Siege Time Series

Tre Hill

Data Cleaning

```
library(tidyverse)
library(dplyr)
library(lubridate)
library(fpp3)
library(fable)
library(forecast)
library(tseries)
library(ggplot2)
library(scales)
```

#Loading and cleaning data

```
#Player data
data<-read.csv("player data.csv")
data <- data |> map_df(rev) |> rename("Avg.Players"="Avg..Players", "%.Gain"="X..Gain")

#New and mid season update data
up<-read.csv("patch updates.csv")
up <- up |> map_df(rev) |> mutate(new.season = ifelse(Note == "Operation Release",1,0),
             mid.season = ifelse(Note == "Mid-Season Reinforcements",1,0),
             Date = sub("^[^-]*-", "", Date))

#Information on free week and weekend deals
weeks<-read.csv("free week.csv")
weeks<- weeks |> mutate(fw=if_else(Label=="Free Weekend",1,
                                   if_else(Label=="Free Week",2,0)),
                        Date = sub("^[^-]*-", "", Date)) |> select(-c("X30.Day.Peak"))

#Remove duplicate and missing from weeks
weeks<-weeks[-c(17,23),]

#Converting variables to proper type
data$Avg.Players<-as.numeric(gsub(',','',data$Avg.Players)) #Avg.Player -> numeric
data$Gain<-as.numeric(gsub(',','',data$Gain)) #Avg number Gained (first difference) -> numeric
data$`%.Gain`<-gsub(',','',data$`%.Gain`)
data$`%.Gain`<-as.numeric(gsub('%','',data$`%.Gain`))  #Percent Gained -> numeric

year_part <- as.integer(substr(data$Month, 1, 2)) + 2000  # e.g. "15" -> 2015
month_part <- substr(data$Month, 4, 6)  # e.g. "Dec"
```

```r
# Create a date string like "2015-Dec-01" and parse
date_str <- paste(year_part, month_part, "01", sep="-")
data$Month <- yearmonth(as.Date(date_str, format="%Y-%b-%d"))

up$Date<-as.Date(paste("01-", up$Date, sep = ""), format = "%d-%b-%y")
up$Date<-yearmonth(up$Date)

weeks$Date<-as.Date(paste("01-", weeks$Date, sep = ""), format = "%d-%b-%y")
weeks$Date<-yearmonth(weeks$Date)

# Join with corrected dates
data <- data %>%
  left_join(up %>% select(Date, new.season, mid.season), by = c("Month" = "Date")) %>%
  left_join(weeks %>% select(Date, fw), by = c("Month" = "Date")) %>%
  mutate(
    new.season = ifelse(is.na(new.season), 0, new.season),
    mid.season = ifelse(is.na(mid.season), 0, mid.season),
    fw = ifelse(is.na(fw), 0, fw)
  ) %>%
  group_by(Month) %>%
  filter(
    if(any(new.season == 1)) new.season == 1 else row_number() == 1
  ) %>%
  filter(
    if(any(mid.season == 1)) mid.season == 1 else row_number() == 1
  ) %>%
  ungroup()
```

#Creating tsibble

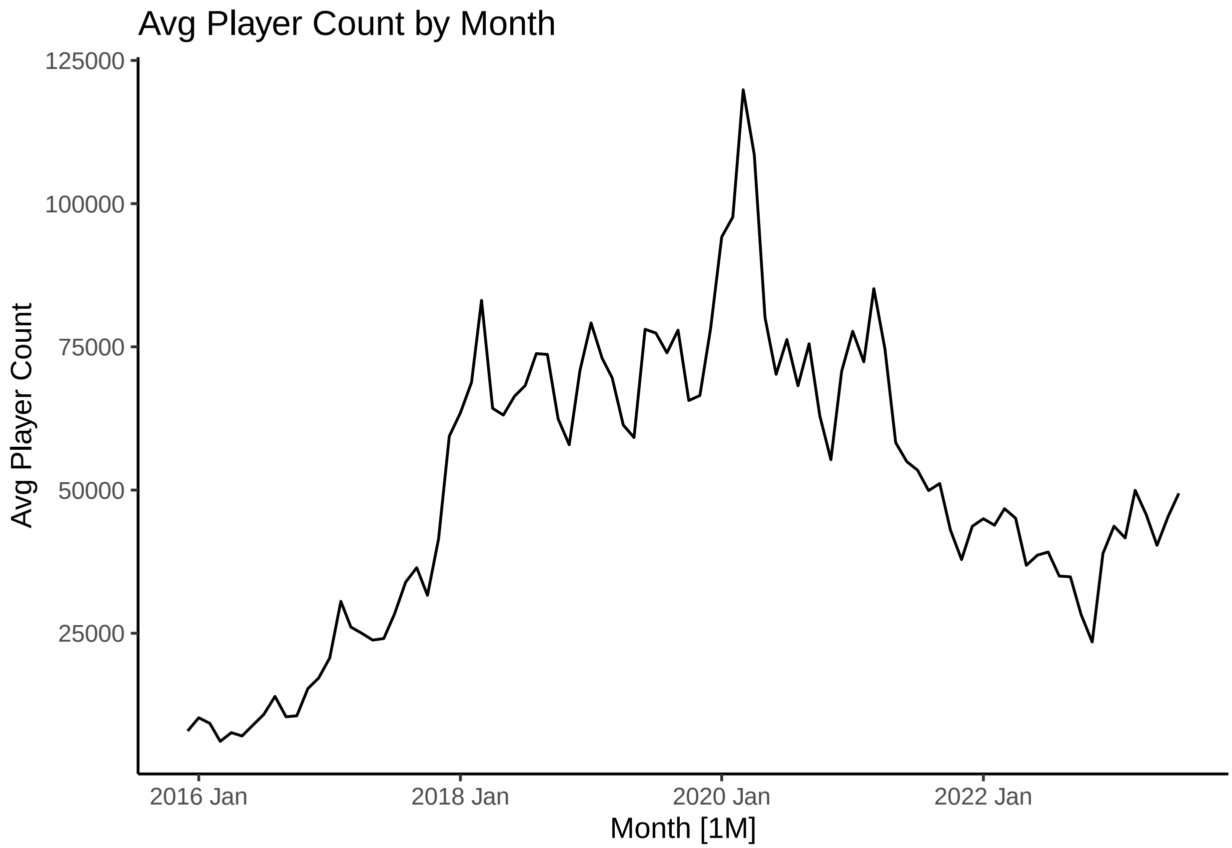
```r
ts<-as_tsibble(data, index=Month)
ts <- ts |>
  mutate(
    covid = ifelse(
      between(Month, as.Date("2020-03-01"), as.Date("2023-05-31")),
      1,
      0
    )
  )
```

