Econ 144 Project 3

Theo Teske

2023-03-15

Introduction

We will forecast a time series of returns on copper based on the global price of copper from January 1990 to January 2023, as reported by the International Monetary fund; the data was retrieved from Kaggle. The frequency of this data is monthly, and the units are U.S. dollars per metric ton, not seasonally adjusted. The global price of copper is calculated based on benchmark prices which are representative of the global market. They are determined by the largest exporter of a given commodity. Prices are period averages in nominal U.S. dollars.

Utilizing this dataset and an 80/20 training-testing split, we will compare the accuracy of six different forecating methods at predicting the test data: ARIMA, exponential smoothing, Holt-Winters', Prophet, a linear combination of these four, and a Neural Network Autoregressive (NNAR) method.

Results

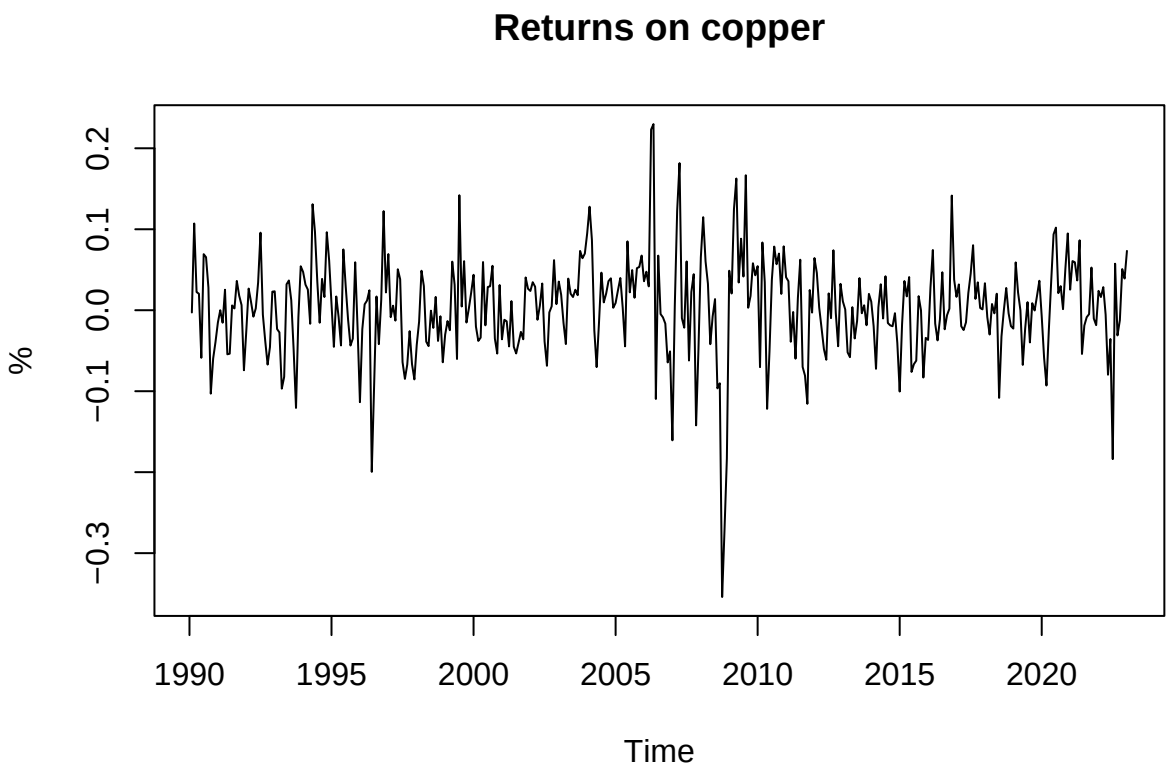
First, we read in our price data p_t, then construct the time series of returns on copper as follows:

