eda_forecast_v3.R

esteban

2025-05-06

```
library(forecast)
```

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

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
df_full=read.csv('~/rug/thesis/data/influencer_sample010525.csv') %>%
  mutate_at(c('comments','video_views','engagements','video_plays','shares','potentialReach'),as.numeri
  # rename(channel_uid=channelId) %>%
  filter(!is.na(engagements) & potentialReach>0) %>% mutate(engagements_rate=engagements/potentialReach
```

```
## Warning: There were 6 warnings in `mutate()`.
## The first warning was:
## i In argument: `comments = .Primitive("as.double")(comments)`.
## Caused by warning:
## ! NAs introduced by coercion
## i Run `dplyr::last_dplyr_warnings()` to see the 5 remaining warnings.
```

```
cor(df_full$engagements_rate,df_full$potentialReach,use = 'complete.obs')
```

```
## [1] -0.007751697
```

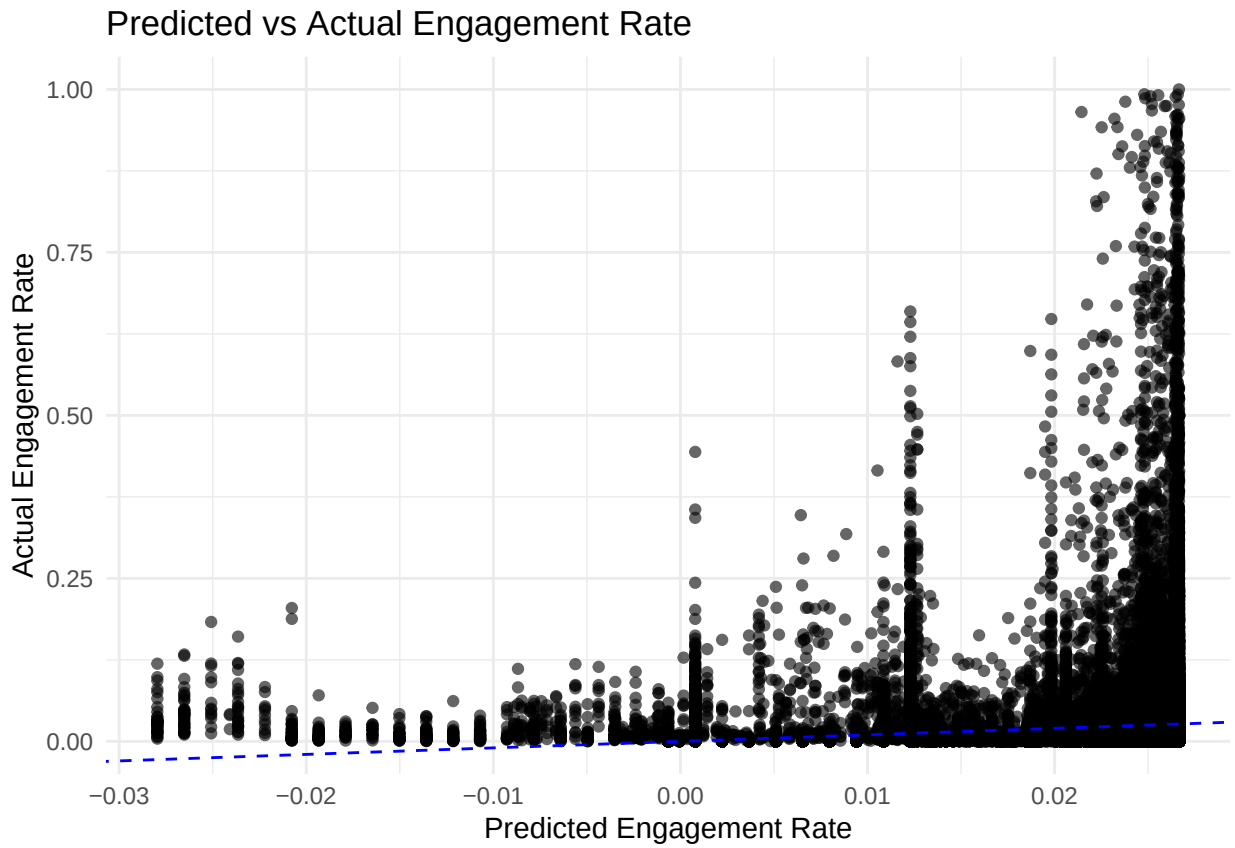
```
model_reg=lm(engagements_rate~potentialReach,data=df_full)
summary(model_reg)
```

