

# Project3

Hwan Cho

2025-03-13

## Introduction

My dataset is from Kaggle called NBA odds that depicts NBA money lines, game totals, spreads and second half totals for 2008-2023 regular season games (82 games per season). Since 2020 is incomplete due to COVID and 2023 season is also incomplete, I only considered from seasons 2008-2019. Furthermore, I focused on my favorite NBA team, Golden State Warriors (GSW) and their money lines. I changed GSW moneyline odds to probability of winning for easier interpretation and modeling. I fit 6 models including ARIMA, ETS, Holt-Winter, NNETAR, Prophet, and a combined model (combined 3 best models).

I wanted to investigate how historical betting odds can be used to predict future odds and illustrate that past behavior of the odds may reveal systematic patterns, trends, or seasonality—even in a competitive market like sports betting. Although this model is simplified compared to models owned by sports books, I wanted to hint at ideas of arbitrage. For example, if my forecasted probabilities differ significantly from those implied by current bookmaker odds, it could highlight potential value bets.

```
data <- read.csv("oddsData.csv")

unique(data$season)

## [1] 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022
## [16] 2023

head(data)

##          date season      team home.visitor     opponent score opponentScore
## 1 2007-10-30  2008      Utah           @ Golden State    117         96
## 2 2007-10-30  2008 LA Lakers           vs     Houston    93         95
## 3 2007-10-30  2008     Houston           @ LA Lakers    95         93
## 4 2007-10-30  2008 San Antonio           vs     Portland   106         97
## 5 2007-10-30  2008     Portland           @ San Antonio   97        106
## 6 2007-10-30  2008 Golden State           vs       Utah    96        117
##   moneyLine opponentMoneyLine total spread secondHalfTotal
## 1      100            -120 212.0     1      105.5
## 2      190            -230 199.0     5      99.0
## 3     -230             190 199.0    -5      99.0
## 4     -1400            900 189.5   -13      95.0
## 5      900            -1400 189.5   13      95.0
## 6     -120             100 212.0    -1      105.5
```

```

# only want to look at GSW games and remove season 2020 because only contains 32 games because it was p

GSW <- data %>%
  dplyr::filter(team == "Golden State" & season != 2020) %>%
  mutate(date = as.Date(date)) %>%
  rename(GSW_moneyline = moneyLine, GSW_score = score, GSW_spread = spread) %>%
  dplyr::select(date, season, team, home.visitor, opponent, GSW_score, opponentScore, GSW_moneyline, op

# create game index

GSW <- GSW %>%
  group_by(season) %>%
  mutate(game_number = row_number()) %>%
  ungroup()

convert_odds_to_prob <- function(odds) {
  ifelse(odds > 0, 100 / (odds + 100), -odds / (-odds + 100))
}

# Create a season-game identifier (e.g., 2008_1 for Game 1 of 2007-08) & convert moneyline to probability

GSW <- GSW %>%
  mutate(season_game = paste0(season, "_", game_number)) %>%
  mutate(probability = convert_odds_to_prob(GSW_moneyline))

head(GSW)

## # A tibble: 6 x 13
##   date      season team    home.visitor opponent  GSW_score opponentScore
##   <date>     <int> <chr>    <chr>       <chr>      <int>        <int>
## 1 2007-10-30  2008 Golden State vs      Utah        96         117
## 2 2007-11-02  2008 Golden State @     LA Clipper~    114         120
## 3 2007-11-03  2008 Golden State @     Utah        110         133
## 4 2007-11-06  2008 Golden State vs     Cleveland    104         108
## 5 2007-11-08  2008 Golden State vs     Dallas       115         120
## 6 2007-11-14  2008 Golden State vs     Detroit      104         111
## # i 6 more variables: GSW_moneyline <dbl>, opponentMoneyLine <dbl>,
## #   GSW_spread <dbl>, game_number <int>, season_game <chr>, probability <dbl>

ts_GSW <- GSW %>%
  dplyr::select(season, game_number, probability) %>%
  tidyr::pivot_wider(names_from = game_number, values_from = probability) %>%
  column_to_rownames("season") %>%
  as.matrix() %>%
  ts(frequency = 82)

head(ts_GSW) # each row represents that season

## Time Series:
## Start = c(1, 1)
## End = c(1, 6)
## Frequency = 82
##
```

1            2            3            4            5            6            7

```

## 1.000000 0.5454545 0.4878049 0.1904762 0.5744681 0.2666667 0.5000000 0.7142857
## 1.012195 0.2857143 0.2941176 0.3773585 0.5652174 0.6000000 0.7619048 0.4255319
## 1.024390 0.6969697 0.3448276 0.7058824 0.6551724 0.6078431 0.7222222 0.3571429
## 1.036585 0.6078431 0.7222222 0.1818182 0.6296296 0.4651163 0.5744681 0.5169082
## 1.048780 0.3636364 0.2857143 0.3125000 0.4761905 0.3846154 0.2564103 0.1666667
## 1.060976 0.4545455 0.5049505 0.2173913 0.4878049 0.7058824 0.2702703 0.4651163
##          8      9      10     11     12     13
## 1.000000 0.3125000 0.5454545 0.13888889 0.57446809 0.6226415 0.3703704
## 1.012195 0.6774194 0.3846154 0.46511628 0.51219512 0.6226415 0.2941176
## 1.024390 0.4255319 0.2500000 0.07142857 0.05714286 0.2500000 0.1250000
## 1.036585 0.4255319 0.2857143 0.27548209 0.73684211 0.6363636 0.1379310
## 1.048780 0.6153846 0.1818182 0.40816327 0.62264151 0.3921569 0.4761905
## 1.060976 0.5305164 0.4761905 0.18181818 0.35087719 0.5833333 0.3125000
##          14     15     16     17     18     19     20
## 1.000000 0.55555556 0.4761905 0.7222222 0.6296296 0.8620690 0.8181818 0.4166667
## 1.012195 0.57446809 0.1052632 0.1428571 0.5652174 0.4878049 0.2352941 0.1428571
## 1.024390 0.08333333 0.2272727 0.5918367 0.0800000 0.4347826 0.2105263 0.2816901
## 1.036585 0.48780488 0.3333333 0.2898551 0.4651163 0.3846154 0.5744681 0.2857143
## 1.048780 0.43478261 0.5555556 0.5744681 0.6296296 0.3636364 0.7619048 0.6969697
## 1.060976 0.60784314 0.5121951 0.6491228 0.8449612 0.5121951 0.3030303 0.6551724
##          21     22     23     24     25     26     27
## 1.000000 0.5833333 0.6666667 0.7297297 0.3333333 0.6551724 0.7560976 0.5744681
## 1.012195 0.5238095 0.5555556 0.4761905 0.0625000 0.4000000 0.3125000 0.1666667
## 1.024390 0.5454545 0.3846154 0.3333333 0.3846154 0.3333333 0.5918367 0.2469136
## 1.036585 0.2222222 0.1666667 0.2083333 0.1428571 0.6153846 0.3448276 0.4545455
## 1.048780 0.4878049 0.4761905 0.3225806 0.5454545 0.6078431 0.6923077 0.1886792
## 1.060976 0.6551724 0.2439024 0.6363636 0.2941176 0.7826087 0.6226415 0.8666667
##          28     29     30     31     32     33     34
## 1.000000 0.4081633 0.9090909 0.6774194 0.4000000 0.43478261 0.3174603 0.5833333
## 1.012195 0.3125000 0.1250000 0.2000000 0.1333333 0.07692308 0.4878049 0.5652174
## 1.024390 0.1904762 0.4255319 0.3571429 0.1333333 0.42553191 0.3225806 0.4444444
## 1.036585 0.4255319 0.6000000 0.6296296 0.2702703 0.40000000 0.1111111 0.1428571
## 1.048780 0.3333333 0.4545455 0.3773585 0.3333333 0.30303030 0.2631579 0.5000000
## 1.060976 0.4761905 0.3921569 0.7014925 0.6610169 0.47619048 0.3125000 0.5348837
##          35     36     37     38     39     40     41
## 1.000000 0.5555556 0.5000000 0.8461538 0.8245614 0.7802198 0.4761905 0.5833333
## 1.012195 0.4651163 0.1818182 0.2352941 0.1250000 0.4878049 0.6774194 0.4761905
## 1.024390 0.6774194 0.3448276 0.5918367 0.6153846 0.4761905 0.3333333 0.7619048
## 1.036585 0.2985075 0.8750000 0.3921569 0.3225806 0.6153846 0.7619048 0.6923077
## 1.048780 0.5121951 0.5454545 0.6226415 0.2631579 0.4166667 0.3571429 0.4761905
## 1.060976 0.7777778 0.2985075 0.3703704 0.1379310 0.3571429 0.3773585 0.5000000
##          42     43     44     45     46     47     48
## 1.000000 0.4761905 0.9090909 0.8461538 0.8360656 0.4444444 0.2105263 0.8076923
## 1.012195 0.6666667 0.7142857 0.3125000 0.8750000 0.2857143 0.2298851 0.2500000
## 1.024390 0.1379310 0.3278689 0.4878049 0.4761905 0.2298851 0.2777778 0.1250000
## 1.036585 0.7222222 0.2222222 0.3636364 0.4878049 0.6428571 0.6491228 0.7500000
## 1.048780 0.2597403 0.5555556 0.4761905 0.2564103 0.5555556 0.1818182 0.2439024
## 1.060976 0.4651163 0.4761905 0.5833333 0.5454545 0.6363636 0.8666667 0.4545455
##          49     50     51     52     53     54     55
## 1.000000 0.9230769 0.7777778 0.8888889 0.60000000 0.2857143 0.4255319 0.7560976
## 1.012195 0.3571429 0.3921569 0.1904762 0.48780488 0.6428571 0.5918367 0.3773585
## 1.024390 0.4081633 0.3636364 0.4444444 0.09090909 0.6551724 0.3571429 0.3508772
## 1.036585 0.4081633 0.6875000 0.5918367 0.31250000 0.5454545 0.5744681 0.3508772
## 1.048780 0.5833333 0.6226415 0.1612903 0.2777778 0.3225806 0.2500000 0.4255319

