Code Appendix

Contents

Research Questions

Prediction

Which data science programs between R and Python had the highest predicted growth rate in from 2019 to 2022?

Inference

Which data science topics (classification, regression, cluster analysis, time series, and machine learning) significantly contributed to the growth rates in R and Python?

Script Setup

Packages Load

```r
# Package Load
library(ggplot2)
library(tidyverse)
library(lubridate)
library(vars)
library(xts)
library(scales)
library(fGarch)
library(ggcorrplot)
library(forecast)
```

Data Load

```r
# Read in the data
#stack.data <- read.csv("C:/Users/cbrig/OneDrive/CMU/Time Series/Course Project/time_series_project_dat
stack.data <- read.csv("C:/Users/Owner/CMU/Spring/36-618/Project/time_series_project_data.csv")

clean.stack.data <- stack.data %>%
  mutate(date = lubridate::mdy(date)) %>%
  dplyr::select(date, r, python, regression, classification, cluster_analysis, time_series, machine_lea
  mutate(id = row_number()) %>%
  dplyr::filter(id > 24) %>%
  dplyr::select(-id)
#head(clean.stack.data)
```

Please note that the first two years were excluded from the data set for a few reasons. To start, some of the topics had no recorded questions in these first two years which negatively affects the modeling process. Similarly, 2009 is around the time period where data science and StackOverflow became popular and utilized by people in the field. Consequently, the data in the first two years looks drastically different than the remainder of the dataset. Therefore, we decided that it was best to focus on the data from 2011 to 2019.

```r
# Data Preparation by putting the relevant variables into their own lists
predictor.list <- c("python", "r", "machine_learning", "time_series",
                                "regression", "classification", "cluster_analysis")
full.var.list <- c("python", "r", "machine_learning", "time_series",
                                "regression", "classification", "cluster_analysis", "date")
pretty.names <- list('Python', "R", "Machine Learning", "Time Series", "Regression", "Classification",
                  "Cluster Analysis")
```

Time Series Objects

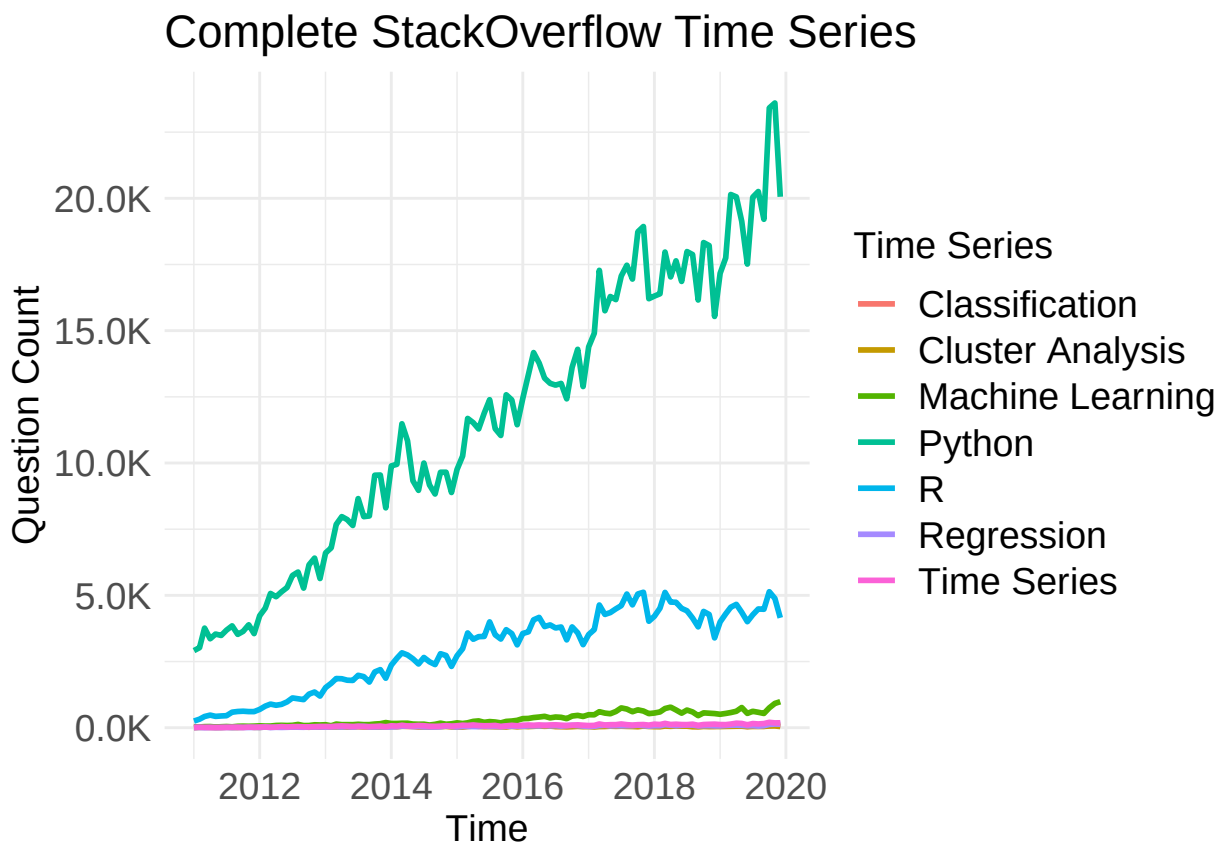
```r
# Create a formal time series for every series
for(variable in predictor.list){
  name <- paste0(variable, '.ts')
  assign(name, ts(clean.stack.data[variable], start = c(2011,1), frequency = 12))
}

# Store the time series in a list so that we can access them easily
```

```
series.list <- list(python.ts, r.ts, machine_learning.ts, time_series.ts,
                                regression.ts, classification.ts, cluster_analysis.ts)
```

Exploratory Data Analysis

Full Time Series Visualization

```
# Visualizes the entire data set

# Create the data needed for the visualization
full.ts.viz <- clean.stack.data %>%
  dplyr::select(all_of(full.var.list)) %>%
  rename(Python = python, R = r, 'Machine Learning' = machine_learning,
         Classification = classification, 'Cluster Analysis' = cluster_analysis,
         'Time Series' = time_series, Regression = regression) %>%
  gather(key = "variable", value = "value", -date)

# Create the visualization
ggplot(full.ts.viz, aes(x = date, y = value, color = variable, group = variable)) +
  geom_line(size = 1) +
  theme_minimal()+
    scale_y_continuous(labels = label_number(suffix = "K", scale = 1e-3))+
  labs(color='Time Series', y = 'Question Count', x = 'Time', title = "Complete StackOverflow Time Seri
  theme(text = element_text(size=14),
        axis.text.x = element_text(size = 14),
        axis.text.y = element_text(size = 14),
        legend.text = element_text(size = 14))
```

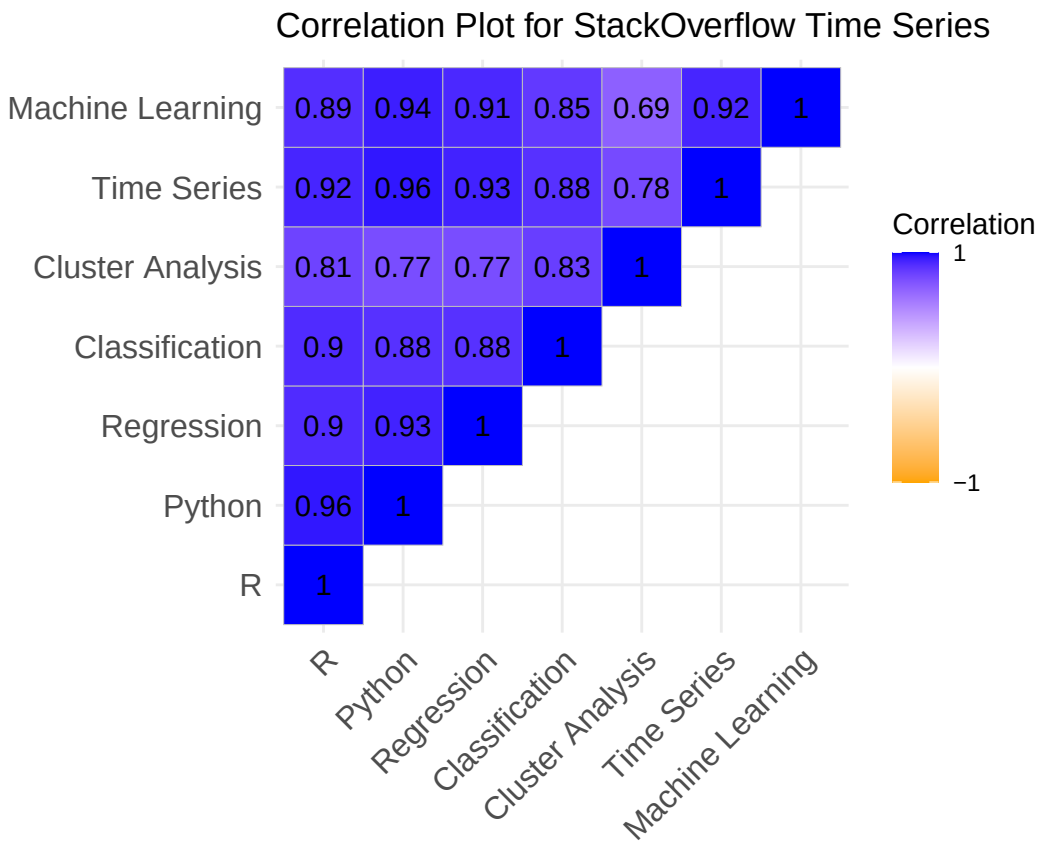


Unsurprisingly, Python stands out among all the time series with the largest number of questions since it is the most popular data science tool. The magnitude can also be explained by the other applications in different fields. R has the second most amount of questions which is also expected since R has other uses. The remaining topics have significantly less monthly questions, with machine learning having the most questions out of the data science topics. While it is hard to see the behavior in the models, every series has an increasing trend. This aligns with the growing data science job market and demand for these skills.

Correlation Plot

```
clean.stack.data %>%
  dplyr::select(-date) %>%
  rename(Python = python, R = r, 'Machine Learning' = machine_learning, Classification = classification
         'Cluster Analysis' = cluster_analysis, Regression = regression, 'Time Series' = time_series) %
  cor() %>%
  ggcorrplot(type = 'upper',
             show.diag = T,
             lab = T,
             title = 'Correlation Plot for StackOverflow Time Series') +
  scale_fill_gradient2(low = 'orange', high = 'blue', breaks=c(-1, 1), limit=c(-1, 1), name = "Correlat
```

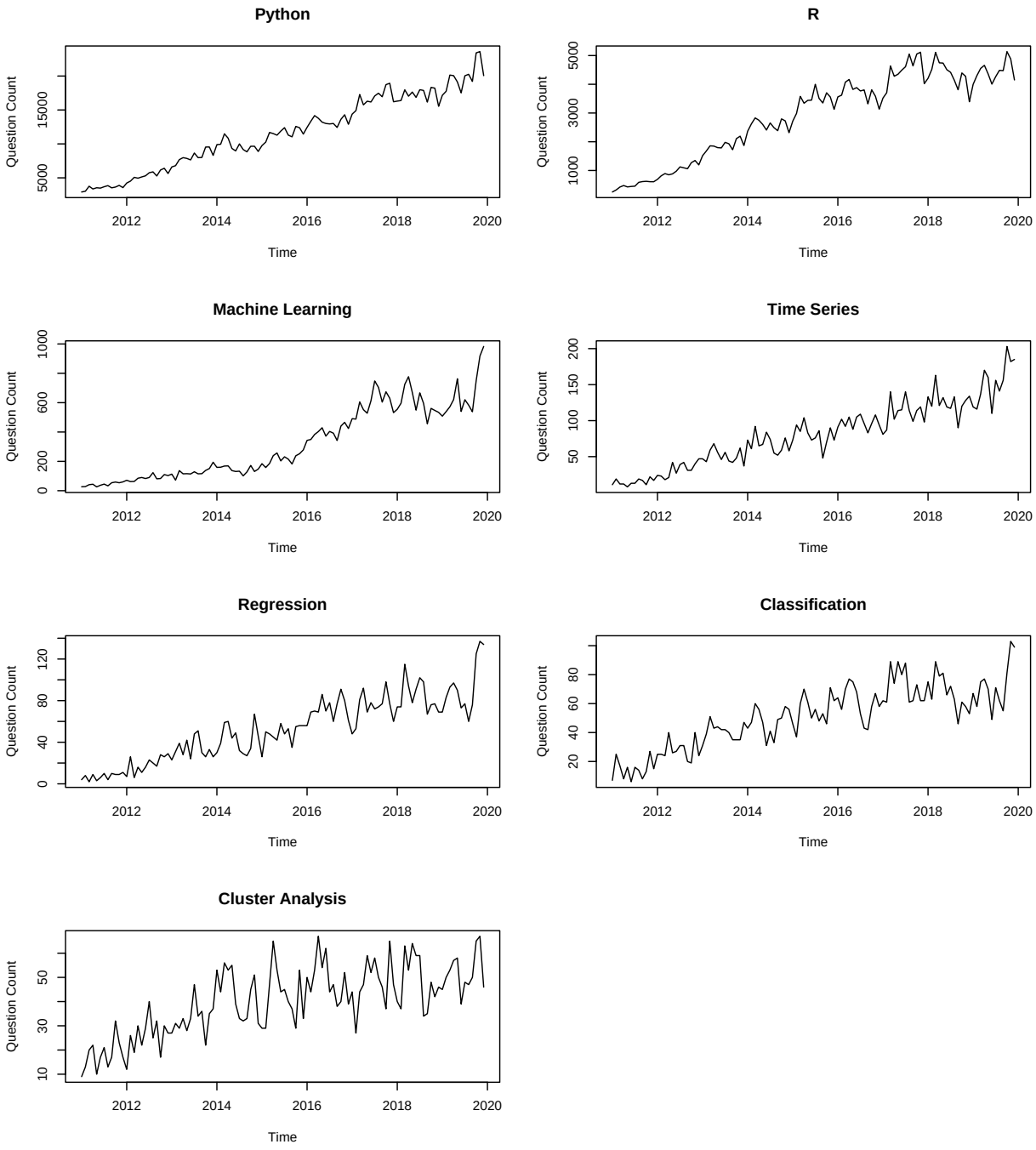
```
## Scale for 'fill' is already present. Adding another scale for 'fill', which
## will replace the existing scale.
```



The correlation plot shows that all the series have a strong positive correlation with eachother. This matches the observations from the full time series since they all have an increasing trend. It is worth noting that R and Python have one of the strong correlations which may cause issues with multicollinearity in the modeling process.

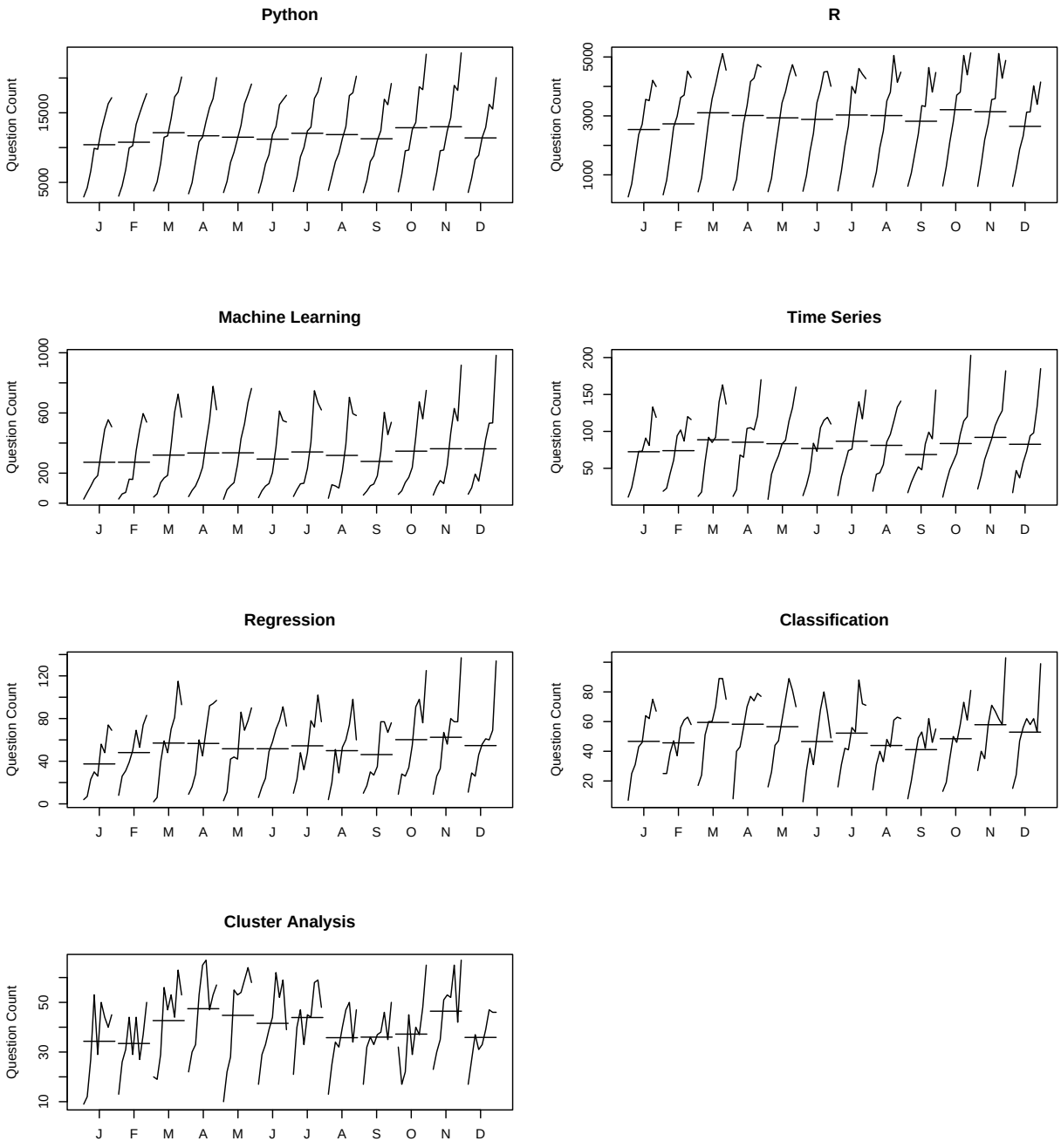
Individual Series Plots

```
par(mfrow = c(4,2))
for(i in 1:length(series.list)){
  plot(series.list[[i]], main = pretty.names[[i]], ylab = 'Question Count')}
```



Month Plots

```
par(mfrow = c(4,2))
for(i in 1:length(series.list)){
  monthplot(series.list[[i]], main = pretty.names[[i]], ylab = 'Question Count')}
```

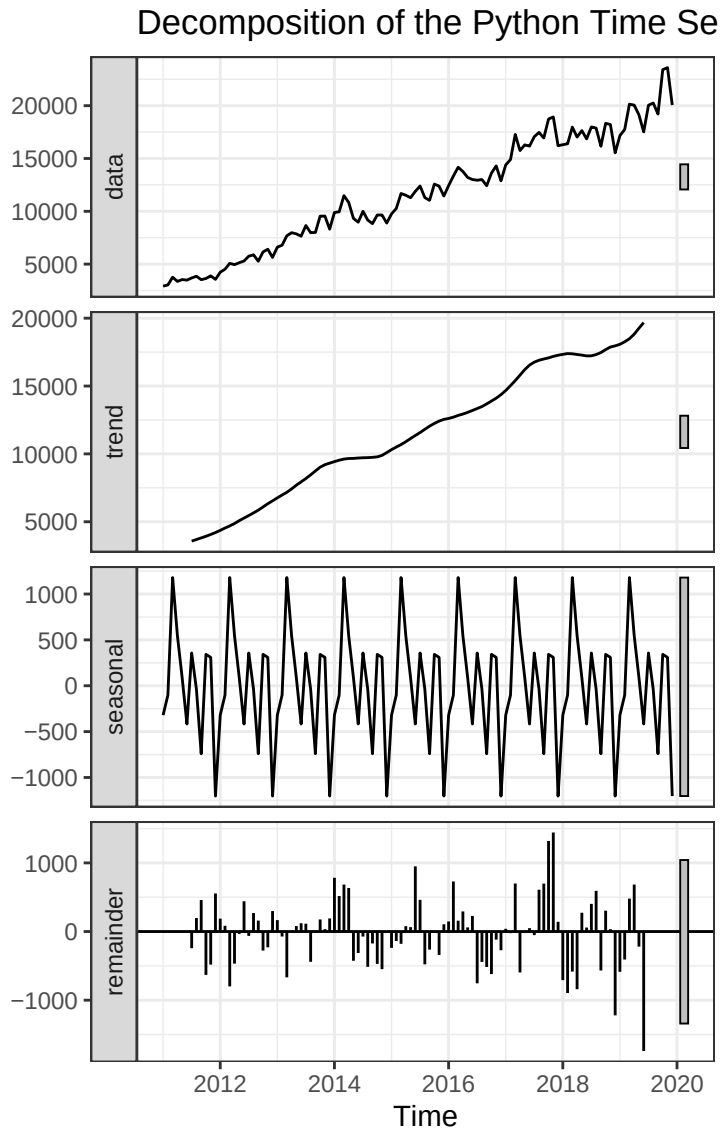


The decomposition of the time series for R and Python show that there may be seasonality in the time series since there are higher question counts in March, April, and November. While there may be several causes for this, one possible explanation could be the academic semester. In other words, students tend to start using these tools during these months.

