Model Fitting of Time Series Data
With Application in R

Nicholas Gallo
Time Series Analysis
December 10, 2018

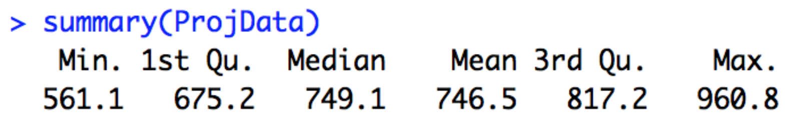
Introduction

This report is meant to illustrate my general thought process throughout the final project and serve as a guide to general analysis of time series datasets. The structure of this report is meant to be chronological, with each section of the report mirroring a section of the R script that I developed. All code is documented and meant to be educational. The script takes a general approach to the entire analysis process and creates a flexible control flow that allows the script to be a universal and general tool for basic analysis of time series data.

For purposes of testing, I sourced a dataset from datamarket.com. This dataset covered the monthly milk production per cow from January 1962 to December 1975. Months were also adjusted for the current duration of time.

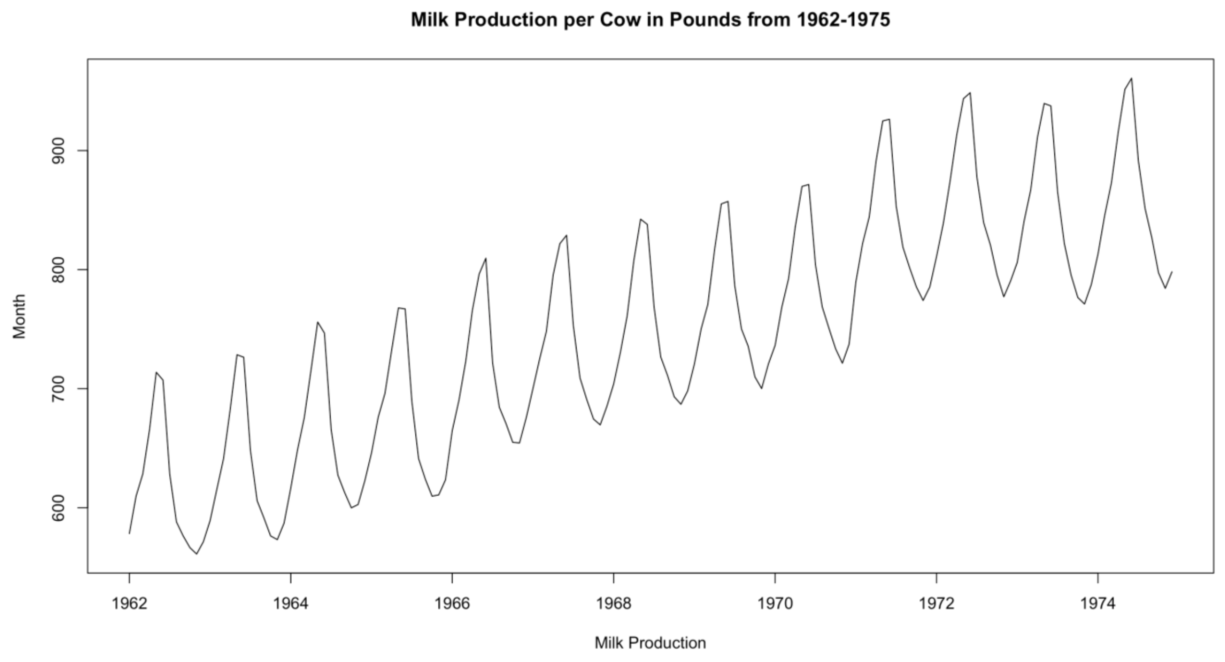
Analysis of the Data

Once the dataset was loaded into R and converted into a monthly time series dataset, analysis was done on the set, showing that it contains a mean of 746.5. Looking at the time series plot shows an upward trend and variability along the trend. Decomposing the dataset exhibits both the trend and seasonal components.

