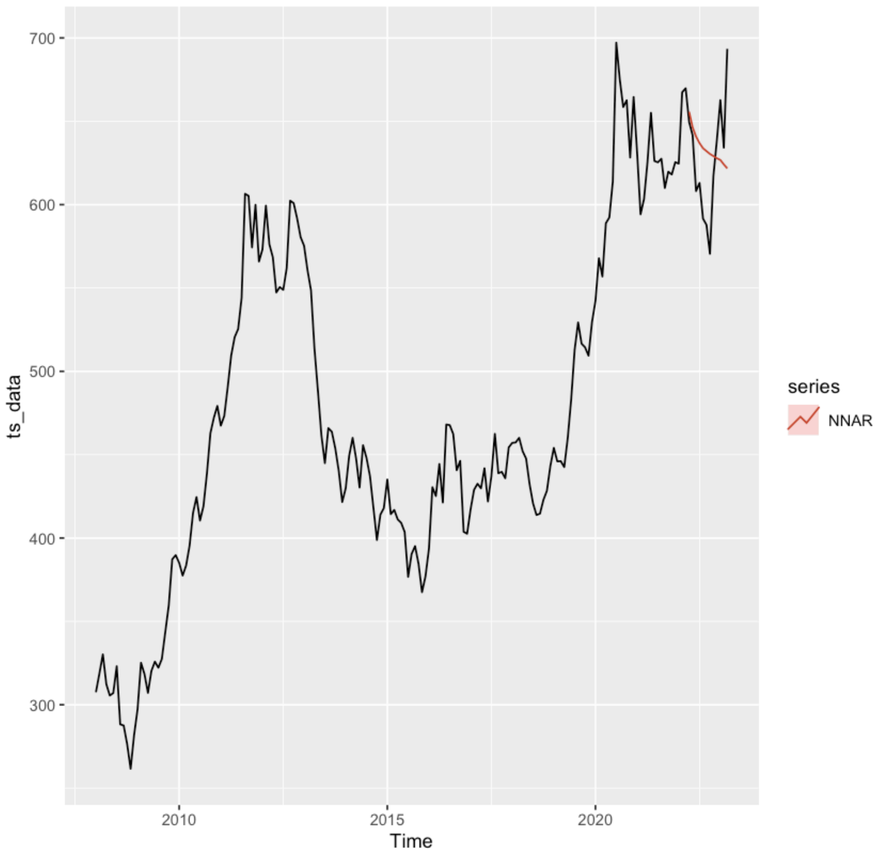
In [47]:
```
current_dir <- getwd()
print(current_dir)
```

```
[1] "/Users/dwx/Documents/spring2023/ECON412 Big Data/Final Project/t
hief"
```

In [48]:
```
mse <- mean((forecast_values_NNAR - test_data)^2)
mrse <- sqrt(mse)
print(mrse)
```

```
[1] 36.40232
```

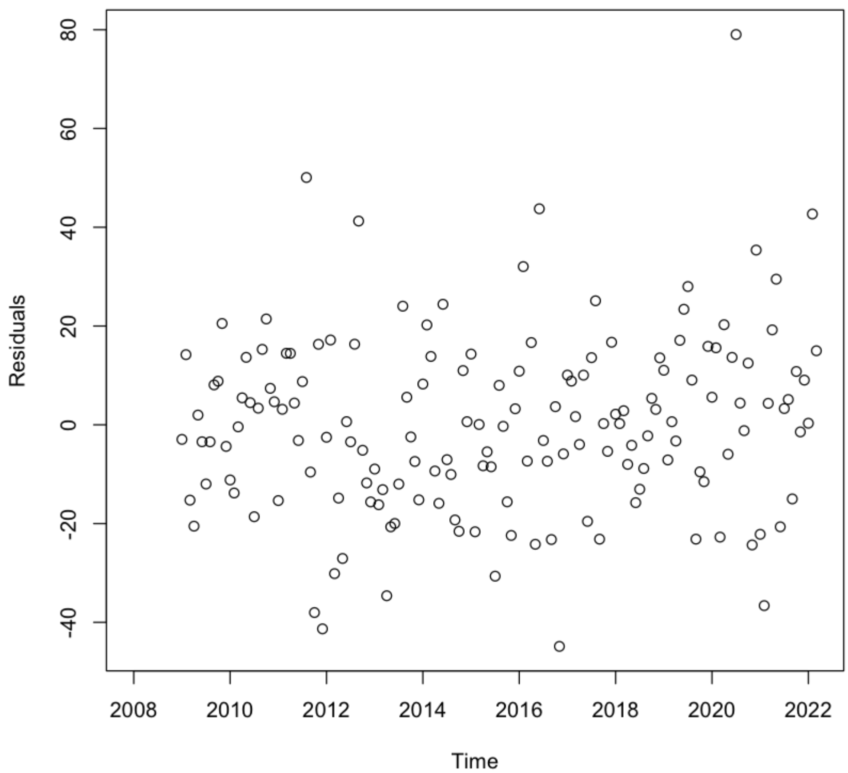
In [49]:
```
autoplot(ts_data) +
 autolayer(NNAR, series="NNAR", PI=FALSE)
```