Decompositions

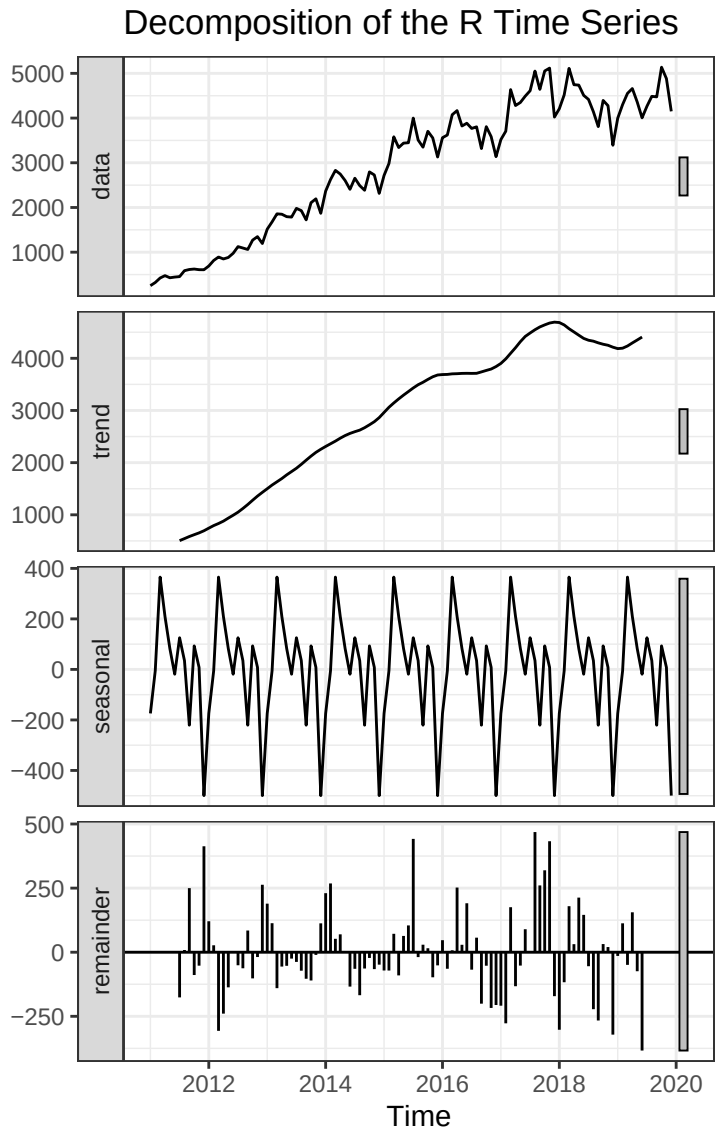
```
# Commented out to save space
# for(i in 1:length(series.list)){
#   plot(decompose(series.list[[i]]))}
```

```
# Nice decomposition plots for the presentation and paper
python.ts %>%
  decompose() %>%
  autoplot()+
  theme_bw() +
  labs(title = 'Decomposition of the Python Time Series')
```



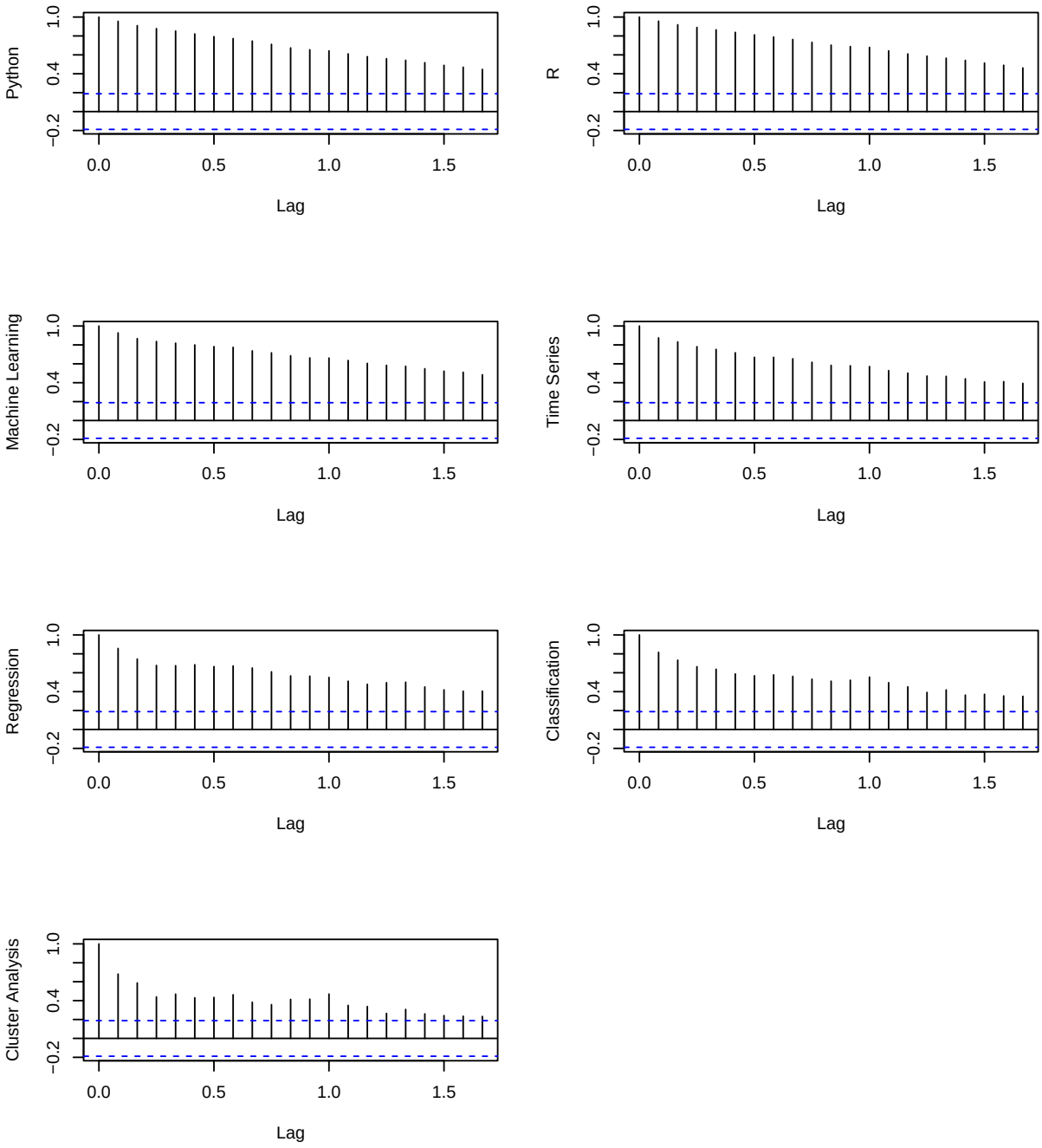

```
r.ts %>%
  decompose() %>%
```

```
  autoplot() +
  theme_bw() +
  labs(title = 'Decomposition of the R Time Series')
```

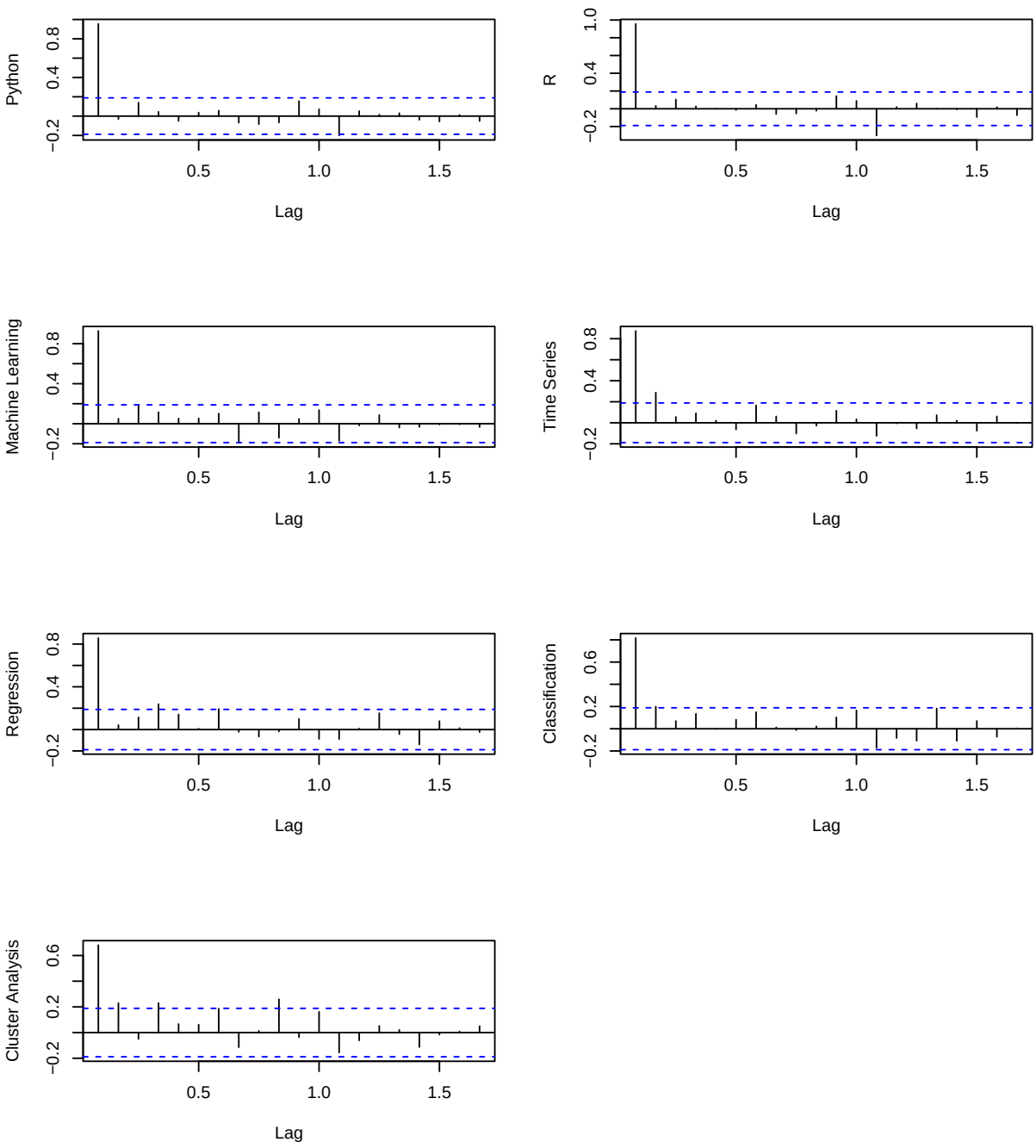


ACF and PACF

```
# ACF
par(mfrow = c(4,2))
for(i in 1:length(series.list)){
  acf(series.list[[i]], ylab = pretty.names[[i]], main = '')}

# PACF
par(mfrow = c(4,2))
```




```
for(i in 1:length(series.list)){
  pacf(series.list[[i]], ylab = pretty.names[[i]], main = '')}
```



The ACF plots for all the time series show a slow decay with increasing lags which suggests that there may be an autoregressive behavior to the series. Additionally, the PACF plots occasionally show a small significant correlation around a year long lag. This supports our observations for the decompositions which suggests a seasonality component to the time series.

ARIMA Modeling

Our exploratory data analysis showed that our time series has increasing trends with a possible annual seasonal component. To better understand our series, an ARIMA model will be built for the R and Python series. The model building process will start by fitting a model based on the findings from the exploratory data analysis. Then, the model will be checked through residual versus observed plots, ACF and PACF plots, QQ plots, and multiple simulations to see if the fitted model is a good fit to the data. After the initial diagnosis, an additional model will be created using the auto.arima() function in R. The parameters for this function will be decided based on the initial model built from the exploratory data analysis. If the model from the auto.arime() function matches the model that was built from the exploratory data analysis and the model diagnostics showed no glaring errors, the ARIMA model will be used to calculate forecasted values. If the auto.arima() function produces a different model, then the model diagnostics listed earlier will be applied again to see if the new model is a better fit to the data. The orders from the final ARIMA model will determine what actions need to be taken before fitting the VARMA model.

Python

Manual ARIMA Modeling

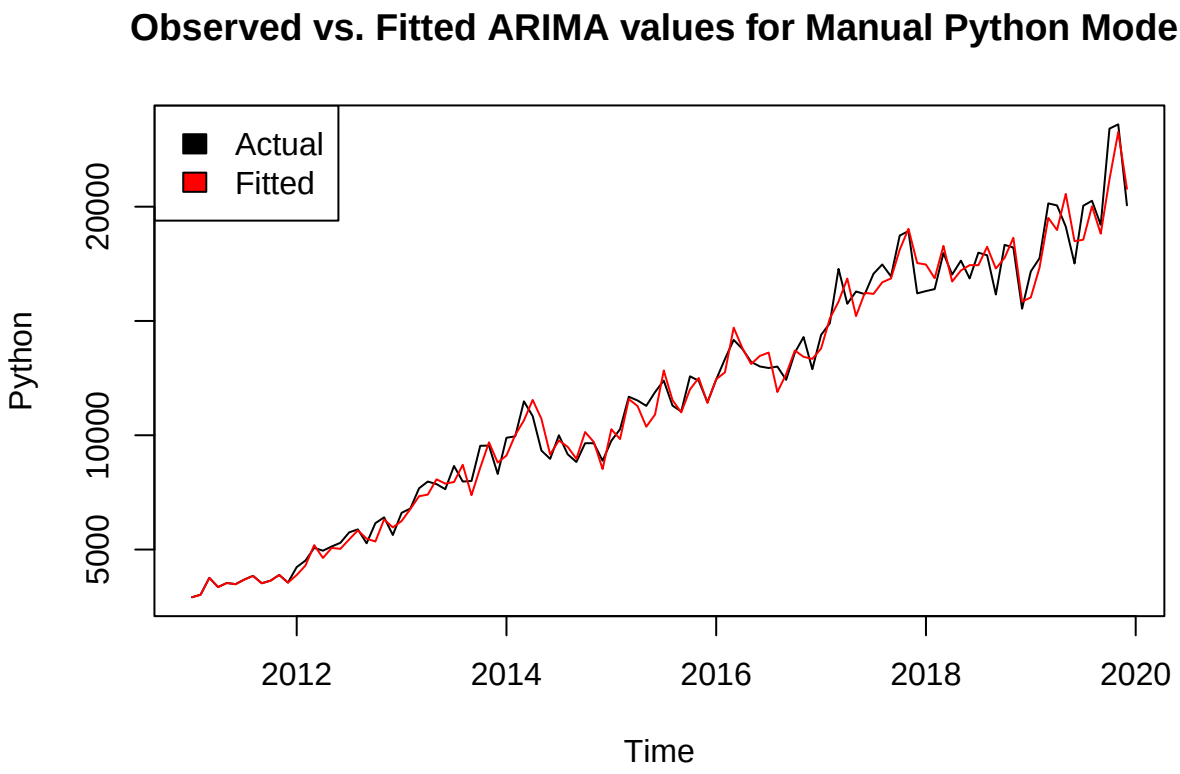
```
# Python Model 1 by Manual Fits
python.mod1 = arima(python.ts, order = c(1,0,0), seasonal = list(order = c(1,1,0), period = 12))
```

The exploratory data analysis showed autoregressive behavior, increasing trend, and annual seasonality. Therefore, the first ARIMA model was fitted with a non-seasonal autoregressive term, a seasonal autoregressive term, and a seasonal trend term.

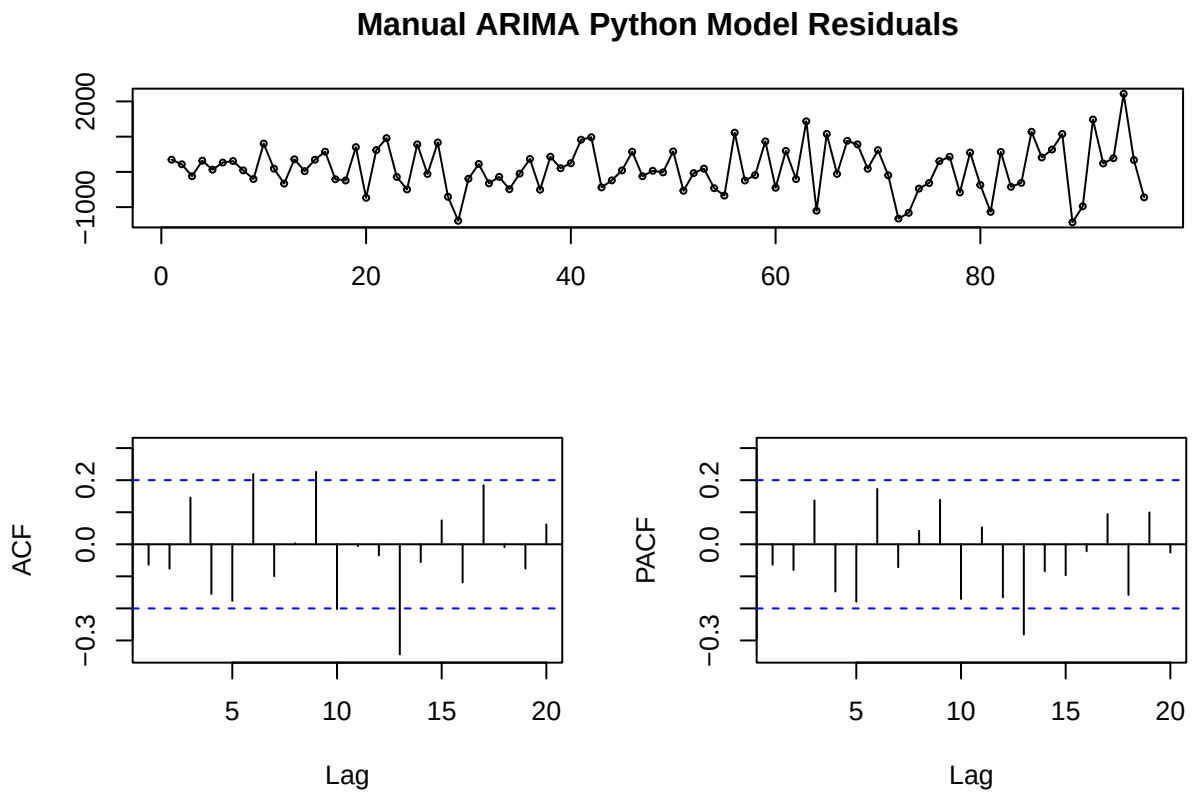
Manual ARIMA Diagnostics

Residuals

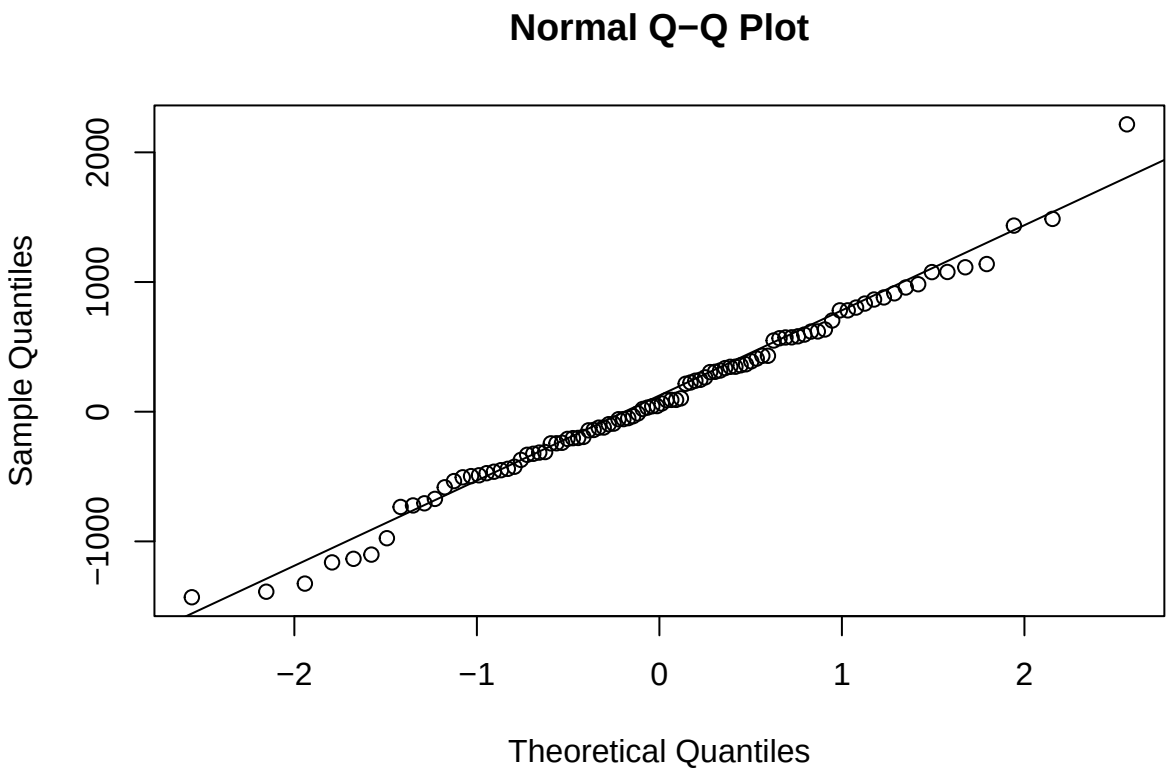
```
# Diagnostics
par(mfrow=c(1,1))
plot(python.ts, col = "black", main = "Observed vs. Fitted ARIMA values for Manual Python Model",
     ylab = "Python")
lines(python.ts - python.mod1$resid, col = "red")
legend("topleft", fill = c("black", "red"), c("Actual", "Fitted"))
```




```
tsdisplay(resid(python.mod1)[13:length(python.mod1$resid)], main= "Manual ARIMA Python Model Residuals"
```