#Train/val/test split

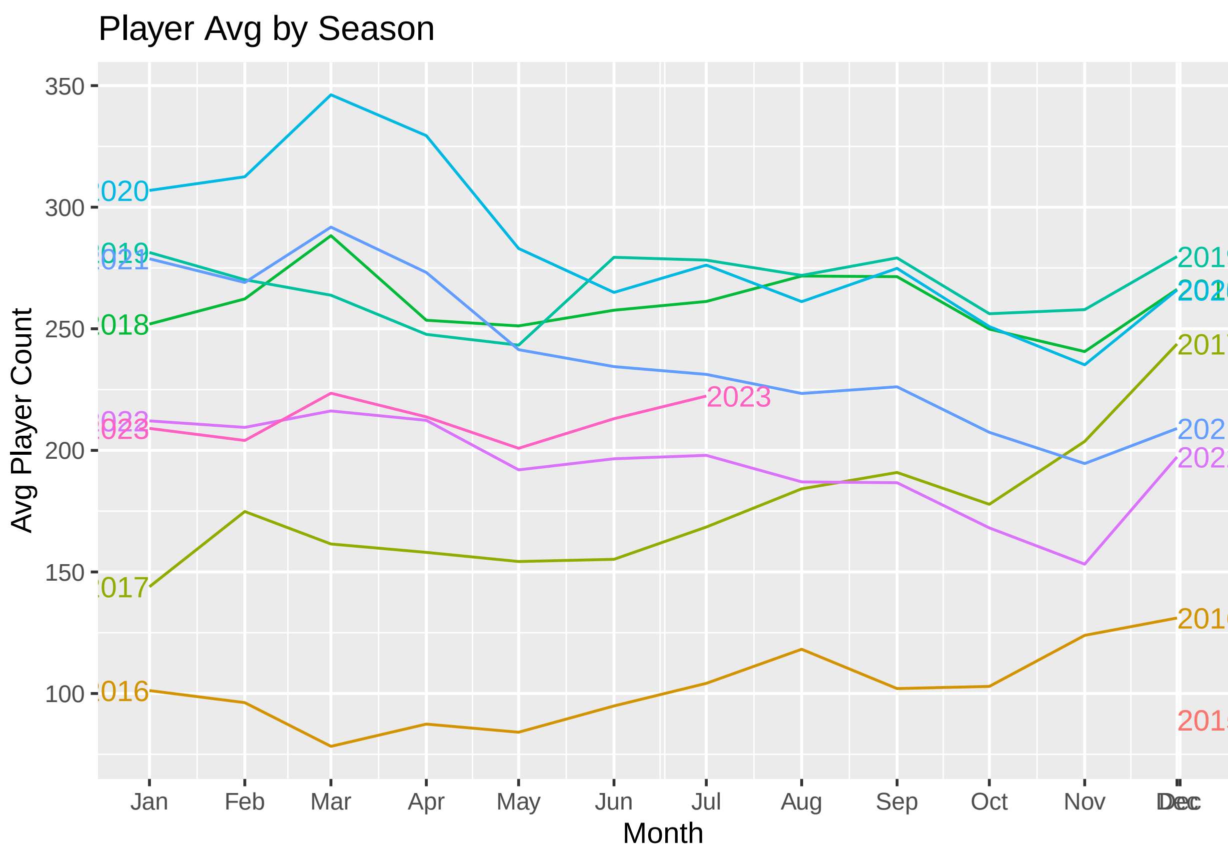
```r
train<-ts[c(1:92),]
val<- ts[c(93:104),]
test<- ts[-c(1:104),]
```

##EDA

```r
#Time plot of training data
autoplot(train,Avg.Players) + labs(title= "Avg Player Count by Month",
                                   y= "Avg Player Count") + theme_classic()
```



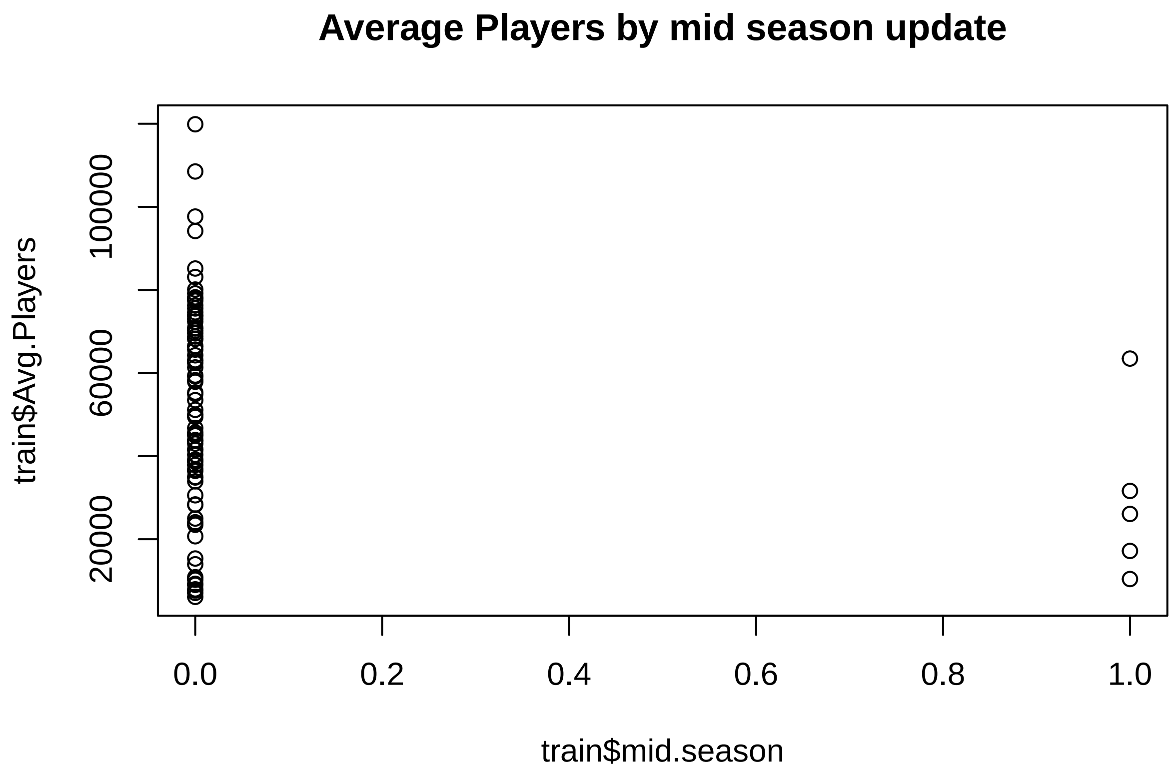

```r
#Season plot
train |> gg_season(sqrt(Avg.Players), labels = "both") + labs(title = "Player Avg by Season",
                                                              y = "Avg Player Count")
```



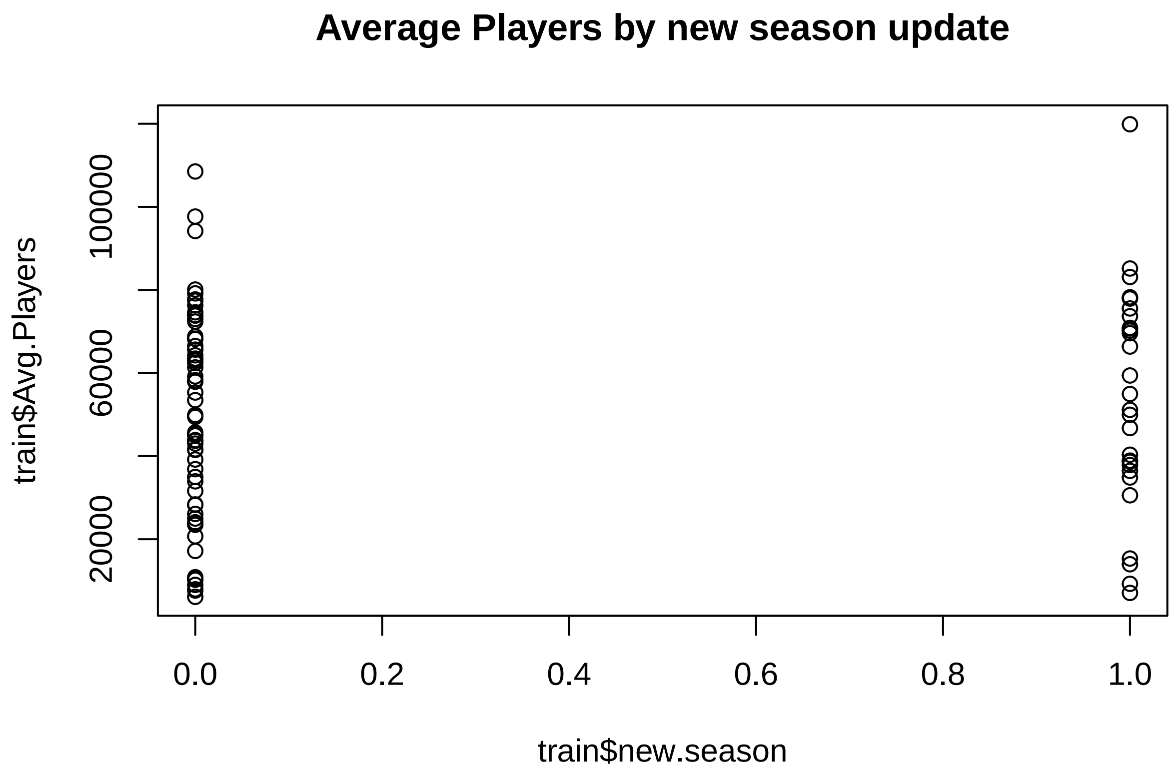
The data trends upward until COVID and then begins to drop.