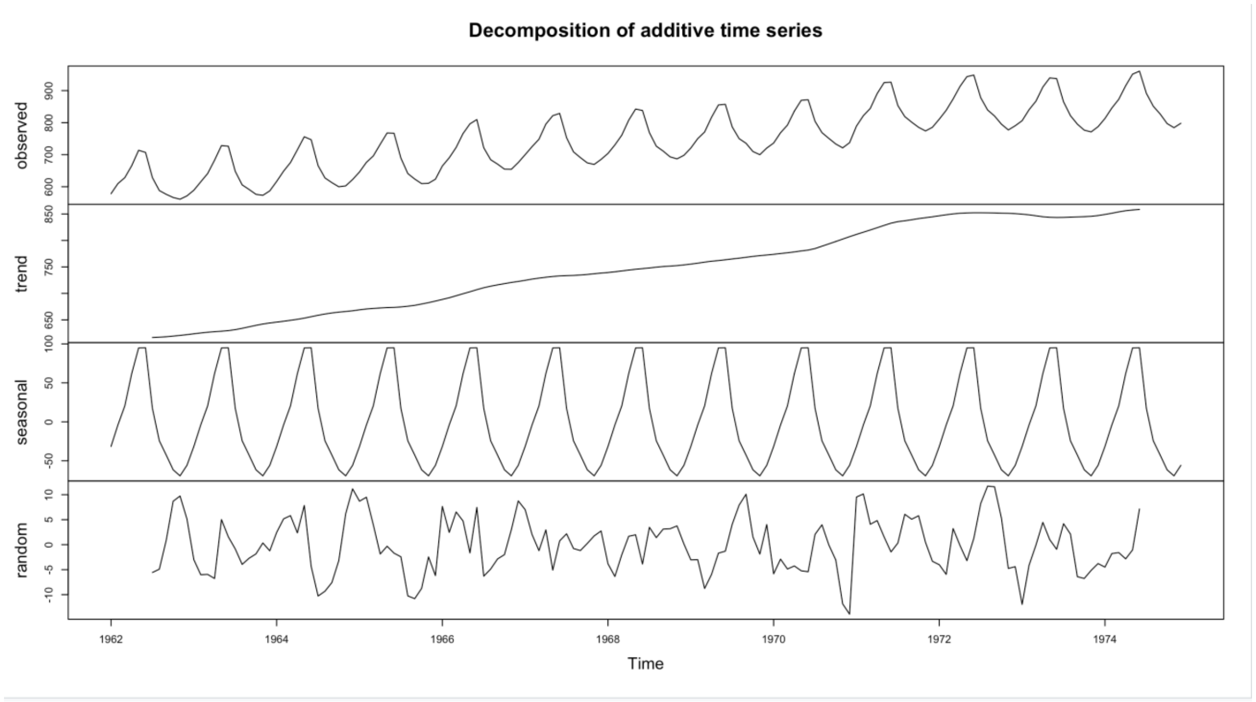


```
> summary(ProjData)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  561.1   675.2   749.1   746.5   817.2   960.8 
```