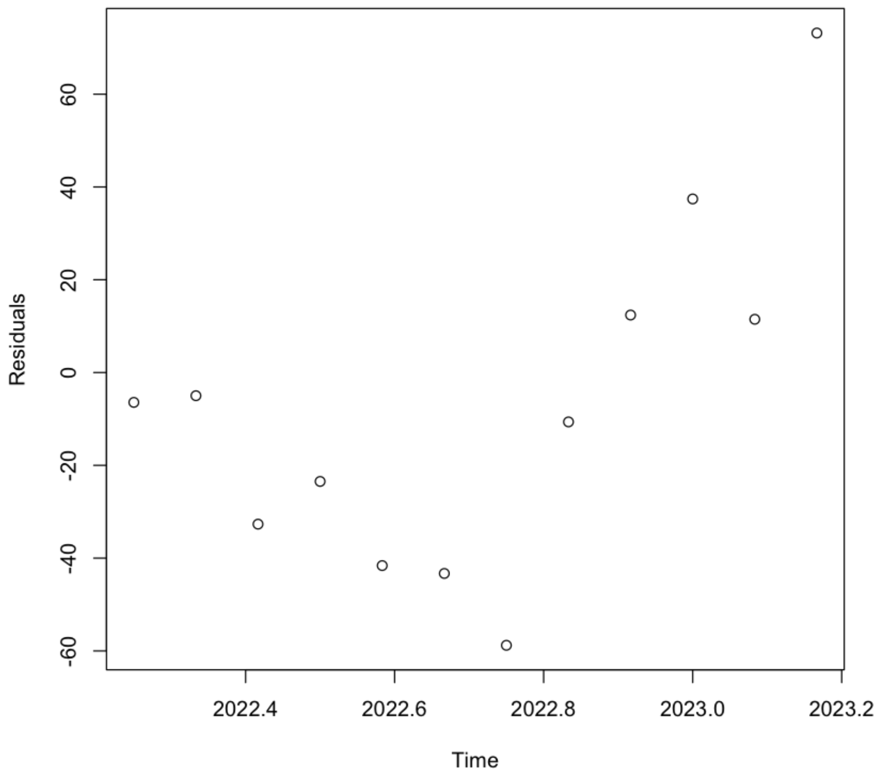
In [50]:
```
mape <- mean(abs((forecast_values_NNAR - test_data) / test_data)) * 10

# Print the MAPE
print(mape)
```

```
[1] 4.77901
```

In [51]:
```
residuals <- train_data-fitted_values_nnar
plot(residuals, type = "p", ylab = "Residuals")
```



In [52]:
```
residuals <- test_data - forecast_values_NNAR

# Create a residual plot
plot(residuals, type = "p", ylab = "Residuals")
```



9. ETS

In [53]:
```
model <- ets(train_data)
forecast <- forecast(model, h=12)
forecast_values_ets <- forecast$mean
forecast_values_ets
```