$$r_t = \log\left(\frac{p_{t+1}}{p_t}\right).$$

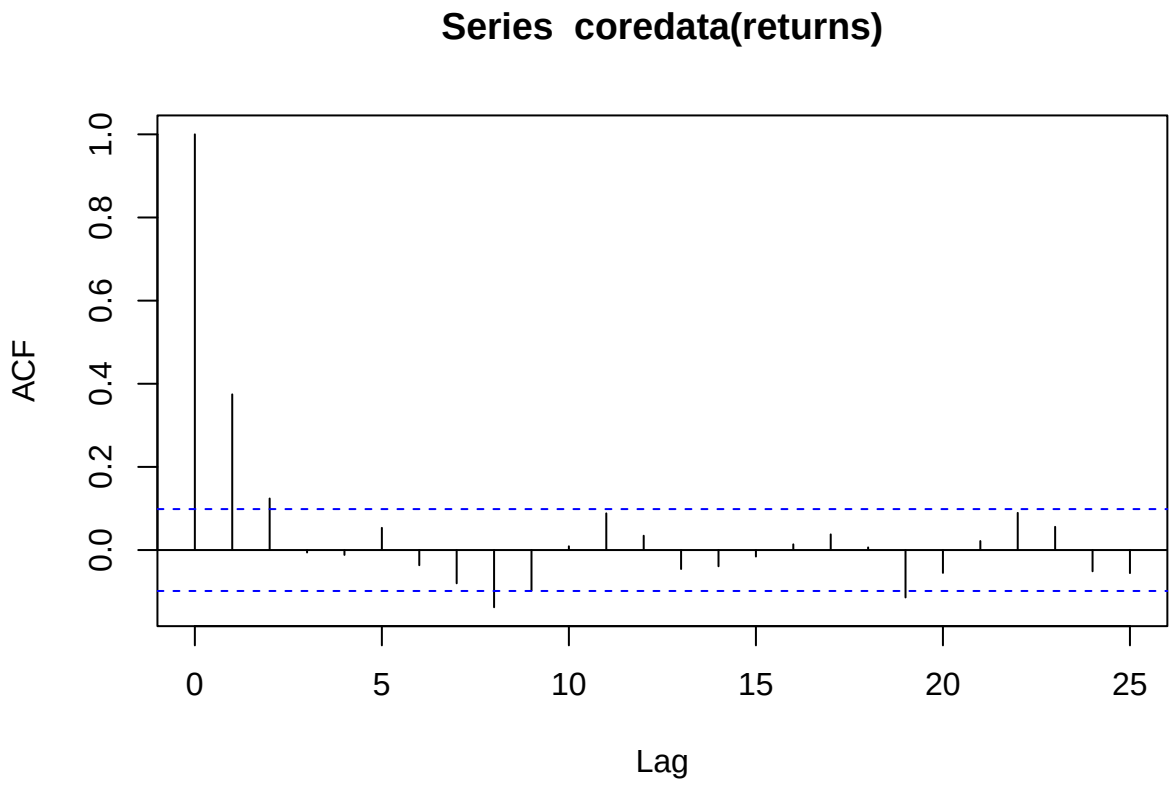
```
copper <- read.table("C:/Users/Theo/Downloads/PCOPPUSDM.csv", header=TRUE,
   sep=",")
copper_ts <- ts(copper$PCOPPUSDM, start=c(1990,1), freq=12)
returns <- diff(log(copper_ts))
```

We plot the time series of returns, and check its ACF and PACF for stationarity.

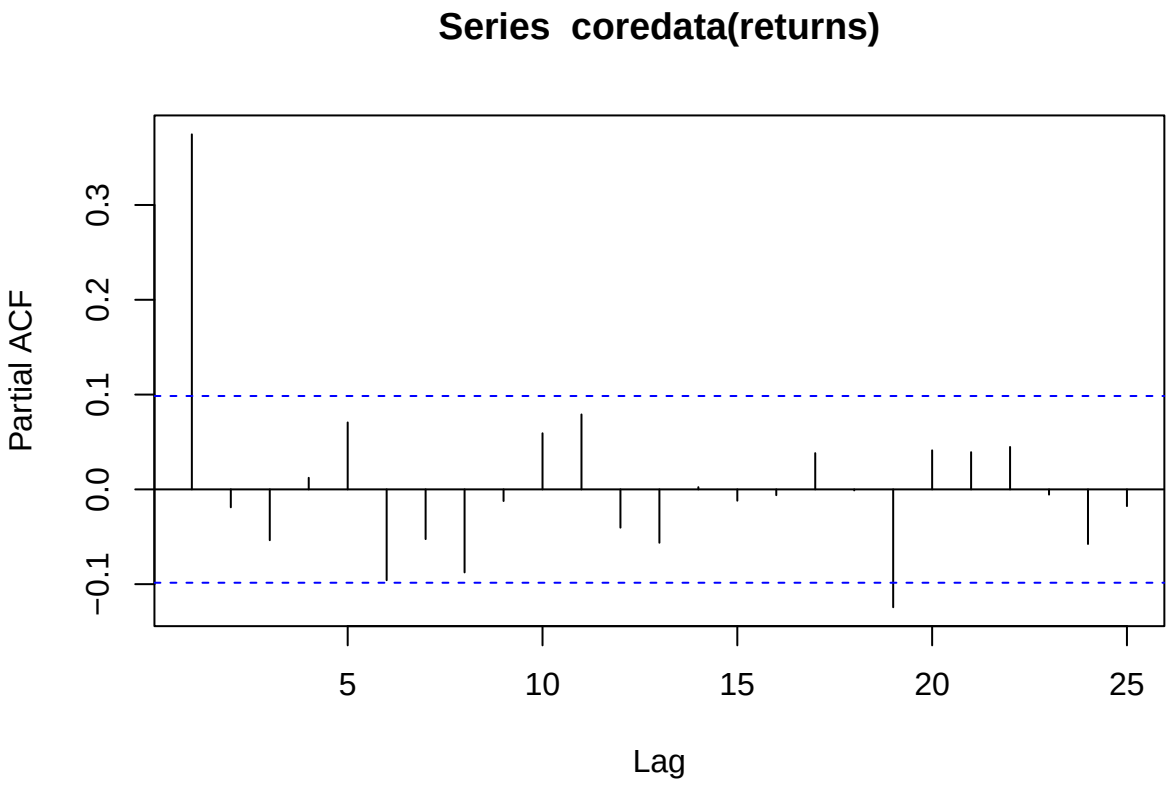
```
ts.plot(returns, main="Returns on copper", ylab="%")
```




```
acf(coredata(returns))
```




```
pacf(coredata(returns))
```