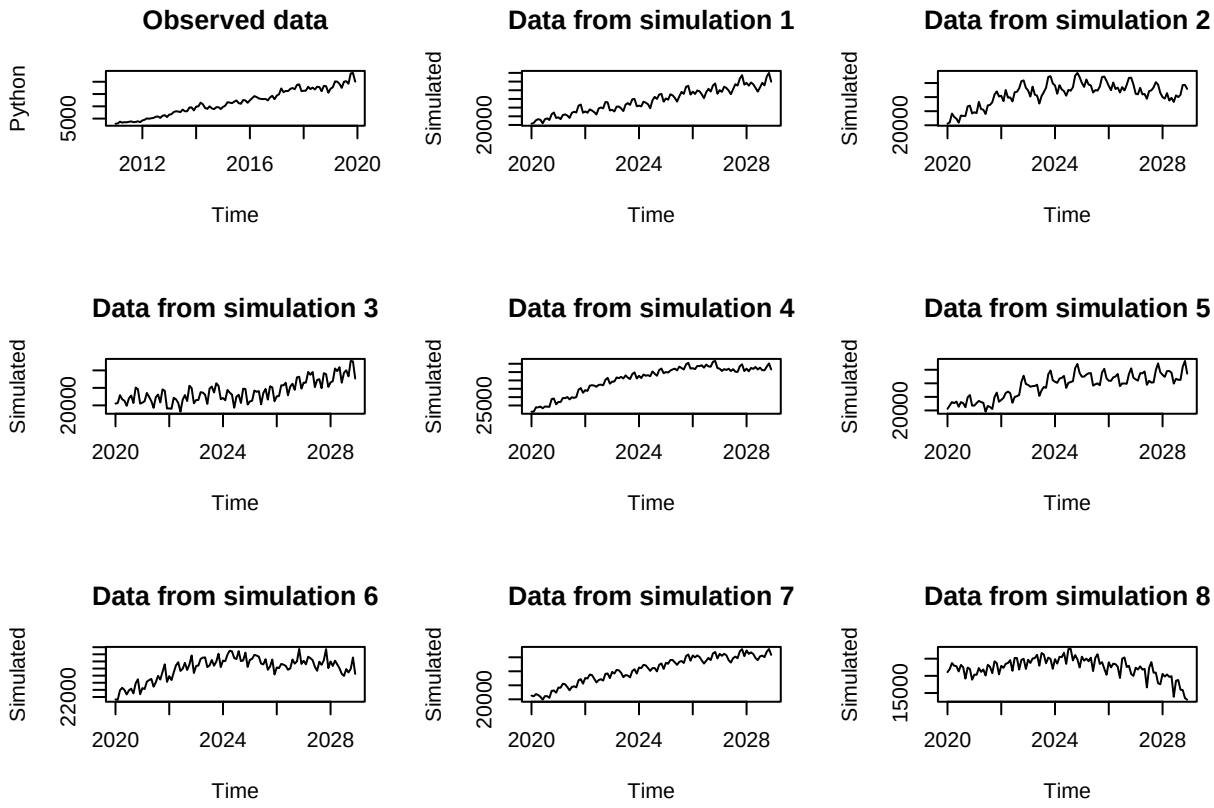

```
qqnorm(python.mod1$resid[13:length(python.mod1$resid)])
qqline(python.mod1$resid[13:length(python.mod1$resid)])
```



The fitted versus observed plot shows that the model fits the data well, with a few possible exceptions in 2018. The residuals also appear to be approximately stationary with a constant mean and variance. However, the ACF and PACF plots display a small significant correlation around the 13th lag. These correlations are not significant enough to warrant any action, but it is worth noting that there may be an underlying behavior in the series. Also, the QQ plot shows that the residuals are approximately normal.

Simulations

```
# Plot Multiple Simulations
par(mfrow=c(3,3))
plot(python.ts, ylab = 'Python', main = 'Observed data')
for (i in 1:8){
  set.seed(i)
  plot(simulate(python.mod1),
     ylab = 'Simulated',
     main = paste('Data from simulation', i, sep = ' '))
}
```




```
par(mfrow=c(1,1))
# In general, diagnostics look pretty good but still missing some seasonality
```

Finally, the simulations support the idea that the model is a good fit to the data since the simulations exhibit similar behaviors to the original StackOverflow time series.

Model Validation

```
# Python auto.arima with these arguments, auto.arima fits the same model we fit
auto.arima(python.ts, trace = T, stepwise = T, approximation = F, allowdrift = F)
```

```
##
##  ARIMA(2,0,2)(1,1,1)[12]                    : 1537.191
##  ARIMA(0,0,0)(0,1,0)[12]                    : 1770.099
##  ARIMA(1,0,0)(1,1,0)[12]                    : 1529.993
##  ARIMA(0,0,1)(0,1,1)[12]                    : 1670.929
##  ARIMA(1,0,0)(0,1,0)[12]                    : 1534.045
##  ARIMA(1,0,0)(2,1,0)[12]                    : 1532.153
##  ARIMA(1,0,0)(1,1,1)[12]                    : 1532.154
##  ARIMA(1,0,0)(0,1,1)[12]                    : 1530.23
##  ARIMA(1,0,0)(2,1,1)[12]                    : 1534.302
##  ARIMA(0,0,0)(1,1,0)[12]                    : 1747.661
##  ARIMA(2,0,0)(1,1,0)[12]                    : 1531.854
```

```
##  ARIMA(1,0,1)(1,1,0)[12]                    : 1531.791
##  ARIMA(0,0,1)(1,1,0)[12]                    : 1664.227
##  ARIMA(2,0,1)(1,1,0)[12]                    : 1534.297
##
##  Best model: ARIMA(1,0,0)(1,1,0)[12]
```

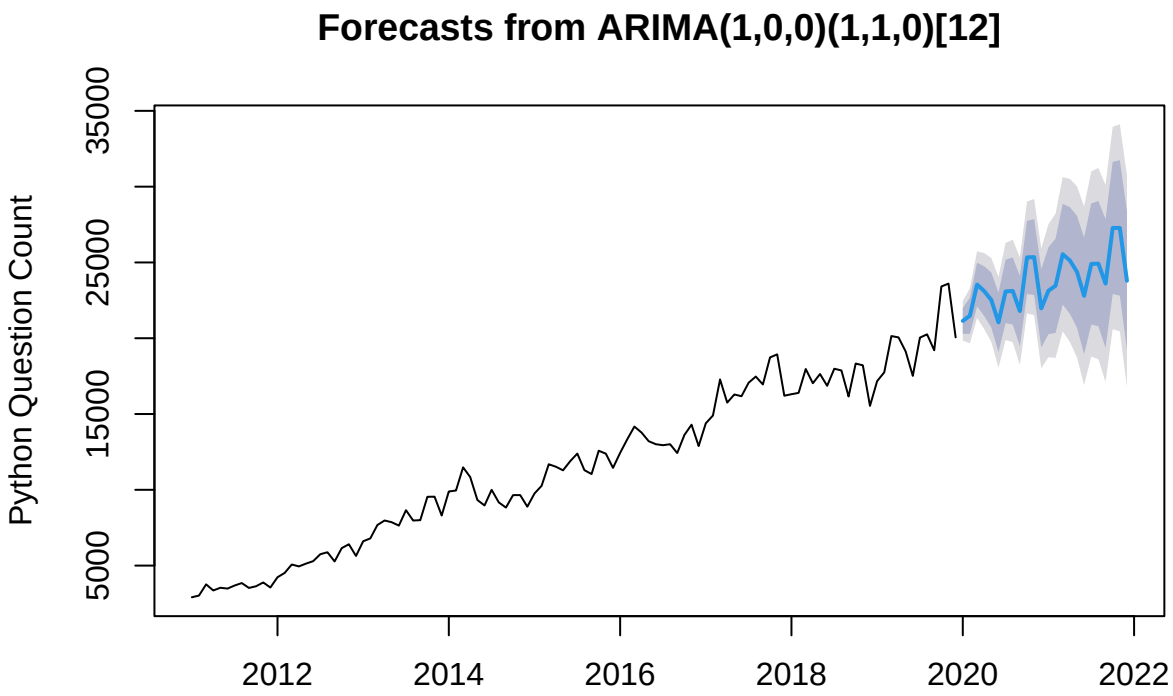
```
## Series: python.ts
## ARIMA(1,0,0)(1,1,0)[12]
##
## Coefficients:
##          ar1     sar1
##       0.9751  -0.2847
## s.e.  0.0206   0.1107
##
## sigma^2 = 450183:  log likelihood = -761.87
## AIC=1529.73   AICc=1529.99   BIC=1537.43
```

```
# stepwise = F very slow for seasonal models
# fits a different model when restrict to stationary model, but diagnostics are worse
# also played around with changing max.p and max.q, but don't get anything new/anything that performs b
```

The auto arima function produced the same model that was suggested by the exploratory data anlaysis. Therefore, the model diganostics were not performed again.

Forecasts

```
# Python forecast
plot(forecast(python.mod1, h = 24), ylab = "Python Question Count")
```



The forecasts from the ARIMA model for Python have an increasing trend with some variations within the year which is what we would expect based on the previous observations. All of these observations suggest that the ARIMA(1,0,0)(1,1,0) model is a good fit to the data.

R

Manual ARIMA Modeling

```
# Python Model 1 by Manual Fits
r.mod1 = arima(r.ts, order = c(1,0,0), seasonal = list(order = c(1,1,0), period = 12))
```

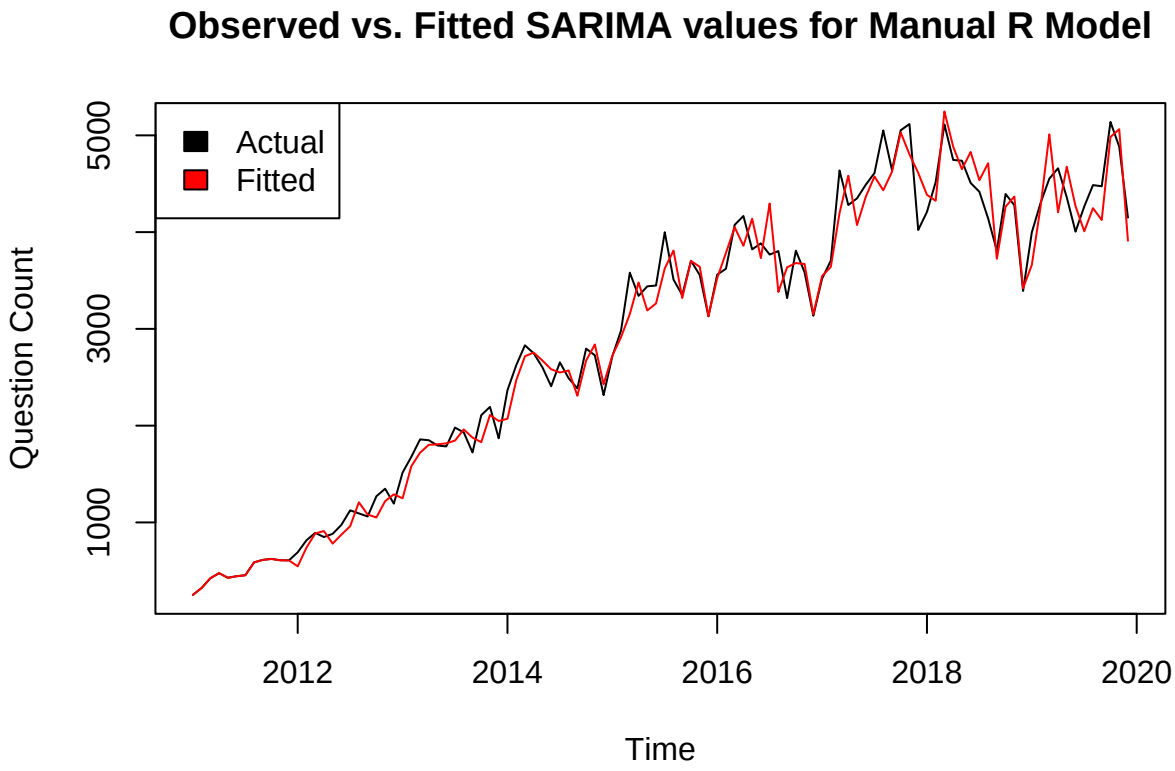
The same initial ARIMA model was fit to the R time series since the exploratory data analysis showed similar behavior for both series.

Manual ARIMA Diagnostics

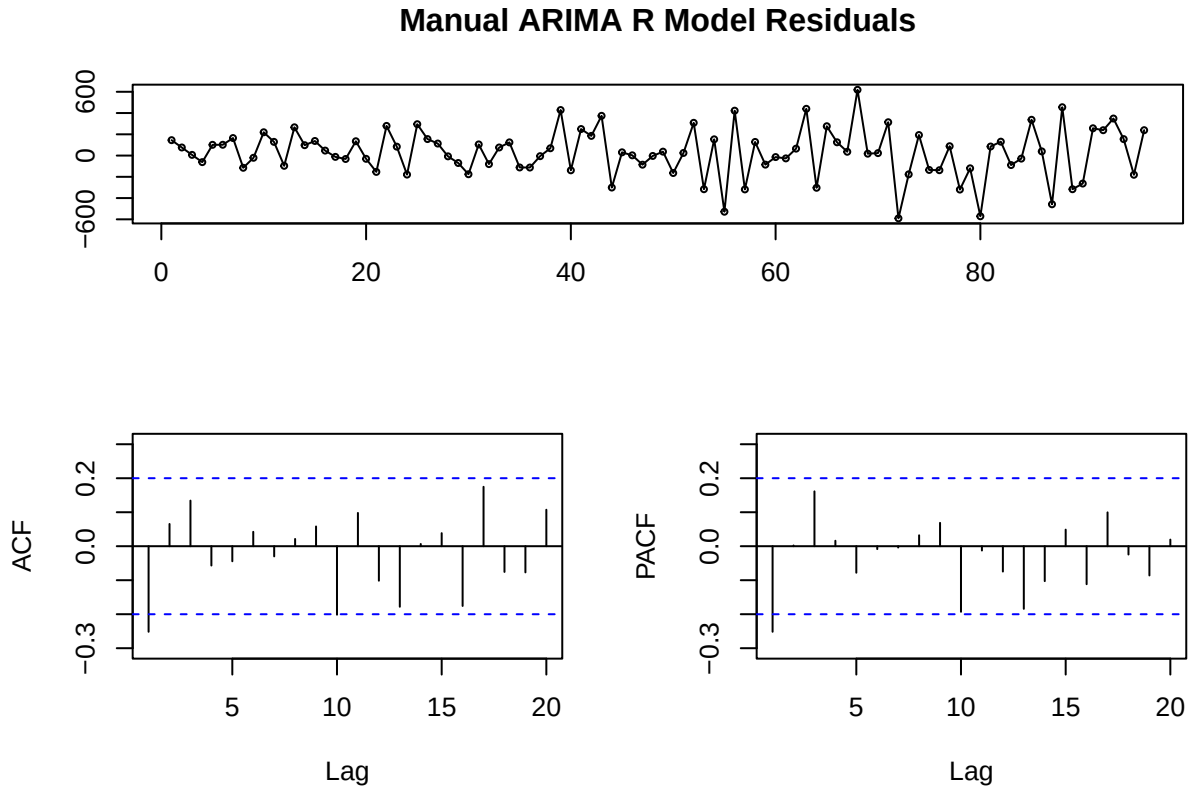
Residuals

```
# Diagnostics
par(mfrow=c(1,1))
plot(r.ts, col = "black", main = "Observed vs. Fitted SARIMA values for Manual R Model",
     ylab = "Question Count")
```

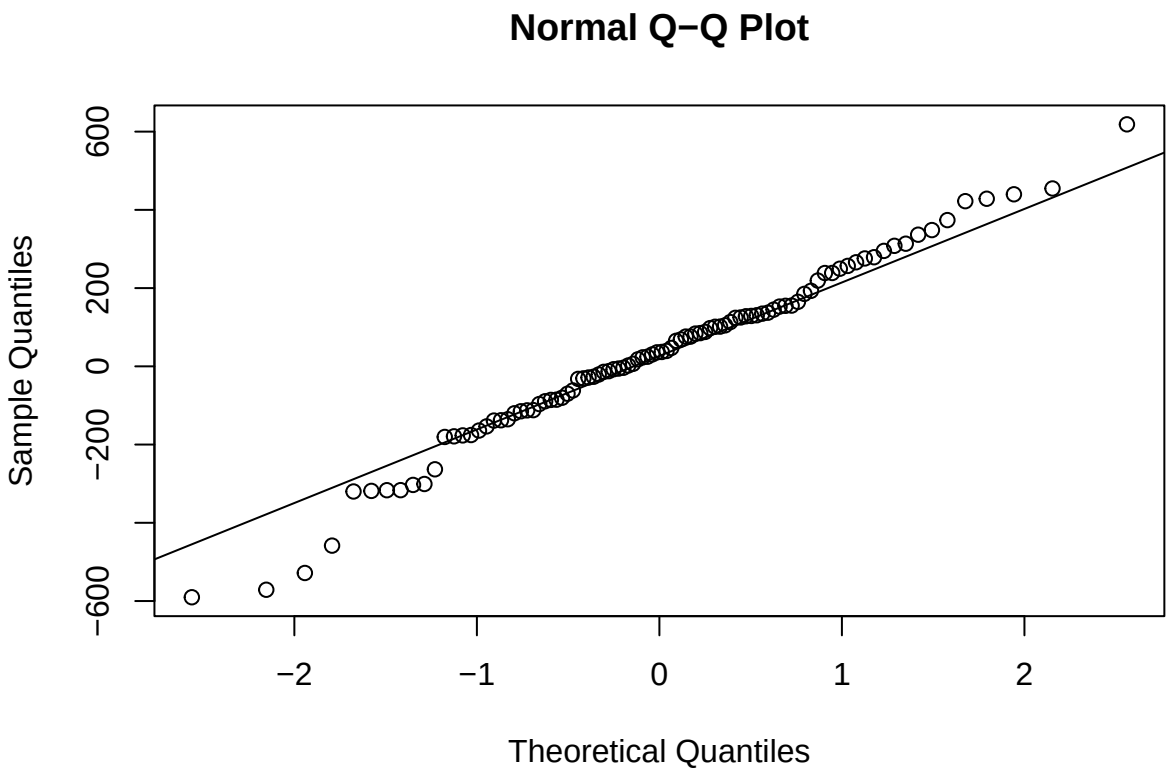
```
lines(r.ts - r.mod1$resid, col = "red")
legend("topleft", fill = c("black", "red"), c("Actual", "Fitted"))
```




```
tsdisplay(resid(r.mod1)[13:length(r.mod1$resid)], main= "Manual ARIMA R Model Residuals")
```




```
qqnorm(r.mod1$resid[13:length(r.mod1$resid)])
qqline(r.mod1$resid[13:length(r.mod1$resid)])
```