Summary of Data, with mean value of 746.5



The plotted time series data

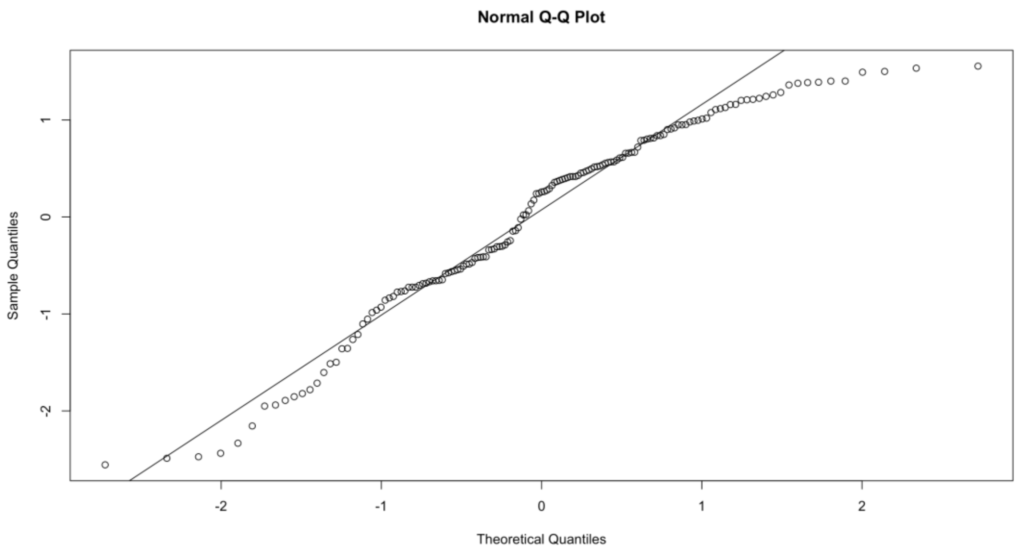


Decomposed time series data, showing the overall increasing trend and seasonal element. White noise appears to be normal and uncorrelated

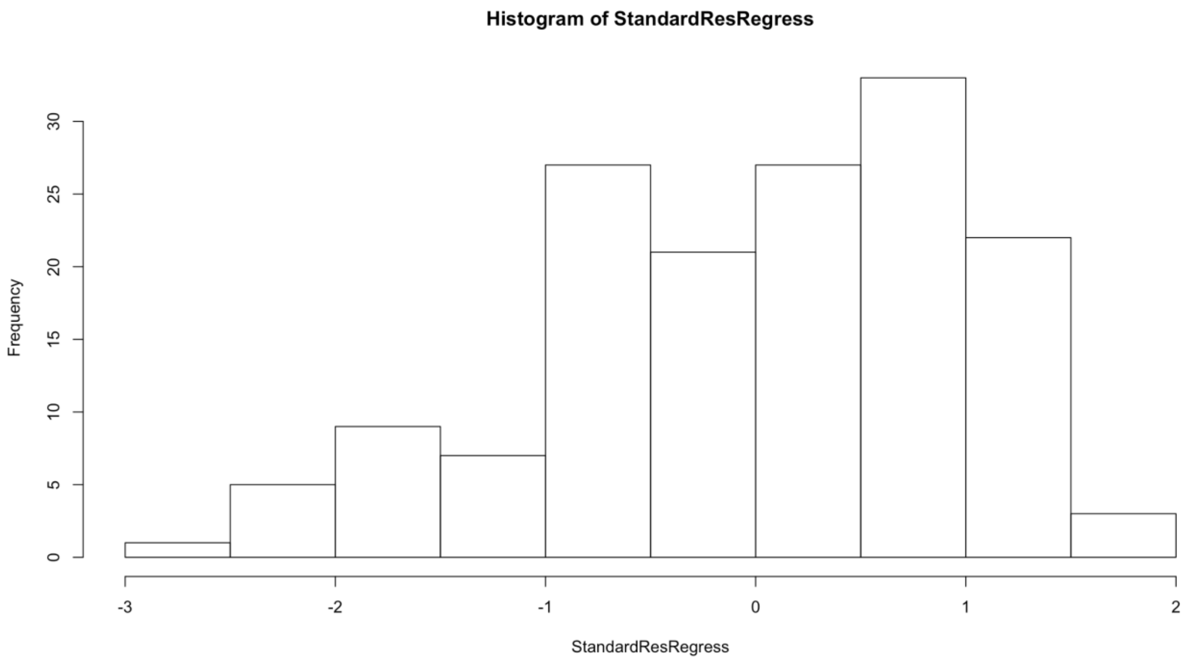
Linear Regression of the Model and Analysis of Models

Taking the project data, a regression model was fit using the lm() function. My function indicates that any given value in the project data set is equal to the value before it (lag one). The R code for the model is included in the R code section.

Analyzing the residuals of the linear regression model shows relative normality, indicating that the linear model could be a good fit for the dataset.