```

```

## 1.060976 0.1408451 0.4166667 0.3278689 0.63636364 0.4166667 0.8000000 0.4000000
##      56      57      58      59      60      61
## 1.000000 0.9565217 0.7014925 0.7435897 0.4651163 0.68253968 0.80198020
## 1.012195 0.7727273 0.6226415 0.5744681 0.3703704 0.54545455 0.25641026
## 1.024390 0.5000000 0.3125000 0.6363636 0.1666667 0.06896552 0.07692308
## 1.036585 0.3703704 0.5833333 0.6226415 0.3333333 0.52380952 0.18181818
## 1.048780 0.2222222 0.3921569 0.2985075 0.1333333 0.18181818 0.28169014
## 1.060976 0.5833333 0.2352941 0.2597403 0.3921569 0.5833333 0.7333333
##      62      63      64      65      66      67
## 1.000000 0.3076923 0.84000000 0.2597403 0.9677419 0.5744681 0.8333333
## 1.012195 0.1666667 0.24390244 0.6000000 0.4878049 0.5555556 0.6363636
## 1.024390 0.1858736 0.32258065 0.3278689 0.4651163 0.2352941 0.6226415
## 1.036585 0.2500000 0.66666667 0.5555556 0.3571429 0.7500000 0.4878049
## 1.048780 0.1273885 0.05405405 0.2105263 0.4545455 0.3225806 NA
## 1.060976 0.8857143 0.57446809 0.5918367 0.5833333 0.8963731 0.6875000
##      68      69      70      71      72      73      74
## 1.000000 0.63636364 0.2898551 0.4444444 0.9047619 0.2531646 0.2941176 0.2127660
## 1.012195 0.08333333 0.4878049 0.2352941 0.2380952 0.1904762 0.0952381 0.7014925
## 1.024390 0.11764706 0.1904762 0.3636364 0.4878049 0.4347826 0.3333333 0.1000000
## 1.036585 0.40816327 0.3389831 0.2222222 0.1455604 0.2173913 0.7701149 0.9000000
## 1.048780 NA NA NA NA NA NA NA NA
## 1.060976 0.27777778 0.6226415 0.2500000 0.8076923 0.5454545 0.8333333 0.8461538
##      75      76      77      78      79      80      81
## 1.000000 0.3703704 0.8571429 0.3636364 0.9375000 0.6296296 0.95238095 0.3846154
## 1.012195 0.7826087 0.3225806 0.4878049 0.7368421 0.3174603 0.05882353 0.3921569
## 1.024390 0.6551724 0.2392344 0.6000000 0.5652174 0.5652174 0.27027027 0.2439024
## 1.036585 0.1481481 0.1739130 0.3333333 0.1818182 0.3333333 0.71428571 0.1081081
## 1.048780 NA NA NA NA NA NA NA NA
## 1.060976 0.77777778 0.7916667 0.6875000 0.7500000 0.3921569 0.26666667 0.7435897
##      82
## 1.000000 0.8181818
## 1.012195 0.1851852
## 1.024390 0.2000000
## 1.036585 0.6875000
## 1.048780 NA
## 1.060976 0.8113208

```

## ARIMA Model

Using our ARIMA Model, we find that differencing of 1 needed to account for trend, MA(2) model for the cycles, and AR(2) to account for the seasonality. We observe that in our ARIMA model's ACF, there are still some lags that are significant, so not all dynamics are explained the model. When we look at the forecasts, we observe that since this is an MA model, the forecast was a horizontal line. Thus, looking at ETS, NNETAR, or Prophet explicitly account for trends and seasonality and typically produce non-flat forecasts when those patterns are present. The testing RMSE is 0.1696273, MAE is 0.1206004, and MAPE is 23.02255.

```

# Create a long time series from the entire matrix (for visualization purposes)
long_ts <- ts(as.vector(t(ts_GSW)), frequency = 82)

# Training: Seasons 2008-2017 (rows 1 to 10)
# Test: Seasons 2018-2019 (rows 11 and 12)
train_seasons <- ts_GSW[1:10, ]

```

```

test_seasons <- ts_GSW[11:12, ]

# Convert the training set to a long vector and then to a time series
long_train <- as.vector(t(train_seasons))
long_train_ts <- ts(long_train, frequency = 82, start = c(1, 1)) # start at "Season 1, Game 1"

# Calculate the number of training observations
n_train <- length(long_train_ts)

# Compute the Correct Start for the Test Time Series
# Since ts() expects start as a vector of (season, game) given frequency = 82,
# we compute these from the total number of training observations.
start_season <- floor(n_train / 82) + 1 # e.g., floor(820/82)+1 = 10+1 = 11
start_game <- ifelse(n_train %% 82 == 0, 1, (n_train %% 82) + 1)
# If training ends exactly at a season boundary, we start at game 1 of the next season.

# Convert the test set to a long vector and then to a time series using the computed start

# Convert the test set to a long vector and then to a time series.
# We set the test series to start immediately after the training data.
long_test <- as.vector(t(test_seasons))
long_test_ts <- ts(long_test, frequency = 82, start = c(start_season, start_game))

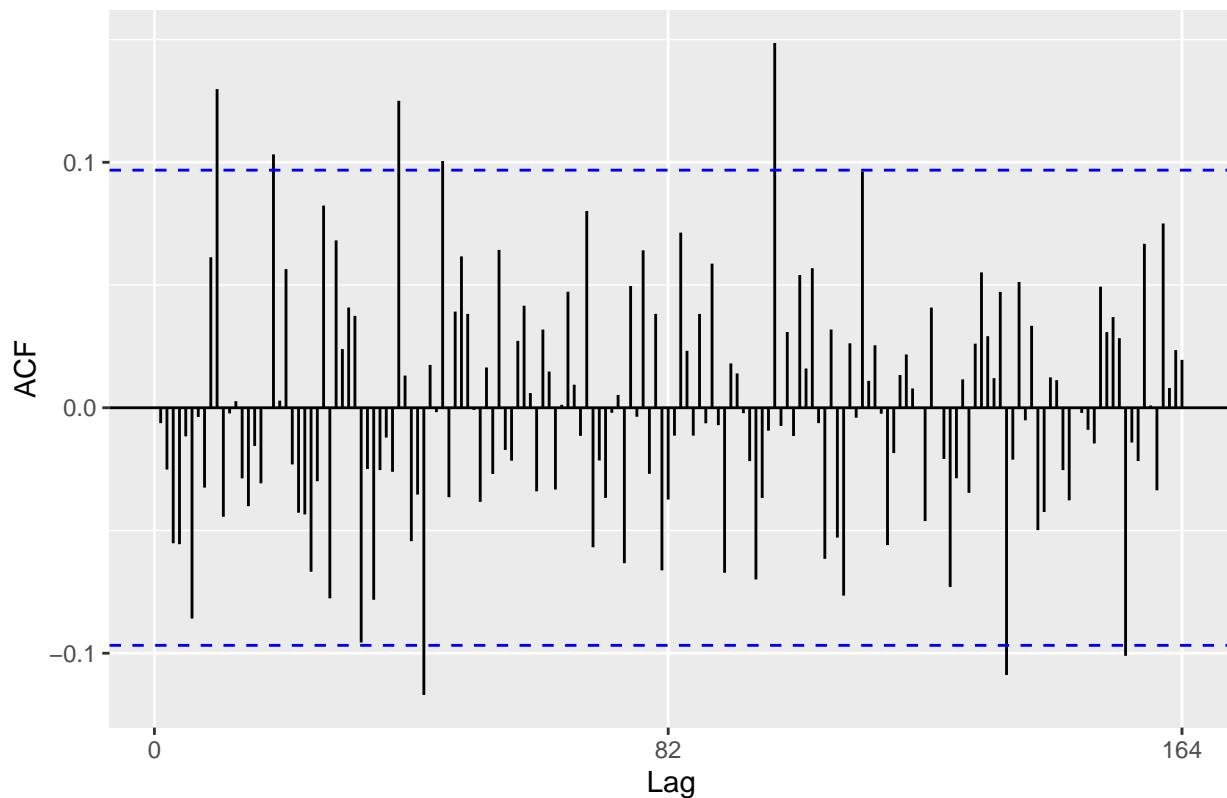
arima_model <- auto.arima(long_train_ts)
summary(arima_model) # arima model is MA(2) with one difference to account for trend and cycles, and AR

## Series: long_train_ts
## ARIMA(0,1,2)(1,0,0)[82]
##
## Coefficients:
##          ma1      ma2      sar1
##        -0.8424 -0.0888 -0.0524
## s.e.    0.0342  0.0342  0.0384
##
## sigma^2 = 0.03538: log likelihood = 202.39
## AIC=-396.78   AICc=-396.73   BIC=-377.95
##
## Training set error measures:
##          ME      RMSE      MAE      MPE      MAPE      MASE
## Training set 0.005767185 0.1876361 0.1546183 -22.46196 44.16782 0.688018
##          ACF1
## Training set 0.0009848986

# check residuals
resid_arima <- residuals(arima_model)
ggAcf(resid_arima) + ggtitle("ACF of ARIMA Residuals")

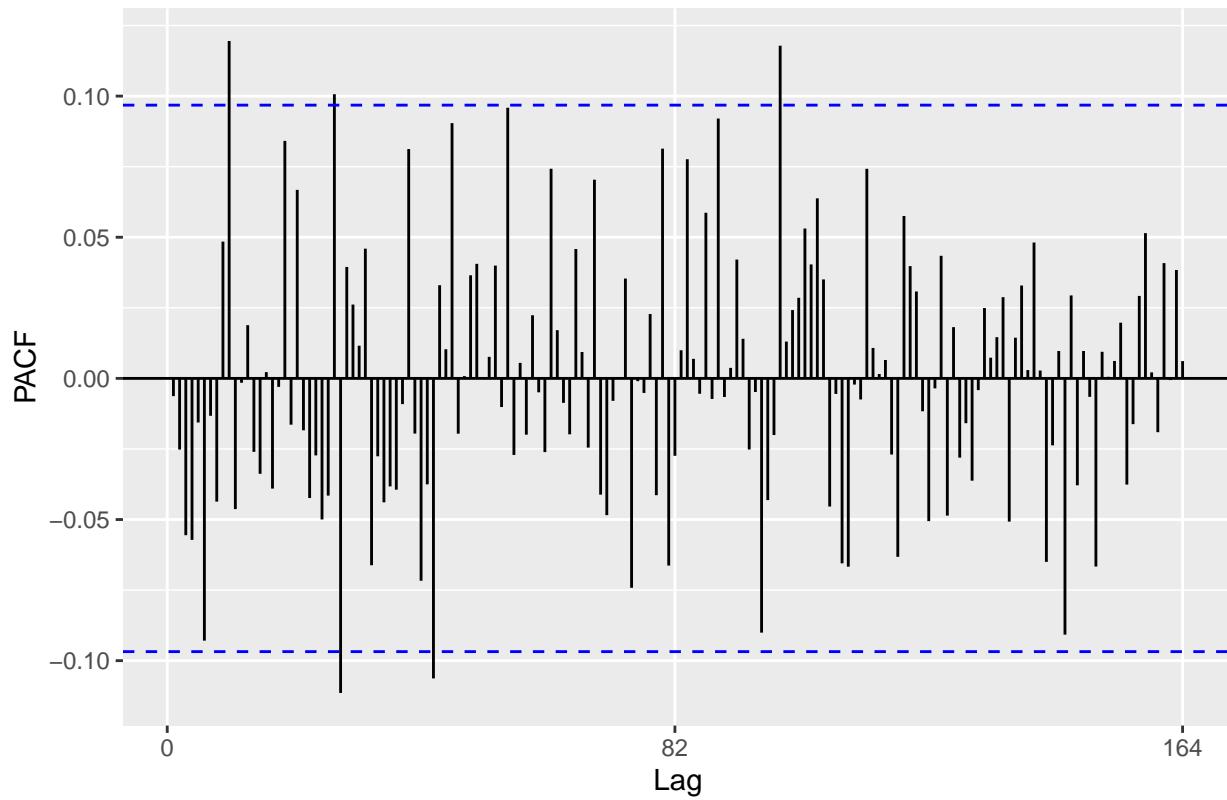
```