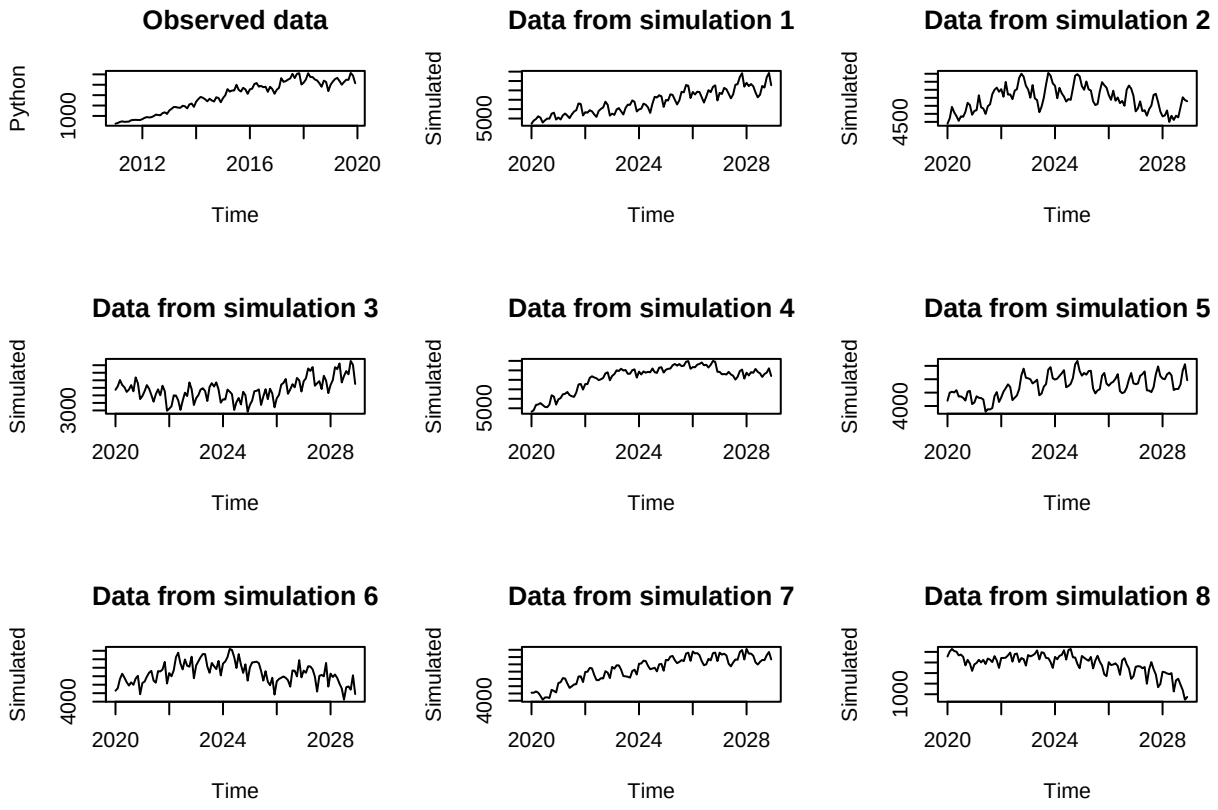



```
# In general, diagnostics look pretty good
```

The fitted versus observed plot shows that the model fits the data well, again with a few possible exceptions in 2018. The residuals also appear to be approximately stationary with a constant mean and variance. But unlike the Python series, there appears to be no significant correlations in the ACF and PACF plots. Also, the QQ plot shows that the residuals are approximately normal.

Simulations

```
# Plot Multiple Simulations
par(mfrow=c(3,3))
plot(r.ts, ylab = 'Python', main = 'Observed data')
for (i in 1:8){
  set.seed(i)
  plot(simulate(r.mod1),
     ylab = 'Simulated',
     main = paste('Data from simulation', i, sep = ' '))
}
```




```
par(mfrow=c(1,1))
```

Several of the simulations exhibit similar behaviors to the original StackOverflow time series since they have constant increasing trends with slight annual variations.

Model Validation

```
# R auto.arima
# tried similar differences in auto.arima arguments here
# fits a different model when restrict to stationary model, but diagnostics are worse
# also played around with changing max.p and max.q, but don't get anything new/anything that performs b
auto.arima(r.ts, trace = T, stepwise = T, approximation = F, allowdrift = F)
```

```
##
##  ARIMA(2,1,2)(1,1,1)[12]                    : 1304.26
##  ARIMA(0,1,0)(0,1,0)[12]                    : 1316.466
##  ARIMA(1,1,0)(1,1,0)[12]                    : 1301.434
##  ARIMA(0,1,1)(0,1,1)[12]                    : 1299.439
##  ARIMA(0,1,1)(0,1,0)[12]                    : 1316.298
##  ARIMA(0,1,1)(1,1,1)[12]                    : 1301.386
##  ARIMA(0,1,1)(0,1,2)[12]                    : 1301.365
##  ARIMA(0,1,1)(1,1,0)[12]                    : 1302.021
##  ARIMA(0,1,1)(1,1,2)[12]                    : 1303.594
```

```
##  ARIMA(0,1,0)(0,1,1)[12]                    : 1301.922
##  ARIMA(1,1,1)(0,1,1)[12]                    : 1301.336
##  ARIMA(0,1,2)(0,1,1)[12]                    : 1300.934
##  ARIMA(1,1,0)(0,1,1)[12]                    : 1299.171
##  ARIMA(1,1,0)(0,1,0)[12]                    : 1315.693
##  ARIMA(1,1,0)(1,1,1)[12]                    : 1301.108
##  ARIMA(1,1,0)(0,1,2)[12]                    : 1301.075
##  ARIMA(1,1,0)(1,1,2)[12]                    : 1303.293
##  ARIMA(2,1,0)(0,1,1)[12]                    : 1301.318
##  ARIMA(2,1,1)(0,1,1)[12]                    : Inf
##
##  Best model: ARIMA(1,1,0)(0,1,1)[12]
```

```
## Series: r.ts
## ARIMA(1,1,0)(0,1,1)[12]
##
## Coefficients:
##           ar1     sma1
##       -0.2231  -0.5008
## s.e.   0.0996   0.1006
##
## sigma^2 = 46908:  log likelihood = -646.45
## AIC=1298.91   AICc=1299.17   BIC=1306.57
```

```
# auto.arima actually fits a different model here than what we tried-fits d and D difference and an MA
```

The auto arima function produced a different model than the one suggested by the exploratory data analysis. Specifically, the auto function determined to add a non-seasonal trend, remove a seasonal autoregressive component, and add a seasonal moving average term.

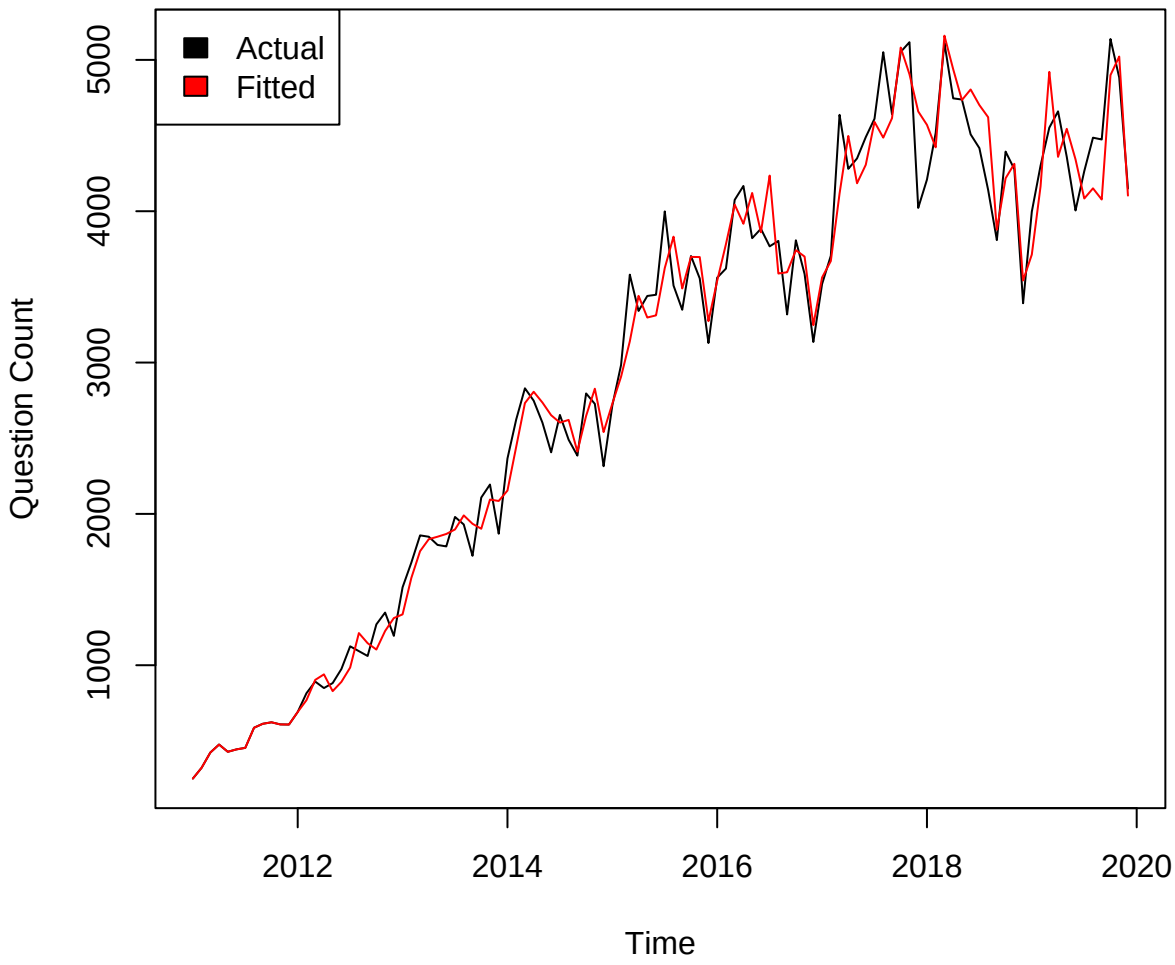
Model Fit - Auto ARIMA

```
# R Model 2 (with auto.arima results)
r.mod2 = arima(r.ts, order = c(1,1,0), seasonal = list(order = c(0,1,1), period = 12))
```

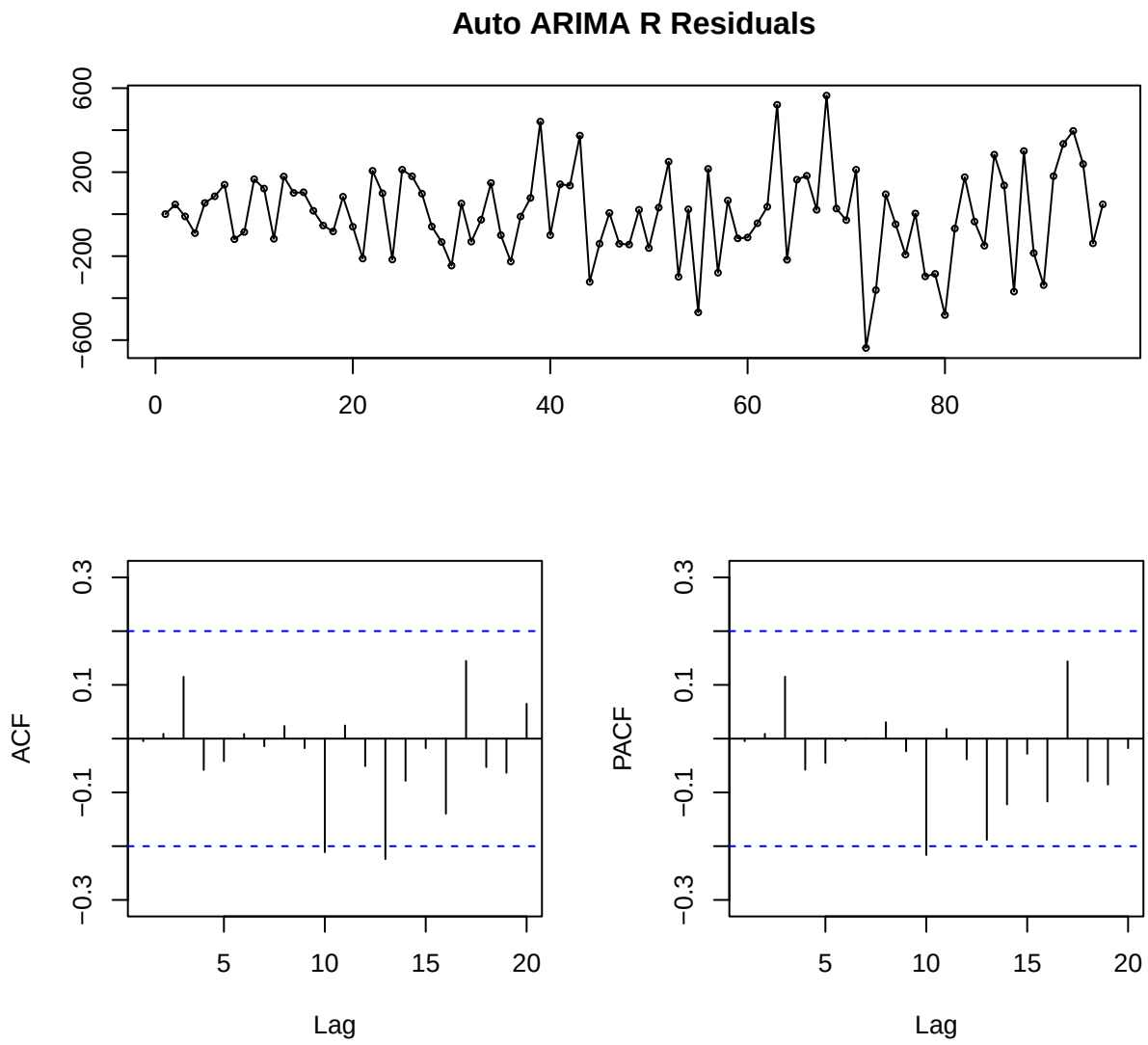
Model Diagnostics - Auto ARIMA

Since the auto arima function produced a different model, we will fit this model and perform the model diagnostics to see if it is a better fit to the data.

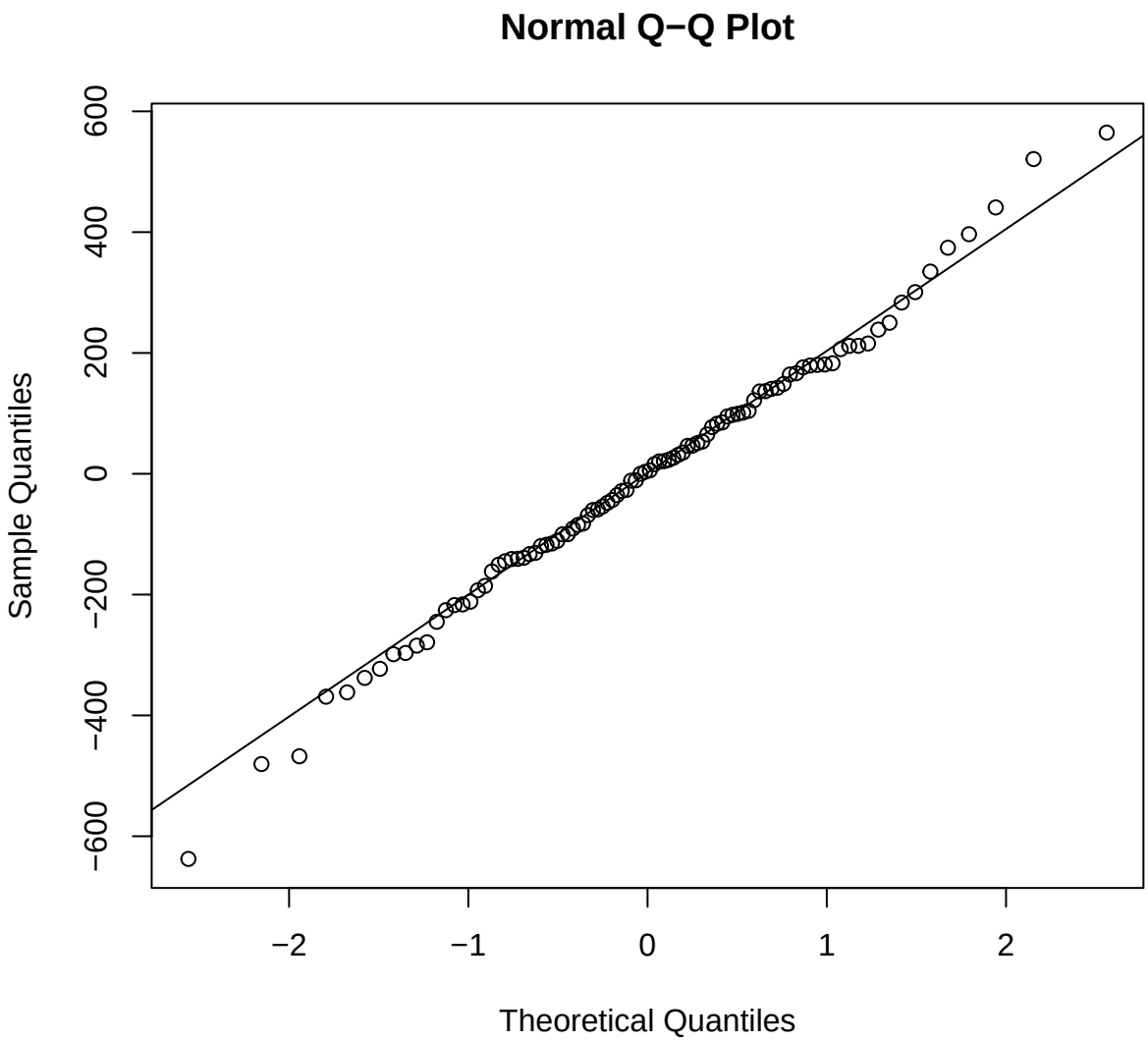
```
par(mfrow=c(1,1))
plot(r.ts, col = "black", main = "Observed vs. Fitted SARIMA values for Auto ARIMA R Model",
     ylab = "Question Count")
lines(r.ts - r.mod2$resid, col = "red")
legend("topleft", fill = c("black", "red"), c("Actual", "Fitted"))
```




```
tsdisplay(resid(r.mod2)[13:length(r.mod2$resid)], main = "Auto ARIMA R Residuals")
```