Exploring possible predictors to add

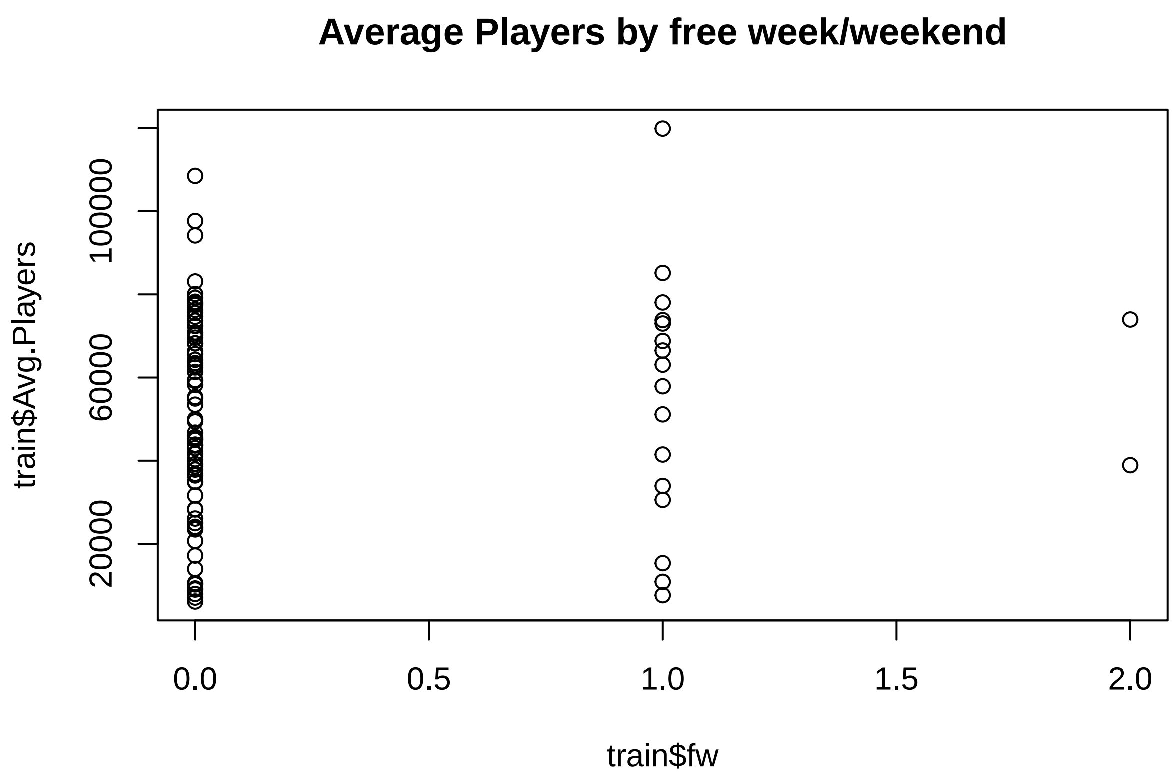
```r
plot(train$mid.season, train$Avg.Players, main = "Average Players by mid season update")
```



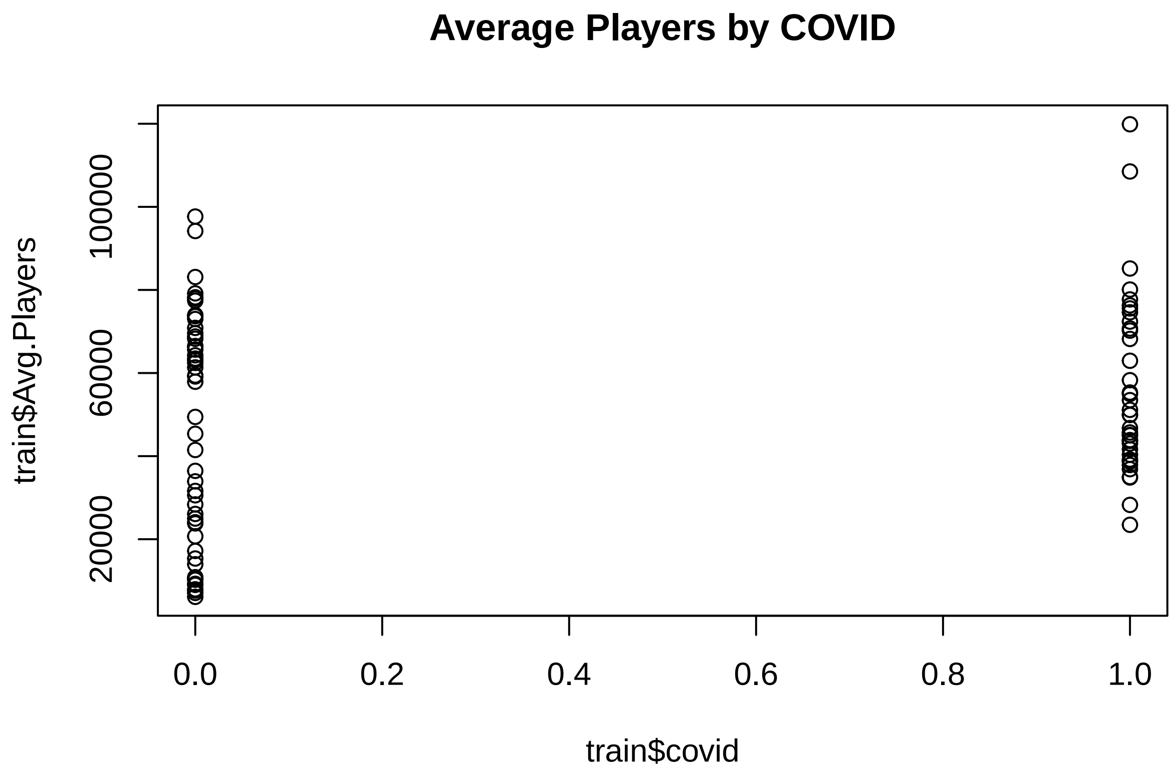

```r
plot(train$new.season, train$Avg.Players, main = "Average Players by new season update")
```




```
plot(train$fw, train$Avg.Players, main = "Average Players by free week/weekend")
```




```r
plot(train$covid, train$Avg.Players, main = "Average Players by COVID")
```

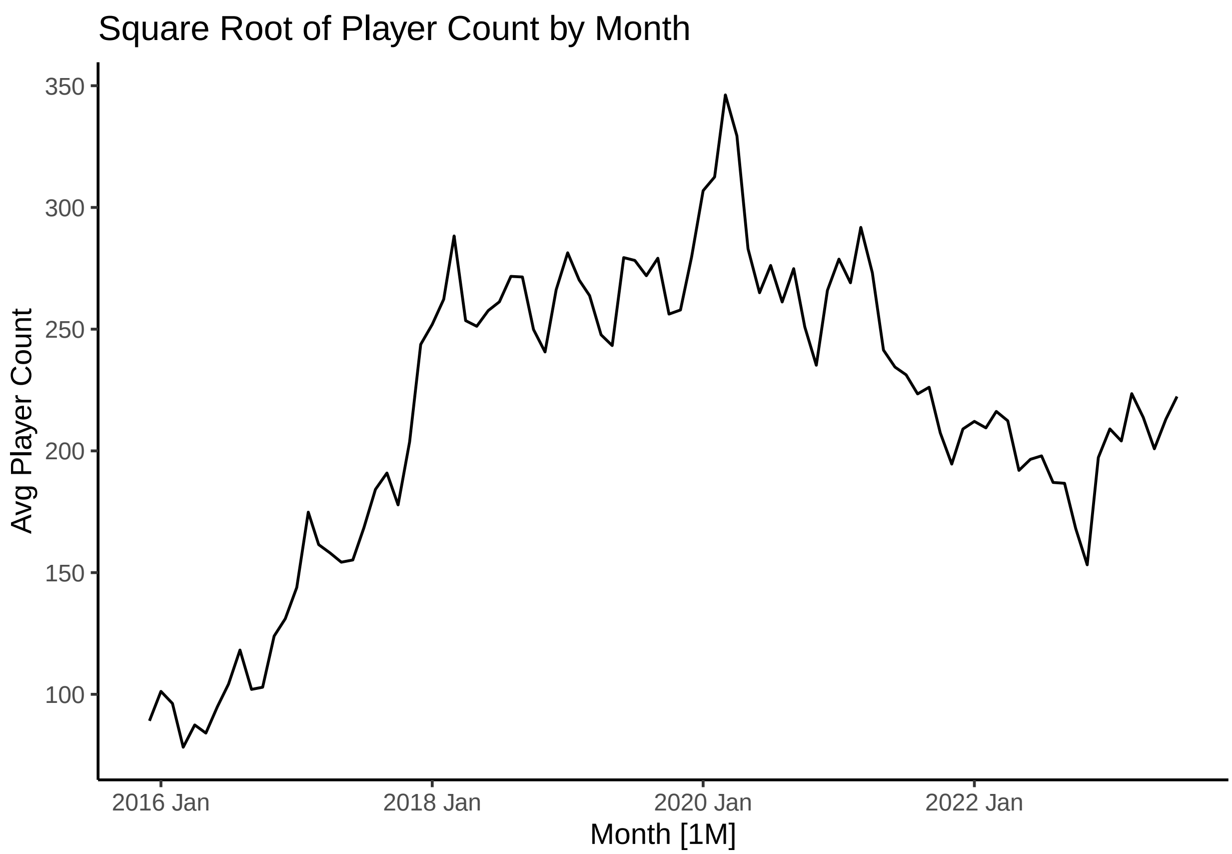


None of these have any separation. So, there is no point in adding these as predictors variables.