## ACF of ARIMA Residuals



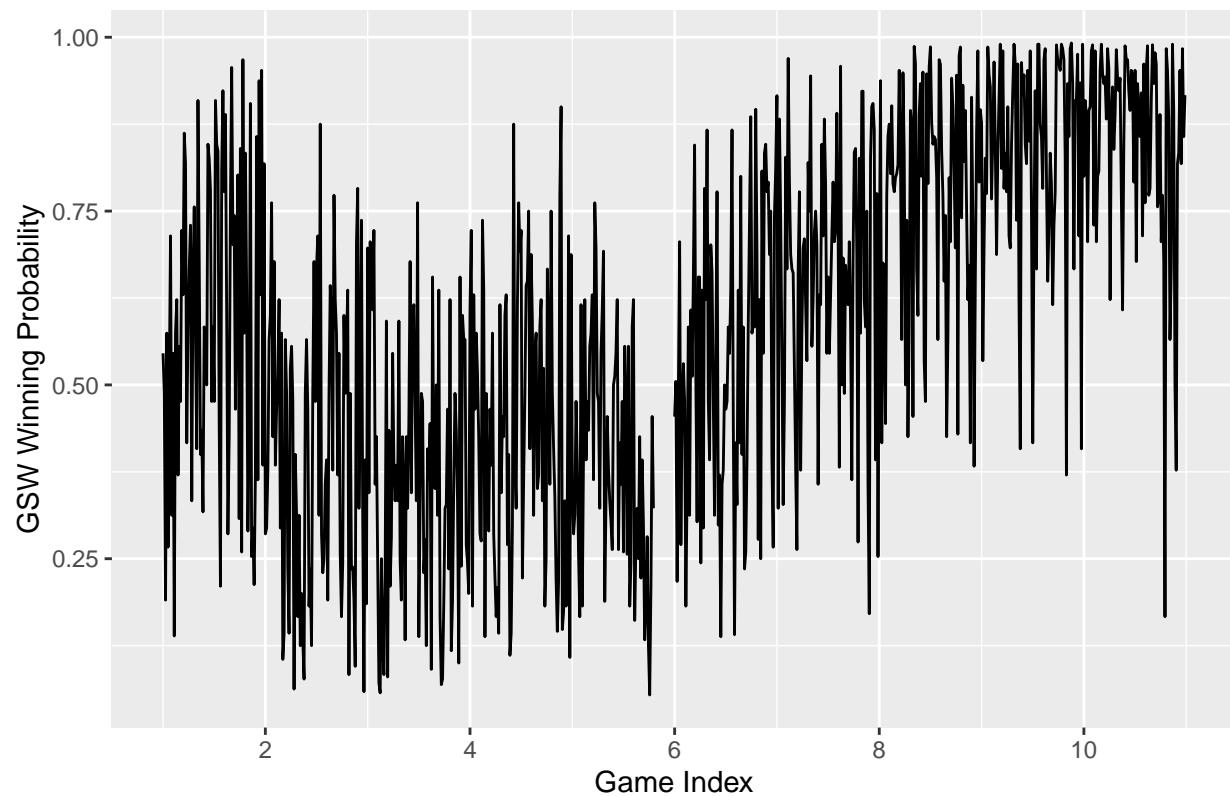
```
ggPacf(resid_arima) + ggttitle("PACF of ARIMA Residuals")
```

## PACF of ARIMA Residuals



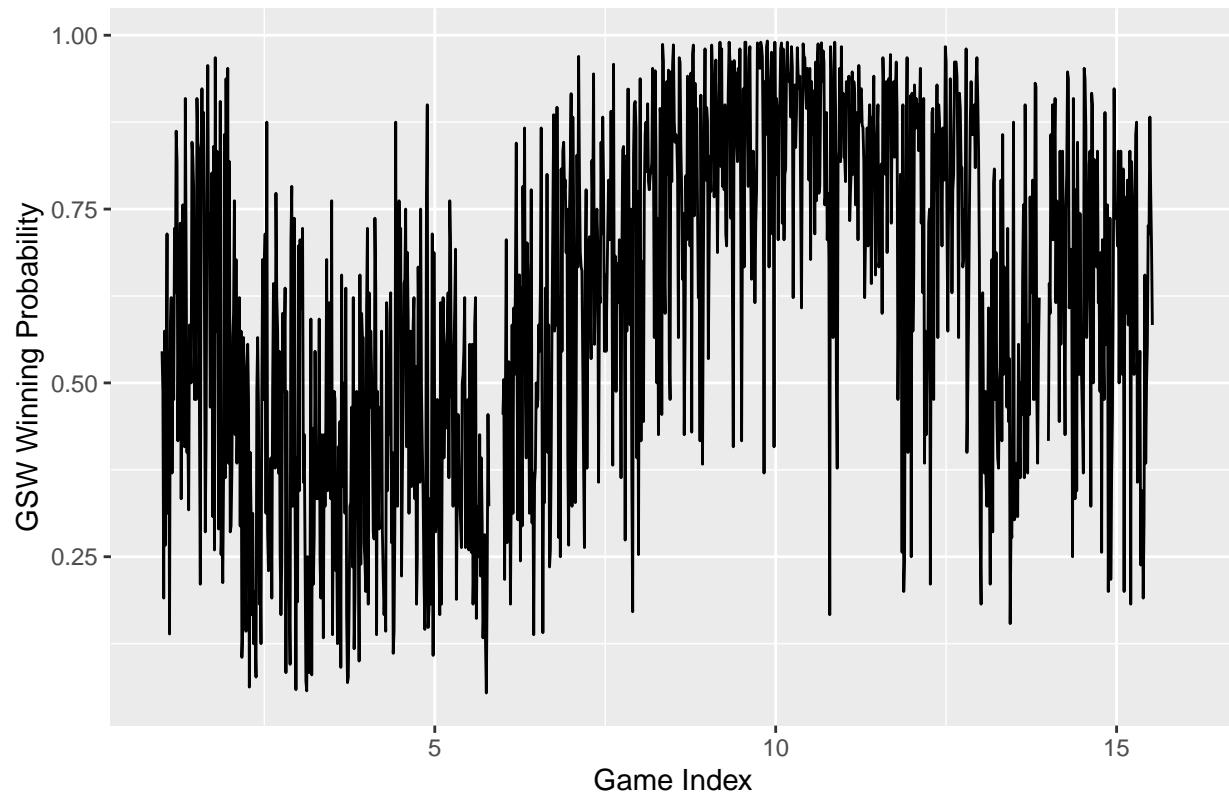
```
# Plot the training time series
autoplot(long_train_ts) +
  ggtitle("Training Time Series (2008-2017)") +
  xlab("Game Index") + ylab("GSW Winning Probability")
```

## Training Time Series (2008–2017)



```
# Plot the entire long time series (all seasons)
autoplot(long_ts) +
  ggtitle("Complete Long Time Series (2008–2019)") +
  xlab("Game Index") + ylab("GSW Winning Probability")
```