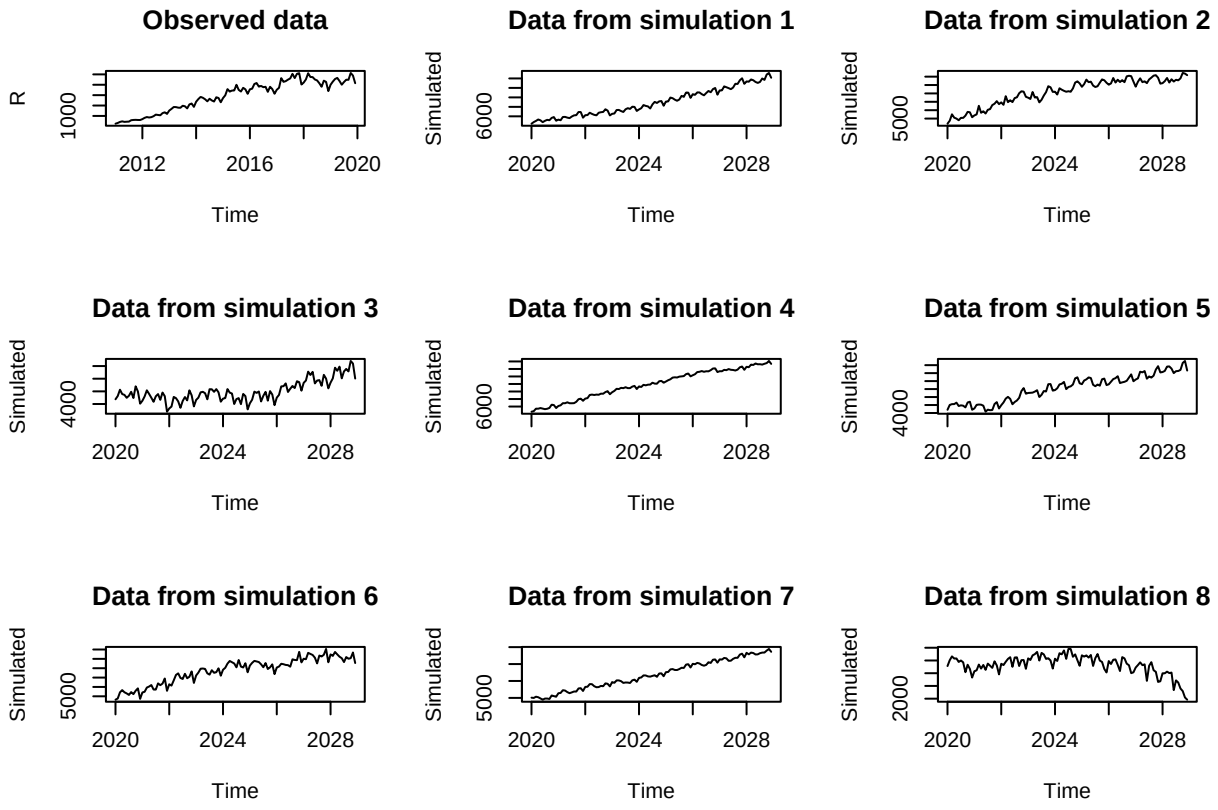



```
qqnorm(r.mod2$resid[13:length(r.mod2$resid)])
qqline(r.mod2$resid[13:length(r.mod2$resid)])
```



The fitted versus observed plot shows that the model fits the data well, again with a few possible exceptions in 2018. The residuals also appear to be approximately stationary with a constant mean and variance. However, there appear to be a few correlations on the border of being significant. Also, the QQ plot shows that the residuals are approximately normal.

```
# Second Round of Simulations
par(mfrow=c(3,3))
plot(r.ts, ylab = 'R', main = 'Observed data')
for (i in 1:8){
  set.seed(i)
  plot(simulate(r.mod2),
     ylab = 'Simulated',
     main = paste('Data from simulation', i, sep = ' '))
}
```

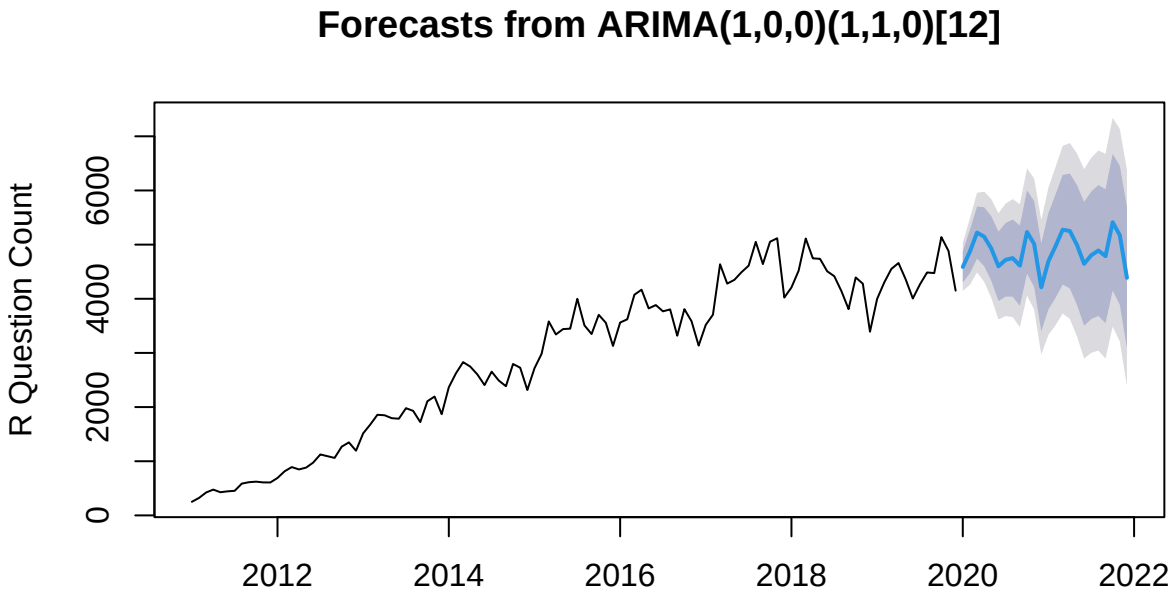


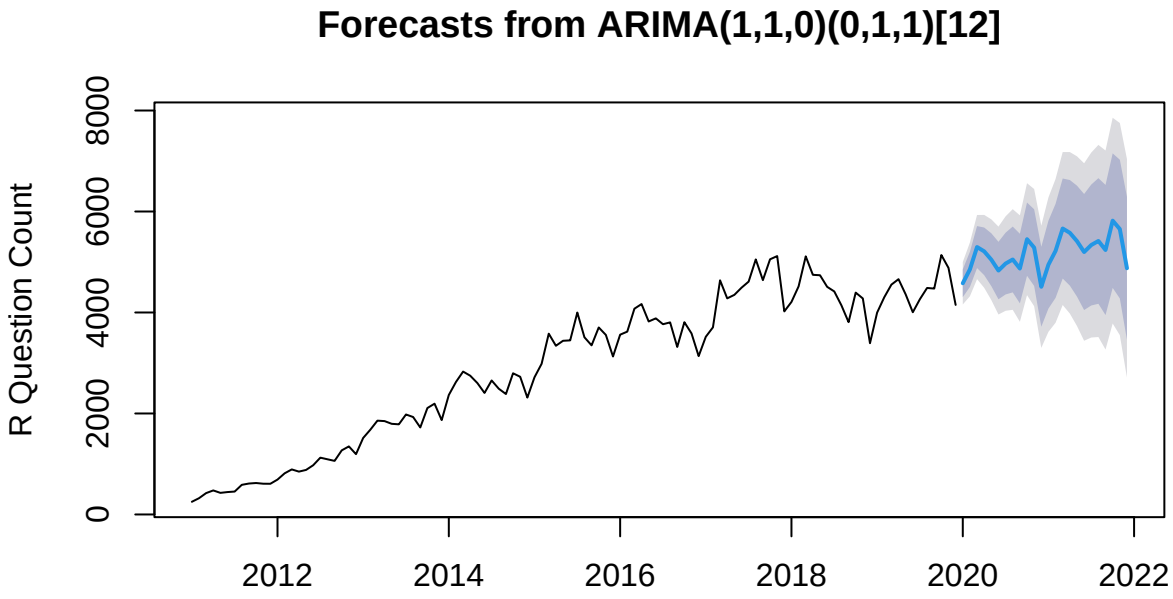

```
par(mfrow=c(1,1))
```

Several of the simulations exhibit similar behaviors to the original StackOverflow time series since they have constant increasing trends with slight annual variations.

Forecasts

```
# R forecasts
par(mfrow = c(2,1))
plot(forecast(r.mod1, h = 24), ylab = "R Question Count")
plot(forecast(r.mod2, h = 24), ylab = "R Question Count")
```





Based on the diagnostics, fit, and forecasts, the models suggested by the exploratory data analysis and by the auto arima functions are very similar. Looking at the summary output, the model that auto.arima picks has slightly lower AIC and RMSE/MAE, so overall this is the best model to the data.

```r
# Summaries of R models
summary(r.mod1)
```

```
##
## Call:
## arima(x = r.ts, order = c(1, 0, 0), seasonal = list(order = c(1, 1, 0), period = 12))
##
## Coefficients:
##          ar1     sar1
##       0.9589  -0.3484
## s.e.  0.0262   0.1008
##
## sigma^2 estimated as 50893:  log likelihood = -658.26,  aic = 1322.52
##
## Training set error measures:
##                     ME    RMSE      MAE      MPE     MAPE      MASE       ACF1
## Training set 26.03888 212.692 154.5326 1.259088 5.221512 0.6355945 -0.2503665
```

```r
summary(r.mod2)
```

```
##
## Call:
## arima(x = r.ts, order = c(1, 1, 0), seasonal = list(order = c(0, 1, 1), period = 12))
##
## Coefficients:
##           ar1     sma1
##       -0.2231  -0.5008
## s.e.   0.0996   0.1006
##
## sigma^2 estimated as 45920:  log likelihood = -646.45,  aic = 1298.91
##
## Training set error measures:
##                      ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -3.195138 200.9797 147.5508 -0.1045264 4.885689 0.6068783
##                       ACF1
## Training set -0.004649645
```

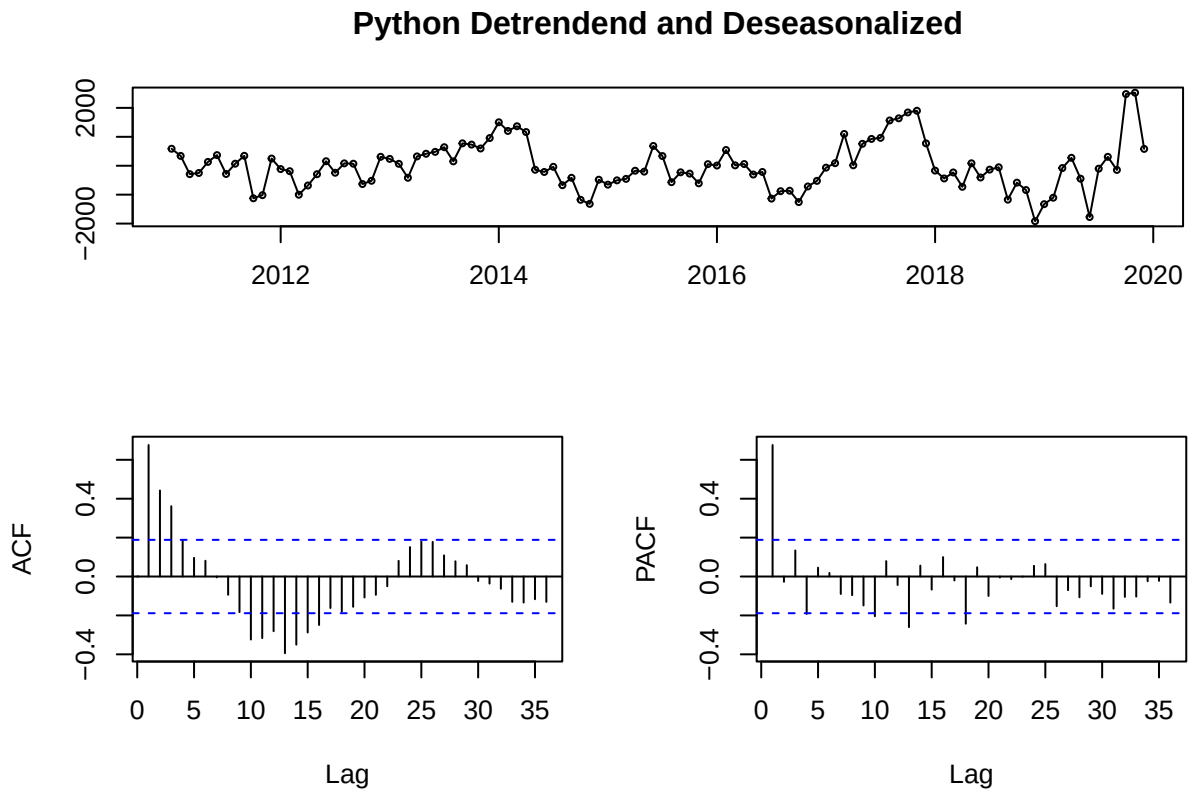
VARMA Modeling

The final ARIMA model and the exploratory data analysis suggests that there is a trend and seasonal component to our time series. Therefore, we will detrend and deseasonalize the data before fitting the VARMA model.

Python

Deseasonalize and Detrend

```
# Python
# Grab the months for the Python series
python.mon <- factor(cycle(python.ts), labels=month.abb)

# Creating a detrended and deseasonalized
# Detrend
python.lm <- lm(python.ts ~ time(python.ts), na.action = NULL)
python.detrend <- resid(python.lm)

# Deseasonalize
python.detrend.lm <- lm(python.detrend ~ python.mon, na.action = NULL)
python.detrend.deseason <- resid(python.detrend.lm)

tsdisplay(python.detrend.deseason, main = 'Python Detrendend and Deseasonalized')
```




```
# Combine the series
# Model 1 - Python
python.series <-  cbind(python.detrend.deseason, machine_learning.ts, classification.ts,
                    regression.ts, time_series.ts, cluster_analysis.ts)
```

Variable Selection

```
VARselect(python.series, lag.max = 13, type = 'none')$selection
```

```
## AIC(n)  HQ(n)  SC(n) FPE(n)
##     13      1      1      1
```

The VARselect function concludes that a VAR(1) model is the best fit to the data.

```
python.fit <- VAR(python.series, p = 1, type = 'none')
summary(python.fit)
```

```
##
## VAR Estimation Results:
## ========================
## Endogenous variables: python.detrend.deseason, machine_learning.ts, classification.ts, regression.ts
## Deterministic variables: none
## Sample size: 107
## Log Likelihood: -2992.646
## Roots of the characteristic polynomial:
## 1.011 0.9199 0.6774 0.3206 0.3206 0.3118
## Call:
## VAR(y = python.series, p = 1, type = "none")
##
##
## Estimation results for equation python.detrend.deseason:
## ========================================================
## python.detrend.deseason = python.detrend.deseason.l1 + machine_learning.ts.l1 + classification.ts.l1
##
##                             Estimate Std. Error t value Pr(>|t|)
## python.detrend.deseason.l1  0.67013    0.07355   9.111 8.11e-15 ***
## machine_learning.ts.l1      0.39098    0.68782   0.568    0.571
## classification.ts.l1        4.86766    6.85791   0.710    0.479
## regression.ts.l1           -5.42537    5.80533  -0.935    0.352
## time_series.ts.l1           0.24284    4.29441   0.057    0.955
## cluster_analysis.ts.l1     -2.65232    7.35109  -0.361    0.719
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 610.4 on 101 degrees of freedom
## Multiple R-Squared: 0.4685,  Adjusted R-squared: 0.4369
## F-statistic: 14.84 on 6 and 101 DF,  p-value: 4.253e-12
##
##
## Estimation results for equation machine_learning.ts:
## ====================================================
## machine_learning.ts = python.detrend.deseason.l1 + machine_learning.ts.l1 + classification.ts.l1 + re
##
##                             Estimate Std. Error t value Pr(>|t|)
## python.detrend.deseason.l1  0.002963   0.007171   0.413   0.6803
```

```
## machine_learning.ts.l1      0.813471   0.067067  12.129   <2e-16 ***
## classification.ts.l1        0.372261   0.668696   0.557   0.5790
## regression.ts.l1            0.711362   0.566062   1.257   0.2118
## time_series.ts.l1           0.763473   0.418737   1.823   0.0712 .
## cluster_analysis.ts.l1     -1.263630   0.716786  -1.763   0.0809 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 59.52 on 101 degrees of freedom
## Multiple R-Squared: 0.9794,  Adjusted R-squared: 0.9782
## F-statistic: 800.9 on 6 and 101 DF,  p-value: < 2.2e-16
##
##
## Estimation results for equation classification.ts:
## ================================================
## classification.ts = python.detrend.deseason.l1 + machine_learning.ts.l1 + classification.ts.l1 + reg
##
##                              Estimate Std. Error t value Pr(>|t|)
## python.detrend.deseason.l1 -0.0007343  0.0013456  -0.546   0.5865
## machine_learning.ts.l1     -0.0114503  0.0125841  -0.910   0.3650
## classification.ts.l1        0.6291672  0.1254694   5.015 2.28e-06 ***
## regression.ts.l1            0.2163720  0.1062118   2.037   0.0442 *
## time_series.ts.l1           0.0689901  0.0785688   0.878   0.3820
## cluster_analysis.ts.l1      0.1310930  0.1344926   0.975   0.3320
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 11.17 on 101 degrees of freedom
## Multiple R-Squared: 0.9617,  Adjusted R-squared: 0.9594
## F-statistic: 422.8 on 6 and 101 DF,  p-value: < 2.2e-16
##
##
## Estimation results for equation regression.ts:
## ============================================
## regression.ts = python.detrend.deseason.l1 + machine_learning.ts.l1 + classification.ts.l1 + regressi
##
##                              Estimate Std. Error t value Pr(>|t|)
## python.detrend.deseason.l1 -5.754e-05  1.525e-03  -0.038  0.96998
## machine_learning.ts.l1      2.772e-02  1.426e-02   1.943  0.05478 .
## classification.ts.l1        4.627e-02  1.422e-01   0.325  0.74556
## regression.ts.l1            3.657e-01  1.204e-01   3.038  0.00303 **
## time_series.ts.l1           1.973e-01  8.905e-02   2.215  0.02899 *
## cluster_analysis.ts.l1      1.788e-01  1.524e-01   1.173  0.24359
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 12.66 on 101 degrees of freedom
## Multiple R-Squared: 0.9598,  Adjusted R-squared: 0.9574
## F-statistic: 401.4 on 6 and 101 DF,  p-value: < 2.2e-16
##
##
```