```
##
## Call:
## lm(formula = engagements_rate ~ potentialReach, data = df_full)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -0.027  -0.026  -0.021  -0.010 137.218
##
```

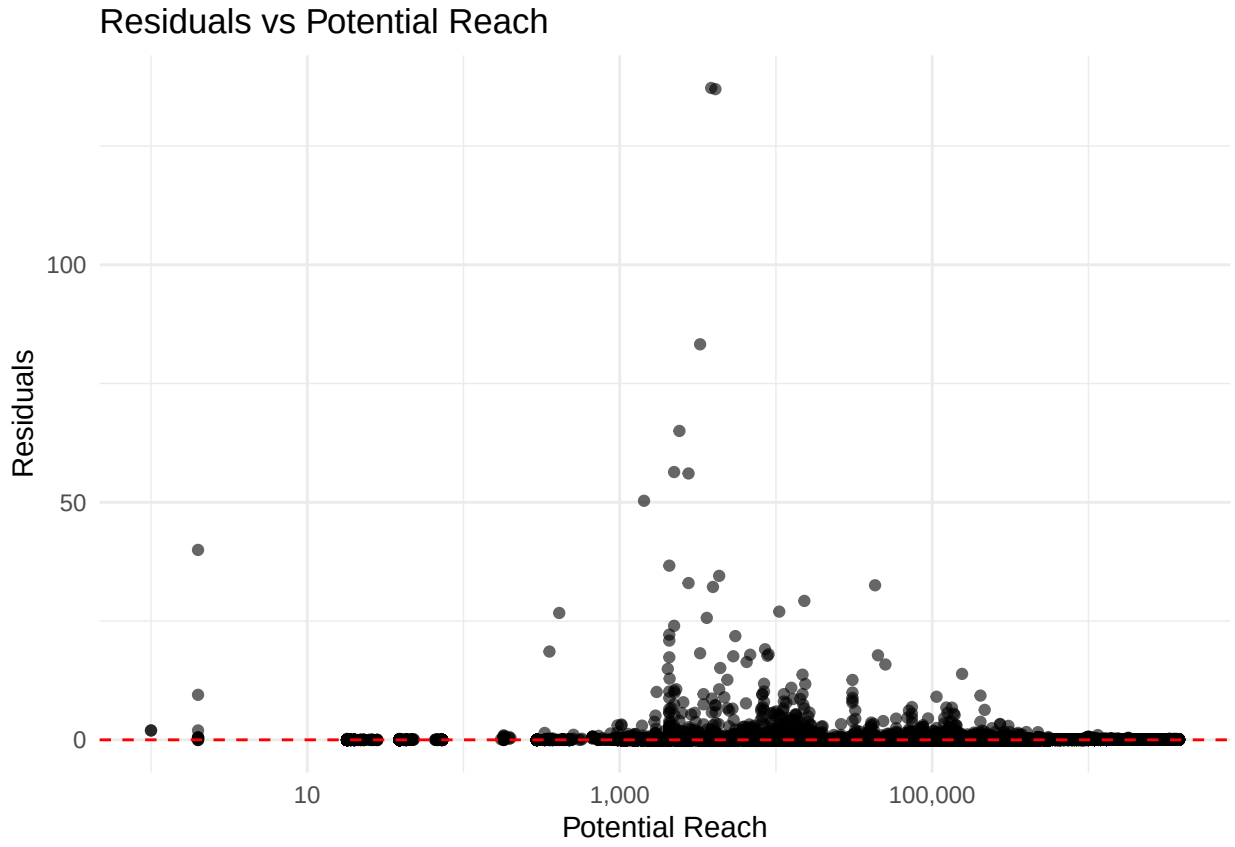
```
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     2.666e-02  1.282e-03  20.802  < 2e-16 ***
## potentialReach -1.437e-08  3.758e-09  -3.825 0.000131 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5781 on 243487 degrees of freedom
## Multiple R-squared:  6.009e-05,  Adjusted R-squared:  5.598e-05
## F-statistic: 14.63 on 1 and 243487 DF,  p-value: 0.0001307
```

```r
model_data <- model.frame(model_reg)
df_full$predicted <- predict(model_reg,newdata = df_full)

# Plot actual vs predicted
ggplot(df_full, aes(x = predicted, y = engagements_rate)) +
  geom_point(alpha = 0.6) +  # scatter plot
  geom_abline(intercept = 0, slope = 1, color = "blue", linetype = "dashed") +  # 45-degree line
  labs(title = "Predicted vs Actual Engagement Rate",
       x = "Predicted Engagement Rate",
       y = "Actual Engagement Rate") +scale_y_continuous(limits=c(0,1))+
  theme_minimal()
```

```
## Warning: Removed 479 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