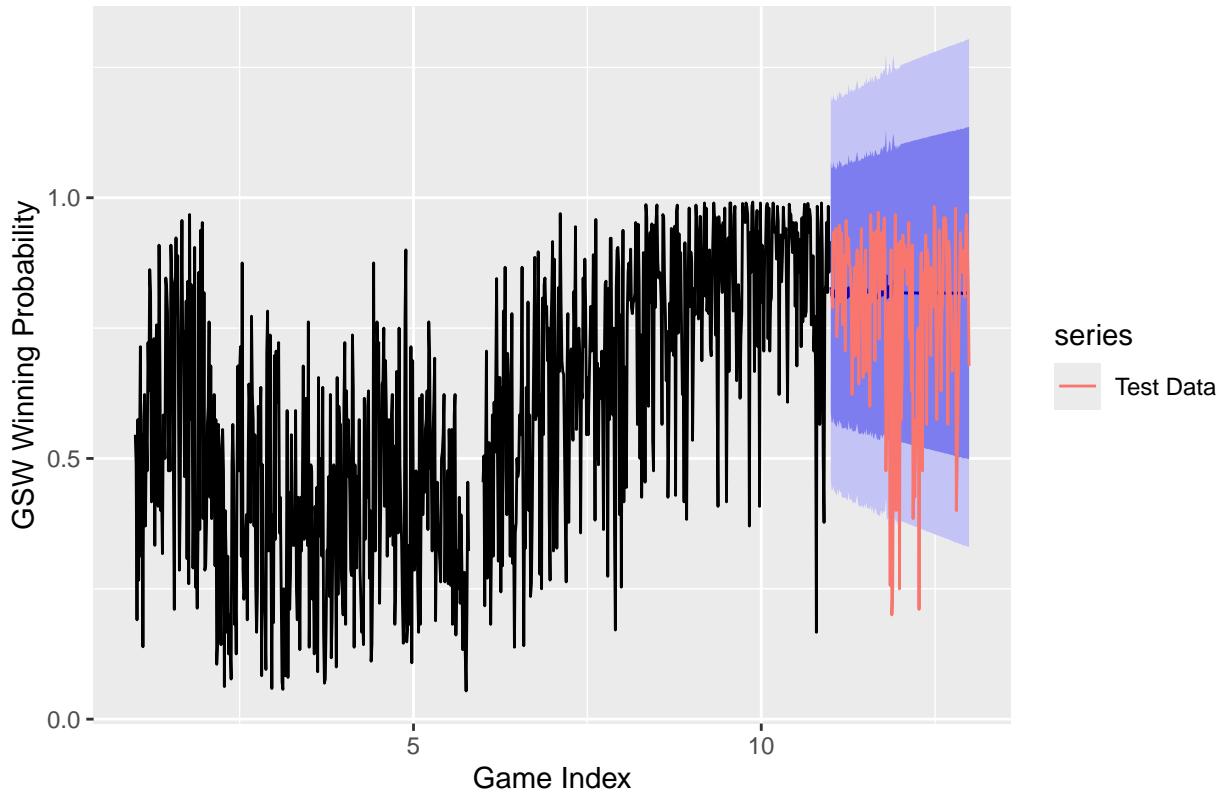
## Complete Long Time Series (2008–2019)



```
# Set forecast horizon equal to the number of test observations
forecast_horizon <- length(long_test_ts)
arima_forecast <- forecast(arima_model, h = forecast_horizon)

# Plot the forecast alongside the actual test data
autoplot(arima_forecast) +
  autolayer(long_test_ts, series = "Test Data") +
  ggtitle("ARIMA Forecast (Training: 2008–2017, Test: 2018–2019)") +
  xlab("Game Index") + ylab("GSW Winning Probability")
```

## ARIMA Forecast (Training: 2008–2017, Test: 2018–2019)



```
# Evaluate forecast accuracy (RMSE)
accuracy_metrics <- accuracy(arima_forecast, long_test_ts)
print(accuracy_metrics)
```

```
##               ME      RMSE      MAE      MPE      MAPE      MASE
## Training set 0.005767185 0.1876361 0.1546183 -22.46196 44.16782 0.6880180
## Test set     -0.033951992 0.1696273 0.1206004 -13.61858 23.02255 0.5366456
##                   ACF1 Theil's U
## Training set 0.0009848986       NA
## Test set     0.2368819549 0.7718857
```

## ETS Model

Since our frequency was too large, ETS ignored seasonality. I assumed that this is not too harmful for the model, because we are dealing with a MA model, so I continued with this approach and did not use stlf(). Like our ARIMA model, we observe that there are some lags in ACF and PACF that are significant. Thus, our model doesn't fully capture all the dynamics. The RMSE is 0.1729378, MAE is 0.1212884, and MAPE is 23.45355 for our test set.

```
ets_model <- ets(long_train_ts)
```

```
## Warning in ets(long_train_ts): Missing values encountered. Using longest
## contiguous portion of time series
```

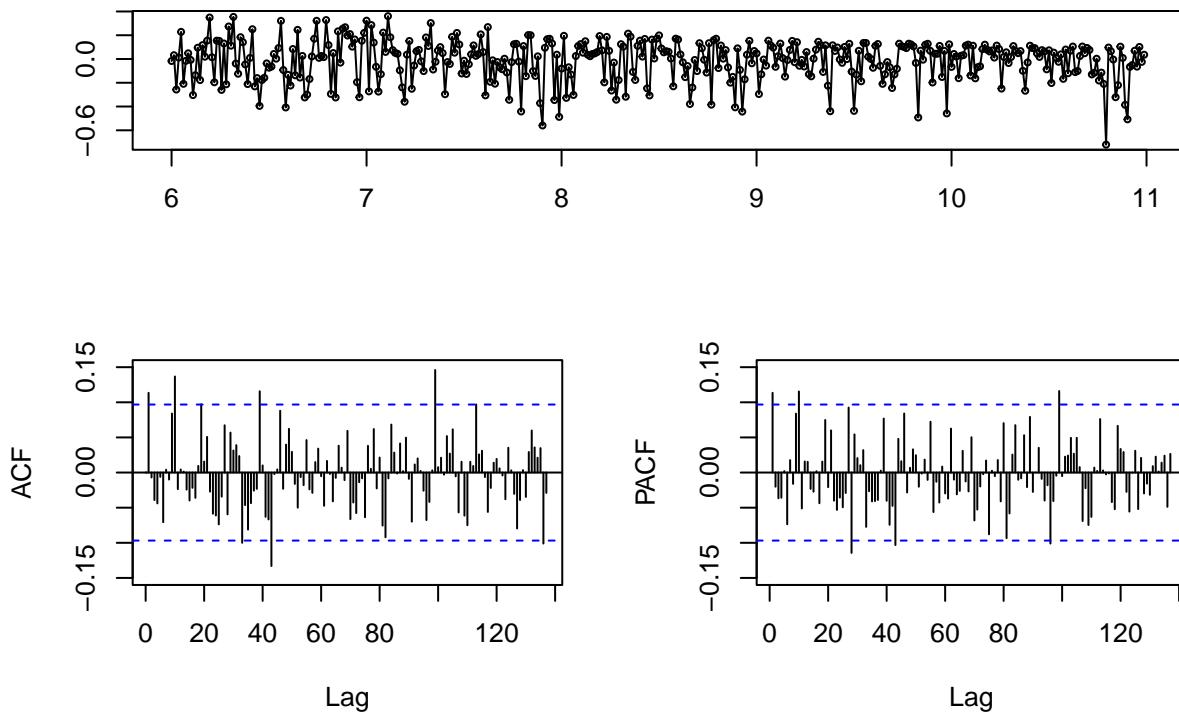
```
## Warning in ets(long_train_ts): I can't handle data with frequency greater than
## 24. Seasonality will be ignored. Try stlf() if you need seasonal forecasts.
```

```
summary(ets_model)
```

```
## ETS(A,A,N)
##
## Call:
## ets(y = long_train_ts)
##
## Smoothing parameters:
##   alpha = 4e-04
##   beta  = 4e-04
##
## Initial states:
##   l = 0.4696
##   b = 0.0017
##
## sigma:  0.1727
##
##      AIC     AICc      BIC
## 1032.602 1032.751 1052.683
##
## Training set error measures:
##           ME      RMSE      MAE      MPE      MAPE      MASE
## Training set -0.01407004 0.1718725 0.1328943 -12.32806 26.15424 0.6415586
##           ACF1
## Training set 0.1134954
```

```
tsdisplay(ets_model$residuals)
```

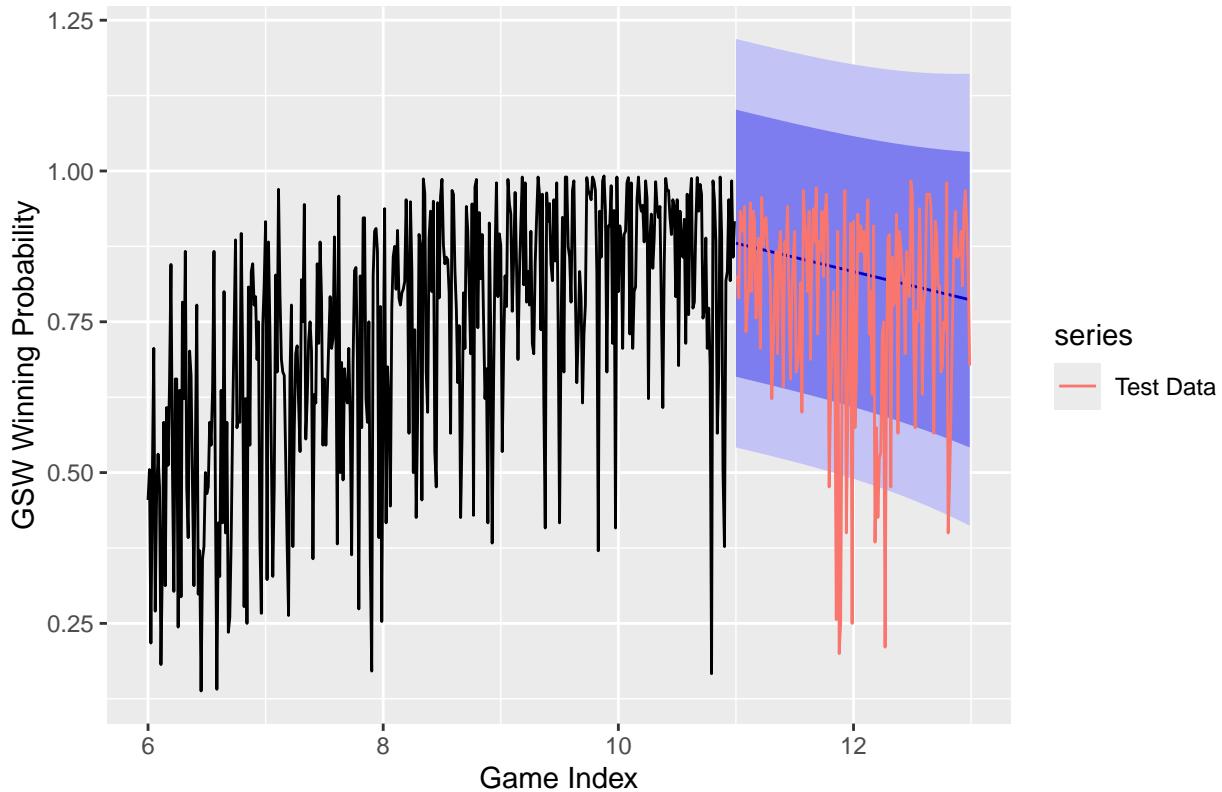
### ets\_model\$residuals



```
forecast_horizon <- length(long_test_ts)
ets_forecast <- forecast(ets_model, h = forecast_horizon)

# Plot the ETS Forecast vs. Test Data
autoplot(ets_forecast) +
  autolayer(long_test_ts, series = "Test Data") +
  ggtitle("ETS Forecast (Training: 2008-2017, Test: 2018-2019)") +
  xlab("Game Index") +
  ylab("GSW Winning Probability")
```