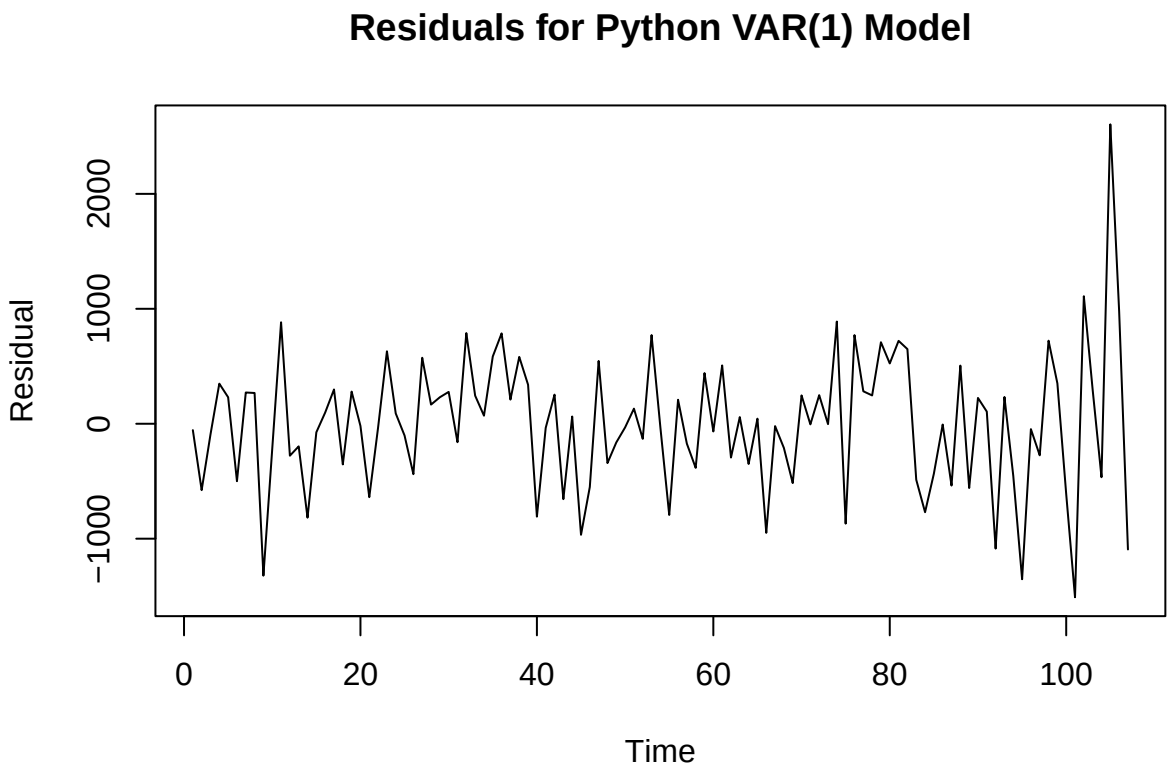
```
## Estimation results for equation time_series.ts:
## ===============================================
## time_series.ts = python.detrend.deseason.l1 + machine_learning.ts.l1 + classification.ts.l1 + regress
##
##                              Estimate Std. Error t value Pr(>|t|)
## python.detrend.deseason.l1 -0.003808   0.002226  -1.711   0.0901 .
## machine_learning.ts.l1      0.037734   0.020815   1.813   0.0728 .
## classification.ts.l1        0.361173   0.207535   1.740   0.0849 .
## regression.ts.l1            0.059839   0.175682   0.341   0.7341
## time_series.ts.l1           0.600359   0.129958   4.620 1.14e-05 ***
## cluster_analysis.ts.l1     -0.010559   0.222460  -0.047   0.9622
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 18.47 on 101 degrees of freedom
## Multiple R-Squared: 0.963,   Adjusted R-squared: 0.9608
## F-statistic: 437.6 on 6 and 101 DF,  p-value: < 2.2e-16
##
##
## Estimation results for equation cluster_analysis.ts:
## ====================================================
## cluster_analysis.ts = python.detrend.deseason.l1 + machine_learning.ts.l1 + classification.ts.l1 + r
##
##                              Estimate Std. Error t value Pr(>|t|)
## python.detrend.deseason.l1 -0.000679   0.001210  -0.561 0.575759
## machine_learning.ts.l1     -0.023510   0.011311  -2.079 0.040198 *
## classification.ts.l1        0.388061   0.112776   3.441 0.000844 ***
## regression.ts.l1            0.075714   0.095466   0.793 0.429582
## time_series.ts.l1           0.091683   0.070620   1.298 0.197157
## cluster_analysis.ts.l1      0.387401   0.120886   3.205 0.001810 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 10.04 on 101 degrees of freedom
## Multiple R-Squared: 0.9475,  Adjusted R-squared: 0.9444
## F-statistic:   304 on 6 and 101 DF,  p-value: < 2.2e-16
##
##
##
## Covariance matrix of residuals:
##                          python.detrend.deseason machine_learning.ts
## python.detrend.deseason                  372634             12443.1
## machine_learning.ts                       12443              3542.0
## classification.ts                          2033               360.0
## regression.ts                              1629               391.7
## time_series.ts                             3646               619.2
## cluster_analysis.ts                        1262               127.8
##                          classification.ts regression.ts time_series.ts
## python.detrend.deseason           2033.49       1628.64         3645.7
## machine_learning.ts                360.04        391.71          619.2
## classification.ts                  123.35         65.54          116.4
## regression.ts                       65.54        160.24          138.4
```

```
## time_series.ts                     116.40        138.44          340.2
## cluster_analysis.ts                 63.37         37.15           85.6
##                         cluster_analysis.ts
## python.detrend.deseason             1262.35
## machine_learning.ts                  127.82
## classification.ts                     63.37
## regression.ts                         37.15
## time_series.ts                        85.60
## cluster_analysis.ts                   98.90
##
## Correlation matrix of residuals:
##                         python.detrend.deseason machine_learning.ts
## python.detrend.deseason                  1.0000              0.3425
## machine_learning.ts                      0.3425              1.0000
## classification.ts                        0.2999              0.5447
## regression.ts                            0.2108              0.5199
## time_series.ts                           0.3238              0.5641
## cluster_analysis.ts                      0.2079              0.2160
##                         classification.ts regression.ts time_series.ts
## python.detrend.deseason            0.2999        0.2108         0.3238
## machine_learning.ts                0.5447        0.5199         0.5641
## classification.ts                  1.0000        0.4662         0.5682
## regression.ts                      0.4662        1.0000         0.5930
## time_series.ts                     0.5682        0.5930         1.0000
## cluster_analysis.ts                0.5737        0.2951         0.4667
##                         cluster_analysis.ts
## python.detrend.deseason              0.2079
## machine_learning.ts                  0.2160
## classification.ts                    0.5737
## regression.ts                        0.2951
## time_series.ts                       0.4667
## cluster_analysis.ts                  1.0000
```

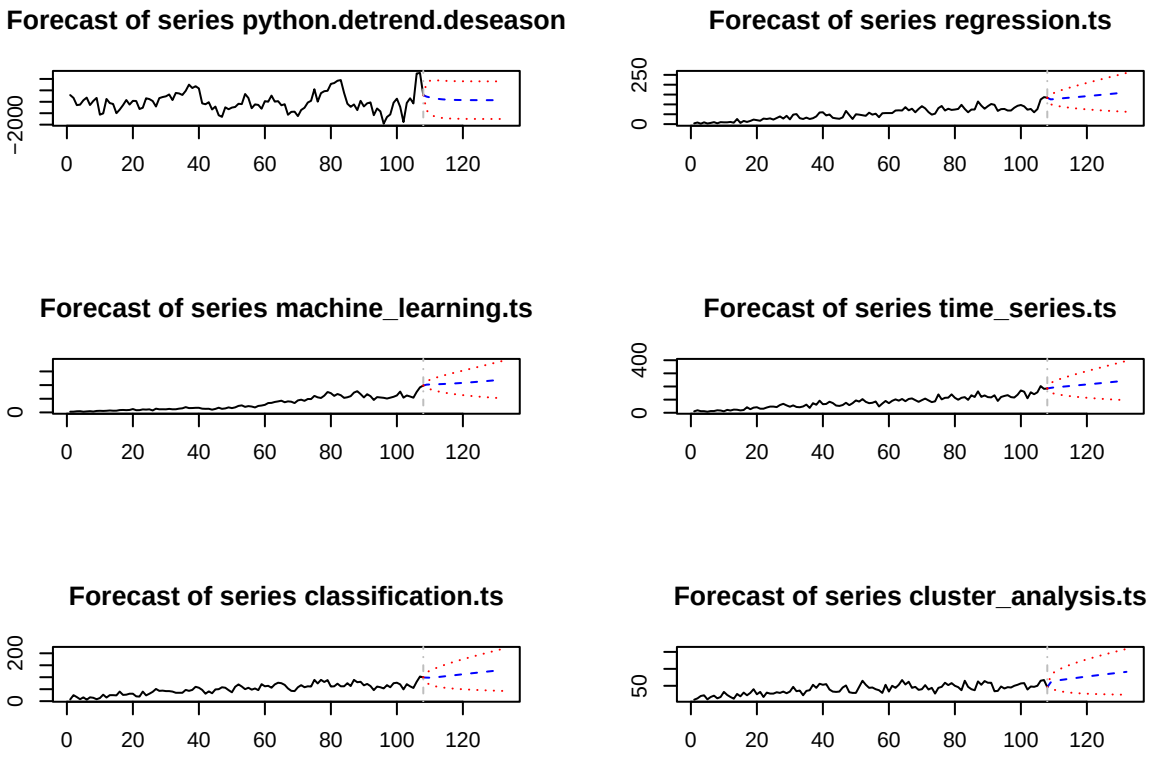
The summary of the VAR(1) on page X shows that the model for the detrended and deseasonalized Python series can be explained by any of the data science topics since all their p-values are above 0.05. The only variable that seems to predict the Python question count is the lagged variable of the Python series. Additionally, the R-squared for the model is 0.43 which suggests that the model does not have strong predictive power.

Model Validation

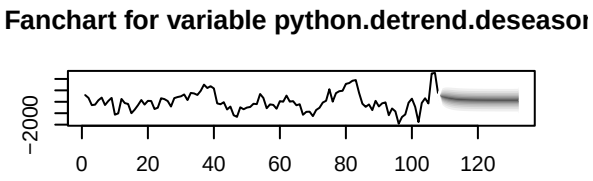
```
# Plot the residuals
plot(ts(resid(python.fit))[,1], main = 'Residuals for Python VAR(1) Model', ylab = 'Residual')
```

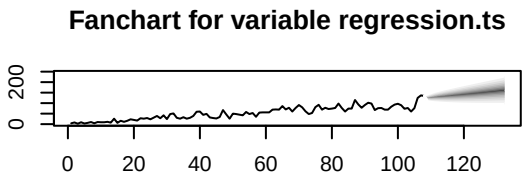


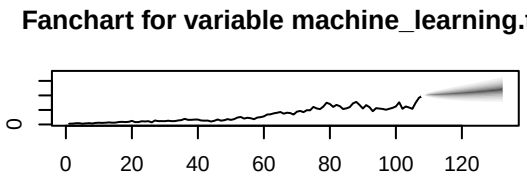

```
# Predictions
python.pred <- predict(python.fit, n.ahead = 24, ci = 0.95)
plot(python.pred)
```

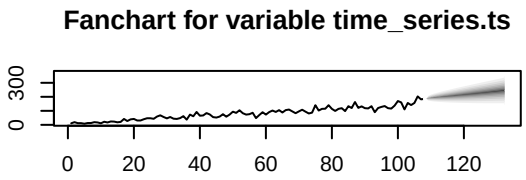


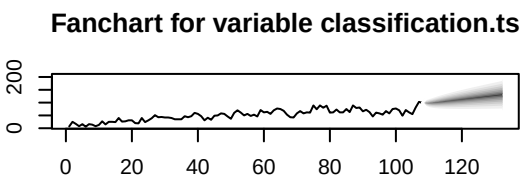

```
fanchart(python.pred)
```

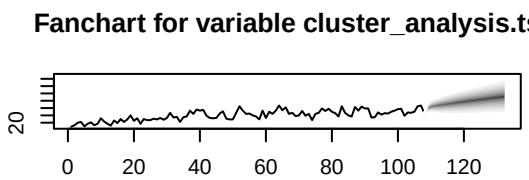




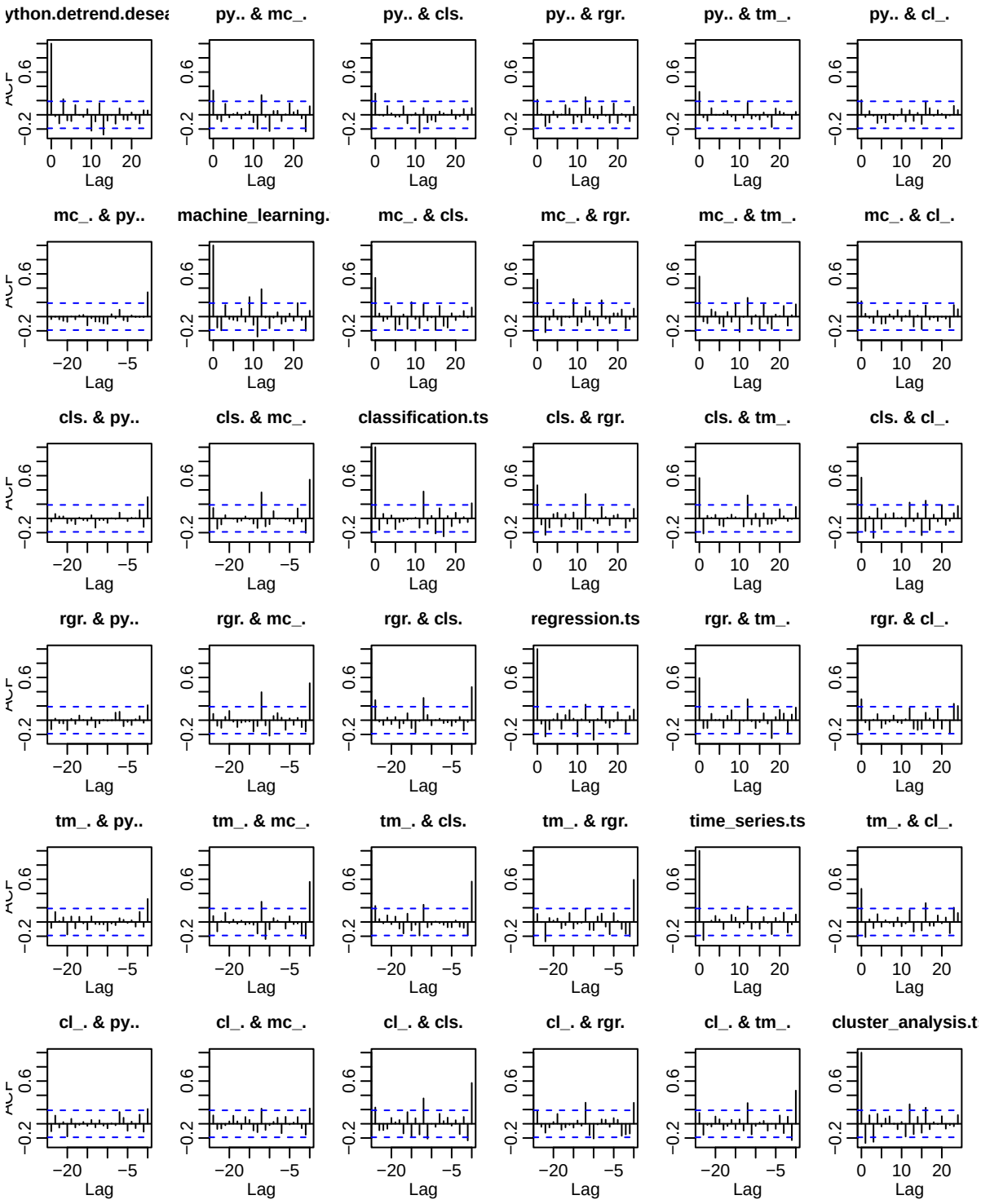









```
# ACF Plots
acf(resid(python.fit), 24)
```

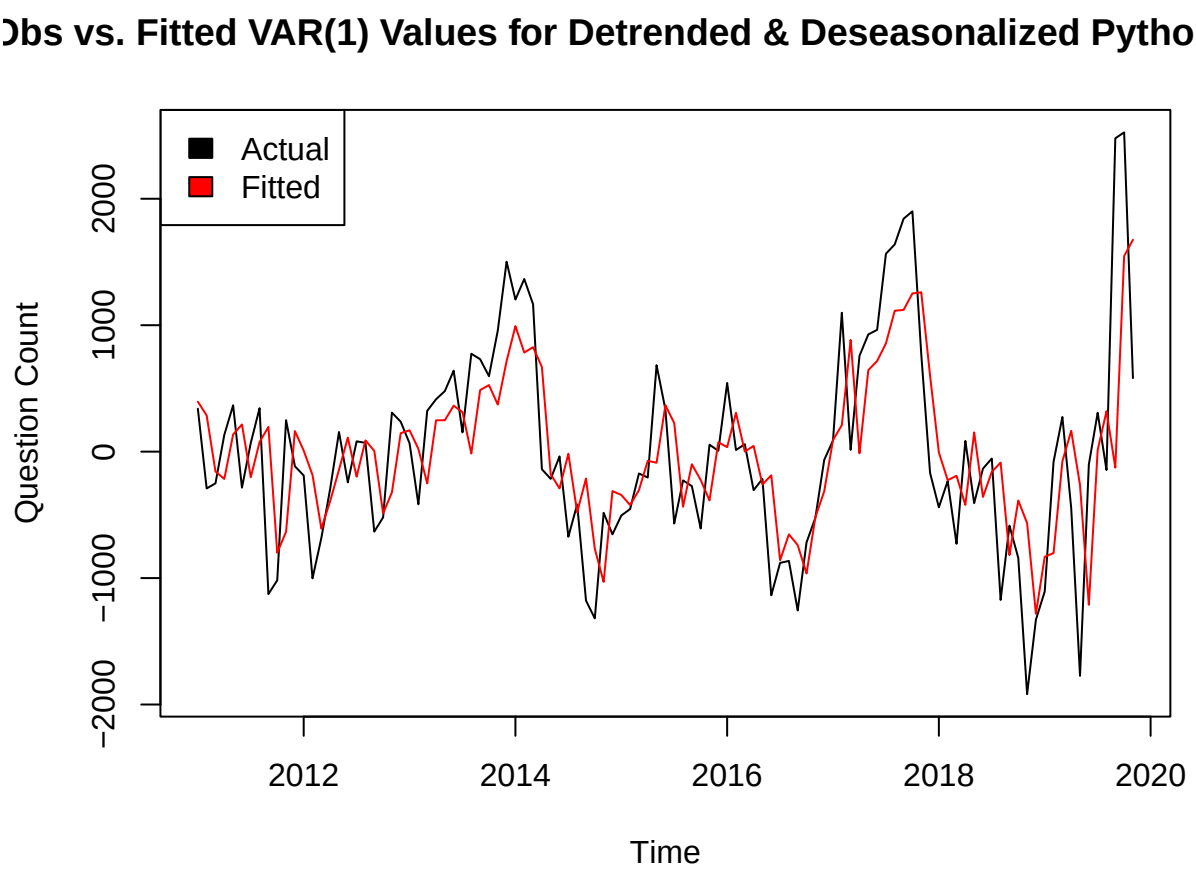


The plot of the residuals versus fitted values for the Python VAR(1) model shows higher variance in the last year of the data. The ACF and CCF plots occassionally show significant correlations around the year-long lag.