A Time Series: 2 × 12

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
2022				669.7925	669.7925	669.7925	669.7925	669.7925	669.7925
2023	669.7925	669.7925	669.7925						

In [54]:
```
fitted_values_ets <- fitted(model)
```

In [55]:
```
print(model)
```

```
ETS(M,N,N)

Call:
 ets(y = train_data)

  Smoothing parameters:
    alpha = 0.9999

  Initial states:
    l = 307.0221

  sigma:  0.0404

     AIC     AICc      BIC
1880.220 1880.363 1889.645
```

In [56]:
```
aic_score <- AIC(model)
bic_score <- BIC(model)

# Print AIC and BIC scores
cat("AIC:", aic_score, "\n")
cat("BIC:", bic_score, "\n")
```

```
AIC: 1880.22
BIC: 1889.645
```

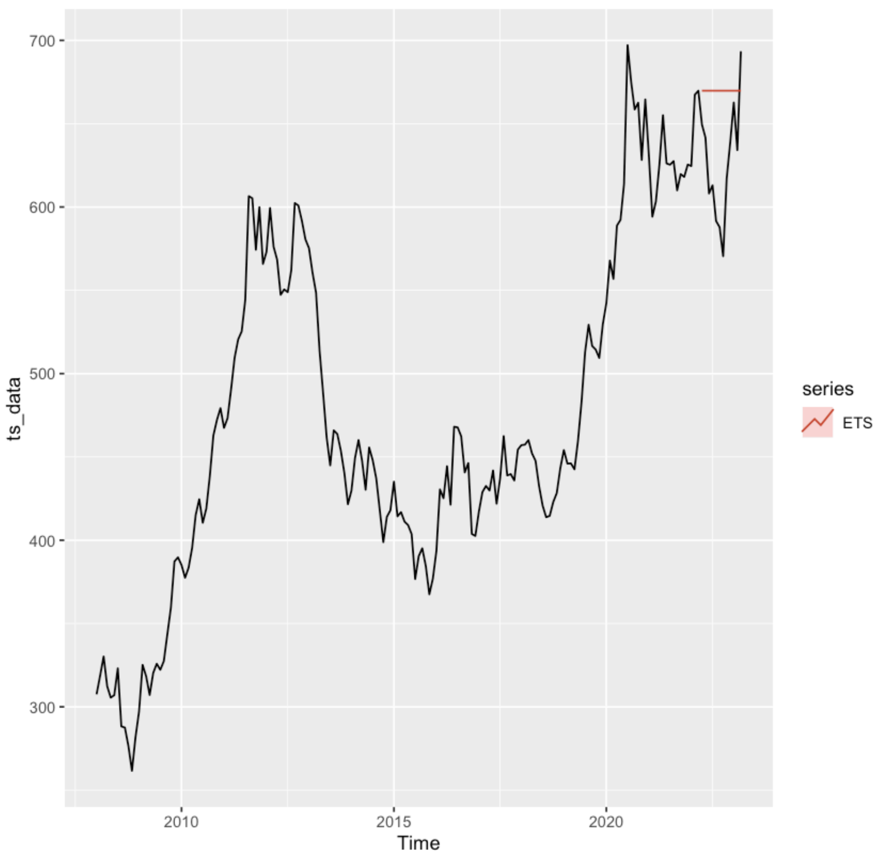
In [57]:
```
mse <- mean((forecast_values - test_data)^2)
rmse <- sqrt(mse)
print(rmse)
```

```
[1] 75.06919
```

In [58]:
```
mape <- mean(abs((forecast_values - test_data) / test_data)) * 100

# Print the MAPE
print(mape)
```

```
[1] 11.2063
```