We observe only two significant spikes in the ACF at lags of 1 and 2 months, and one barely significant spike in the PACF at a lag of 19 months. We conclude that the data are sufficiently stationary.

Now, we split our data approximately into 80% training data (from January 1990 to December 2016) and 20% testing data (from January 2017 to January 2023). We then fit an ARIMA model to the training data using `auto.arima()`, which uses a variation of the Hyndman-Khandakar algorithm, combining unit root tests, minimization of the AICc and MLE to obtain an ARIMA model.

```
training <- window(returns, start=c(1990,1), end=c(2016,12))
testing <- window(returns, start=c(2017,1), end=c(2023,1))

ret_ar <- auto.arima(training)
summary(ret_ar)
```

```
## Series: training
## ARIMA(2,0,0) with zero mean
##
## Coefficients:
##          ar1      ar2
##       0.4024  -0.0401
## s.e.  0.0555   0.0562
##
## sigma^2 = 0.003419:  log likelihood = 459.67
## AIC=-913.34   AICc=-913.26   BIC=-902.01
##
## Training set error measures:
```

```
##                       ME       RMSE        MAE MPE MAPE      MASE         ACF1
## Training set 0.001747608 0.05829026 0.04252609 Inf  Inf 0.6836433 -0.001344549
```

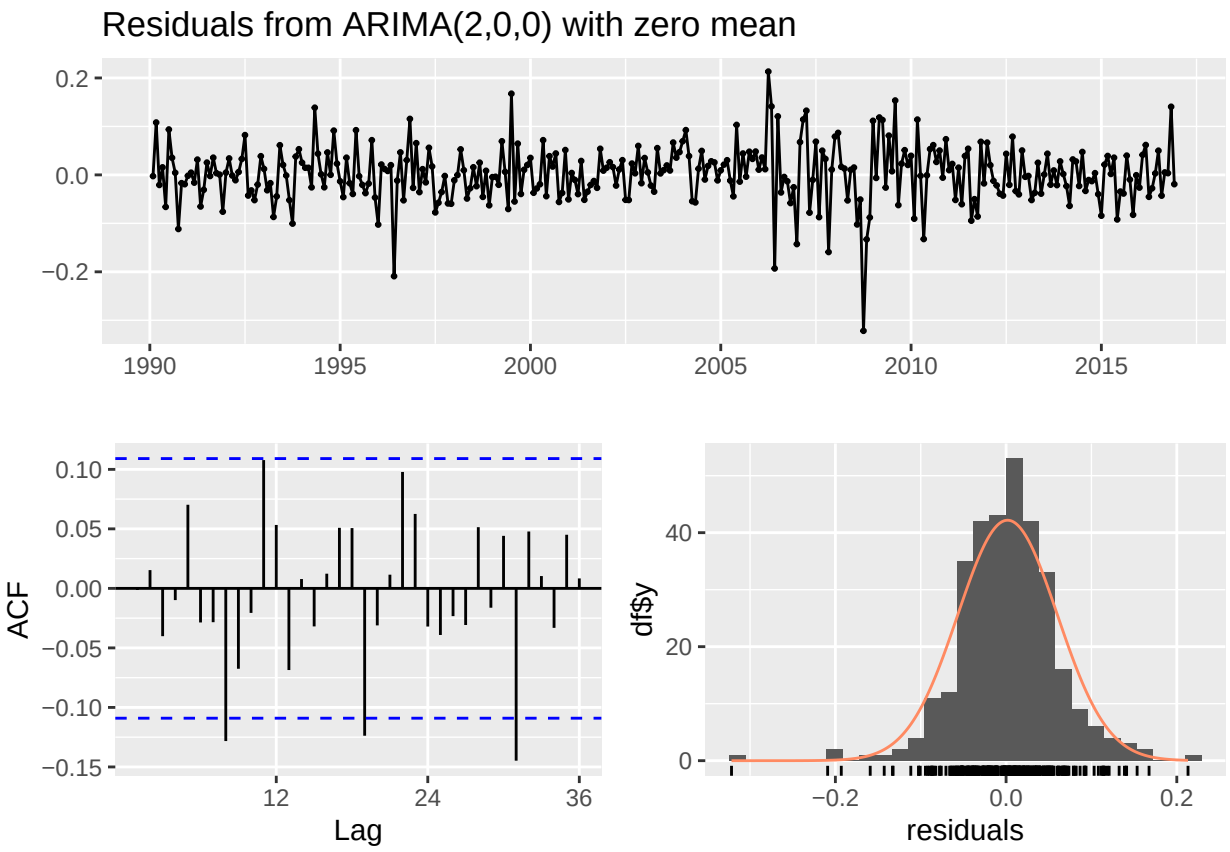
We find that `auto.arima()` has selected an AR(2) model, where we forecast r_t using a linear combination of lagged values of r_t. This can be written as:

$$r_t = c + \phi_1 r_{t-1} + \phi_2 r_{t-2} + \varepsilon_t,$$

where ε_t is white noise, $-1 < \phi_2 < 1, \phi_1 + \phi_2 < 1, \phi_2 - \phi_1 < 1$.

Now, we check to make sure that the residuals from the fitted ARIMA model resemble white noise.

```
checkresiduals(ret_ar)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(2,0,0) with zero mean
## Q* = 29.349, df = 22, p-value = 0.1352
##
## Model df: 2.   Total lags used: 24
```

At the $\alpha = 0.05$ significance level, we have a large enough p-value that we fail to reject the null hypothesis of the Ljung-Box Test, which indicates that the residuals are not autocorrelated. This is supported by the ACF,

which shows no large significant spikes; the residuals also appear to be approximately normally distributed. Thus, the residuals resemble white noise.

Next, we fit exponential smoothing models. First, we use the `ets()` function to select an appropriate model by minimizing the AICc.

```
ret_ets <- ets(training)
summary(ret_ets)
```

```
## ETS(A,N,N)
##
## Call:
##  ets(y = training)
##
##   Smoothing parameters:
##     alpha = 0.226
##
##   Initial states:
##     l = 0.0215
##
##   sigma:  0.0639
##
##       AIC      AICc       BIC
##  93.48450  93.55973 104.81745
##
## Training set error measures:
##                         ME      RMSE        MAE MPE MAPE      MASE      ACF1
## Training set 0.0001386197 0.0637108 0.04618985 Inf  Inf 0.7425414 0.2526455
```

The `ets()` function selected an ETS(A,N,N) model, which is a simple exponential smoothing model with additive errors. In this model, the forecast at time $T+1$ is equal to a weighted average of the most recent observation r_T and the previous forecast $\hat{r}_{T|T-1}$:

$$\hat{r}_{T+1|T} = \alpha r_T + (1-\alpha)\hat{r}_{T|T-1},$$

where $0 \leq \alpha \leq 1$ is the smoothing parameter. The equations of the model can be written as:

$$\begin{aligned} r_t &= \ell_{t-1} + \varepsilon_t \\ \ell_t &= \ell_{t-1} + \alpha\varepsilon_t, \end{aligned}$$

where ℓ_t is the level (or the smoothed value) of the series at time t, and where $\varepsilon_t = r_t - \ell_{t-1} = r_t - \hat{r}_{t|t-1}$ is the residual at time t, for $t = 1, 2, ..., T$.

We continue by applying Holt-Winters' seasonal method, choosing an additive seasonality as the seasonal variations appear to be roughly constant through the series.

```
#holt-winters
ret_hw <- hw(training,seasonal="additive")
```

The additive Holt-Winters' method for forecasting is:

$$
\begin{aligned}
\hat{r}_{t+h|t} &= \ell_t + hb_t + s_{t+h-m(k+1)} \\
\ell_t &= \alpha(r_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\
b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1} \\
s_t &= \gamma(r_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m},
\end{aligned}
$$