QQplot of the residuals



Residuals are slightly skewed, however, the residuals are relatively normally distributed

Fitting First Order Auto-Regressive Model

A first order auto-regressive (AR(1)) model was fit to the training set as a test model. The coefficient was estimated using the Arima function Then, the next element in the series after the end of the training subset was predicted using the AR(1) prediction function.

Forecast error was tested using the test set by comparing the first value of the test set to the forecasted error. The forecast error was relatively low, indicating that the AR(1) model may a decent fit for the data. However, this still does not account for the seasonal component of the dataset. Checking Lower and Upper 95% confidence intervals of the predicted value also showed a large range in which the data point could fall, indicating that the model will not be suitable for long term predictions.

Fitting Custom ARIMA Model

The custom ARIMA model used was:

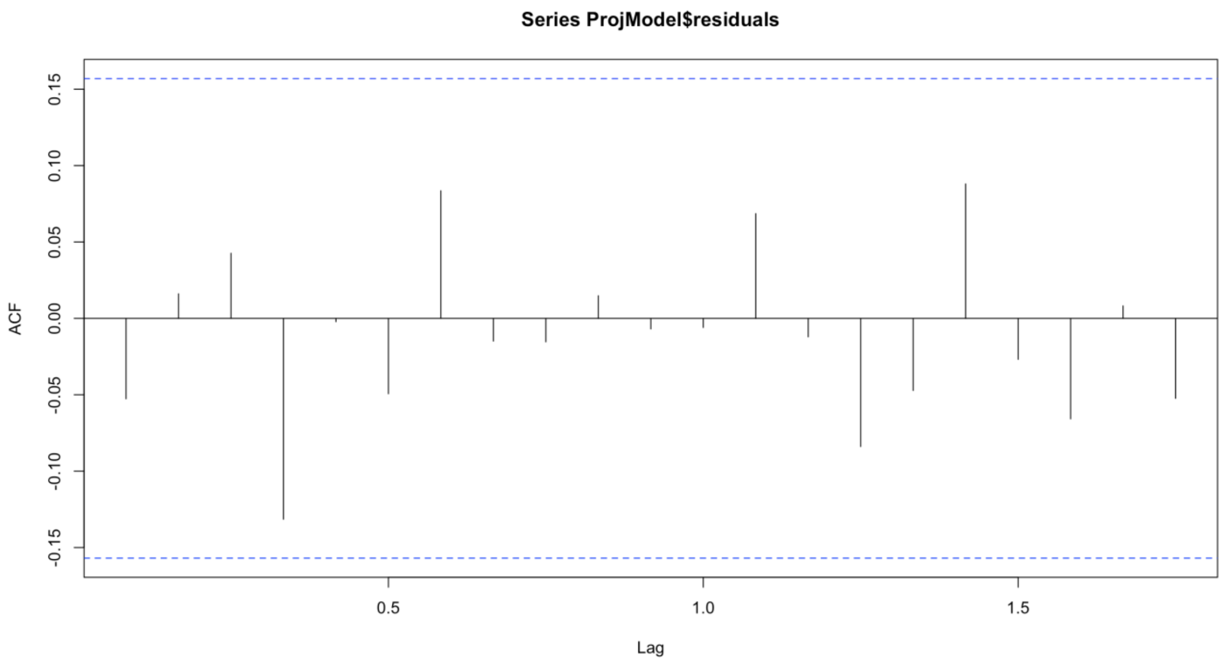
$$Arima(1,0,0)(1,1,1)_{12}$$

The model, using the Arima function, was fit as a first order auto-regressive model with a monthly seasonal auto-regressive and moving average component, with a drift. Analyzing the residuals shows normality, indicating that the model is most likely a good fit.

```
                Box-Pierce test

data:  ProjModel$residuals
X-squared = 0.43256, df = 1, p-value = 0.5107
```