## ETS Forecast (Training: 2008–2017, Test: 2018–2019)



```
# Evaluate Forecast Accuracy
ets_accuracy <- accuracy(ets_forecast, long_test_ts)
print(ets_accuracy)
```

```
##               ME      RMSE      MAE      MPE      MAPE      MASE
## Training set -0.01407004 0.1718725 0.1328943 -12.32806 26.15424 0.6415586
## Test set     -0.05119971 0.1729378 0.1212884 -15.87176 23.45355 0.5855300
##             ACF1 Theil's U
## Training set 0.1134954       NA
## Test set     0.2317142 0.7791461
```

## Holt Winter's Model

Using our Holt Winter's Model, we first interpolated missing NA values. When looking at the acf and pacf of the residuals in the holt winter's model, we found that some ACF and PACF lags are significant indicating that not all dynamics are accounted for. The RMSE of the test set is 0.2071309, MAE is 0.1454064, and MAPE is 27.39466.

```
# our time series has NA values, so we interpolate missing values

if(any(is.na(long_train_ts))) {
  message("Missing values found in long_train_ts. Interpolating missing values using na.interp().")
  long_train_ts <- na.approx(long_train_ts)
}
```

```

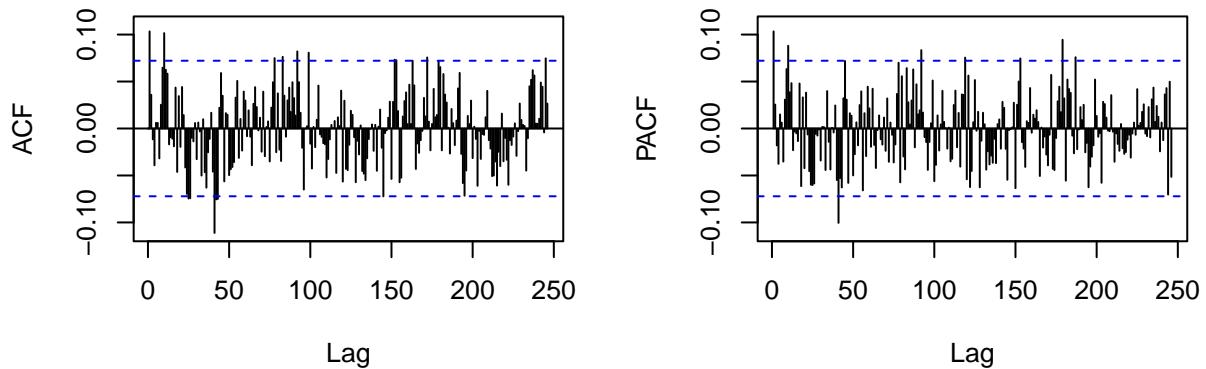
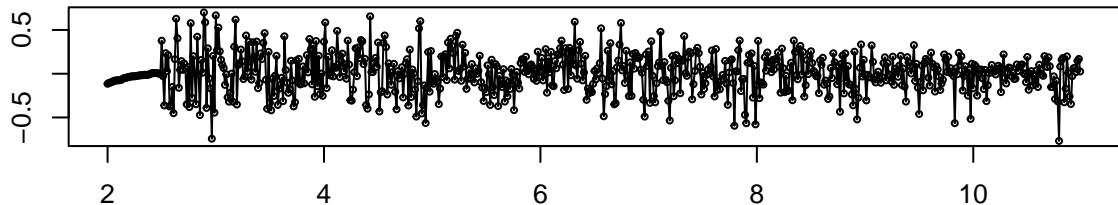
## Missing values found in long_train_ts. Interpolating missing values using na.interp().

hw_model <- HoltWinters(long_train_ts) # HoltWinters can't have NA values

tsdisplay(residuals(hw_model))

```

### residuals(hw\_model)



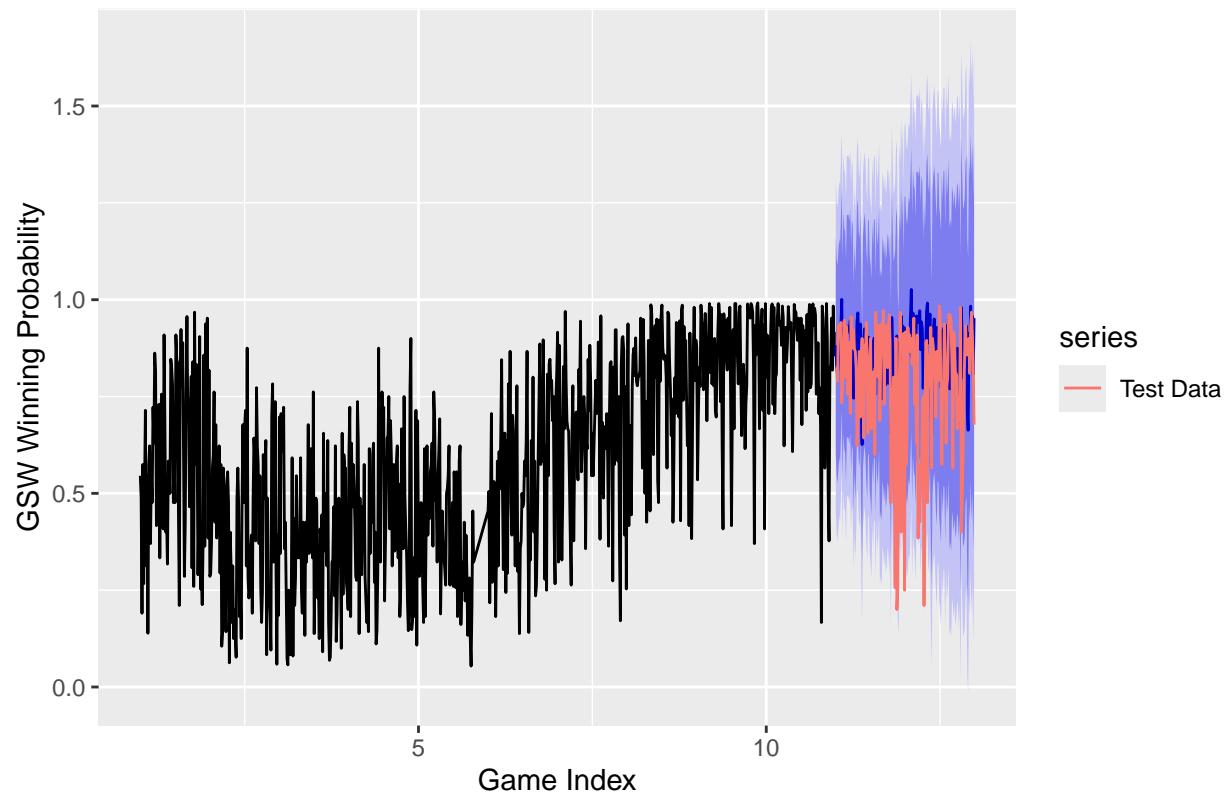
```

# Forecast Using Holt-Winters
# Set forecast horizon equal to the number of observations in the test set
forecast_horizon <- length(long_test_ts)
hw_forecast <- forecast(hw_model, h = forecast_horizon)

# Plot the Forecast vs. the Actual Test Data
autoplot(hw_forecast) +
  autolayer(long_test_ts, series = "Test Data") +
  ggtitle("Holt-Winters Forecast (Training: 2008-2017, Test: 2018-2019)") +
  xlab("Game Index") +
  ylab("GSW Winning Probability")

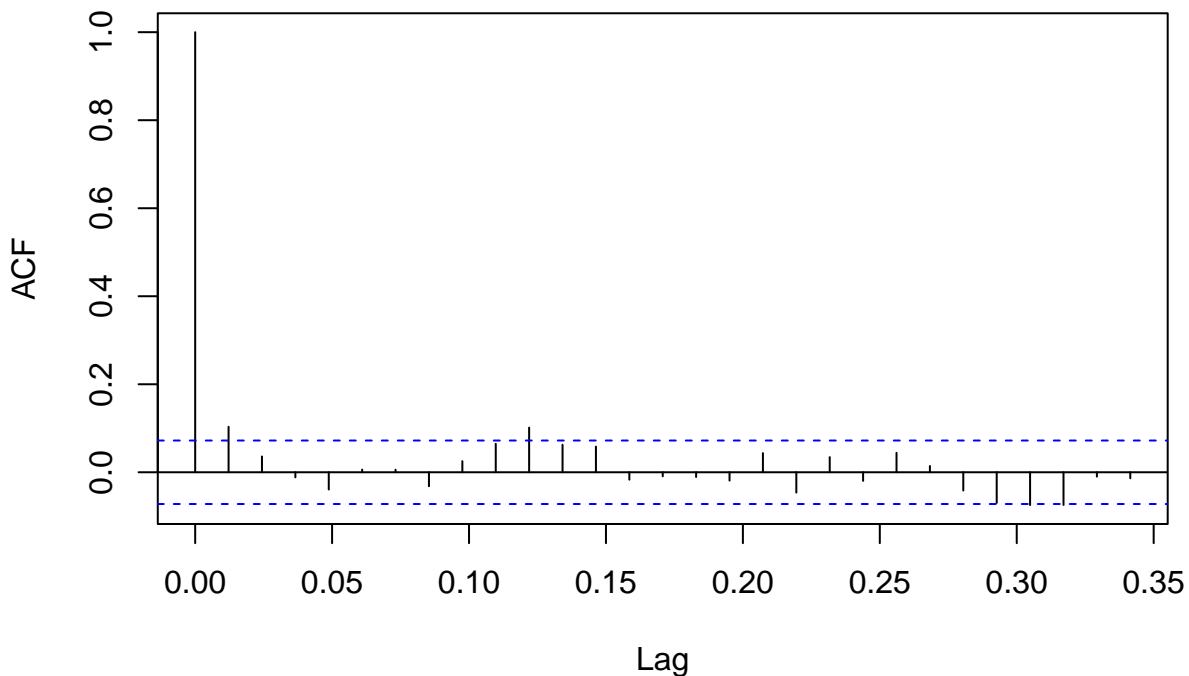
```