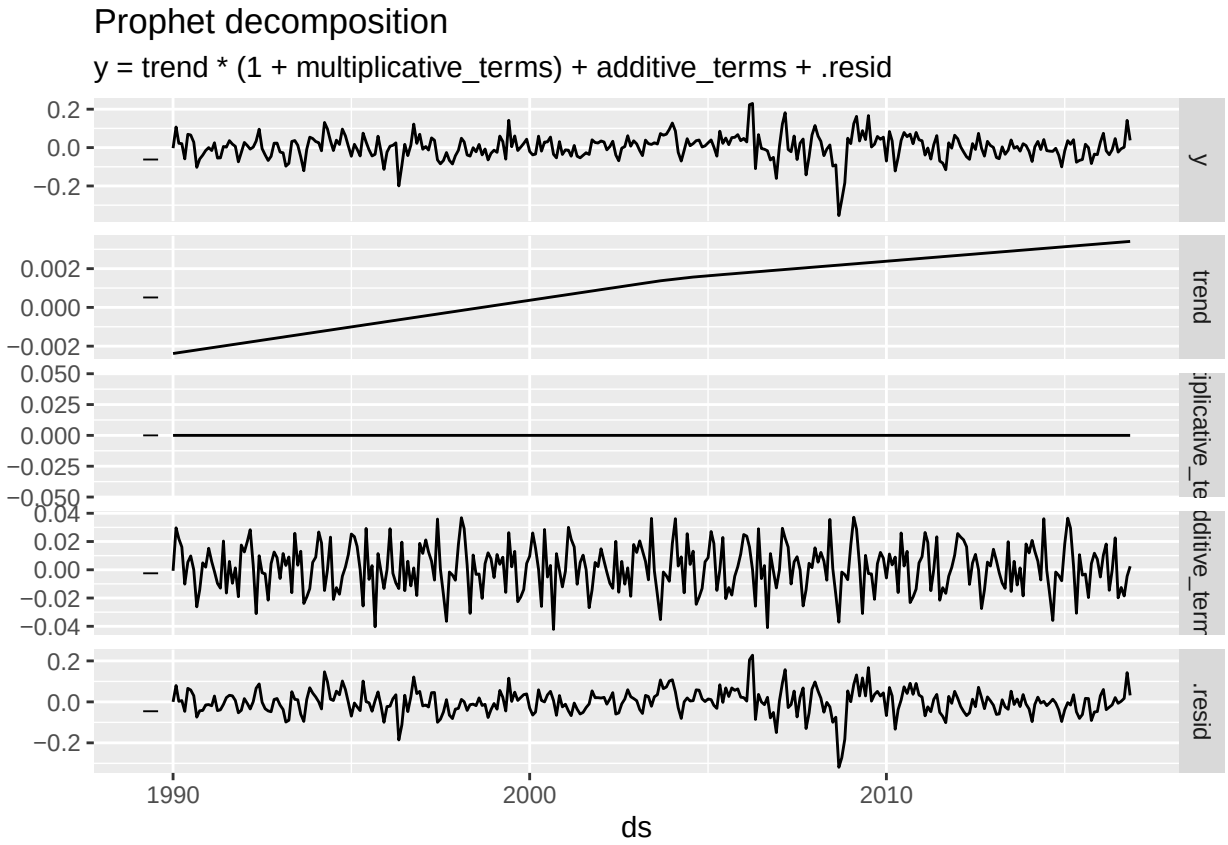
where k is the integer part of $\frac{h-1}{m}$. The level equation shows a weighted average of the seasonally adjusted observation $(r_t - s_{t-m})$ and the non-seasonal forecast $(\ell_{t-1} + b_{t-1})$ for time t. The trend equation shows that a weighted average of the estimated trend at time t based on $(\ell_t - \ell_{t-1})$ and b_{t-1}, the previous estimate of the trend. The seasonal equation shows a weighted average of the current seasonal index, $(r_t - \ell_{t-1} - b_{t-1})$, and the seasonal index of the same season last year s_{t-m}. The model we fitted has $\alpha = 0.0528$, $\beta^* = 0.0001$, and $\gamma = 0.0187$.

The next model we will fit to the time series is a Prophet model, as introduced by Facebook. This is a nonlinear regression model of the form

$$
r_t = g(t) + s(t) + h(t) + \varepsilon_t,
$$

where $g(t)$ is a piecewise-linear trend term, $s(t)$ captures seasonality, $h(t)$ accounts for holiday effects, and ε_t is a white noise error term.