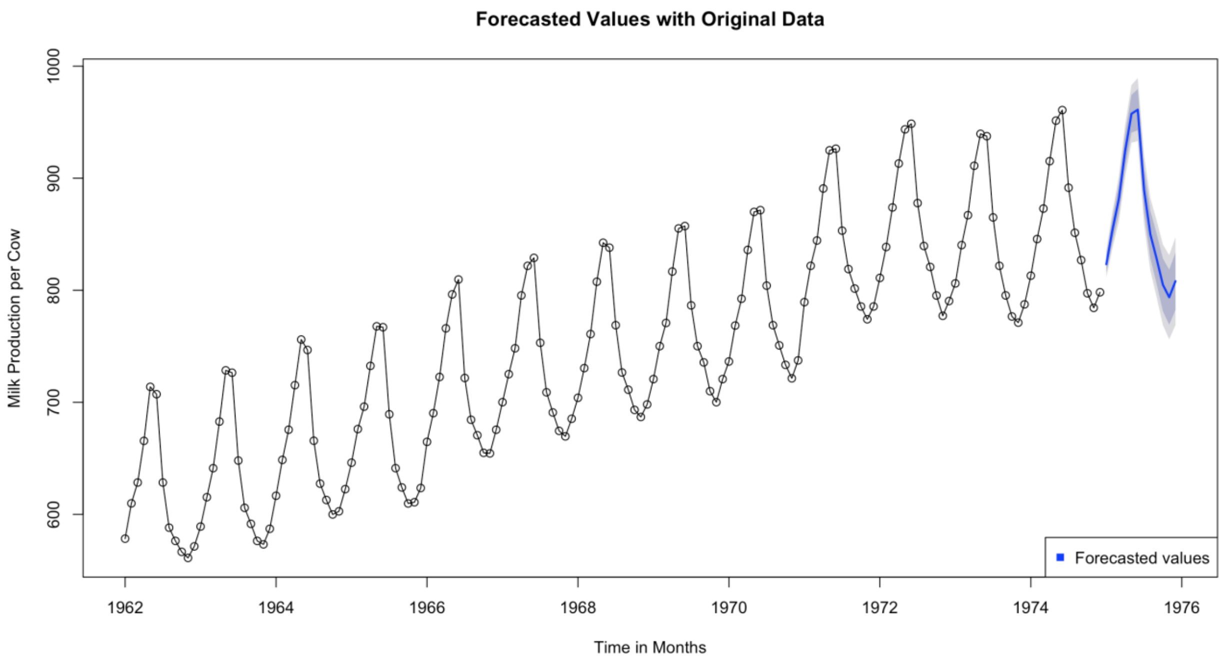
Box-Pierce test, p-value is suspect



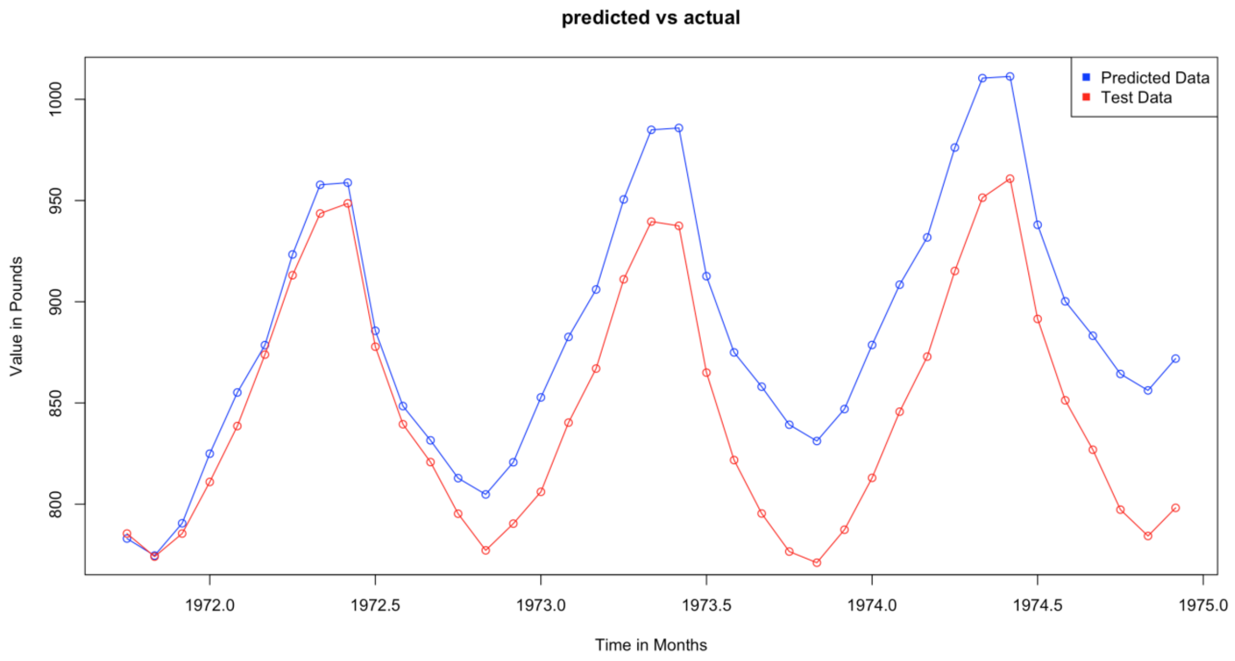
ACF of the residuals show no significant correlation between lags, indicating normality