In [66]:
```
autoplot(ts_data) +
 autolayer(ETS, series="ETS", PI=FALSE)
```



10. LSTM

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from arch import arch_model
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
data = pd.read_excel("G&O.xls")
data.set_index('Date', inplace=True)
```

In [3]:
```
data
```

Out[3]:

Date	Oil	Gold
1994-01-01	55.11754	137.31491
1994-02-01	52.44123	136.52517
1994-03-01	53.45389	133.95854
1994-04-01	61.19349	134.74827
1994-05-01	66.18445	133.46496
...	...	...
2022-12-01	289.90958	617.17670
2023-01-01	285.53345	638.79566
2023-02-01	278.04702	662.68509
2023-03-01	273.70705	634.15597
2023-04-01	277.68535	693.48470

352 rows × 2 columns

In [4]:
```
train_data = data["Gold"][:-12]
test_data = data["Gold"][-12:]
```

In [5]:
```
test_data
```

Out[5]:
```
Date
2022-05-01    649.55577
2022-06-01    641.65844
2022-07-01    608.09477
2022-08-01    613.03060
2022-09-01    591.60908
2022-10-01    587.85785
2022-11-01    570.48371
2022-12-01    617.17670
2023-01-01    638.79566
2023-02-01    662.68509
2023-03-01    634.15597
2023-04-01    693.48470
Name: Gold, dtype: float64
```

In [6]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
train_data_scaled = scaler.fit_transform(train_data.values.reshape(-1, 1
test_data_scaled = scaler.transform(test_data.values.reshape(-1, 1))
```

In [7]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = Sequential()
model.add(LSTM(units=50, activation='relu', input_shape=(1, 1)))
model.add(Dense(units=1))
```

In [8]:
```
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(train_data_scaled.reshape(-1, 1, 1), train_data_scaled, epochs
```

```
Epoch 1/40
34/34 [==============================] - 2s 2ms/step - loss: 0.1716
Epoch 2/40
34/34 [==============================] - 0s 2ms/step - loss: 0.1110
Epoch 3/40
34/34 [==============================] - 0s 2ms/step - loss: 0.0679
Epoch 4/40
34/34 [==============================] - 0s 2ms/step - loss: 0.0443
Epoch 5/40
34/34 [==============================] - 0s 2ms/step - loss: 0.0347
Epoch 6/40
34/34 [==============================] - 0s 2ms/step - loss: 0.0284
Epoch 7/40
34/34 [==============================] - 0s 3ms/step - loss: 0.0224
Epoch 8/40
34/34 [==============================] - 0s 3ms/step - loss: 0.0170
Epoch 9/40
34/34 [==============================] - 0s 3ms/step - loss: 0.0120
Epoch 10/40
34/34 [==============================] - 0s 3ms/step - loss: 0.0080
Epoch 11/40
34/34 [==============================] - 0s 3ms/step - loss: 0.0050
Epoch 12/40
34/34 [==============================] - 0s 3ms/step - loss: 0.0029
Epoch 13/40
34/34 [==============================] - 0s 3ms/step - loss: 0.0016
Epoch 14/40
34/34 [==============================] - 0s 3ms/step - loss: 8.7218e-04
Epoch 15/40
34/34 [==============================] - 0s 3ms/step - loss: 5.1604e-04
Epoch 16/40
34/34 [==============================] - 0s 3ms/step - loss: 3.5598e-04
Epoch 17/40
34/34 [==============================] - 0s 3ms/step - loss: 2.9312e-04
Epoch 18/40
34/34 [==============================] - 0s 3ms/step - loss: 2.6440e-04
Epoch 19/40
34/34 [==============================] - 0s 2ms/step - loss: 2.5467e-04
Epoch 20/40
34/34 [==============================] - 0s 2ms/step - loss: 2.4419e-04
Epoch 21/40
34/34 [==============================] - 0s 2ms/step - loss: 2.4051e-04
Epoch 22/40
34/34 [==============================] - 0s 2ms/step - loss: 2.3575e-04
Epoch 23/40
34/34 [==============================] - 0s 2ms/step - loss: 2.3809e-04
Epoch 24/40
34/34 [==============================] - 0s 2ms/step - loss: 2.2725e-04
Epoch 25/40
34/34 [==============================] - 0s 2ms/step - loss: 2.2605e-04
Epoch 26/40
34/34 [==============================] - 0s 2ms/step - loss: 2.2354e-04
Epoch 27/40
34/34 [==============================] - 0s 2ms/step - loss: 2.1807e-04
Epoch 28/40
34/34 [==============================] - 0s 2ms/step - loss: 2.1431e-04
Epoch 29/40
```