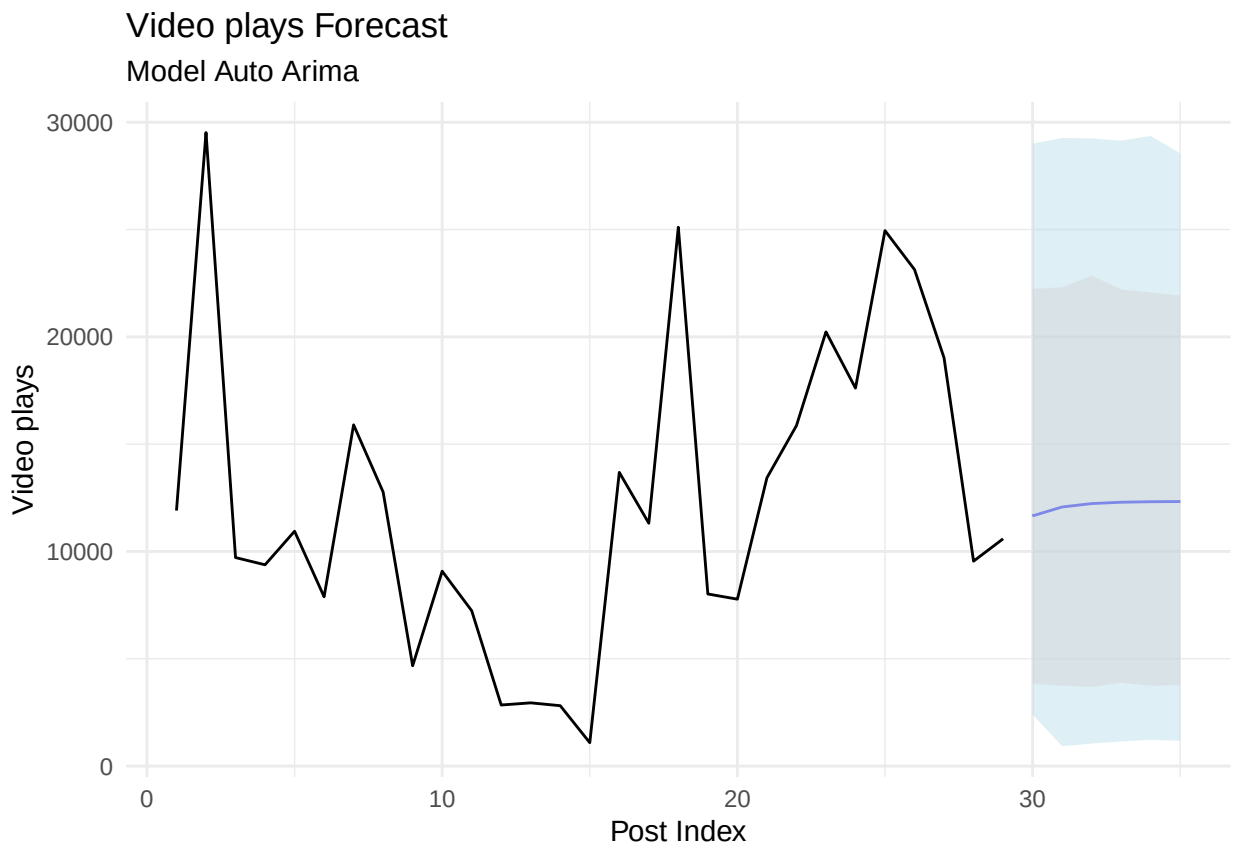
```r
resid_vals <- residuals(model_reg)
model_data$residuals <- resid_vals

# Plot residuals vs potentialReach
ggplot(model_data, aes(x = potentialReach, y = residuals)) +
  geom_point(alpha = 0.6) +
  geom_hline(yintercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Residuals vs Potential Reach",
       x = "Potential Reach",
       y = "Residuals") +scale_x_log10(labels=scales::comma_format())+
  theme_minimal()
```