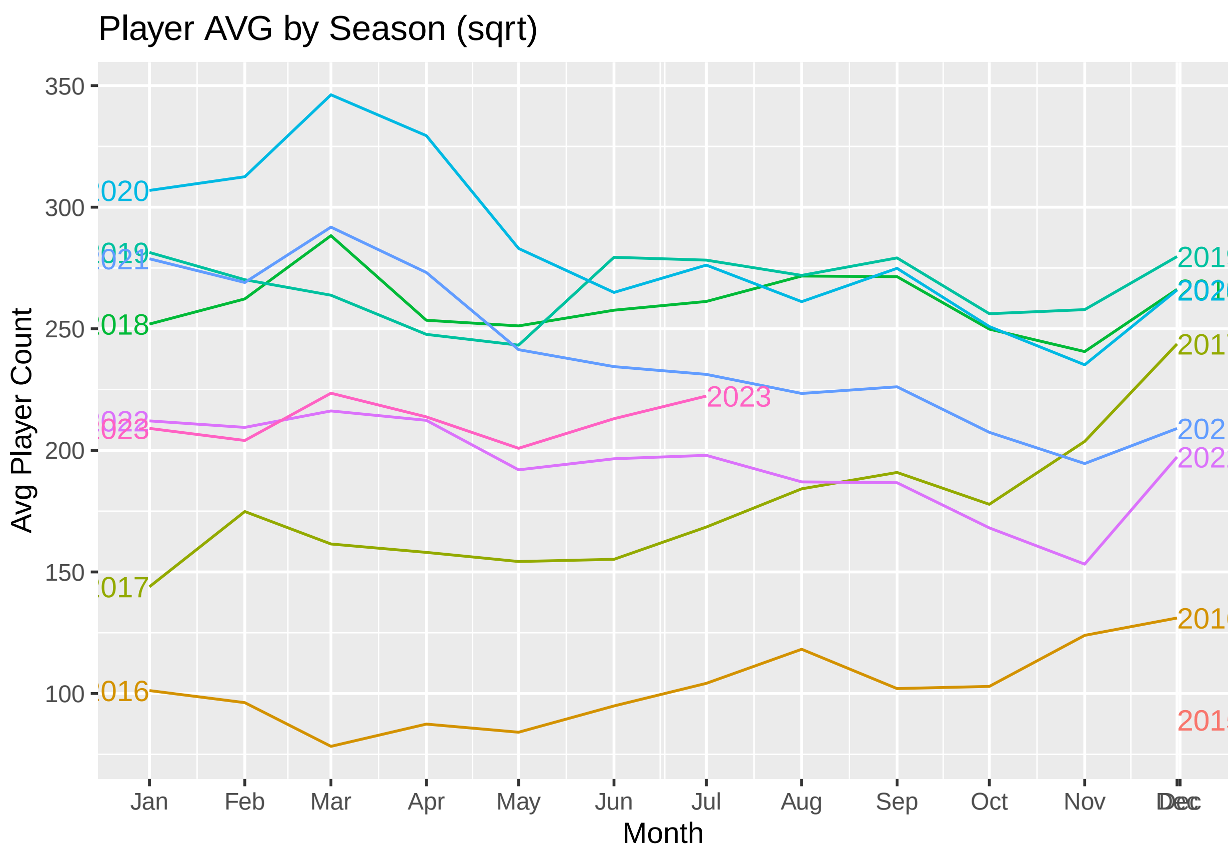
```
#Checking good box-cox
train |> features(Avg.Players, features = guerrero)
```

```
## # A tibble: 1 x 1
##   lambda_guerrero
##             <dbl>
## 1           0.383
```

```
autoplot(train,sqrt(Avg.Players)) + labs(title= "Square Root of Player Count by Month",
                                         y= "Avg Player Count") + theme_classic()
```




```r
#Seasonal plot
train |> gg_season(sqrt(Avg.Players), labels = "both") + labs(title = "Player AVG by Season (sqrt)",
                                                                y = "Avg Player Count")
```

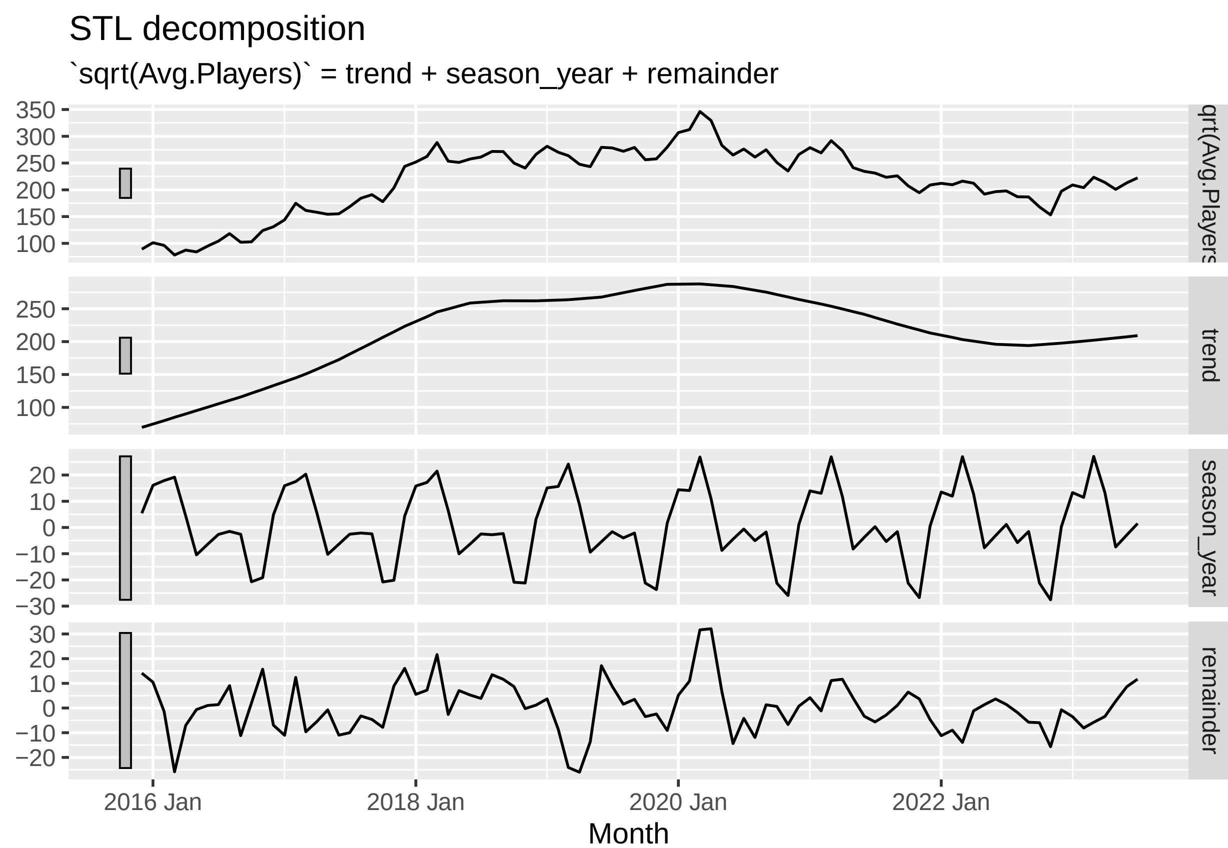


The variances are more stable when transformed. So, modeling will focus on transformed data.

```
#STL Decomposition to estimate seasonality
dcmp<- train |> model(stl=STL(sqrt(Avg.Players)))


train |> features(sqrt(Avg.Players), feat_stl)
```

```
## # A tibble: 1 x 9
##   trend_strength seasonal_strength_year seasonal_peak_year seasonal_trough_year
##            <dbl>                  <dbl>              <dbl>                <dbl>
## 1          0.972                  0.645                  4                    0
## # i 5 more variables: spikiness <dbl>, linearity <dbl>, curvature <dbl>,
## #   stl_e_acf1 <dbl>, stl_e_acf10 <dbl>
```

```
components(dcmp) |> autoplot()
```




```
dcmp[[1]][[1]][["fit"]][["seasons"]][["season_year"]][["period"]]
```

```
## [1] 12
```