### Holt–Winters Forecast (Training: 2008–2017, Test: 2018–2019)



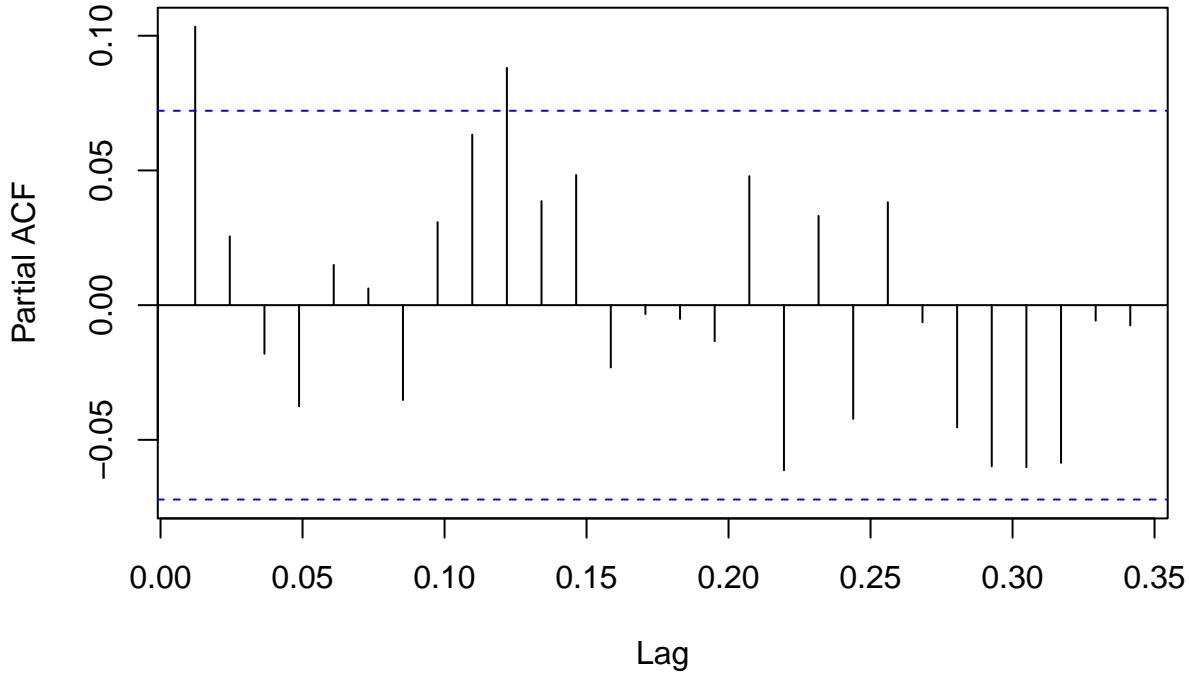
```
acf(hw_model$fitted[,1] - long_train_ts)
```

**Series hw\_model\$fitted[, 1] – long\_train\_ts**



```
pacf(hw_model$fitted[, 1] ~ long_train_ts)
```

### Series hw\_model\$fitted[, 1] ~ long\_train\_ts



```
# Evaluate Forecast Accuracy
```

```
hw_accuracy <- accuracy(hw_forecast, long_test_ts)
print(hw_accuracy)
```

```
##               ME      RMSE      MAE      MPE      MAPE      MASE
## Training set 0.009417707 0.2186848 0.1668483 -17.51961 45.45134 0.7390142
## Test set     -0.079611958 0.2050808 0.1440108 -20.15747 27.13473 0.6378613
##               ACF1 Theil's U
## Training set 0.1033790      NA
## Test set     0.1722173 0.7788986
```

## NNETAR model

Using our NNETAR model, we find that all our ACF and PACF of residuals are within the confidence bounds, indicating that NNETAR model did a fairly excellent job explaining the dynamics in our model. The RMSE is 0.1820251, MAE is 0.12503591, and MAPE is 24.61647.

```
nnetar_model <- nnetar(long_train_ts)
summary(nnetar_model)
```

```
##           Length Class      Mode
## x            820   ts     numeric
```

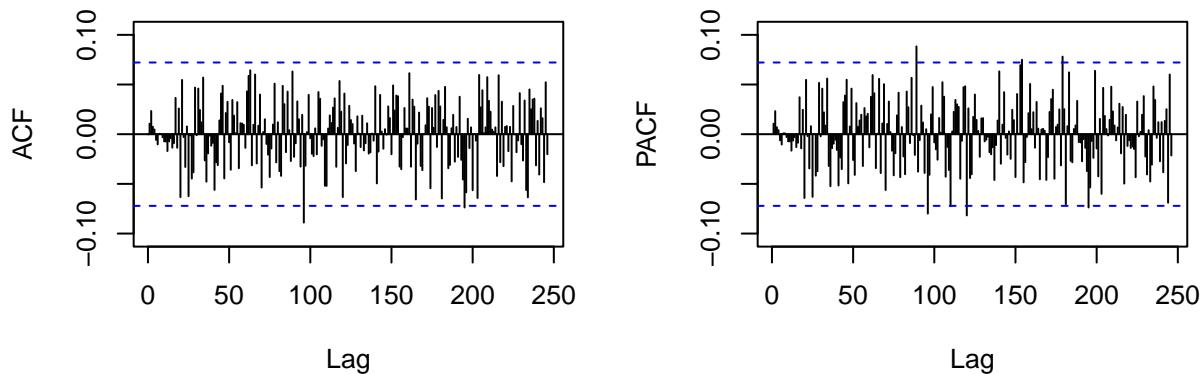
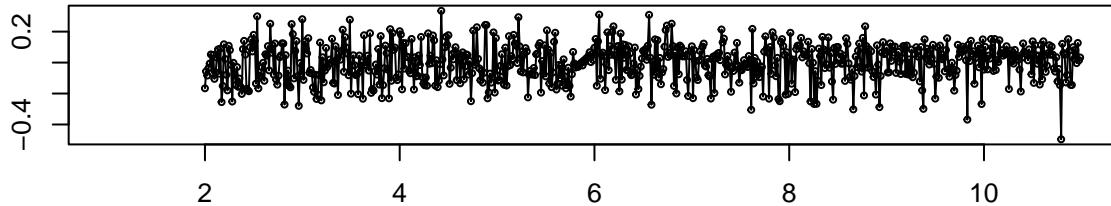
```

## m      1 -none-    numeric
## p      1 -none-    numeric
## P      1 -none-    numeric
## scalex 2 -none-    list
## size   1 -none-    numeric
## subset 820 -none-   numeric
## model  20 nnetarmodels list
## nnetargs 0 -none-    list
## fitted 820 ts       numeric
## residuals 820 ts       numeric
## lags    13 -none-    numeric
## series  1 -none-    character
## method  1 -none-    character
## call    2 -none-    call

```

```
tsdisplay(nnetar_model$residuals)
```

**nnetar\_model\$residuals**



```

forecast_horizon <- length(long_test_ts)
nnetar_forecast <- forecast(nnetar_model, h = forecast_horizon)

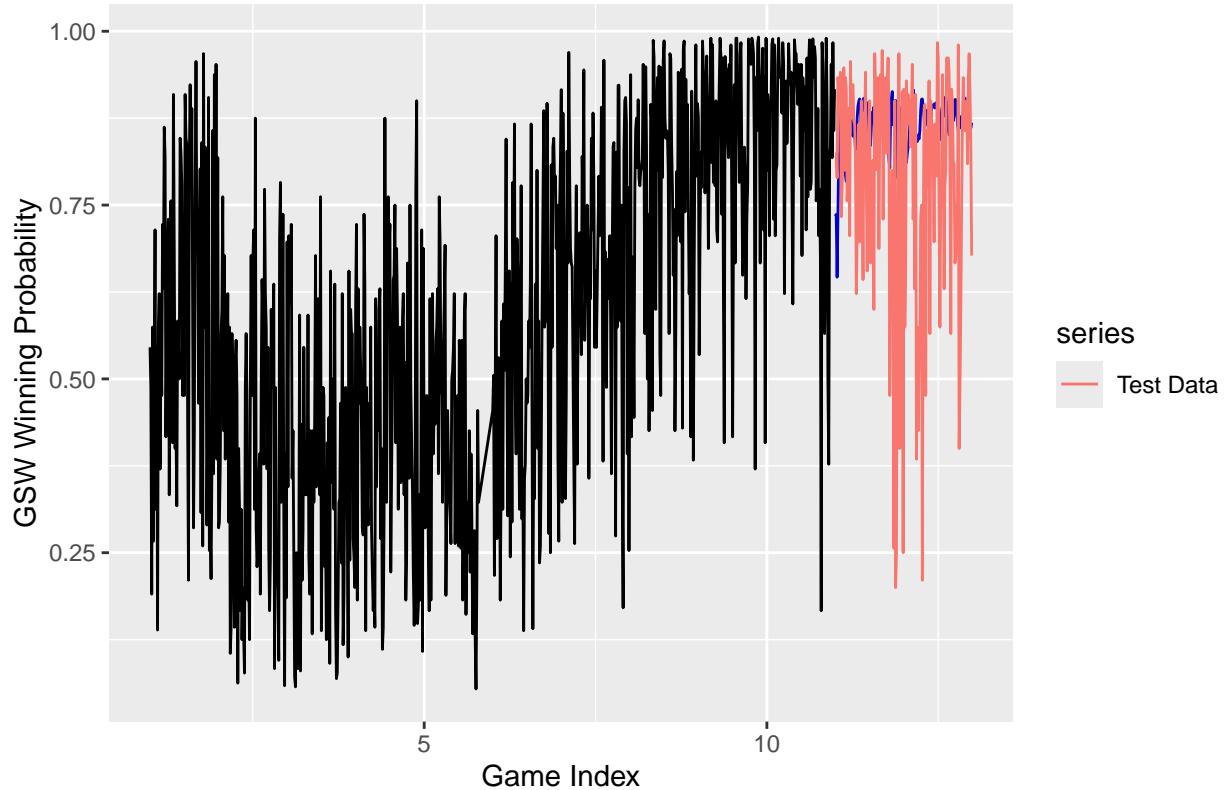
```

```

# Plot the Forecast vs. the Actual Test Data
autoplot(nnetar_forecast) +
  autolayer(long_test_ts, series = "Test Data") +
  ggtitle("NNETAR Forecast (Training: 2008-2017, Test: 2018-2019)") +
  xlab("Game Index") +
  ylab("GSW Winning Probability")