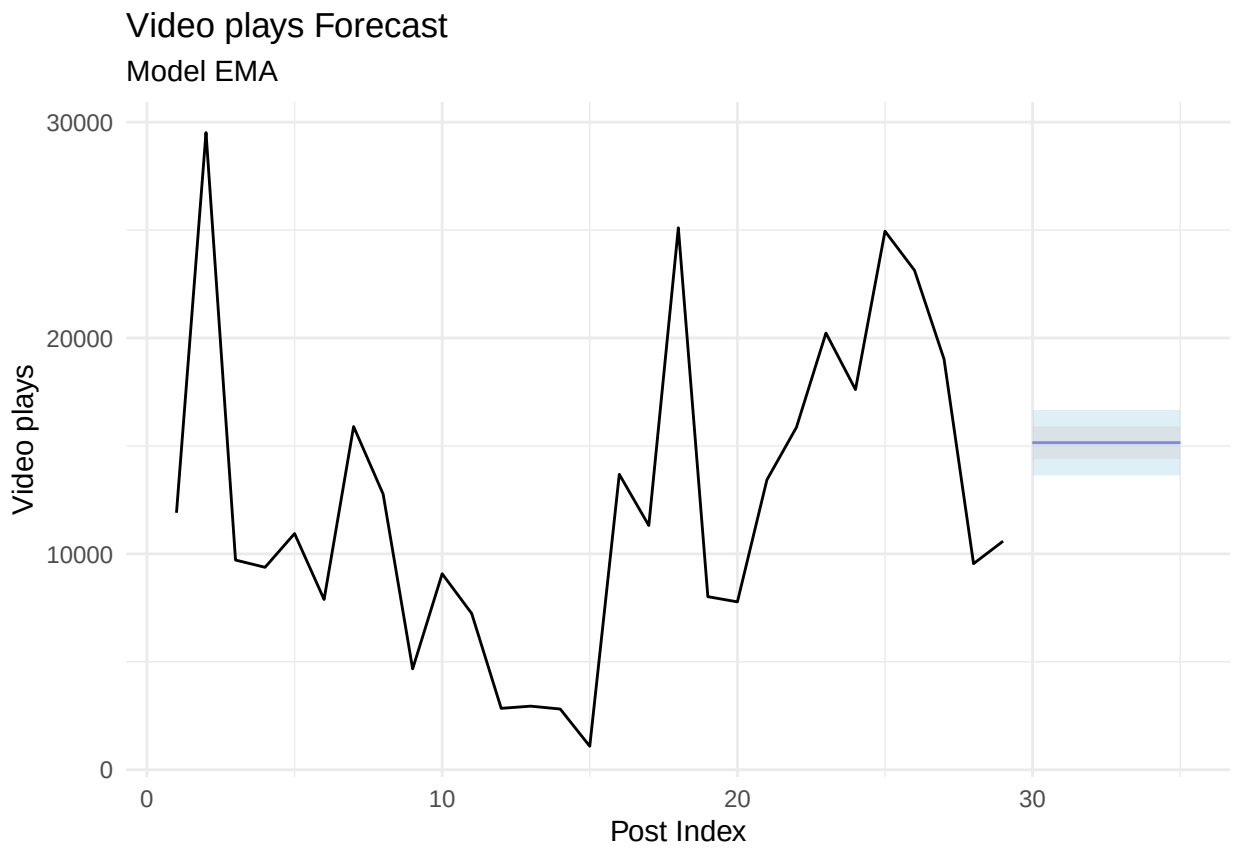

```r
plot_forecast <- function(x_ts, model,model_name,metric_name, h = 10, ci_width = 0.05) {

  n_obs <- length(x_ts)
  time_index <- seq_len(n_obs)
  future_index <- seq(n_obs + 1, n_obs + h)

  # Historical DataFrame
  historical_df <- data.frame(
    time = time_index,
    engagement_rate = as.numeric(x_ts)
  )

  if (is.numeric(model)) {
    # Proper forecasting model
```

```
    last_value <- tail(model[!is.na(model)], 1) # use last non-NA

    forecast_values <- rep(last_value, h)

    forecast_df <- data.frame(
      time = future_index,
      forecast = forecast_values,
      lower_80 = forecast_values * (1 - ci_width),
      upper_80 = forecast_values * (1 + ci_width),
      lower_95 = forecast_values * (1 - ci_width * 2),
      upper_95 = forecast_values * (1 + ci_width * 2)
    )

  } else {
    # TTR models (SMA, EMA, etc.) are just smoothed numeric vectors

    fc <- forecast::forecast(model, h = h,bootstrap=T,PI=T)

    forecast_df <- data.frame(
      time = future_index,
      forecast = as.numeric(fc$mean),
      lower_80 = as.numeric(fc$lower[, 1]),
      upper_80 = as.numeric(fc$upper[, 1]),
      lower_95 = as.numeric(fc$lower[, 2]),
      upper_95 = as.numeric(fc$upper[, 2])
    )
  }
  # Final Plot (consistent for all models)
  ggplot() +
    geom_line(data = historical_df, aes(x = time, y = engagement_rate), color = "black") +
    geom_line(data = forecast_df, aes(x = time, y = forecast), color = "blue") +
    geom_ribbon(data = forecast_df, aes(x = time, ymin = lower_95, ymax = upper_95), fill = "lightblue"
    geom_ribbon(data = forecast_df, aes(x = time, ymin = lower_80, ymax = upper_80), fill = "lightgrey"
    labs(title = glue::glue("{metric_name} Forecast"),
         subtitle = glue::glue("Model {model_name}"),
         x = "Post Index",
         y = metric_name) +
    theme_minimal()
}

# Example of usage
# x_ts <- df$engagement_rate %>% tail(24)
# df_temp=df_full %>% filter(channel_uid=="51b52b775df43f1ea0abf5db8789446d"& !is.na(video_plays))

df_temp=read.csv('~/rug/thesis/data/test_vv230225.csv') %>% mutate(engagement_rate=(reactions+comments),
                                                    published_date=lubridate::as_date(publishedDate)) %>%
  filter(!is.na(video_plays))
h=6
x_ts <- df_temp%>% pull(video_plays)
model_arima <- forecast::auto.arima(x_ts)
plot_forecast(x_ts, model_arima,model_name = 'Auto Arima' ,metric_name = 'Video plays',h = h)
```