#Baseline models The models, mean naive, seasonal naive and drift will be the baseline to see if something better can be found.

```
root.baseline <- train |>
  model(
    mean.model = MEAN((Avg.Players)^(1/2)), # Forecasts mean of entire training dataset
    naive = NAIVE((Avg.Players)^(1/2)), # Forecasts last observation into future predictions
    snaive = SNAIVE((Avg.Players)^(1/2)), # Forecasts last observation of each season into future
    drift = RW((Avg.Players)^(1/2) ~ drift())
  )

#MAPE value for square root transformations
root.base.fore<- root.baseline |> fabletools::forecast(h = 12)
fabletools::accuracy(root.base.fore, val) |> arrange(MAPE)
```

```
## # A tibble: 4 x 10
##   .model     .type     ME   RMSE    MAE    MPE  MAPE  MASE RMSSE  ACF1
##   <chr>      <chr>  <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 drift      Test   3914. 18374. 11583. -0.434  17.4   NaN   NaN 0.457
## 2 naive      Test   8444. 20334. 12506.  7.33   17.6   NaN   NaN 0.497
```

```
## 3 mean.model Test   9184. 20937. 12957.  8.49   18.2   NaN   NaN 0.515
## 4 snaive     Test  17051. 23025. 17106. 25.1    25.2   NaN   NaN 0.447
```

Drift model performed the best. Other models will be compared to the drift model as a baseline.

##Exponential Smoothing Models

```
#Square root models
root.fit <- train |>
  model(
    SES = ETS((Avg.Players)^(1/2) ~ error("A") + trend("N") + season("N")),
    `Linear` = ETS(sqrt(Avg.Players)^(1/2) ~ error("A") + trend("A") + season("N")),
    `Damped Linear` = ETS(sqrt(Avg.Players)^(1/2) ~ error("A") + trend("Ad") +
                        season("N")),
    `Damp Mult` = ETS(sqrt(Avg.Players)^(1/2) ~ error("M") + trend("Ad") +
                    season("M")),
    HWAdd = ETS((Avg.Players)^(1/2) ~ error("A") + trend("A") + season("A")),
    HWMult = ETS((Avg.Players)^(1/2) ~ error("M") + trend("A") + season("M")),
    algo = ETS((Avg.Players)^(1/2)) # "algo" uses automated procedure to determine best model
  )
report(root.fit) |> arrange(AICc)
```

```
## # A tibble: 7 x 9
##   .model           sigma2 log_lik   AIC  AICc   BIC    MSE   AMSE    MAE
##   <chr>             <dbl>   <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>  <dbl>
## 1 Damp Mult       0.00129   -138.  311.  321.  357.   0.186  0.322 0.0241
## 2 Linear          0.348     -157.  325.  325.  337.   0.332  0.665 0.469
## 3 Damped Linear   0.345     -157.  325.  326.  340.   0.327  0.635 0.465
## 4 HWAdd         194.        -442.  917.  925.  960. 161.   297.    9.80
## 5 algo          194.        -442.  917.  925.  960. 161.   297.    9.80
## 6 HWMult          0.00543   -449.  932.  940.  975. 156.   293.    0.0490
## 7 SES           296.        -469.  944.  944.  951. 290.   592.   13.6
```

Checking MAPE of the best ESM model (Damped Multiplicative)

```
root.fore <- root.fit |> select(`Damp Mult`) |> fabletools::forecast(h = 12)
fabletools::accuracy(root.fore, val) |> select(.model, MAPE)
```

```
## # A tibble: 1 x 2
##   .model     MAPE
##   <chr>     <dbl>
## 1 Damp Mult  19.3
```

##ARIMA

#ARIMA EDA

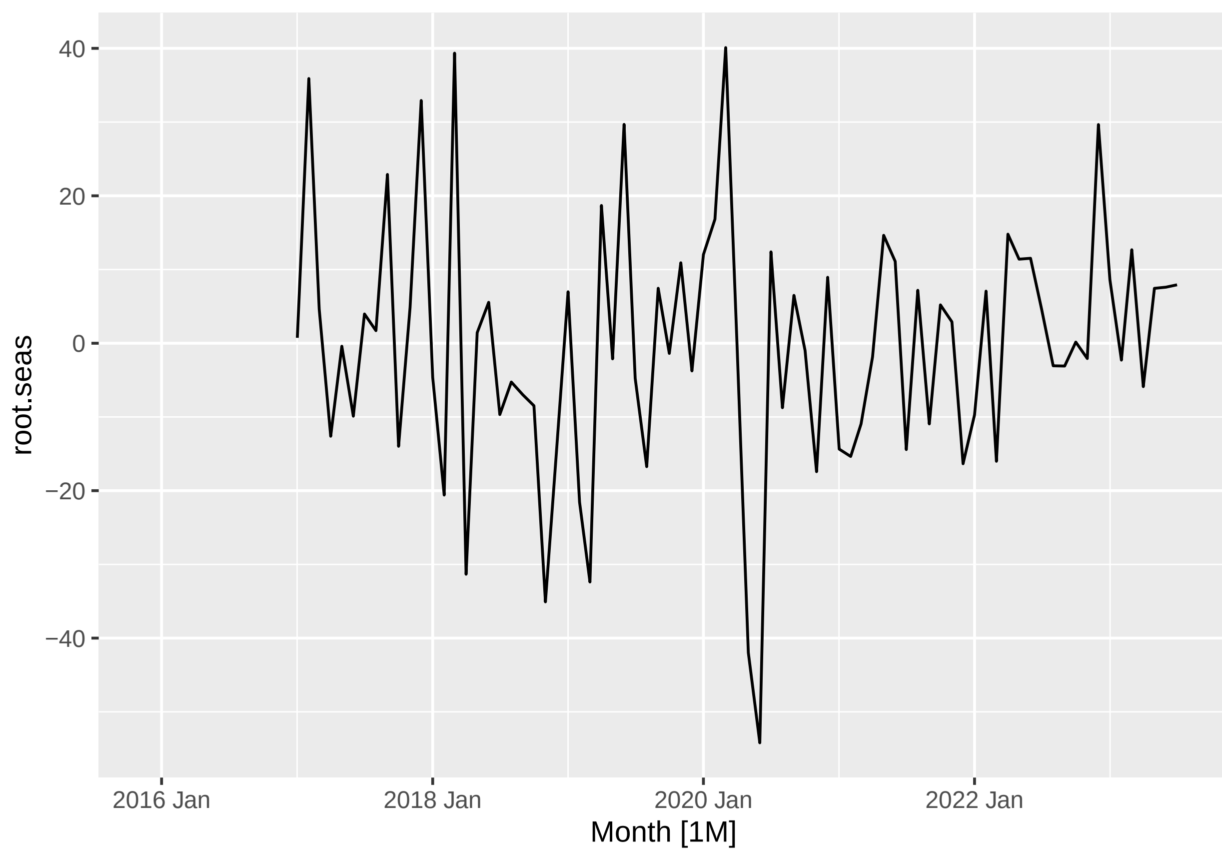
```
#Checking for ARIMA Seasonality
train |>
features(sqrt(Avg.Players), unitroot_nsdiffs) #1 seasonal difference needed
```

```
## # A tibble: 1 x 1
##   nsdiffs
##     <int>
## 1       1
```

```r
train %>%
  mutate(root.seas = difference(sqrt(Avg.Players), lag = 12)) %>%
  features(root.seas, unitroot_ndiffs) #1 regular difference needed
```

```
## # A tibble: 1 x 1
##   ndiffs
##    <int>
## 1      1
```

```r
train %>%
  mutate(root.seas = difference(difference(sqrt(Avg.Players), lag = 12)),1) %>% autoplot(root.seas)
```