```

## NNETAR Forecast (Training: 2008–2017, Test: 2018–2019)



```
# Evaluate Forecast Accuracy
accuracy_metrics <- accuracy(nnetar_forecast, long_test_ts)
print(accuracy_metrics)
```

```
##               ME      RMSE      MAE      MPE      MAPE      MASE
## Training set 0.0001158611 0.1184579 0.09571236 -15.28347 28.57468 0.4239348
## Test set     -0.0829699425 0.1890938 0.13223894 -20.48310 25.82065 0.5857205
##             ACF1 Theil's U
## Training set 0.01052075       NA
## Test set     0.25488828 0.8230631
```

## Prophet Model

In our prophet model, we found that the residuals in ACF and PACF exhibit many significant lags, indicating that prophet may not be a good fit to explain our model. Furthermore, prophet model has the highest RMSE at 0.3100337, MAE at 0.2489537, and MAPE at 43.69948.

```
# Assume long_train_ts is your training time series (numeric vector)
# and long_test_ts is your test series (numeric vector)
# n_train and n_test represent their lengths respectively.
n_train <- length(long_train_ts)
n_test <- length(long_test_ts)

# Prepare Training Data Frame for Prophet
```

```

# Create a synthetic date sequence for the training period
start_date <- as.Date("2008-10-01") # Adjust as needed
train_dates <- seq.Date(from = start_date, by = "day", length.out = n_train)

# Build the training data frame for Prophet (ds = date, y = moneyline)
prophet_train <- data.frame(ds = train_dates, y = as.numeric(long_train_ts))

# Fit the Prophet Model
prophet_model <- prophet(prophet_train)

## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

# Create a Future Data Frame and Forecast
# This future data frame will contain dates for both training and test periods
future <- make_future_dataframe(prophet_model, periods = n_test)
prophet_forecast <- predict(prophet_model, future)

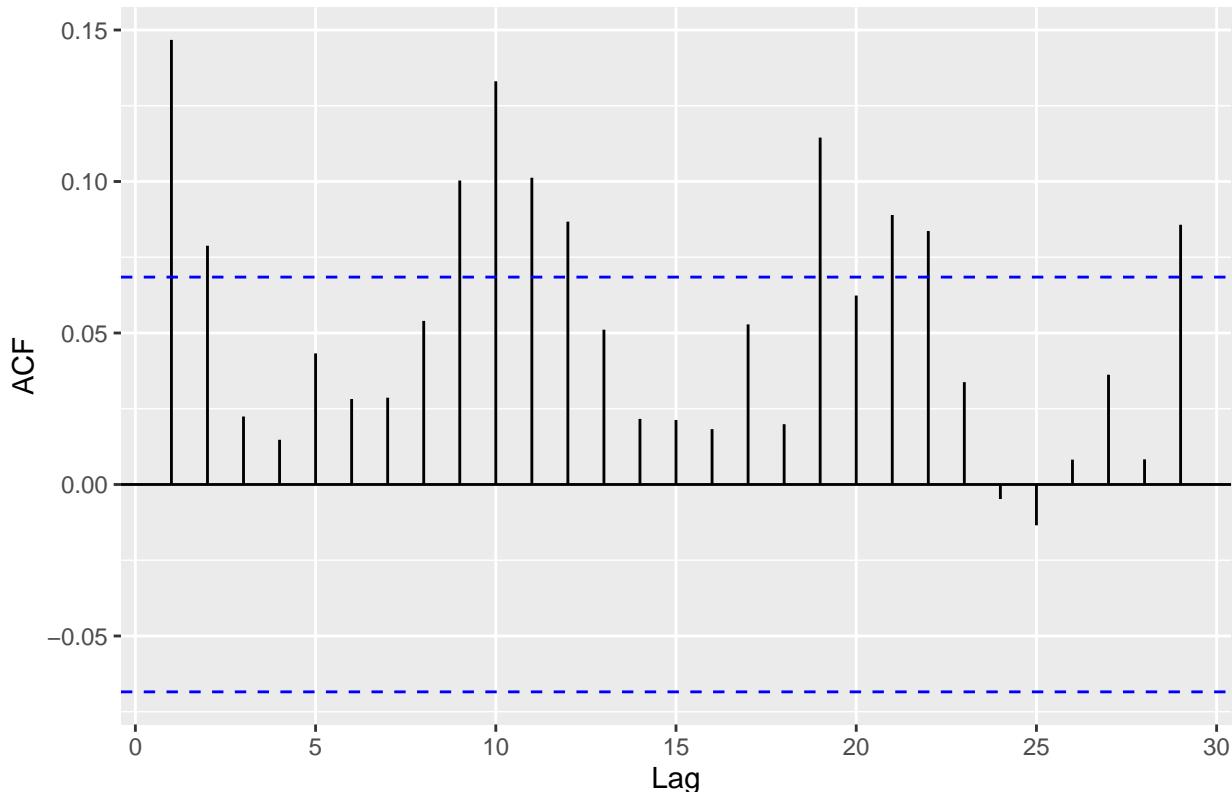
train_forecast <- prophet_forecast[1:n_train, ]

# Compute residuals: actual training values minus fitted values
residuals_prophet <- prophet_train$y - train_forecast$yhat

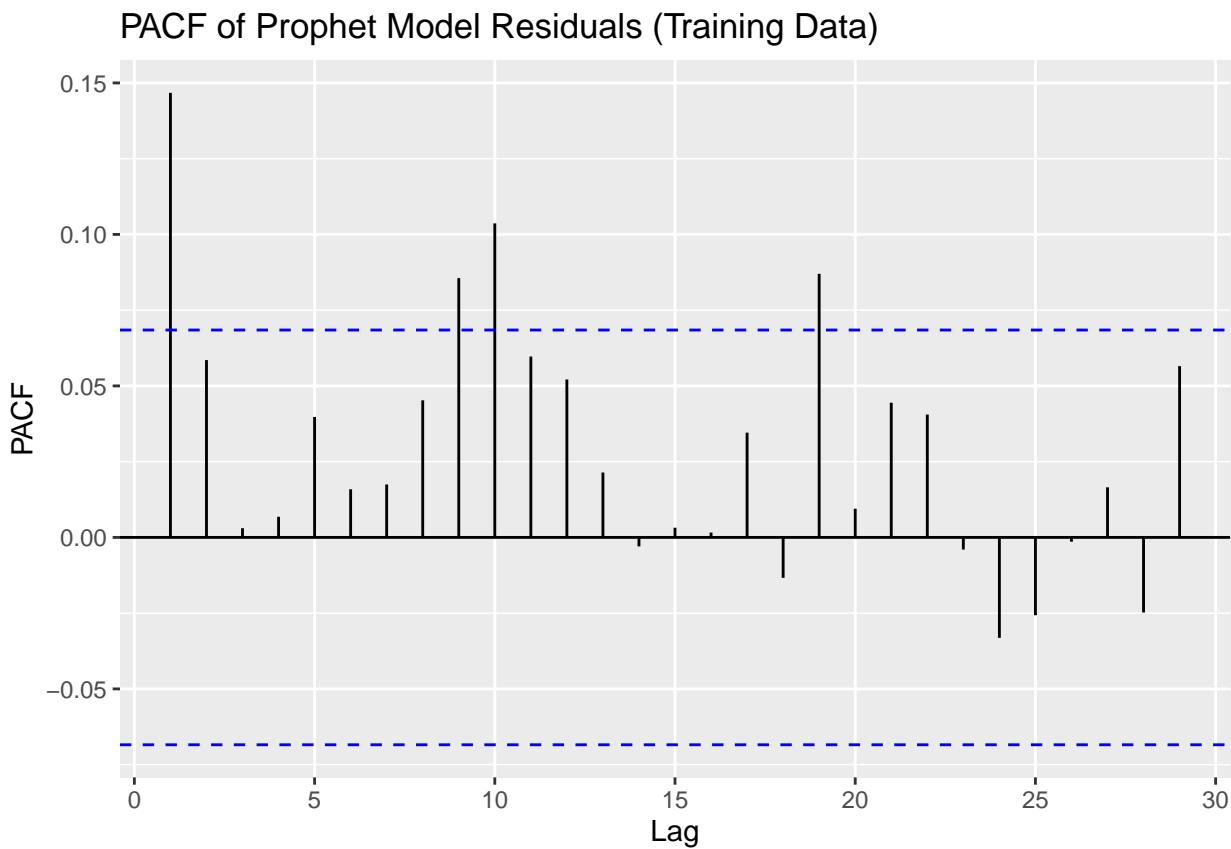
# Plot the ACF of the Residuals
ggAcf(residuals_prophet) +
  ggtitle("ACF of Prophet Model Residuals (Training Data)")