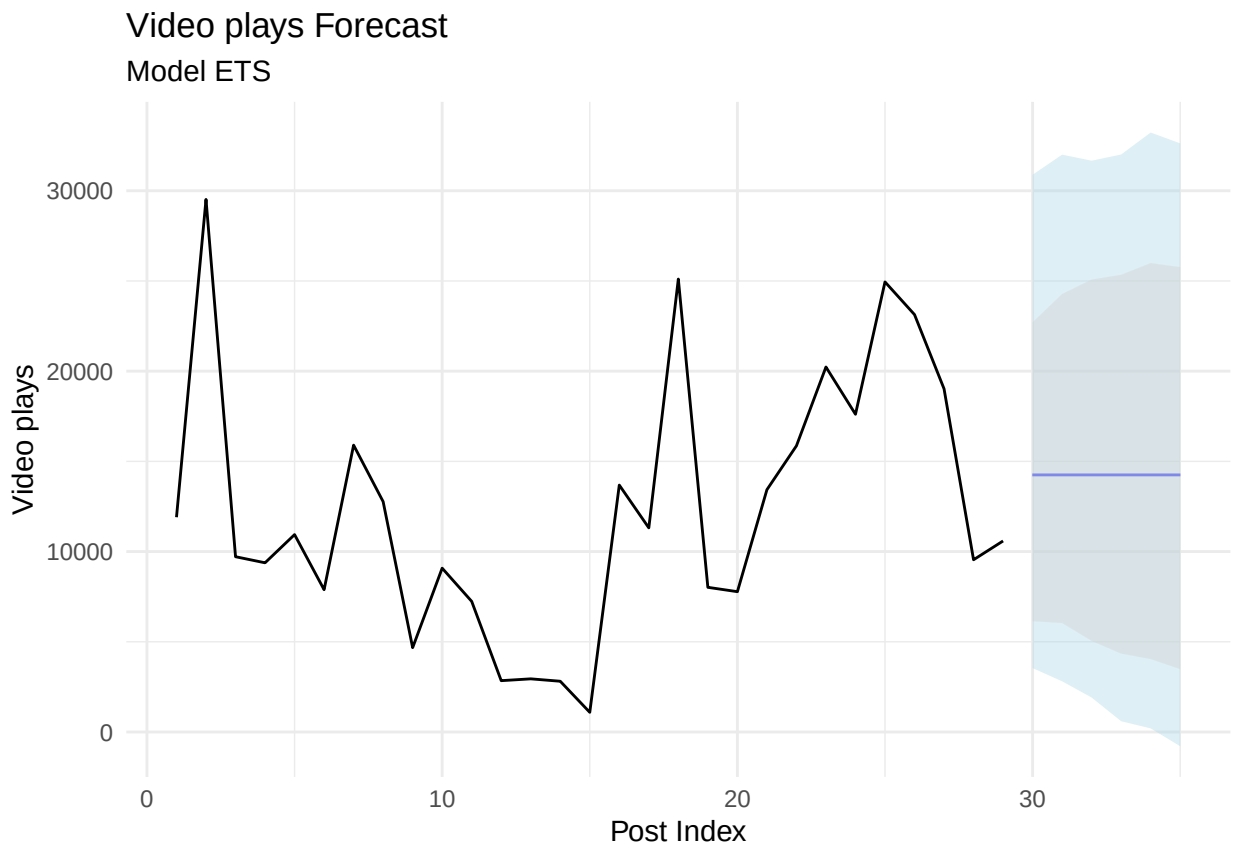
```
## Warning in forecast.forecast_ARIMA(model, h = h, bootstrap = T, PI = T): The
## non-existent PI arguments will be ignored.
```



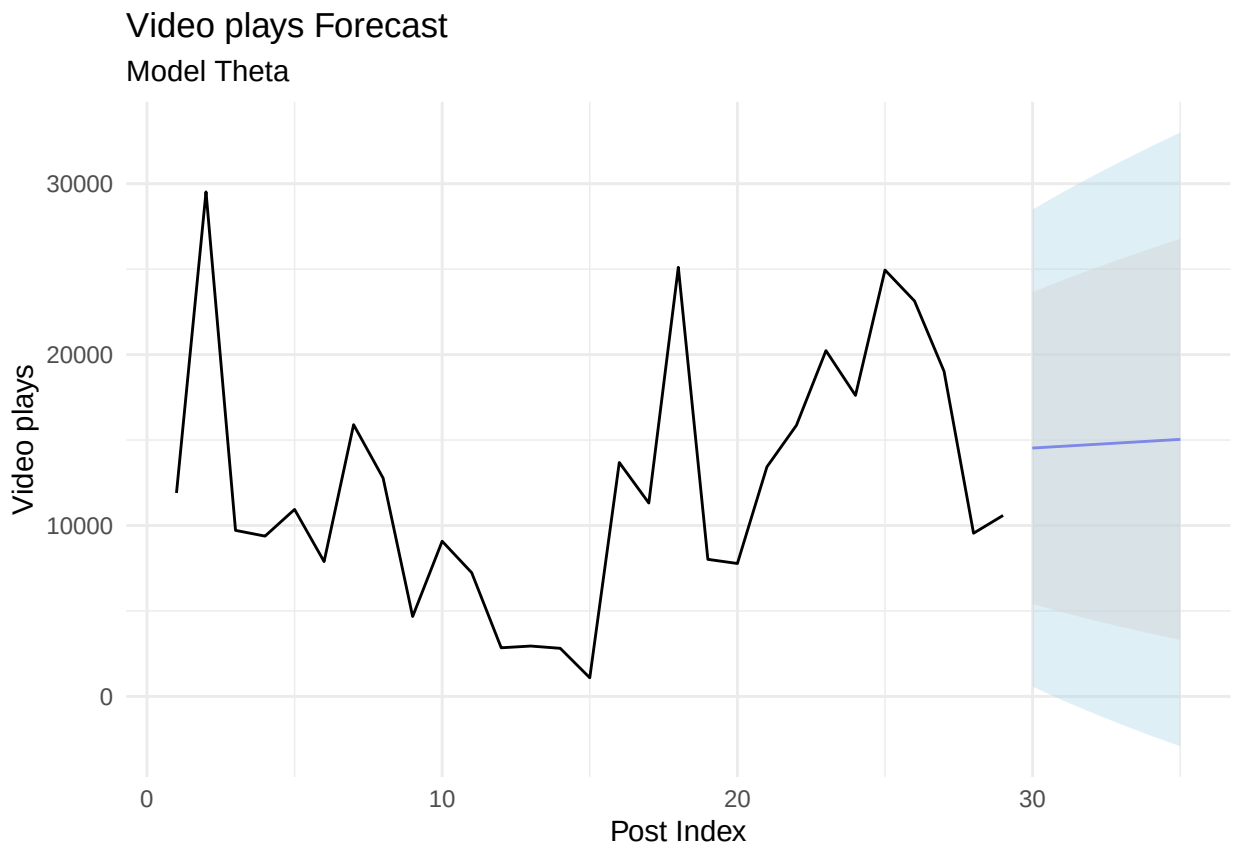

```
model_ema <- TTR::EMA(x_ts)
plot_forecast(x_ts, model_ema,model_name = 'EMA' ,metric_name = 'Video plays' ,h = h)
```