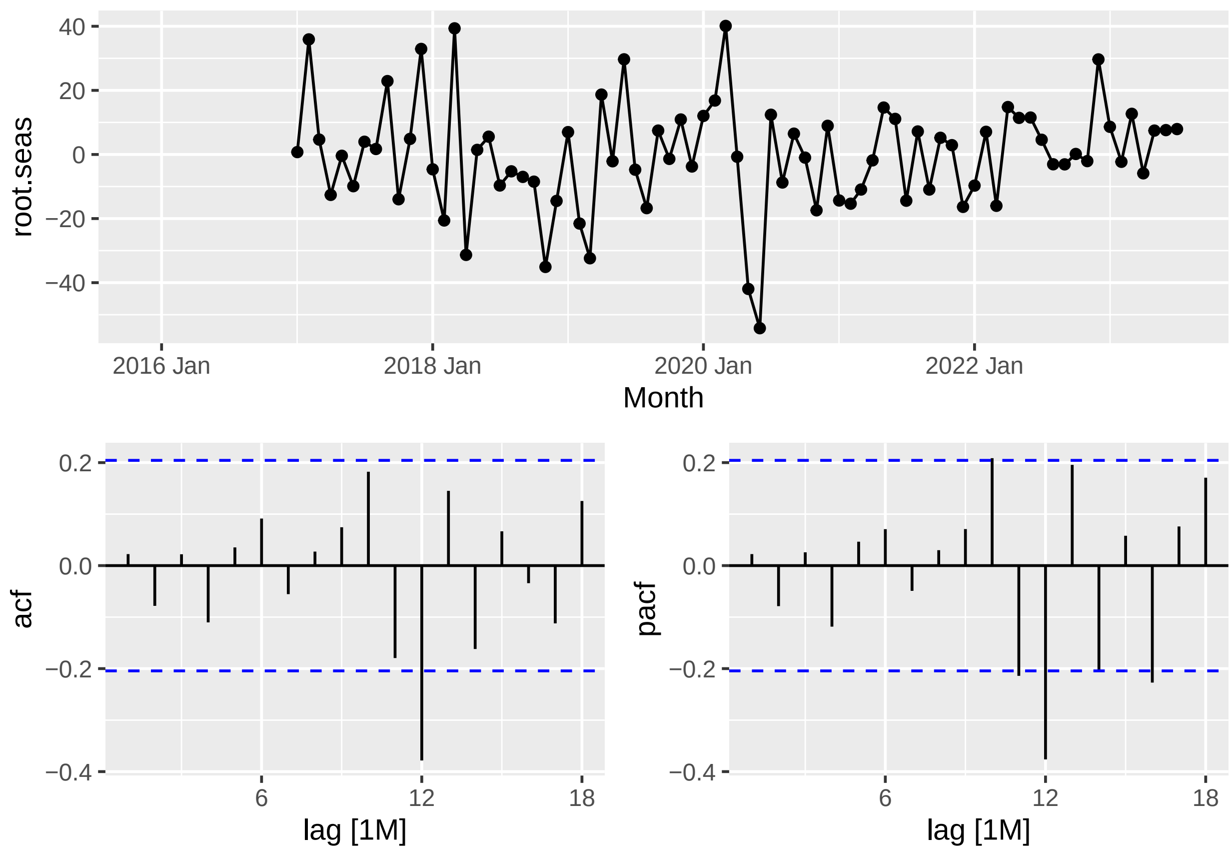

```r
#Checking if transformed data is stationary; alpha = 0.05
root_diff <- diff(diff(train$Avg.Players^(1/2), lag = 12), lag = 1)
adf.test(root_diff) #Ha: Stationary; Reject Ho-> Stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  root_diff
## Dickey-Fuller = -3.9576, Lag order = 4, p-value = 0.01599
## alternative hypothesis: stationary
```

```r
kpss.test(root_diff, null = "Level") #Ha: Non-stationary; FTR Ho-> Stationary
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  root_diff
## KPSS Level = 0.12652, Truncation lag parameter = 3, p-value = 0.1
```

```r
train %>%
  mutate(root.seas = difference(difference(sqrt(Avg.Players), lag = 12)),1) %>% gg_tsdisplay(root.seas,
```



#ARIMA modeling

Fitting 2 stochastic models: 1. with an automated procedure, and another based on the significant spikes from the ACF and pACF plots

Also exploring deterministic models with Fourier terms

```r
root.arima<-train |> model(  stoch=ARIMA(sqrt(Avg.Players)),
                             arma11=ARIMA(sqrt(Avg.Players)~pdq(0,1,0)+PDQ(1,1,1)),
                             `K = 1` = ARIMA(Avg.Players^(1/2) ~ fourier(K = 1) + PDQ(0,0,0)),
                             `K = 2` = ARIMA(Avg.Players^(1/2) ~ fourier(K = 2) + PDQ(0,0,0)),
                             `K = 3` = ARIMA(Avg.Players^(1/2) ~ fourier(K = 3) + PDQ(0,0,0)),
                             `K = 4` = ARIMA(Avg.Players^(1/2) ~ fourier(K = 4) + PDQ(0,0,0)),
                             `K = 5` = ARIMA(Avg.Players^(1/2) ~ fourier(K = 5) + PDQ(0,0,0)),
                             `K = 6` = ARIMA(Avg.Players^(1/2) ~ fourier(K = 6) + PDQ(0,0,0)))
root.arima |> select(stoch)
```

```
## # A mable: 1 x 1
##                        stoch
##                      <model>
## 1 <ARIMA(0,1,0)(0,1,1)[12]>
```

```
report(root.arima) |> arrange(AICc)
```

```
## # A tibble: 8 x 8
##   .model sigma2 log_lik   AIC  AICc   BIC ar_roots   ma_roots
##   <chr>   <dbl>   <dbl> <dbl> <dbl> <dbl> <list>     <list>
## 1 stoch    219.   -327.  657.  658.  662. <cpl [0]>  <cpl [12]>
## 2 arma11   221.   -327.  659.  660.  667. <cpl [12]> <cpl [12]>
## 3 K = 2    160.   -357.  732.  734.  755. <cpl [2]>  <cpl [2]>
## 4 K = 4    175.   -360.  738.  740.  760. <cpl [0]>  <cpl [0]>
## 5 K = 5    179.   -360.  742.  745.  769. <cpl [0]>  <cpl [0]>
## 6 K = 6    180.   -360.  743.  747.  773. <cpl [0]>  <cpl [0]>
## 7 K = 3    215.   -370.  755.  756.  772. <cpl [0]>  <cpl [0]>
## 8 K = 1    242.   -376.  767.  768.  784. <cpl [0]>  <cpl [4]>
```

Testing best stochastic and deterministic ARIMAs against validation

```
root.arima.fore<- root.arima |> select(c(stoch,`K = 2`)) |> fabletools::forecast(h=12)
fabletools::accuracy(root.arima.fore, val) |> arrange(MAPE)
```

```
## # A tibble: 2 x 10
##   .model .type     ME   RMSE    MAE   MPE  MAPE  MASE RMSSE  ACF1
##   <chr>  <chr>  <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 stoch  Test  10679. 18335. 12440.  13.4  17.3   NaN   NaN 0.475
## 2 K = 2  Test  10936. 19242. 13082.  13.4  18.2   NaN   NaN 0.495
```