Forecasting and Testing Model Accuracy

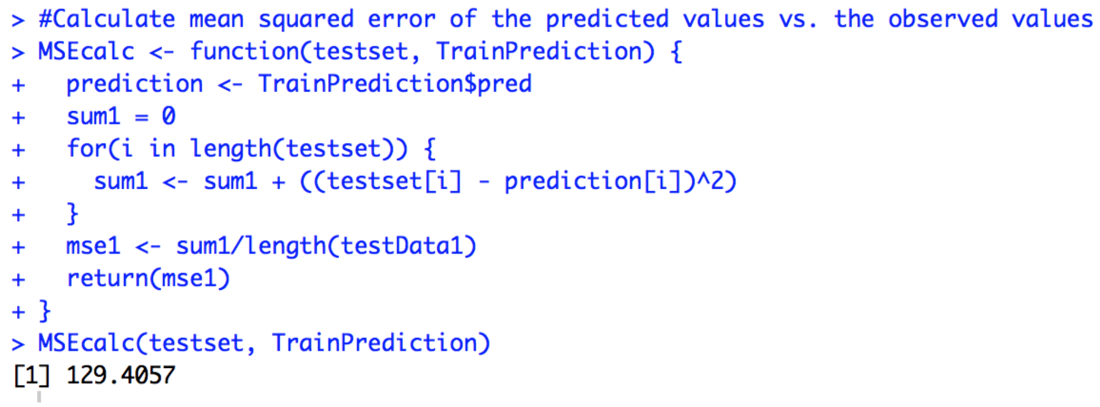
For forecasting, the forecast library was imported into R and the forecast function was utilized to predict one year ahead of the original dataset using the ARIMA model that I developed for the data. The model was also trained using both the training and test subsets of the data, with values being fit from the end of the training data to the end of the test data.



Plot of the one year forecast with the original dataset



Plot containing both the actually observed data and the predicted values



```
> #Calculate mean squared error of the predicted values vs. the observed values
> MSEcalc <- function(testset, TrainPrediction) {
+   prediction <- TrainPrediction$pred
+   sum1 = 0
+   for(i in length(testset)) {
+     sum1 <- sum1 + ((testset[i] - prediction[i])^2)
+   }
+   mse1 <- sum1/length(testData1)
+   return(mse1)
+ }
> MSEcalc(testset, TrainPrediction)
[1] 129.4057
```

Calculation of MSE using an R function

Discussion and Conclusion

By taking an overall general approach to the project, I was able to develop a decently flexible script for handling time series dataset analysis. The data I used was meant for demonstration purposes only and the script can be easily modified to handle a different dataset if necessary.