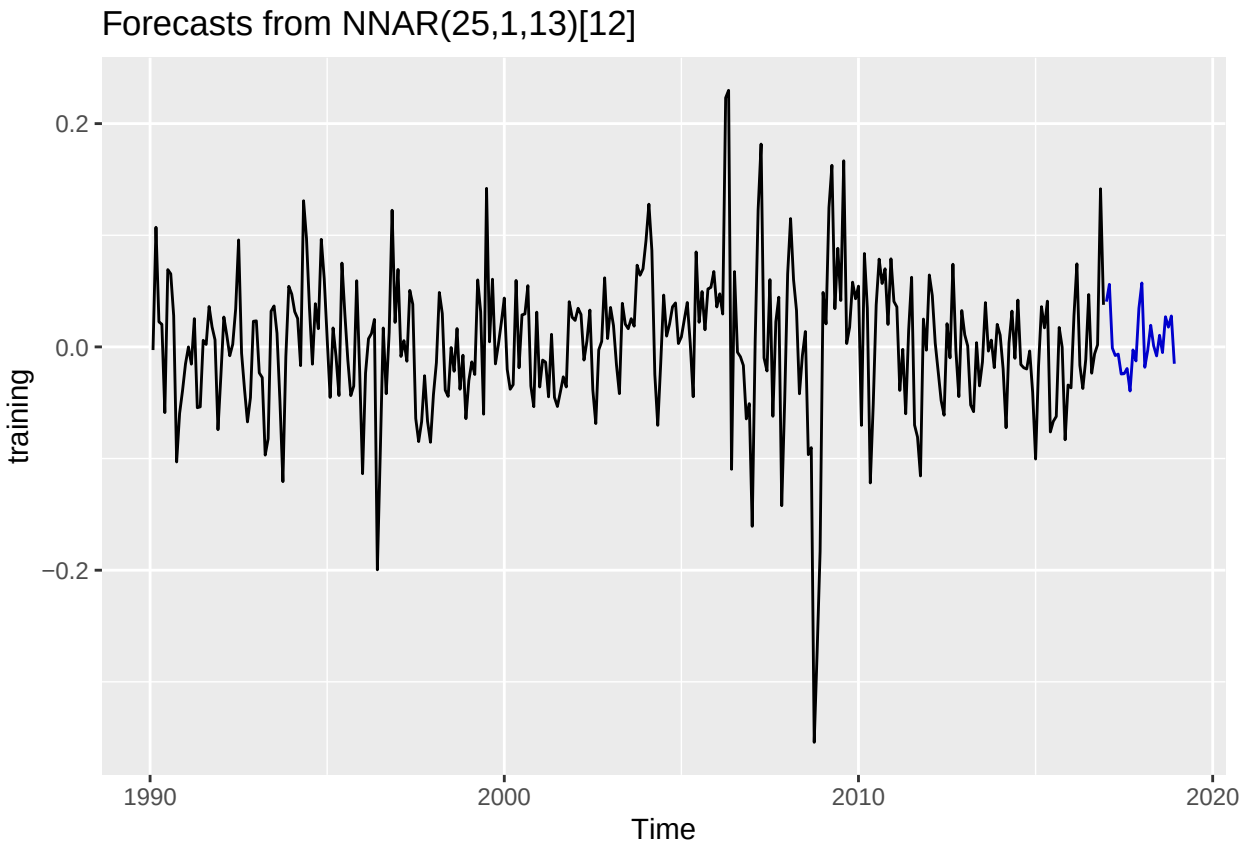
```
ds <- seq(as.Date("1990/1/1"), by = "month", length.out = 323)
te_dates <- seq(as.Date("2017/1/1"), by = "month", length.out = 73)
y <- coredata(training)

df <- data.frame(ds, y)
ret_tbl <- as_tsibble(df, index = ds)

ret_pro <- ret_tbl %>%
  model(
    prophet = fable.prophet::prophet(y)
  )



autoplot(components(ret_pro[[1]][[1]]))
```



Above, the components of the Prophet model we fitted are visible. We observe a clear linear trend, additive terms, and residuals, but no multiplicative term, agreeing with our earlier observation that the series' seasonality is additive.

Lastly, we fit an NNAR (Neural Network Autoregressive) model to the time series of returns using the `nnetar()` function, which uses lagged values of the time series as inputs to a neural network. We consider a feed-forward network with one hidden layer, so an NNAR$(p, P, k)_m$ model has seasonality m, P observed values from the same season as the input (so the inputs $r_{t-m}, r_{t-2m}, ..., r_{t-Pm}$), and p lagged inputs (so the inputs $r_{t-1}, r_{t-2}, ..., r_{t-p}$), as well as k neurons in the hidden layer.

```
ret_nnar <- nnetar(training)
autoplot(forecast::forecast(ret_nnar))
```



We see that the `nnetar()` function has selected a seasonal period $m = 12$, along with $p = 25$ lagged inputs, $P = 1$ seasonally lagged input, and $k = 13$ neurons in the hidden layer.

We finally combine all of our forecasts excluding the neural network into one by regressing the returns on each model's fitted values, like so:

$$r_t = \alpha + \beta_1 \text{AR}(2) + \beta_2 \text{ETS(A,N,N)} + \beta_3 \text{HW} + \beta_4 \text{Prophet},$$