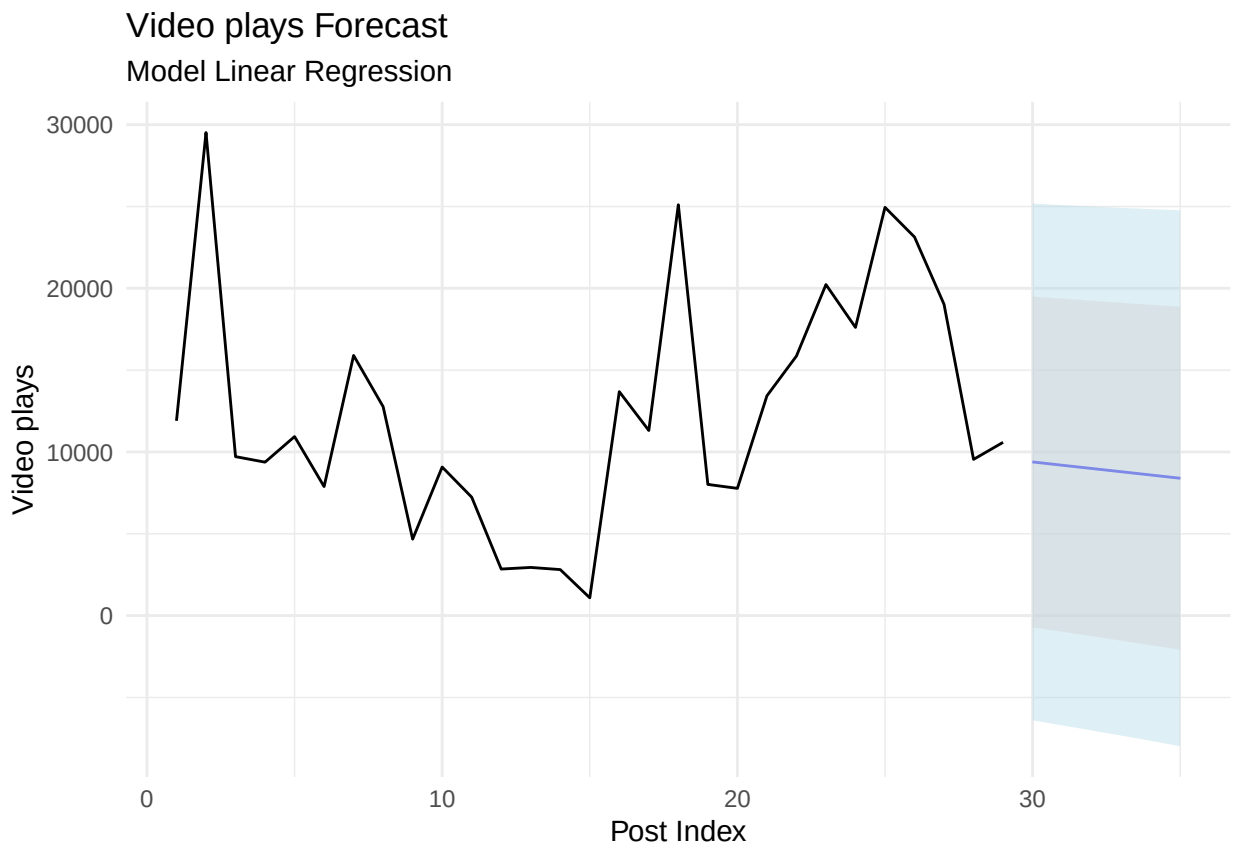

```
model_ets=forecast::ets(x_ts)
plot_forecast(x_ts,model_ets,model_name = 'ETS' ,metric_name = 'Video plays',h = h)
```