R code

library(TSA)
library(forecast)

```
########
#Nicholas Gallo
#Time Series Model Fitting Project
########

#Read in CSV file, however, there are libraries for JSON objects and Excel tables
Project <- read.csv(file.choose(), header = TRUE)

#Convert dataset to seasonal time series
ProjData <- ts(Project$Data, frequency = 12, start = c(1962, 1))
summary(ProjData)
plot.ts(ProjData, main = "Milk Production per Cow in Pounds from 1962-1975", xlab = "Milk
Production", ylab = "Month", type = "o")
#Decompose to check trend and seasonality of data, as well as randomness
plot(decompose(ProjData))

#Split dataset into training and testing set
#The training set is used to fit models and the test set is used for analysis of predicted values up
to the end of the dataset
trainset <- subset(ProjData, end = floor(0.75*length(ProjData)))
testset <- subset(ProjData, start = floor(0.75*length(ProjData)) + 1)

#Fitting a linear regression model, using lagged data
ProjDataRegress <- lm(formula = ProjData ~ zlag(ProjData), data = ProjData)
summary(ProjDataRegress)
#Check residuals for normality. Histogram is especially useful
StandardResRegress <- rstandard(ProjDataRegress)
qqnorm(StandardResRegress)
qqline(StandardResRegress)
hist(StandardResRegress)

#Analyze the data set, differencing if needed
DiffProjData <- diff(ProjData)
plot(DiffProjData)
acf(DiffProjData)
pacf(DiffProjData)

#Test first order auto-regressive (AR(1)) model on the training set
ProjAR1 <- Arima(trainset, order = c(1, 0, 0))
ProjAR1

#Predict one value ahead of the last value in the training set
#Use last value in training set to predict the next value
#This is just a test using the AR(1) Forecast function Yt(1)
Yt1 <- 0.9243*(trainset[length(trainset)] - 709.2583) + 709.2583
Yt1
#Alternative method that also works, using predict function
ProjARforecast <- predict(ProjAR1, n.ahead = 1)
ProjARforecast
#Forecast error of the model
```

```
ARforecastError <- testset[1] - ProjARforecast$pred
ARforecastError

#Lower and Upper Confidence intervals for the predicted data points
LCI <- (ProjARforecast$pred - 1.96*ProjARforecast$se)
UCI <- (ProjARforecast$pred + 1.96*ProjARforecast$se)
LCI
UCI

#Build custom model for the training set, in this case I utilize the ARIMA model and the
associated libraries
auto.arima(ProjData)
ProjModel <- Arima(ProjData, order = c(1, 0, 0), seasonal = list(order = c(1, 1, 1), period = 12))
ProjModel
#Test the normality of the Residuals
acf(ProjModel$residuals)
Box.test(ProjModel$residuals)
#Forecast predicted values up to one year and plot the forecast with the original data
ProjForecast <- forecast(ProjData, model = ProjModel, 12)
plot(ProjForecast)
legend("bottomright", legend = c("Forecasted values"), col = c("blue"), pch = c(15))

#Fit a custom training model with the training set and the Arima model specified for the entire
dataset
TrainProjModel <- Arima(trainset, order = c(1, 0, 0), seasonal = list(order = c(1, 1, 1), period =
12))
#Use the predict function to fit values up through the length of the test set
TrainPrediction <- predict(TrainProjModel, n.ahead = length(testset))
#Plot the predicted data against the observed data
plot(TrainPrediction$pred, main = "predicted vs actual", ylab = "Value in Pounds", xlab = "Time
in Months", col = "blue", type ="o")
points(testset, col = "red", type = "o")
legend("topright", legend = c("Predicted Data", "Test Data"), col = c("blue", "red"), pch = c(15,
15))

#Calculate mean squared error of the predicted values vs. the observed values
MSEcalc <- function(testset, TrainPrediction) {
  prediction <- TrainPrediction$pred
  sum1 = 0
  for(i in length(testset)) {
    sum1 <- sum1 + ((testset[i] - prediction[i])^2)
  }
  mse1 <- sum1/length(testData1)
  return(mse1)
}
MSEcalc(testset, TrainPrediction)
```