where AR(2) represents the prediction of the AR(2) model, and so on.

```
arreg <- ret_ar$fitted
etsreg <- ret_ets$fitted
hwreg <- ret_hw$fitted
proreg <- ret_pro[[1]][[1]]$fit$est$.fitted

ourdata <- data.frame(training, arreg, etsreg, hwreg, proreg)

ret_combo <- tslm(training ~ arreg+etsreg+hwreg+proreg, data=ourdata)

summary(ret_combo)
```

```
##
## Call:
## tslm(formula = training ~ arreg + etsreg + hwreg + proreg, data = ourdata)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
```

```
## -0.290079 -0.033411 -0.000318  0.030093  0.199217
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.0003598  0.0032229  -0.112 0.911182
## arreg        0.9402548  0.1698068   5.537 6.44e-08 ***
## etsreg       0.0363786  0.1725770   0.211 0.833181
## hwreg       -0.1470495  0.2403723  -0.612 0.541135
## proreg       0.9835668  0.2513002   3.914 0.000111 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05681 on 318 degrees of freedom
## Multiple R-squared:  0.2052, Adjusted R-squared:  0.1952
## F-statistic: 20.53 on 4 and 318 DF,  p-value: 4.611e-15
```

Looking at the p-values of each coefficient, it seems that the Prophet model is most predictive of the returns time series, followed closely by the AR(2) model, then the Holt-Winters' model, then the ETS(A,N,N) model.

We compute the MSE (Mean Squared Error), MAE (Mean Absolute Error), and MAPE (Mean Absolute Percentage Error) for each forecast we've created. The MSE tells us the average squared distance between the observed and predicted values. The MAE tells us the average absolute difference between the observed values and the predicted values. MAPE indicates the average absolute percentage difference between the predicted value and the original value.

```
cast_ar <- forecast::forecast(ret_ar, h=73)
pred_ar <- cast_ar$mean

cast_ets <- forecast::forecast(ret_ets, h=73)
pred_ets <- cast_ets$mean

cast_hw <- hw(training,seasonal="additive", h=73)
pred_hw <- cast_hw$mean

pro_fc <- ret_pro |> forecast(h = 73)
pred_pro <- pro_fc$.mean

cast_nnar <- forecast::forecast(ret_nnar, h=73)
pred_nnar <- cast_nnar$mean

#use coefficients from model
pred_combo <- 1.0411225528*pred_ar-0.1412586593*pred_ets+0.2197996840*pred_hw+0.5571246669*pred_pro-0.0
```

Now we can set up a table to see which forecast performs the best.

```
library(huxtable)
```

```
##
## Attaching package: 'huxtable'

## The following object is masked from 'package:dplyr':
##
##     add_rownames
```

```
## The following object is masked from 'package:seasonal':
##
##     final

## The following object is masked from 'package:ggplot2':
##
##     theme_grey
```

```
MSE = c()
MAE = c()
MAPE = c()

#for AR(2)
MSE[1] = mean((testing-pred_ar)^2)
MAE[1] = mean(abs(testing-pred_ar))
MAPE[1] = mean(abs((testing-pred_ar)/testing))*100

#for ETS(A,N,N)
MSE[2] = mean((testing-pred_ets)^2)
MAE[2] = mean(abs(testing-pred_ets))
MAPE[2] = mean(abs((testing-pred_ets)/testing))*100

#for HW
MSE[3] = mean((testing-pred_hw)^2)
MAE[3] = mean(abs(testing-pred_hw))
MAPE[3] = mean(abs((testing-pred_hw)/testing))*100

#for Prophet
MSE[4] = mean((testing-pred_pro)^2)
MAE[4] = mean(abs(testing-pred_pro))
MAPE[4] = mean(abs((testing-pred_pro)/testing))*100

#for NNAR
MSE[5] = mean((testing-pred_nnar)^2)
MAE[5] = mean(abs(testing-pred_nnar))
MAPE[5] = mean(abs((testing-pred_nnar)/testing))*100

#for combo
MSE[6] = mean((testing-pred_combo)^2)
MAE[6] = mean(abs(testing-pred_combo))
MAPE[6] = mean(abs((testing-pred_combo)/testing))*100

table <- hux(
        Forecast  = c("ARIMA", "ETS", "HW","Prophet", "NNAR","Combo"),
        MSE=MSE,
        MAE=MAE,
        MAPE=MAPE
      )

table %>%
      set_all_padding(4) %>%
      set_outer_padding(0) %>%
      set_number_format(6) %>%
      set_bold(row = 1, col = everywhere) %>%
```