```
34/34 [==============================] - 0s 2ms/step - loss: 2.1536e-04
Epoch 30/40
34/34 [==============================] - 0s 3ms/step - loss: 2.1189e-04
Epoch 31/40
34/34 [==============================] - 0s 3ms/step - loss: 2.0698e-04
Epoch 32/40
34/34 [==============================] - 0s 3ms/step - loss: 2.0512e-04
Epoch 33/40
34/34 [==============================] - 0s 2ms/step - loss: 1.9947e-04
Epoch 34/40
34/34 [==============================] - 0s 2ms/step - loss: 1.9775e-04
Epoch 35/40
34/34 [==============================] - 0s 3ms/step - loss: 1.9441e-04
Epoch 36/40
34/34 [==============================] - 0s 3ms/step - loss: 1.9238e-04
Epoch 37/40
34/34 [==============================] - 0s 3ms/step - loss: 1.8688e-04
Epoch 38/40
34/34 [==============================] - 0s 2ms/step - loss: 1.8172e-04
Epoch 39/40
34/34 [==============================] - 0s 3ms/step - loss: 1.8507e-04
Epoch 40/40
34/34 [==============================] - 0s 3ms/step - loss: 1.7378e-04
```

Out[8]:

```
<keras.callbacks.History at 0x2059b4e90a0>
```

In [16]:

```python
predictions_scaled = model.predict(train_data_scaled[-12:].reshape(-1, 1
predictions = scaler.inverse_transform(predictions_scaled)
```

```
1/1 [==============================] - 0s 194ms/step
```

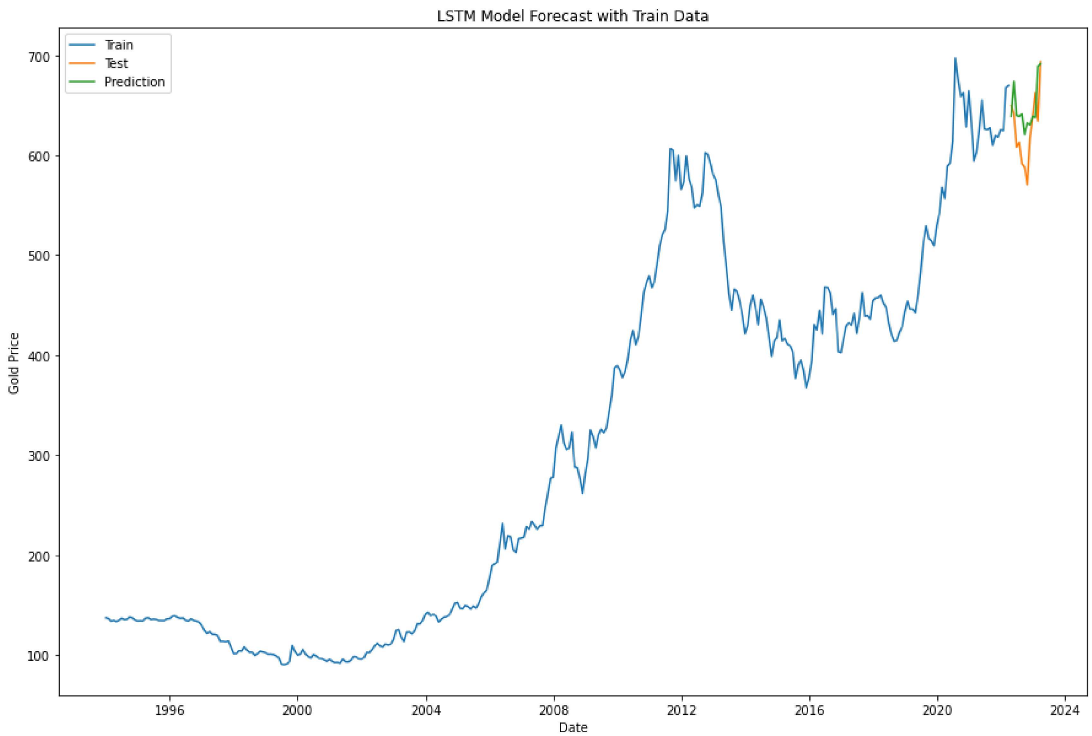
In [15]:

```python
predictions = np.squeeze(predictions)
predictions = pd.Series(predictions, index=test_data.index)
predictions
```

Out[15]:

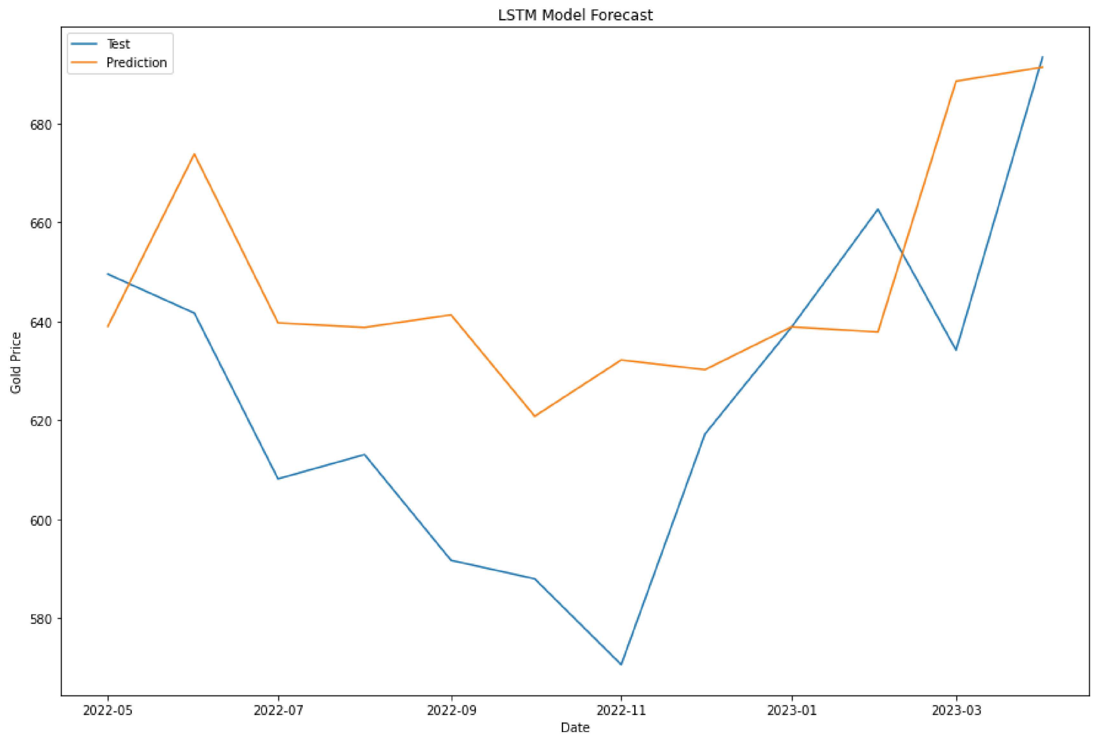
```
Date
2022-05-01    638.990967
2022-06-01    673.876282
2022-07-01    639.685547
2022-08-01    638.759399
2022-09-01    641.307068
2022-10-01    620.780640
2022-11-01    632.172791
2022-12-01    630.212219
2023-01-01    638.875122
2023-02-01    637.833801
2023-03-01    688.624817
2023-04-01    691.467590
dtype: float32
```

In [11]:
```
plt.figure(figsize=(15, 10))
plt.plot(train_data, label='Train')
plt.plot(test_data, label='Test')
plt.plot(predictions, label='Prediction')
plt.legend()
plt.xlabel('Date')
plt.ylabel('Gold Price')
plt.title('LSTM Model Forecast with Train Data')
plt.show()
```



In [12]:

```
plt.figure(figsize=(15, 10))
plt.plot(test_data, label='Test')
plt.plot(predictions, label='Prediction')
plt.legend()
plt.xlabel('Date')
plt.ylabel('Gold Price')
plt.title('LSTM Model Forecast')
plt.show()
```



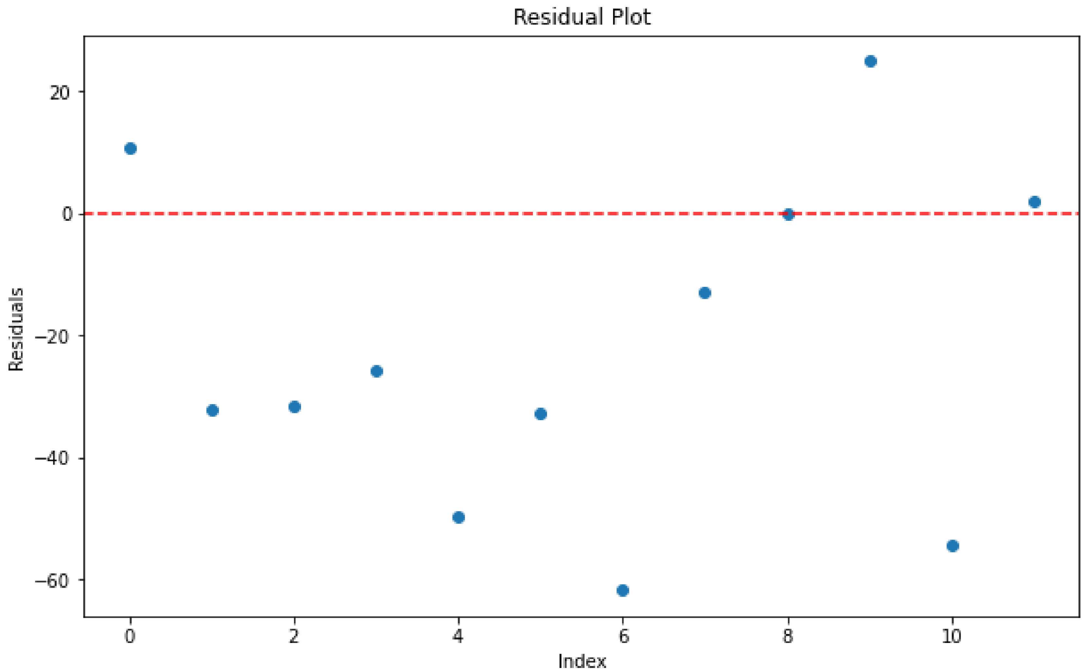
In [13]:

```
from sklearn.metrics import mean_squared_error
import numpy as np

rmse = np.sqrt(mean_squared_error(test_data, predictions))
mape = np.mean(np.abs((test_data - predictions) / test_data)) * 100
print("RMSE:", rmse)
print("MAPE:", mape)
```

```
RMSE: 34.06774754057272
MAPE: 4.634053137000738
```

In [14]:

```
residuals = test_data - predictions
plt.figure(figsize=(10, 6))
plt.scatter(range(len(residuals)), residuals)
plt.axhline(0, color='r', linestyle='--')
plt.xlabel('Index')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()
```



11. Reference

1. U.S. Bureau of Labor Statistics, Export Price Index (End Use): Nonmonetary Gold [IQ12260], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/IQ12260, June 9, 2023.
2. U.S. Energy Information Administration, Crude Oil Prices: West Texas Intermediate (WTI) - Cushing, Oklahoma [DCOILWTICO], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/DCOILWTICO, June 9, 2023.

Coding Source:

[1] pandas (Version 1.3.5). McKinney, W. Retrieved from https://pandas.pydata.org
[2] Seabold, S. & Perktold, J. Retrieved from http://conference.scipy.org/proceedings/scipy2010/seabold.html
[3] NumPy (Version 1.22.0). Oliphant, T. E. Retrieved from https://numpy.org
[4] Matplotlib (Version 3.5.1). Hunter, J. D.. Retrieved from https://matplotlib.org

在此处键入文本