Forecasts

> Recall that before fitting the VARMA model, the original time series was detrended and deseasonalized in order to get the best model fit. But, for the forecasted series, it is better to report the predictions in their original scale. Therefore, here we are transforming the forecasted values from the VAR(1) model back into their original scale.

```
# Python
# Plot of actual vs fitted values
plot(ts(python.detrend.deseason[2:length(python.detrend.deseason)], start = c(2011,1), frequency = 12),
     main = paste("Obs vs. Fitted VAR(1) Values for Detrended & Deseasonalized Python Series"),
     ylab = paste("Question Count"))
lines(ts(python.detrend.deseason[2:length(python.detrend.deseason)] - python.fit$varresult$python.detren
         start = c(2011,1), frequency = 12), col="red")
legend("topleft",legend = c("Actual", "Fitted"), fill = c("black", "red"))
```



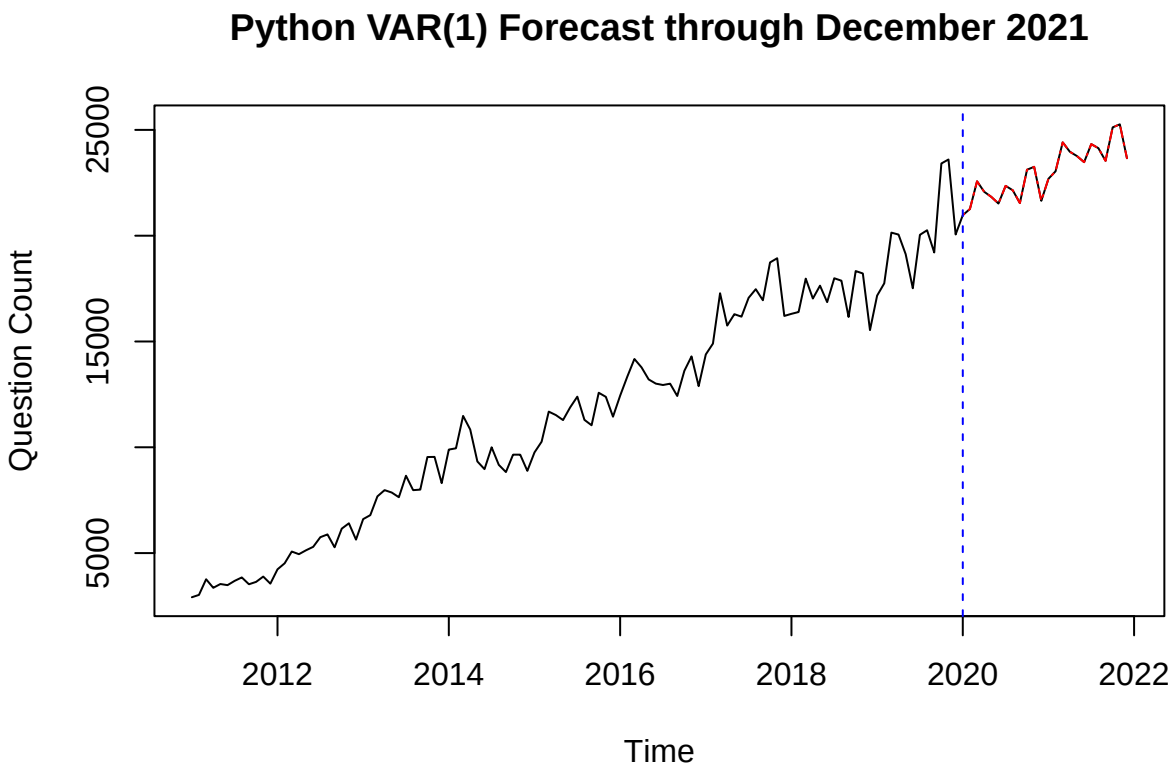

```
# Python
# Creates the time period for forecasting the trend
python.months.past.current.projection <- 24
python.forecast.time <- seq(max(time(python.ts)) + min(cycle(python.ts))/max(cycle(python.ts)),
                      max(time(python.ts)) + (python.months.past.current.projection/max(cycle(python.ts)),
                      by = min(cycle(python.ts)) / max(cycle(python.ts)))
```

```
python.fcsts <- NULL

# Creates a matrix that stores the forecasts for the linear model with trend
python.fcsts <- rbind(rep(coef(python.lm)[[1]],length(python.forecast.time)),
               coef(python.lm)[[2]]*python.forecast.time)
```

```
# getting seasonality model coefficient estimates
tmp <- matrix(rep(0, length(coef(python.detrend.lm))^2), ncol = length(coef(python.detrend.lm)))
diag(tmp) <- coef(python.detrend.lm)
tmp[1,] <- tmp[1,1] # since deasonalized model has dummy variables measuring difference from intercept
python.fcsts <- rbind(python.fcsts, rep(colSums(tmp), length.out = python.months.past.current.projectio
```

```
# Gets the forecasted values based on the VAR model and appends to matrix
python.fcst.detrend.deseason <- ts(python.pred$fcst$python.detrend.deseason[,1], start = c(2020,1), fre
python.fcsts <- rbind(python.fcsts, python.fcst.detrend.deseason)

# Now, each column is a forecast for a future month (24 cols = 2 years)
# each row (except the last one) is the forecasted value at that month associated with a coefficient
# from the original models fit when detrending/deasonalizing
# colSums will give us this forecast
python.fcsts <- ts(colSums(python.fcsts), start = c(2020,1), frequency = 12)
python.total <- ts(c(python.ts, python.fcsts), start = c(2011,1), frequency = 12)
```

```
# Plot the transformed predicted values
# Plotting over the whole time series initially so R sets the correct boundaries
plot(python.total,
     main = paste('Python VAR(1) Forecast through',
                  month.name[(max(time(python.total))
                              - floor(max(time(python.total))))*max(cycle(python.total)) + 1],
                  floor(max(time(python.total)))),
     ylab = 'Question Count')
# Plot the forecasted values
lines(python.fcsts, col = 'red', lty = 6)
# Cutoff for data and predicted values
abline(v = max(time(python.ts)) + 1/max(cycle(python.ts)), lty = 2, col = 'blue')
```




```
paste('Ending actual data count:', python.ts[length(python.ts)])
```

```
## [1] "Ending actual data count: 20058"
```

```
paste('Ending projection data count:', floor(python.fcsts[length(python.fcsts)]))
```

```
## [1] "Ending projection data count: 23661"
```

```
paste('Forecasted growth %:',round((floor(python.fcsts[length(python.fcsts)]) - python.ts[length(python
```

```
## [1] "Forecasted growth %: 17.96"
```

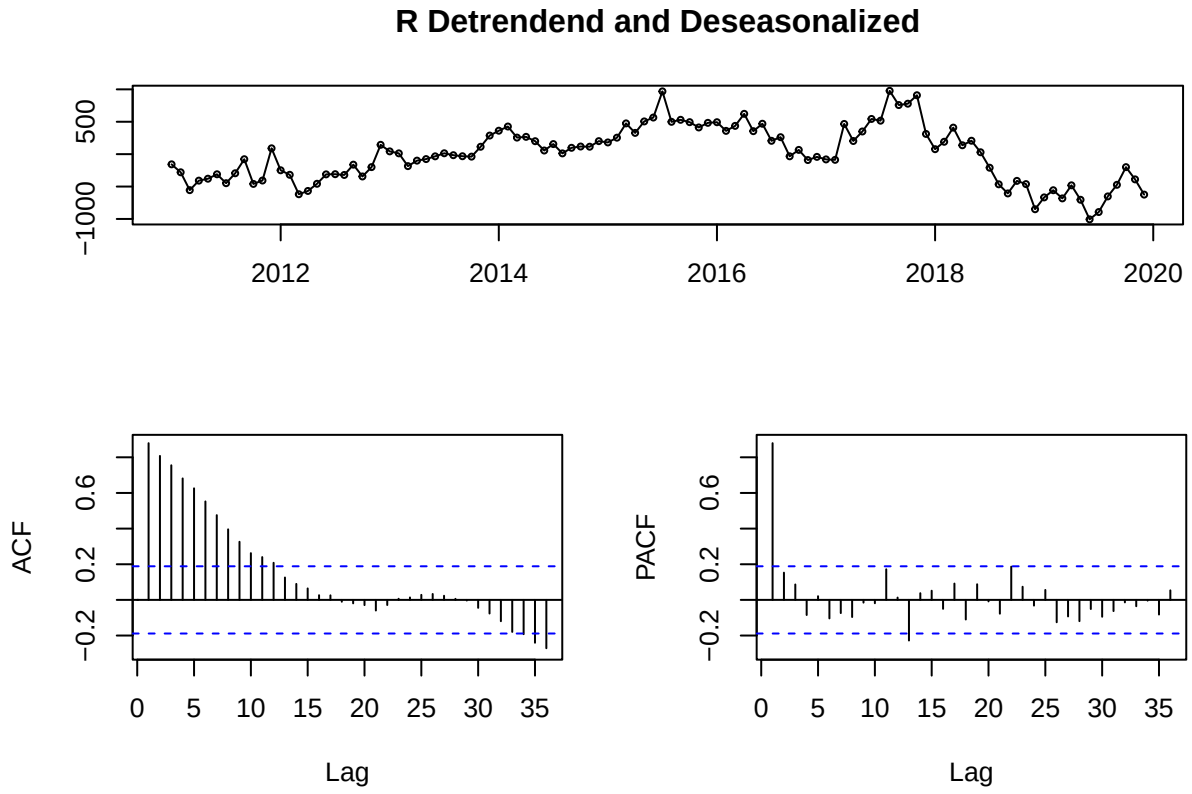
The forecasted values indicated that the positive trend will continue to 2022. Python's predicted number of questions is forecasted to grow from 20,058 in December 2019 to 23,661 in December 2022, a 17.96% growth rate.

R

Again, the final ARIMA model and the exploratory data analysis suggests that there is a trend and seasonal component to our time series. Therefore, we will detrend and deseasonalize the data before fitting the VARMA model.

Deseasonalize and Detrend

```
# Grab the months for the R series
r.mon <- factor(cycle(r.ts), labels=month.abb)

# Creating a detrended and deseasonalized
# Detrend
r.lm <- lm(r.ts ~ time(r.ts), na.action = NULL)
r.detrend <- resid(r.lm)

# Deseasonalize
r.detrend.lm <- lm(r.detrend ~ r.mon, na.action = NULL)
r.detrend.deseason <- resid(r.detrend.lm)

tsdisplay(r.detrend.deseason, main = 'R Detrendend and Deseasonalized')
```




```
# Model 2 - R
r.series <-  cbind(r.detrend.deseason, machine_learning.ts, classification.ts,
                   regression.ts, time_series.ts, cluster_analysis.ts)
```

Variable Selection

```
VARselect(r.series, lag.max = 13, type = 'none')$selection
```

```
## AIC(n)  HQ(n)  SC(n) FPE(n)
##     13      1      1     13
```

The VARselect model also suggests that a VAR(1) model is the best fit to the data.

```
r.fit <- VAR(r.series, p = 1, type = 'none')
summary(r.fit)
```

```
##
## VAR Estimation Results:
## ========================
## Endogenous variables: r.detrend.deseason, machine_learning.ts, classification.ts, regression.ts, time
## Deterministic variables: none
## Sample size: 107
## Log Likelihood: -2868.287
## Roots of the characteristic polynomial:
## 1.015 0.9244 0.8683 0.328 0.3082 0.3082
## Call:
## VAR(y = r.series, p = 1, type = "none")
##
##
## Estimation results for equation r.detrend.deseason:
## ===================================================
## r.detrend.deseason = r.detrend.deseason.l1 + machine_learning.ts.l1 + classification.ts.l1 + regressi
##
##                         Estimate Std. Error t value Pr(>|t|)
## r.detrend.deseason.l1    0.87379    0.04623  18.902  < 2e-16 ***
## machine_learning.ts.l1  -0.23731    0.20841  -1.139 0.257540
## classification.ts.l1     7.78787    2.15498   3.614 0.000472 ***
## regression.ts.l1        -2.57262    1.78455  -1.442 0.152506
## time_series.ts.l1        0.73486    1.36172   0.540 0.590622
## cluster_analysis.ts.l1  -6.17744    2.24920  -2.747 0.007133 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 186.8 on 101 degrees of freedom
## Multiple R-Squared: 0.8179,  Adjusted R-squared: 0.8071
## F-statistic: 75.63 on 6 and 101 DF,  p-value: < 2.2e-16
##
##
## Estimation results for equation machine_learning.ts:
## ====================================================
## machine_learning.ts = r.detrend.deseason.l1 + machine_learning.ts.l1 + classification.ts.l1 + regress
##
##                          Estimate Std. Error t value Pr(>|t|)
## r.detrend.deseason.l1   -0.009753   0.014714  -0.663   0.5089
## machine_learning.ts.l1   0.816558   0.066338  12.309   <2e-16 ***
## classification.ts.l1     0.488736   0.685923   0.713   0.4778
## regression.ts.l1         0.740550   0.568015   1.304   0.1953
```

```
## time_series.ts.l1       0.670931   0.433429   1.548   0.1248
## cluster_analysis.ts.l1 -1.268998   0.715911  -1.773   0.0793 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 59.44 on 101 degrees of freedom
## Multiple R-Squared: 0.9795,  Adjusted R-squared: 0.9782
## F-statistic: 803.1 on 6 and 101 DF,  p-value: < 2.2e-16
##
##
## Estimation results for equation classification.ts:
## ================================================
## classification.ts = r.detrend.deseason.l1 + machine_learning.ts.l1 + classification.ts.l1 + regressi
##
##                          Estimate Std. Error t value Pr(>|t|)
## r.detrend.deseason.l1  -0.0001731  0.0027685  -0.063   0.9503
## machine_learning.ts.l1 -0.0124192  0.0124817  -0.995   0.3221
## classification.ts.l1    0.6282735  0.1290598   4.868 4.17e-06 ***
## regression.ts.l1        0.2203418  0.1068749   2.062   0.0418 *
## time_series.ts.l1       0.0710195  0.0815517   0.871   0.3859
## cluster_analysis.ts.l1  0.1301398  0.1347022   0.966   0.3363
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 11.18 on 101 degrees of freedom
## Multiple R-Squared: 0.9616,  Adjusted R-squared: 0.9593
## F-statistic: 421.6 on 6 and 101 DF,  p-value: < 2.2e-16
##
##
## Estimation results for equation regression.ts:
## ============================================
## regression.ts = r.detrend.deseason.l1 + machine_learning.ts.l1 + classification.ts.l1 + regression.t
##
##                         Estimate Std. Error t value Pr(>|t|)
## r.detrend.deseason.l1  0.002147   0.003126   0.687  0.49370
## machine_learning.ts.l1 0.027810   0.014094   1.973  0.05121 .
## classification.ts.l1   0.022868   0.145732   0.157  0.87562
## regression.ts.l1       0.356656   0.120681   2.955  0.00389 **
## time_series.ts.l1      0.214878   0.092087   2.333  0.02161 *
## cluster_analysis.ts.l1 0.180630   0.152103   1.188  0.23780
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 12.63 on 101 degrees of freedom
## Multiple R-Squared: 0.9599,  Adjusted R-squared: 0.9576
## F-statistic: 403.4 on 6 and 101 DF,  p-value: < 2.2e-16
##
##
## Estimation results for equation time_series.ts:
## ============================================
## time_series.ts = r.detrend.deseason.l1 + machine_learning.ts.l1 + classification.ts.l1 + regression.
```

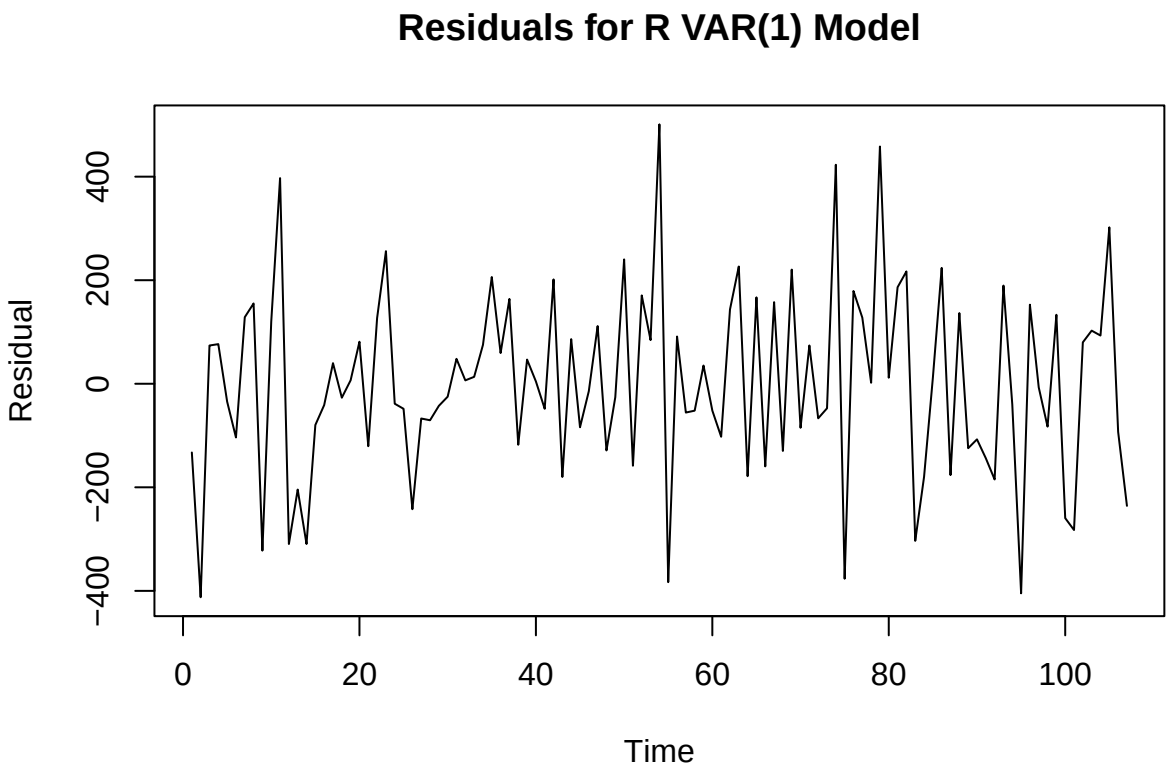
```
##
##                          Estimate Std. Error t value Pr(>|t|)
## r.detrend.deseason.l1  -0.009391   0.004543  -2.067   0.0413 *
## machine_learning.ts.l1  0.032040   0.020484   1.564   0.1209
## classification.ts.l1    0.448254   0.211798   2.116   0.0368 *
## regression.ts.l1        0.117166   0.175391   0.668   0.5056
## time_series.ts.l1       0.542339   0.133833   4.052 9.97e-05 ***
## cluster_analysis.ts.l1 -0.022992   0.221058  -0.104   0.9174
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 18.35 on 101 degrees of freedom
## Multiple R-Squared: 0.9634,  Adjusted R-squared: 0.9613
## F-statistic: 443.5 on 6 and 101 DF,  p-value: < 2.2e-16
##
##
## Estimation results for equation cluster_analysis.ts:
## ====================================================
## cluster_analysis.ts = r.detrend.deseason.l1 + machine_learning.ts.l1 + classification.ts.l1 + regress
##
##                          Estimate Std. Error t value Pr(>|t|)
## r.detrend.deseason.l1  -0.001159   0.002486  -0.466 0.642056
## machine_learning.ts.l1 -0.024484   0.011208  -2.185 0.031236 *
## classification.ts.l1    0.398022   0.115890   3.434 0.000863 ***
## regression.ts.l1        0.083706   0.095969   0.872 0.385160
## time_series.ts.l1       0.085497   0.073230   1.168 0.245751
## cluster_analysis.ts.l1  0.385638   0.120957   3.188 0.001906 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 10.04 on 101 degrees of freedom
## Multiple R-Squared: 0.9475,  Adjusted R-squared: 0.9444
## F-statistic: 303.7 on 6 and 101 DF,  p-value: < 2.2e-16
##
##
##
## Covariance matrix of residuals:
##                     r.detrend.deseason machine_learning.ts classification.ts
## r.detrend.deseason             34877.2              2071.9            440.96
## machine_learning.ts             2071.9              3533.5            359.07
## classification.ts                441.0               359.1            123.66
## regression.ts                    242.4               395.0             65.49
## time_series.ts                   677.6               597.6            118.44
## cluster_analysis.ts              395.9               125.6             63.68
##                     regression.ts time_series.ts cluster_analysis.ts
## r.detrend.deseason         242.41         677.57              395.95
## machine_learning.ts        394.99         597.55              125.56
## classification.ts           65.49         118.44               63.68
## regression.ts              159.46         141.79               37.42
## time_series.ts             141.79         336.43               86.11
## cluster_analysis.ts         37.42          86.11               99.08
##
```

```
## Correlation matrix of residuals:
##                     r.detrend.deseason machine_learning.ts classification.ts
## r.detrend.deseason              1.0000              0.1866            0.2123
## machine_learning.ts             0.1866              1.0000            0.5432
## classification.ts               0.2123              0.5432            1.0000
## regression.ts                   0.1028              0.5262            0.4663
## time_series.ts                  0.1978              0.5481            0.5807
## cluster_analysis.ts             0.2130              0.2122            0.5753
##                     regression.ts time_series.ts cluster_analysis.ts
## r.detrend.deseason         0.1028         0.1978              0.2130
## machine_learning.ts        0.5262         0.5481              0.2122
## classification.ts          0.4663         0.5807              0.5753
## regression.ts              1.0000         0.6122              0.2977
## time_series.ts             0.6122         1.0000              0.4716
## cluster_analysis.ts        0.2977         0.4716              1.0000
```

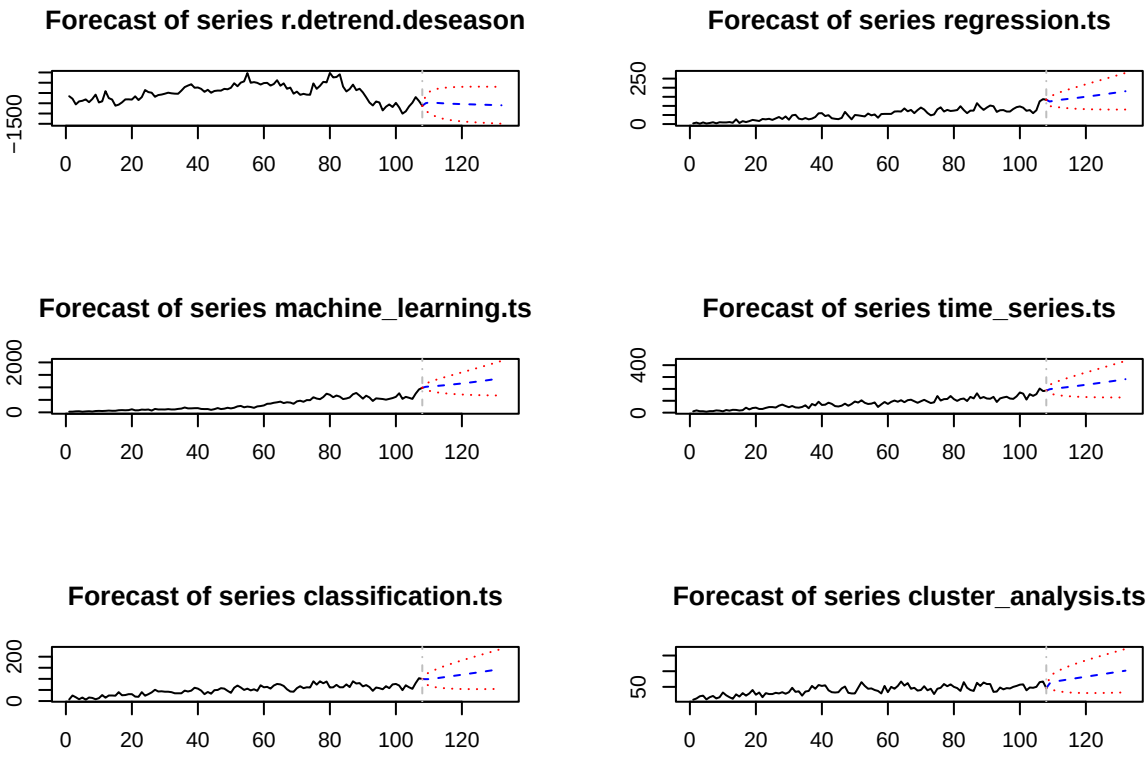
Unlike the Python model, the VAR(1) model for R has two variables that significant contribute to the model: classification and cluster analysis. Both of these variables have p-values below 0.05 which suggests that they explain a significant amount of variance in the Python series. Additionally, the R-squared value is 0.81 which suggests that the model has strong predictive power.