```

ACF of Prophet Model Residuals (Training Data)



```
# Plot the PACF of the Residuals
ggPacf(residuals_prophet) +
  ggtitle("PACF of Prophet Model Residuals (Training Data)")
```



```
# Extract Forecasts for the Test Period
# The forecast output contains predictions for the entire period.
# The test forecasts are the rows after the training period.
forecast_test <- prophet_forecast[(n_train + 1):(n_train + n_test), ]

# Compute Forecast Accuracy Metrics
# Actual test values from your series:
actual_test <- as.numeric(long_test_ts)
# Predicted values from Prophet:
predicted_test <- forecast_test$yhat

# Calculate error metrics:
rmse <- sqrt(mean((actual_test - predicted_test)^2))
mae <- mean(abs(actual_test - predicted_test))
mape <- mean(abs((actual_test - predicted_test) / actual_test)) * 100

cat("Forecast Accuracy Metrics for Prophet Model:\n")
```

```
## Forecast Accuracy Metrics for Prophet Model:
```

```

cat("RMSE:", rmse, "\n")

## RMSE: 0.3007236

cat("MAE :", mae, "\n")

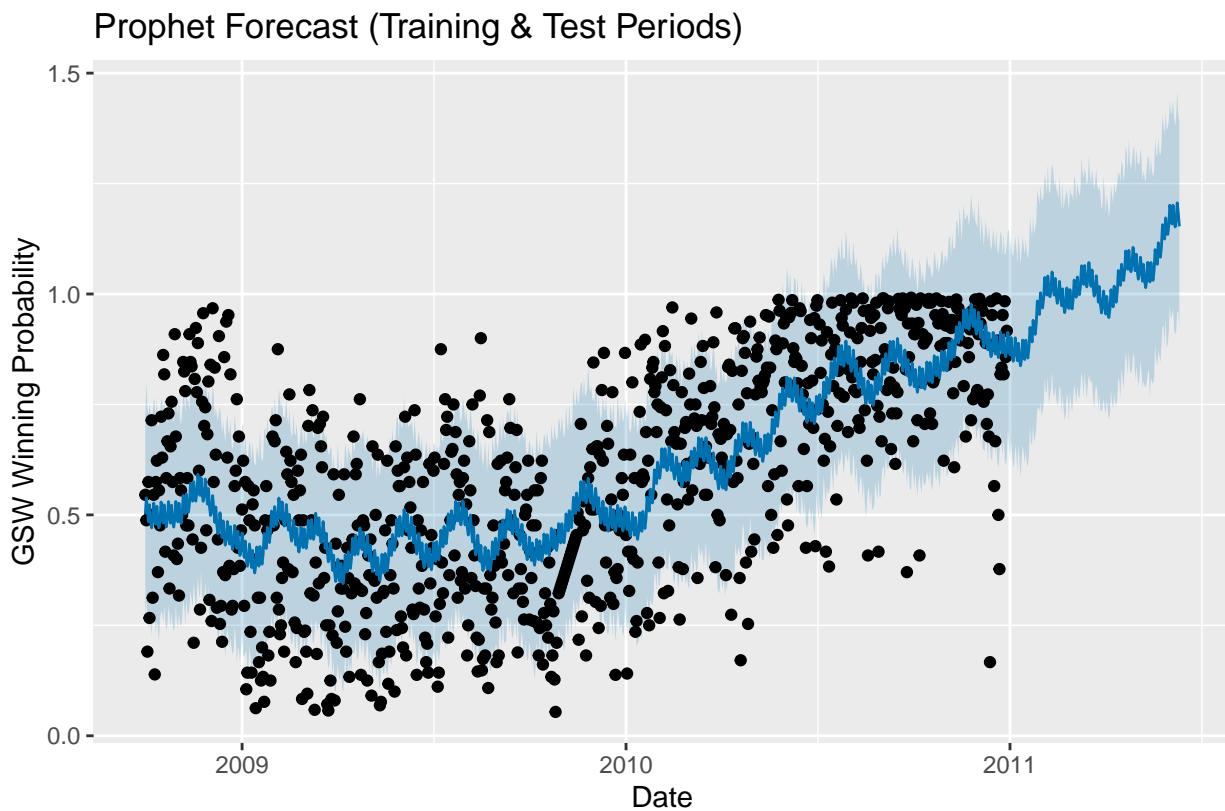
## MAE : 0.2394153

cat("MAPE:", mape, "\n")

## MAPE: 42.33196

# Plot the forecast
plot(prophet_model, prophet_forecast) +
  ggtitle("Prophet Forecast (Training & Test Periods)") +
  xlab("Date") + ylab("GSW Winning Probability")

```



## Combined Model

When looking at the three models I found ets, arima, and nnetar to provide the best forecasting using three metrics: RMSE, MAE, and MAPE. Thus, I combined the forecasts of the three models by averaging the means of each forecast. I found that the RMSE of the test set is 0.1731986, MAE is 0.1204005, and MAPE is 23.46671

```

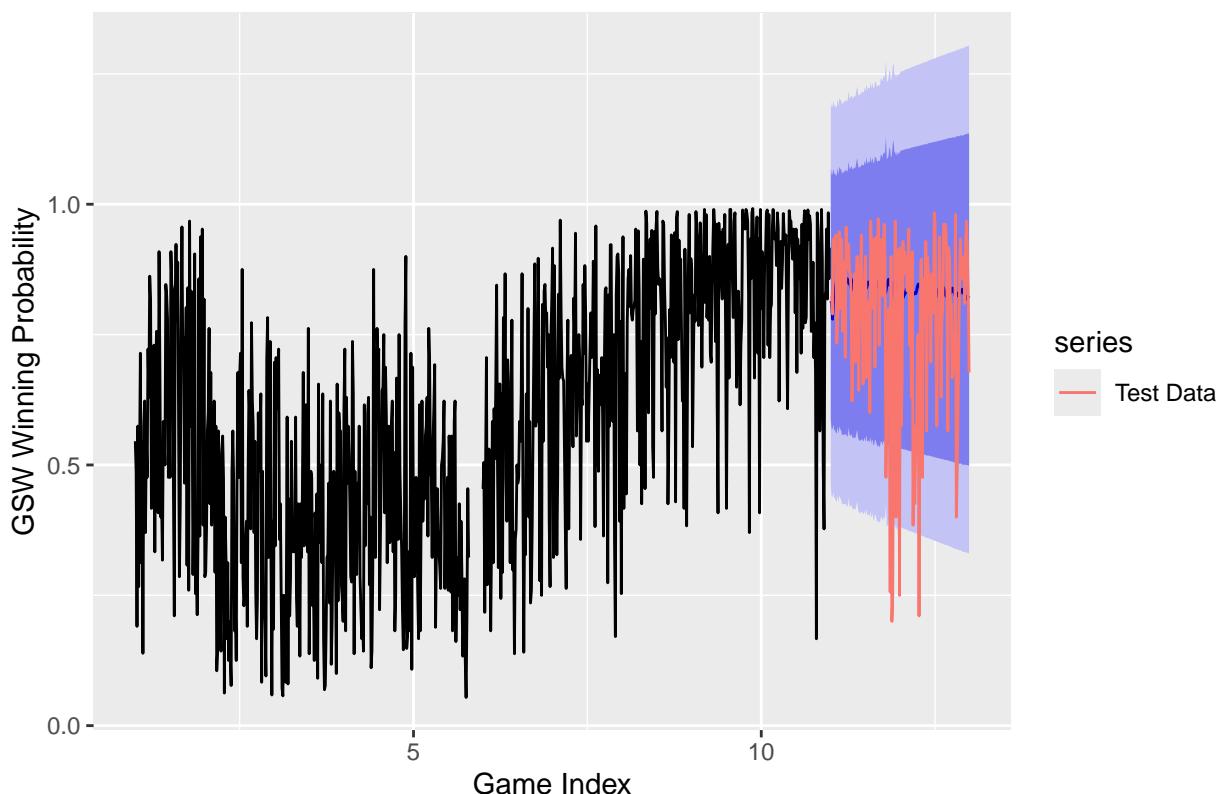
combined_forecast_mean <- (arima_forecast$mean + ets_forecast$mean + nnetar_forecast$mean) / 3

# Create a combined forecast object. Here we copy the structure of one of the forecast objects.
combined_forecast <- arima_forecast
combined_forecast$mean <- combined_forecast_mean

# Plot the Combined Forecast vs. the Actual Test Data
autoplot(combined_forecast) +
  autolayer(long_test_ts, series = "Test Data") +
  ggtitle("Combined Forecast: ARIMA + ETS + NNETAR") +
  xlab("Game Index") +
  ylab("GSW Winning Probability")

```

### Combined Forecast: ARIMA + ETS + NNETAR



```

# Evaluate the Accuracy of the Combined Forecast
accuracy_combined <- accuracy(combined_forecast, long_test_ts)
print(accuracy_combined)

```

	ME	RMSE	MAE	MPE	MAPE	MASE
## Training set	0.005767185	0.1876361	0.1546183	-22.46196	44.16782	0.6880180
## Test set	-0.056040549	0.1747171	0.1215605	-16.65781	23.72235	0.5409181
## ACF1	Theil's U					
## Training set	0.0009848986		NA			
## Test set	0.2302477779	0.7853238				

## Conclusion and References

Among the five models I tested (ARIMA, ETS, Holt-Winters, NNETAR, and Prophet), ARIMA provided the lowest forecasting errors (RMSE, MAE, and MAPE). Even when I combined forecasts from multiple models, ARIMA alone still outperformed the combined approach. This finding is important because it shows that even in a seemingly complex and dynamic environment like sports betting, simpler models can sometimes yield superior forecasting performance. Moreover when converting raw moneyline odds into probabilities, it shows a clearer basis for evaluating betting value, potentially guiding better decision-making in sports betting contexts.

Some limitations are present in our ARIMA model. All models tested including ARIMA are univariate, meaning that they do not consider external variables. Every model considers only past market conditions, and doesn't really capture future changes such as rule changes, trades, player injuries, etc.

dataset link- <https://www.kaggle.com/datasets/christophertreasure/nba-odds-data/data>