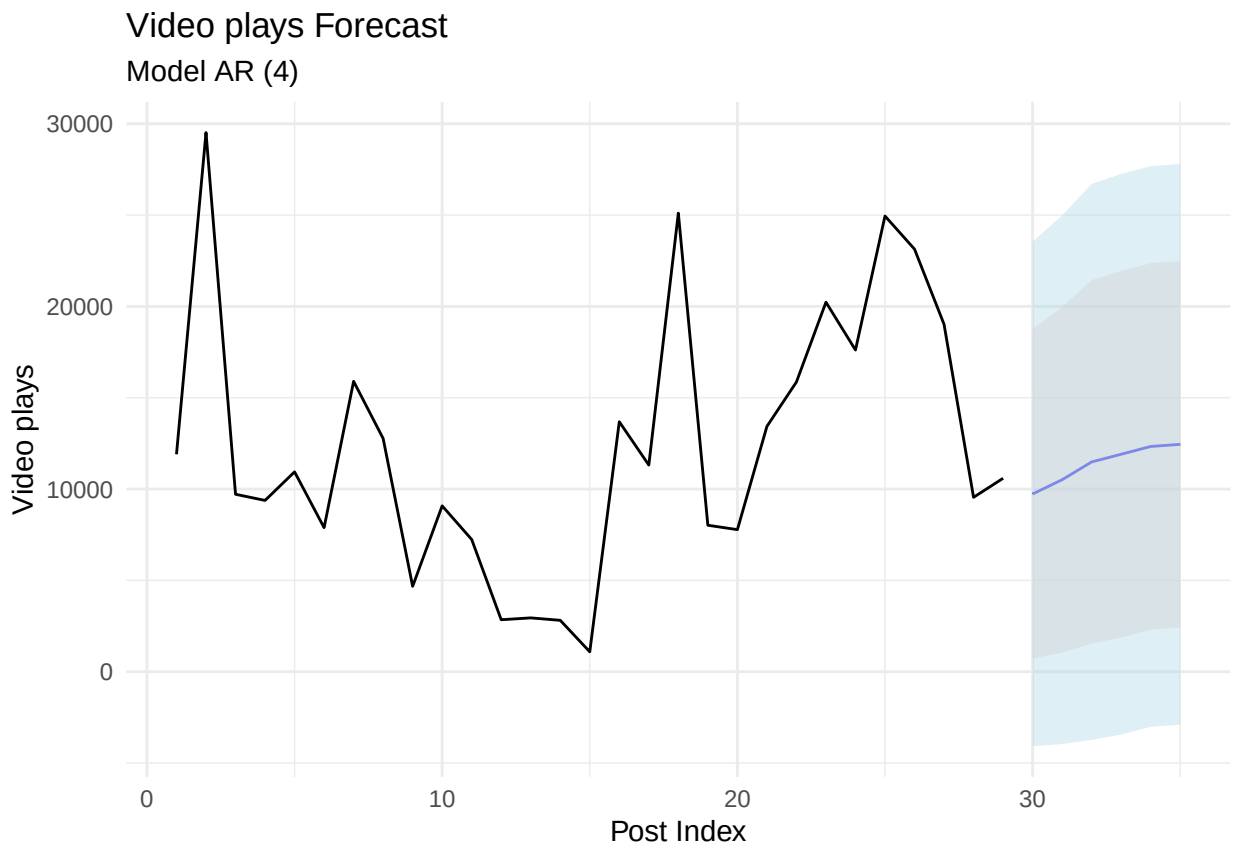

```
model_theta=forecast::thetaf(x_ts)
plot_forecast(x_ts,model_theta,model_name = 'Theta' ,metric_name = 'Video plays',h = h)
```