##Final Test

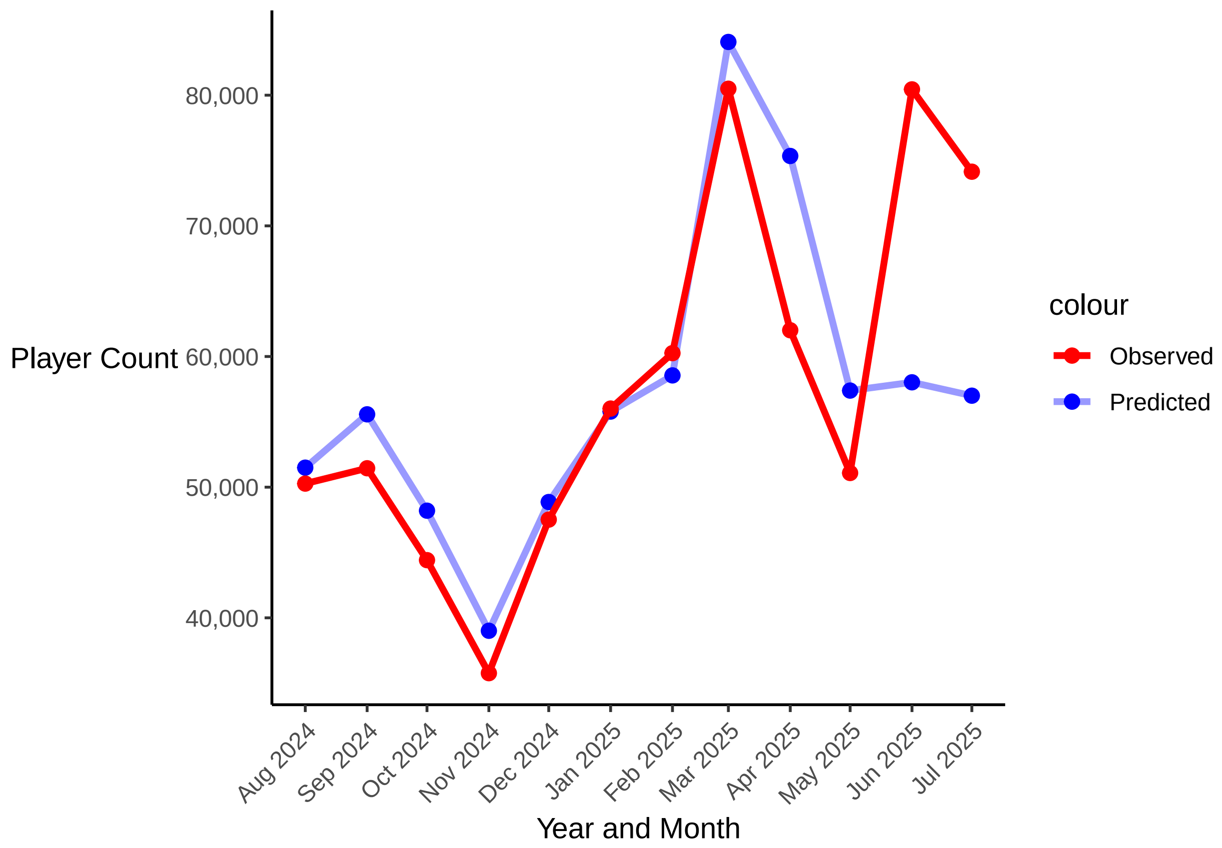
```
tune.train<-ts[c(1:104),]

best.ar<- tune.train |> model(ARIMA(sqrt(Avg.Players)~pdq(0,1,0)+PDQ(0,1,1)))

best.fore<-best.ar |> fabletools::forecast(h = 12)
fabletools::accuracy(best.fore,test)
```

```
## # A tibble: 1 x 10
##   .model                        .type    ME  RMSE   MAE   MPE  MAPE  MASE RMSSE  ACF1
##   <chr>                         <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 ARIMA(sqrt(Avg.Players)~ Test   379. 9466. 6538. -1.25  10.3   NaN   NaN 0.382
```

Graph of the forecasted values against test data

```
graph.data<- data.frame(Month=as.Date(best.fore$Month),
                        predicted=best.fore$.mean,
                        observed=test$Avg.Players)
ggplot(data = graph.data, aes(x = Month)) +
  geom_line(aes(y = predicted, color = "Predicted"), linewidth = 1.2,alpha =.4) +
```

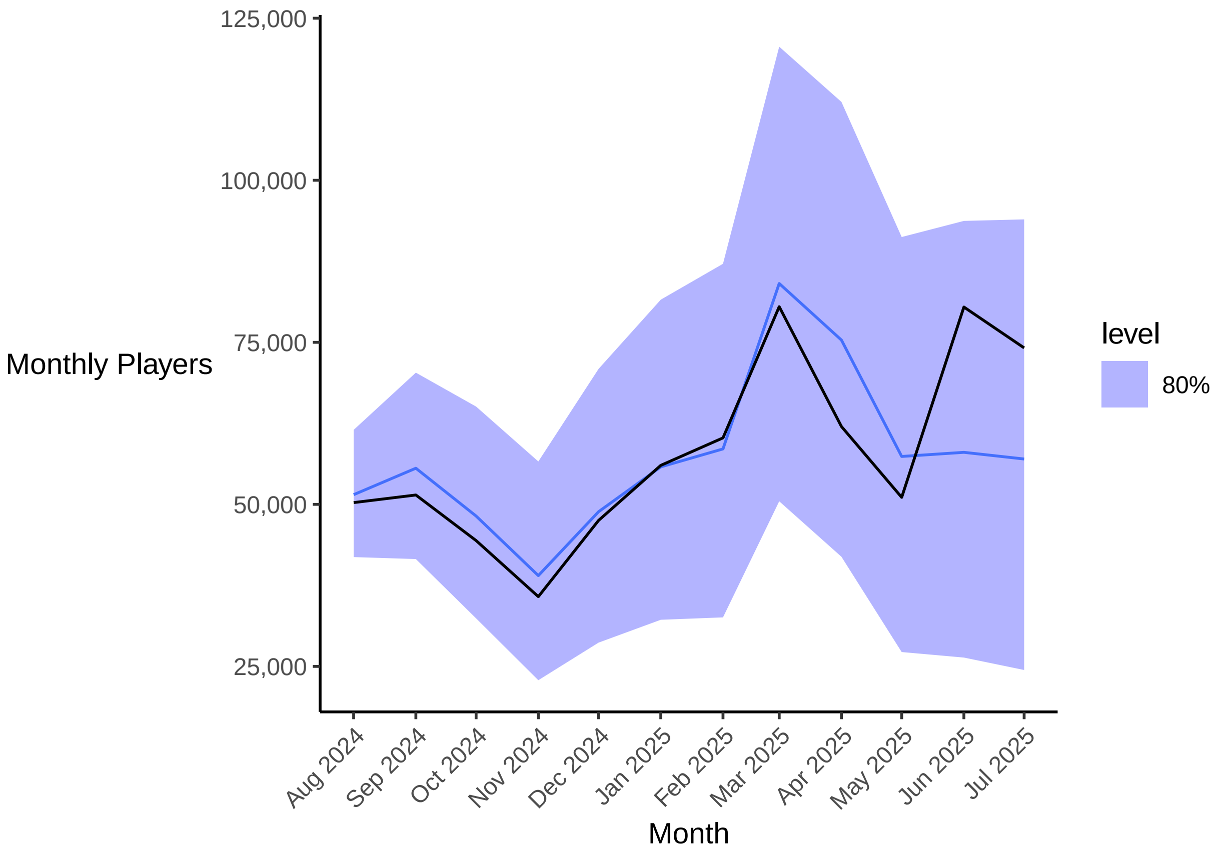
```r
  geom_point(aes(y = predicted,color = "Predicted"),size = 2.2) +
  geom_line(aes(y = observed, color = "Observed"), linewidth = 1.2) +
  geom_point(aes(y = observed,color = "Observed"),size = 2.2) +
  scale_color_manual(values = c("Predicted" = "blue", "Observed" = "red")) +
  scale_y_continuous(breaks = c(40000, 50000, 60000, 70000, 80000),
                     labels = c("40,000", "50,000", "60,000", "70,000", "80,000"))+labs(x = "Year and Mo
  theme(axis.title.y = element_text(angle = 0,vjust = .5),
        axis.text.x = element_text(angle= 45,hjust= 1,vjust= 1,
                                   margin= margin(t= 3))) +
  scale_x_date(date_breaks= "1 month", date_labels= "%b %Y")
```