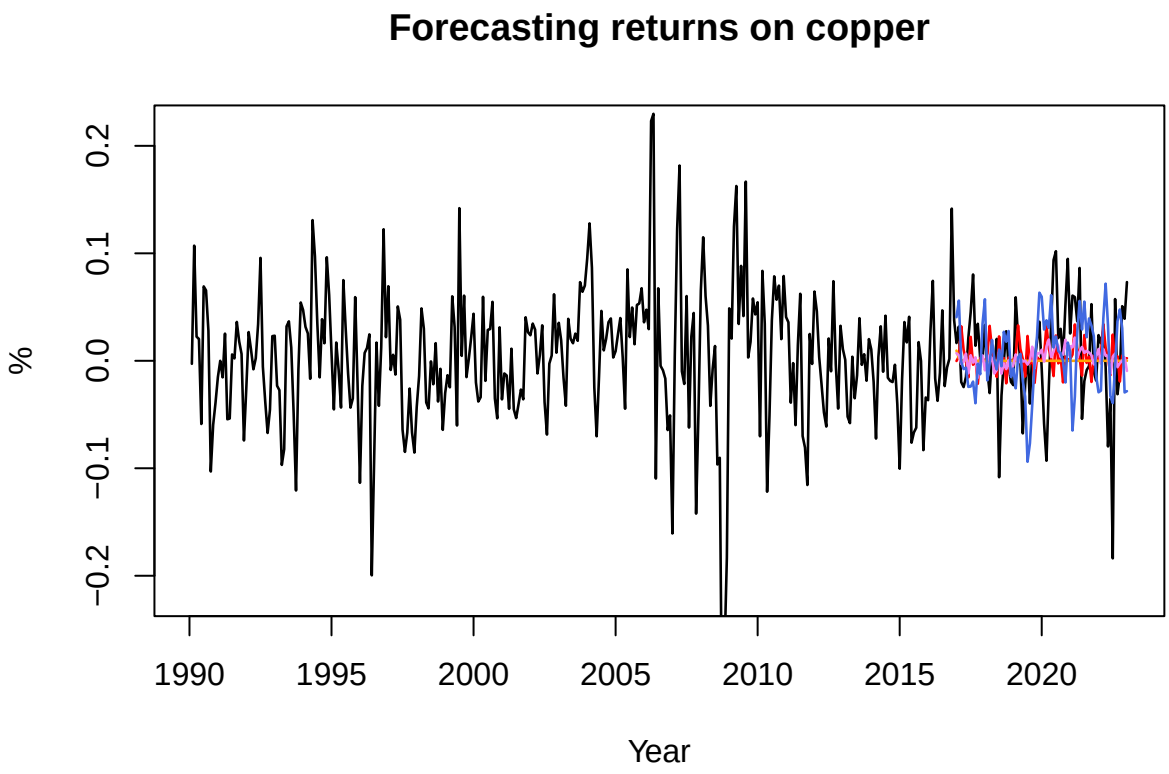
```
    set_bottom_border(row = 1, col = everywhere) %>%
    set_width(0.4) %>%
    set_caption("Forecast accuracy by various measures")
```

Table 1: Forecast accuracy by various measures

Forecast	MSE	MAE	MAPE
ARIMA	0.002307	0.035411	99.160658
ETS	0.002910	0.040355	1561.723751
HW	0.002685	0.037836	373.192501
Prophet	0.002235	0.034640	337.419016
NNAR	0.003306	0.045002	1086.126994
Combo	0.002257	0.034761	222.041077

In terms of MSE and MAE, the Prophet model performs the best, but in terms of MAPE, the AR(2) model performs the best. The next best model in terms of MSE and MAE is the combination forecast, which isn't surprising as the Prophet term of the combination regression model had the largest coefficient in terms of absolute value. The Holt-Winters' model also predicts the testing data reasonably well. The ETS(A,N,N) model fared poorly, but the NNAR model had the worst MSE, MAE, and MAPE. Although many of the models have seemingly large MAPE values, this statistic is especially distorted because the returns time series has very small values to begin with, and MAPE is calculated with a value from the returns time series in the denominator.

```
plot(returns, lwd=1.3, ylab="%", main="Forecasting returns on copper", xlab="Year", ylim=c(-0.22,0.22))
lines(pred_hw, col="red", lwd=1.3)
lines(pred_ar, col="orange", lwd=1.3)
lines(pred_pro, col="darkgreen", lwd=1.3)
lines(pred_combo, col="violet", lwd=1.3)
lines(pred_nnar, col="royalblue", lwd=1.3)
```



Conclusions and Future Work

We conclude that the Prophet model performed the best in terms of predicting the test data, as it had the lowest MSE and MAE, and we prefer MSE and MAE over MAPE because the latter was somewhat distorted by the small values in the returns time series. The ARIMA model also had very low MSE and MAE values and had the lowest MAPE, and the combination forecast which weighted both the Prophet and ARIMA models very heavily performed almost as well as the Prophet model. The NNAR model performed especially badly, likely because there weren't very many data points with which to train the model. Future work could analyze a daily time series or even intraday data, to allow for the neural network to be trained on a larger data set. Our analysis also only considered one time series, so we can't gauge the effectiveness of each forecasting method in general. In the future, comparing the performance of these forecasting methods across many different time series would allow for a more general analysis.

References

International Monetary Fund, retrieved from FRED Economic Data. (2023). *Global price of Copper* [Data set]. https://fred.stlouisfed.org/series/PCOPPUSDM