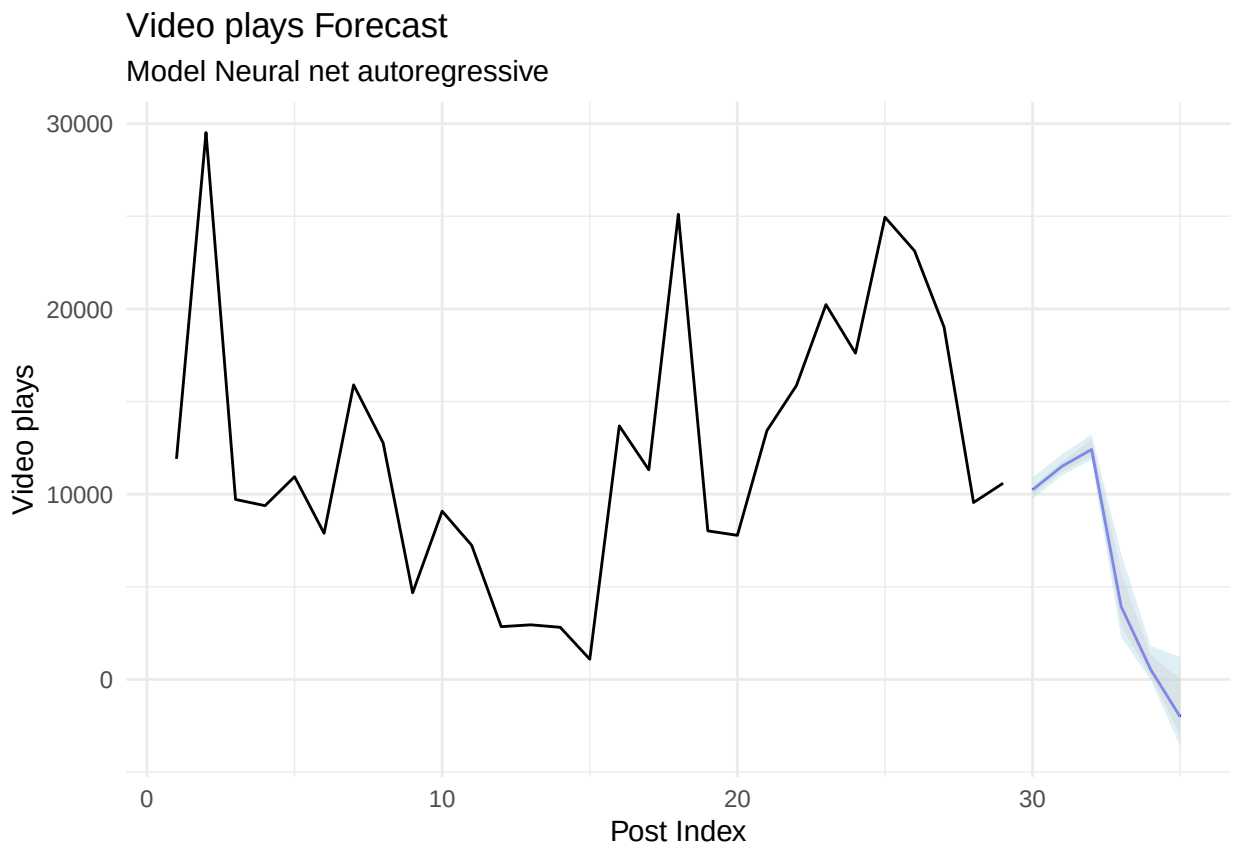

```
x_xts=xts::as.xts(x_ts,order.by=df_temp$published_date)
x_ts_ts <- ts(as.numeric(x_xts), frequency = 365, start = c(as.numeric(format(start(x_xts), "%Y")), as.n

model_lm=forecast::tslm(x_ts_ts~trend)
plot_forecast(x_ts,model_lm,model_name = 'Linear Regression',metric_name = 'Video plays',h = h)
```




```
ar_fun <- function(x, h){forecast(Arima(x, order=c(4,0,0)), h=h)}
model_ar=ar_fun(x_ts,h)
plot_forecast(x_ts,model_ar,model_name = 'AR (4)',metric_name = 'Video plays',h = h)
```




```
model_nnet=nnetar(x_ts,h)
plot_forecast(x_ts,model_nnet,model_name = 'Neural net autoregressive',metric_name = 'Video plays',h = h
```




```
test_metrics=forecast::tsCV(x_ts,ar_fun,h=6)
# Remove NAs created by cross-validation
# Calculate RMSE, MAE, MAPE safely ignoring NA
# RMSE
rmse <- sqrt(colMeans(test_metrics^2, na.rm = TRUE))

# MAE
mae <- colMeans(abs(test_metrics), na.rm = TRUE)

# MAPE
# For MAPE, you need actual values aligned properly
# Here's the trick:
n <- length(x_ts)
mape <- numeric(6)

for (h in 1:6) {
  # Predicted errors are x_ts[t+h] - forecast[t]
  actuals <- x_ts[(h + 1):n]
  forecasts <- x_ts[1:(n - h)] + test_metrics[1:(n - h), h]  # reconstruct forecast = actual - error
  mape[h] <- mean(abs((actuals - forecasts) / actuals), na.rm = TRUE) * 100
}

# Final metrics table
metrics <- data.frame(
  Horizon = 1:6,
  RMSE = rmse,
```

```
  MAE = mae,
  MAPE = mape
)

print(metrics)
```

```
##     Horizon      RMSE      MAE     MAPE
## h=1       1  9477.404 8161.348 88.04156
## h=2       2  8753.058 7992.076 95.97237
## h=3       3  8251.098 7060.064 81.24161
## h=4       4  8129.099 6420.866 68.95560
## h=5       5  9107.525 7257.831 68.84498
## h=6       6 10537.761 8490.968 98.33846
```