Distributional Test Forecast

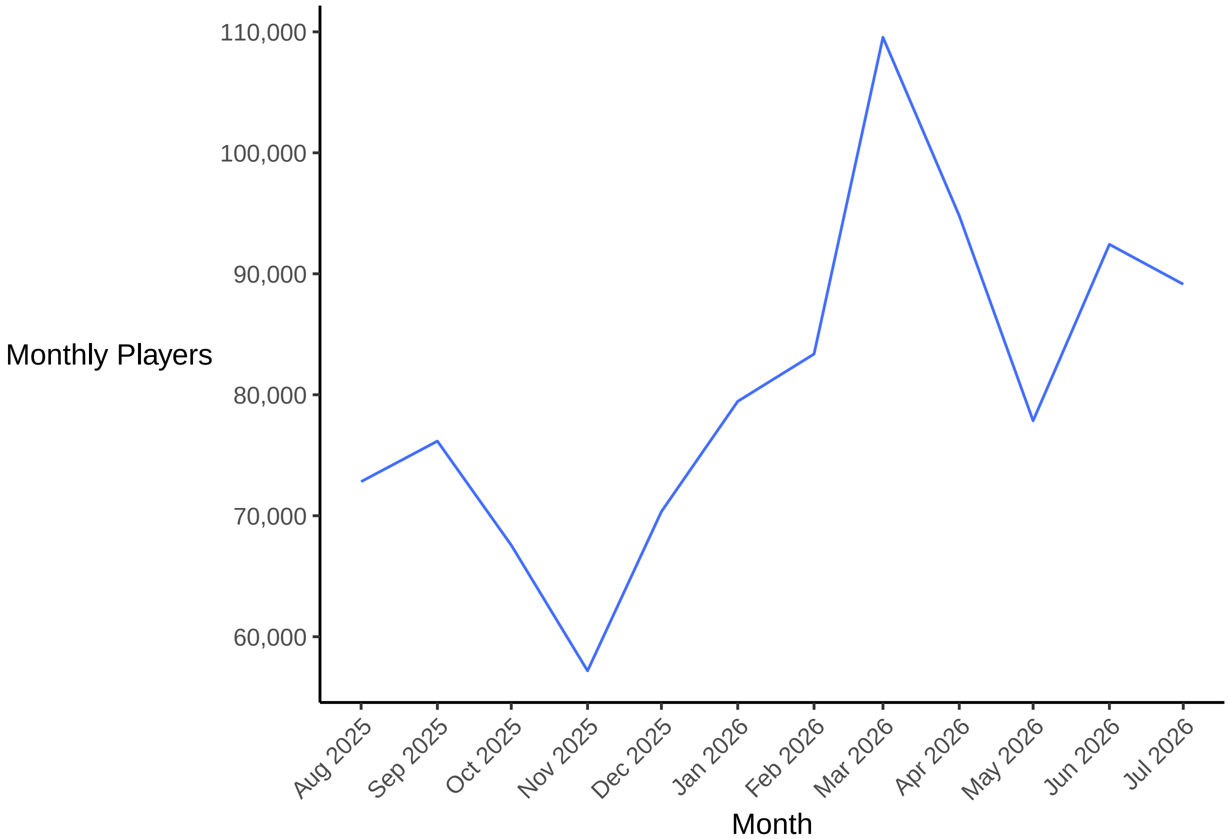
```r
#Distributional forecast
index_col <- tsibble::index_var(test)
test_months <- test[[index_col]] # Extract yearmonth vector from the test set

# Plot with only test months shown on x-axis
best.ar |>
  forecast(test) |>
  autoplot(test, level = 80) +
  labs(y = "Monthly Players") +
  scale_x_yearmonth(
    breaks = test_months,
    labels = label_date("%b %Y")
  ) +
```

```
  scale_y_continuous(labels = label_comma()) +
  theme_classic() +
  theme(
    axis.title.y = element_text(angle = 0, vjust = 0.5),
    axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1)
  )
```

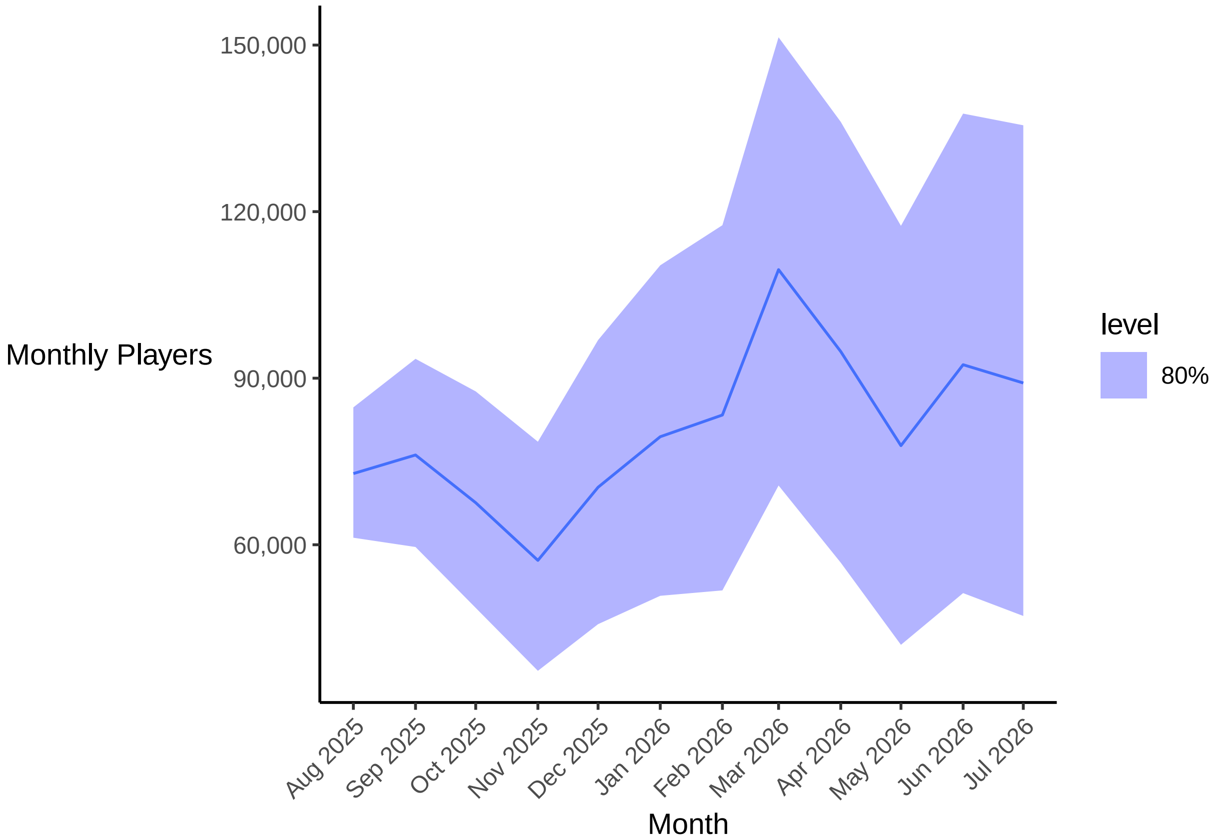


##Future Forecast

```
fore.model<- ts |> model(ARIMA(sqrt(Avg.Players)~pdq(0,1,0)+PDQ(0,1,1)))

#Point estimate forecast
autoplot(fore.model |> forecast(h=12), level = NULL) +
  labs(y = "Monthly Players") +
  scale_x_yearmonth(
    breaks = yearmonth(c("2025 Aug", "2025 Sep", "2025 Oct", "2025 Nov",
                         "2025 Dec", "2026 Jan", "2026 Feb", "2026 Mar",
                         "2026 Apr", "2026 May", "2026 Jun", "2026 Jul")),
    labels = label_date("%b %Y")
  ) +
  scale_y_continuous(labels = label_comma()) +
  theme_classic() +
  theme(
    axis.title.y = element_text(angle = 0, vjust = 0.5),
    axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1)
  )
```




```r
#Distributional forecast
autoplot(fore.model |> forecast(h=12), level = 80) +
  labs(y = "Monthly Players") +
  scale_x_yearmonth(
    breaks = yearmonth(c("2025 Aug", "2025 Sep", "2025 Oct", "2025 Nov",
                         "2025 Dec", "2026 Jan", "2026 Feb", "2026 Mar",
                         "2026 Apr", "2026 May", "2026 Jun", "2026 Jul")),
    labels = label_date("%b %Y")
  ) +
  scale_y_continuous(labels = label_comma()) +
  theme_classic() +
  theme(
    axis.title.y = element_text(angle = 0, vjust = 0.5),
    axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1)
  )
```


Monthly Players
150,000
120,000
90,000
60,000
Aug 2025
Sep 2025
Oct 2025
Nov 2025
Dec 2025
Jan 2026
Feb 2026
Mar 2026
Apr 2026
May 2026
Jun 2026
Jul 2026
Month
level
80%