Model Validation

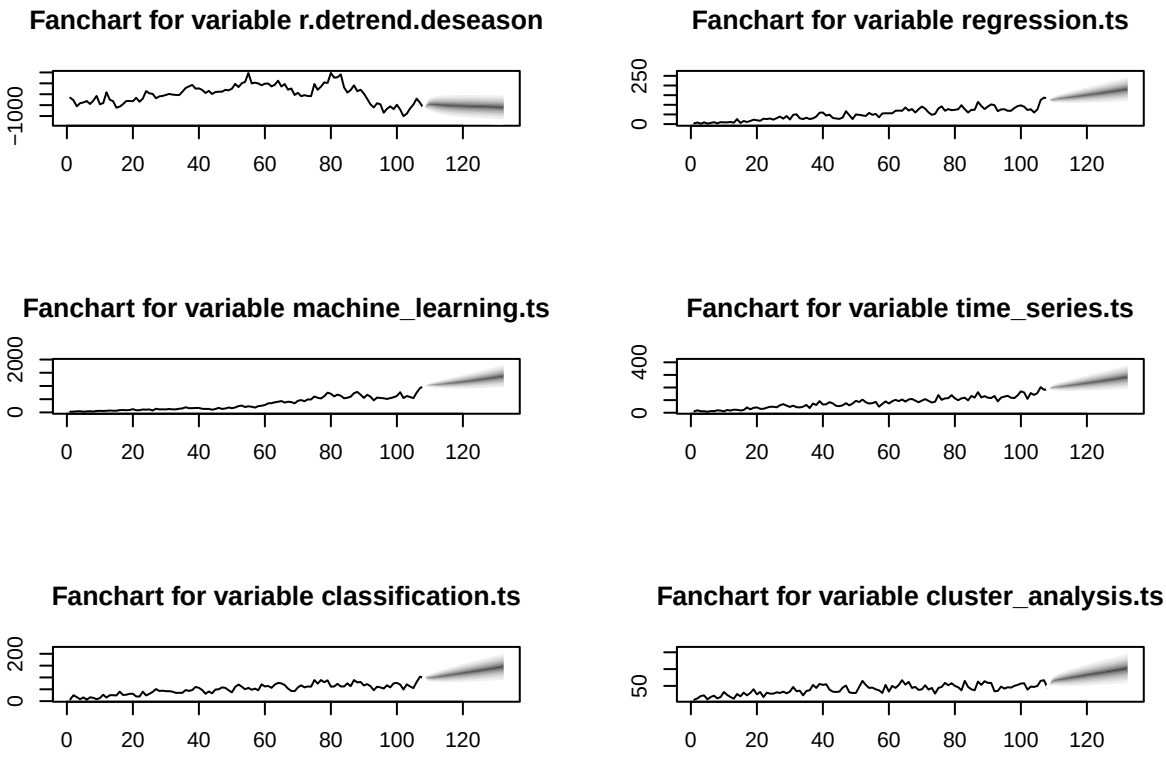
```
plot(ts(resid(r.fit))[,1], main = 'Residuals for R VAR(1) Model', ylab = 'Residual')
```



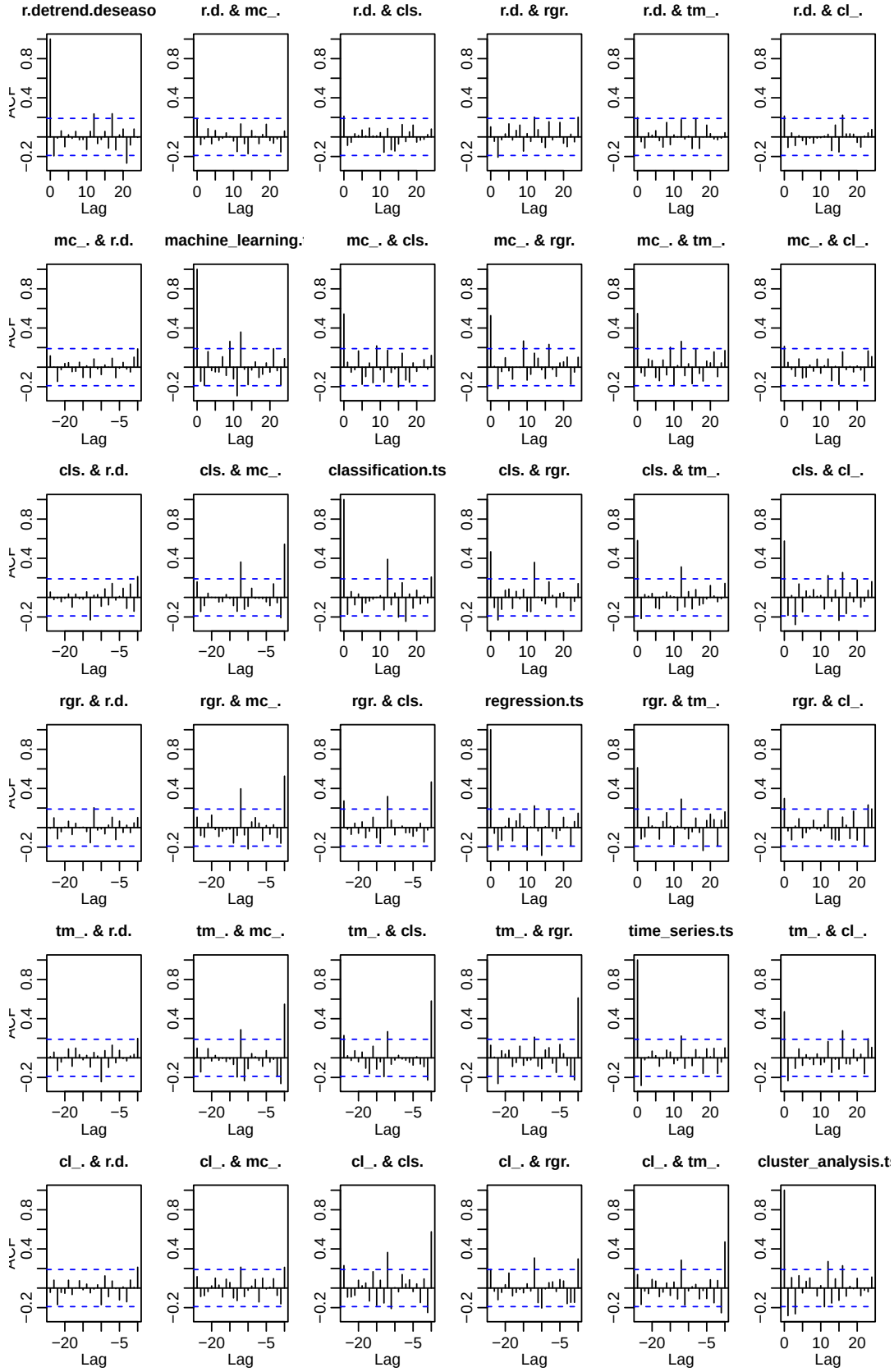

```
r.pred <- predict(r.fit, n.ahead = 24, ci = 0.95)
plot(r.pred)
```




```
fanchart(r.pred)
```



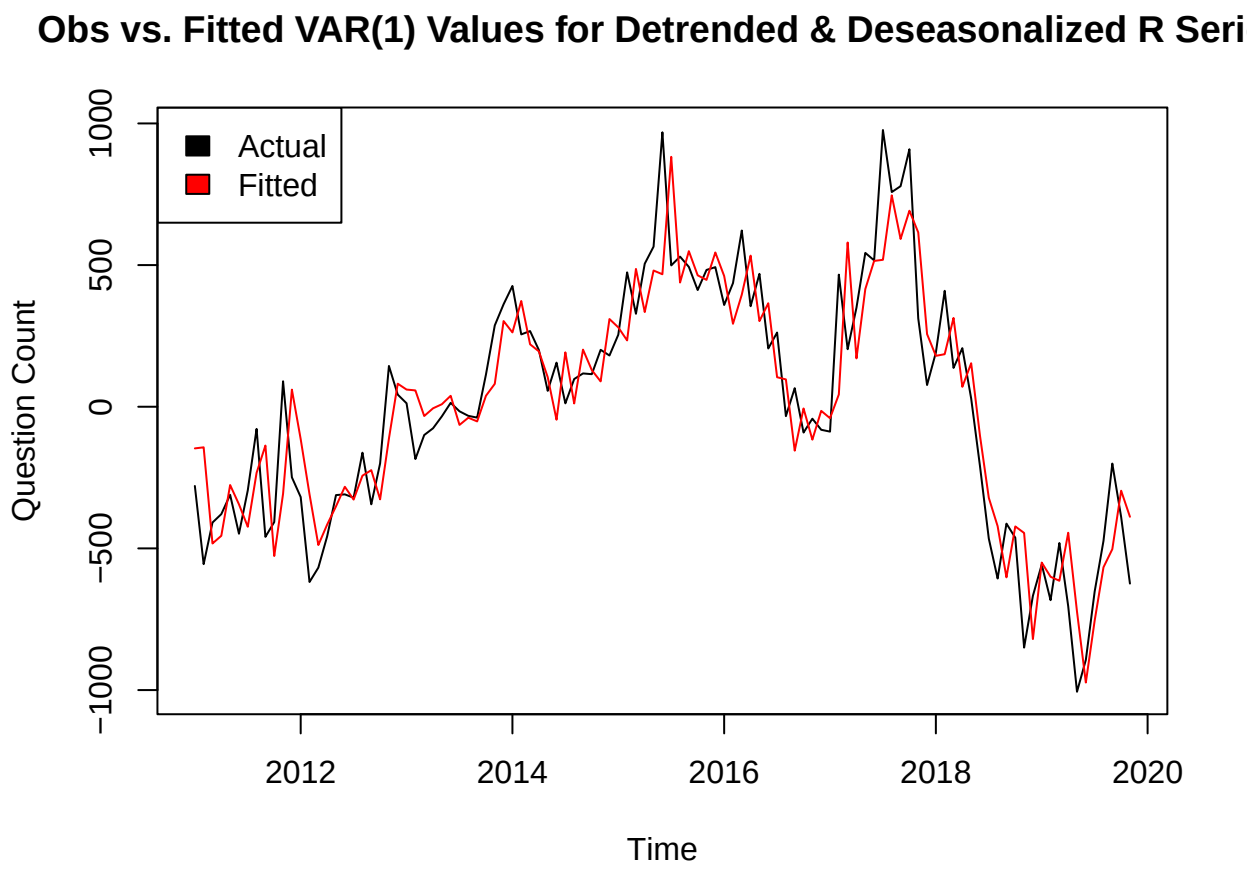

```
acf(resid(r.fit), 24)
```


r.detrend.deseaso
r.d. & mc_.
r.d. & cls.
r.d. & rgr.
r.d. & tm_.
r.d. & cl_.
mc_. & r.d.
machine_learning.
mc_. & cls.
mc_. & rgr.
mc_. & tm_.
mc_. & cl_.
cls. & r.d.
cls. & mc_.
classification.ts
cls. & rgr.
cls. & tm_.
cls. & cl_.
rgr. & r.d.
rgr. & mc_.
rgr. & cls.
regression.ts
rgr. & tm_.
rgr. & cl_.
tm_. & r.d.
tm_. & mc_.
tm_. & cls.
tm_. & rgr.
time_series.ts
tm_. & cl_.
cl_. & r.d.
cl_. & mc_.
cl_. & cls.
cl_. & rgr.
cl_. & tm_.
cluster_analysis.t
ACF
Lag

The residuals for the detrended and deasonalized R series follow a stationary behavior, with a constant mean and variance. The ACF and PACF plots also show less instances of significant correlations around the annual lag point.

Forecasts

```
# R
# Plot of actuals vs fitted values
plot(ts(r.detrend.deseason[2:length(r.detrend.deseason)], start = c(2011,1), frequency = 12),
     main = paste("Obs vs. Fitted VAR(1) Values for Detrended & Deseasonalized R Series"),
     ylab = paste("Question Count"))
lines(ts(r.detrend.deseason[2:length(r.detrend.deseason)] - r.fit$varresult$r.detrend.deseason$residuals
         start = c(2011,1), frequency = 12), col="red")
legend("topleft",legend = c("Actual", "Fitted"), fill = c("black", "red"))
```



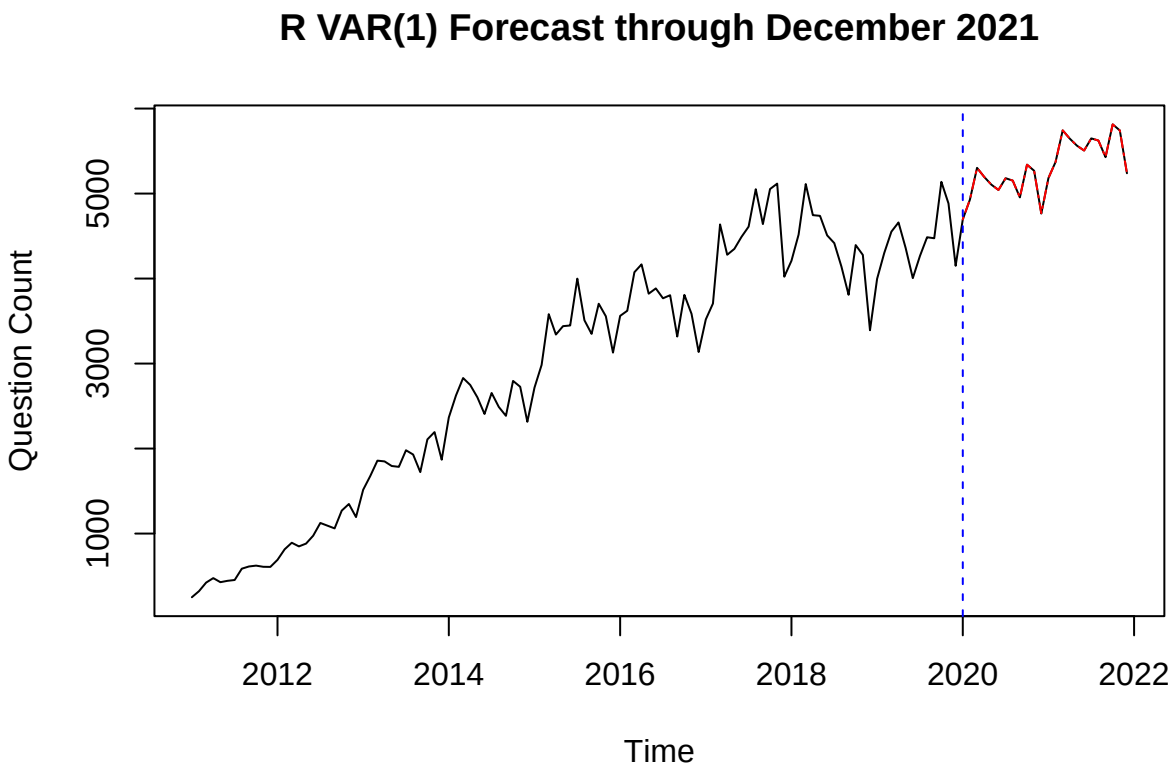

```
# R
# Creates the time period for forecasting the trend
r.months.past.current.projection <- 24
r.forecast.time <- seq(max(time(r.ts)) + min(cycle(r.ts))/max(cycle(r.ts)),
                       max(time(r.ts)) + (r.months.past.current.projection/max(cycle(r.ts))),
                       by = min(cycle(r.ts)) / max(cycle(r.ts)))
```

```
r.fcsts <- NULL
# creates a matrix that stores the forecasts for the linear model with trend
r.fcsts <- rbind(rep(coef(r.lm)[[1]],length(r.forecast.time)),
                 coef(r.lm)[[2]]*r.forecast.time)
```

```
# getting seasonality model coefficient estimates
tmp <- matrix(rep(0, length(coef(r.detrend.lm))^2), ncol = length(coef(r.detrend.lm)))
diag(tmp) <- coef(r.detrend.lm)
tmp[1,] <- tmp[1,1] # since deasonalized model has dummy variables measuring difference from intercept
r.fcsts <- rbind(r.fcsts, rep(colSums(tmp), length.out = r.months.past.current.projection))
```

```
# R translated forecasts
# Get the forecasted values based on the VAR model and appends to matrix
r.fcst.detrend.deseason <- ts(r.pred$fcst$r.detrend.deseason[,1], start = c(2020,1), frequency = 12)
r.fcsts <- rbind(r.fcsts, r.fcst.detrend.deseason)

# Now, each column is a forecast for a future month (24 cols = 2 years)
# each row (except the last one) is the forecasted value at that month associated with a coefficient
# from the original models fit when detrending/deasonalizing
# colSums will give us this forecast
r.fcsts <- ts(colSums(r.fcsts), start = c(2020,1), frequency = 12)
r.total <- ts(c(r.ts, r.fcsts), start = c(2011,1), frequency = 12)
```

```
# Plot the transformed forecasted values
# Plotting over the whole time series initially so R sets the correct boundaries
plot(r.total, main = paste('R VAR(1) Forecast through',
             month.name[(max(time(r.total)) - floor(max(time(r.total))))*max(cycle(r.total)) + 1],
             floor(max(time(r.total)))),
     ylab = 'Question Count')
# Plot the forecasted values
lines(r.fcsts, col = 'red', lty = 6)
# Line to show the divide between actual and predicted values
abline(v = max(time(r.ts)) + 1/max(cycle(r.ts)), lty = 2, col = 'blue')
```




```
paste('Ending projection data count:', floor(r.fcsts[length(r.fcsts)]))
```

```
## [1] "Ending projection data count: 5238"
```

```
paste('Ending actual data count:', r.ts[length(r.ts)])
```

```
## [1] "Ending actual data count: 4150"
```

```
paste('Forecasted growth %:',round((floor(r.fcsts[length(r.fcsts)]) - r.ts[length(r.ts)])/r.ts[length(r
```

```
## [1] "Forecasted growth %: 26.22"
```

The forecasted values indicated that the positive trend will continue to 2022. R's predicted number of questions is forecasted to grow from 4,150 in December 2020 to 5,238 in December 2021, a 26.2% growth rate.