Experimenting with Time Series/Forecasting

James Burnard

11/20/2024

This litte project provides an overview of working with time series data, modeling time series and evaluating and selecting models for use in forecasting future values. There are two parts to this project. In part 1, I will be fitting and evaluating several models on a time series of a single variable. In part 2, I'll fit a linear regression model to a multivariate time series.

1 Part 1: Exploring and Modeling Time Series

In this project, I'm workign with time series data on birth rates in New York city observed monthly over a period of 12 years. This series starts in January 1948 and ends in December 1959.

1.1 Data Processing

```
birth.df <- read.csv("nybirths.csv")
```

Since there is missing data, we can use the na.approx function to impute values.

```
birth.df$birth_rate<-na.approx(birth.df$birth_rate)
```

The summary does not indicate the presence of any outliers. Confirm with a boxplot of the bith_rate.

```
boxplot(birth.df$birth_rate,
        main = "Boxplot of Birth Rates (1948-1958)",
        ylab = "Birth Rate",
        col = "skyblue",
        border = "darkblue",
        notch = TRUE)
```

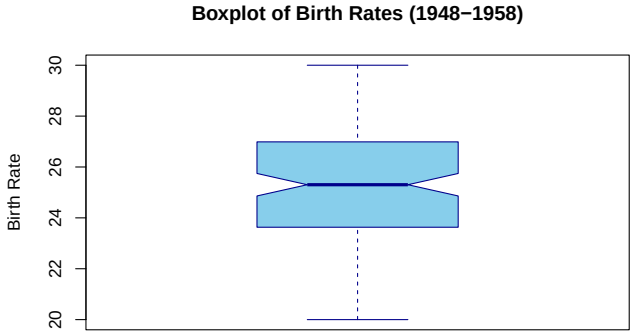

Figure 1: Distribution of Birth rate 1948-1958.

1.1.1 The ts Function

Note that I am not using the date column at all. I am manually setting the dates and months in the time series object.

```
birth.ts <- ts(birth.df$birth_rate, frequency=4, start=c(1948,1))
```

This code calls the autoplot function to visualize the time series.

```
autoplot(birth.ts) + ylab("Births") + xlab("Year")
```

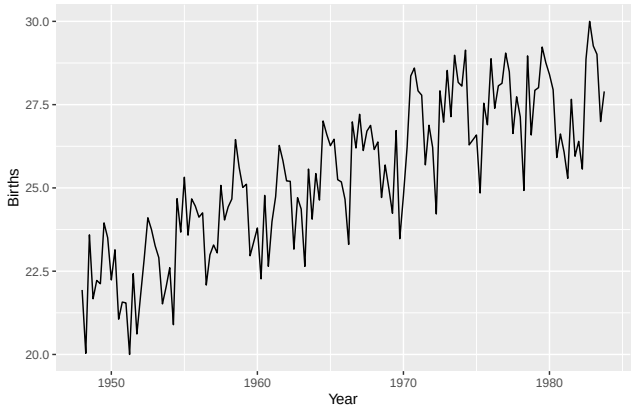

Figure 2: NY Births 1948-1959.

I see a positive trend that is happening. I also see a seasonal cycle in the repeated spiking of the plot that roughly have the same spacing.

To visualize the components of this time series, I wrote the code below that calls the decompose function and then shows the components in one plot.

```
birth.ts.comp <- decompose(birth.ts)
autoplot(birth.ts.comp)
```

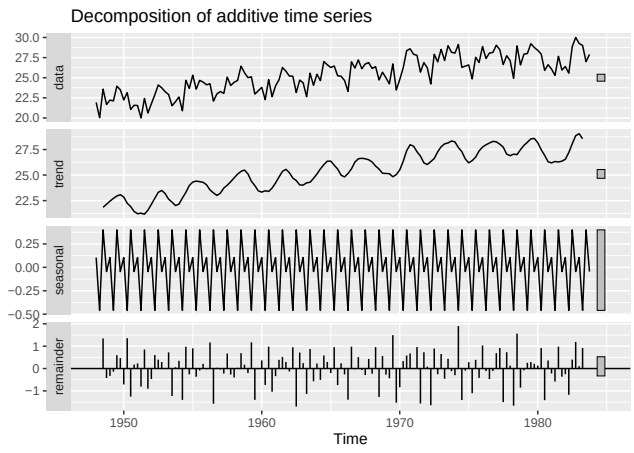

Figure 3: NY Births Components.

Used the "type" attribute to see if this is an additive or multiplicative series (type birth.ts.comp$type in the console). It revealed this series is additive.

```
adf.test(birth.ts)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  birth.ts
## Dickey-Fuller = -6.1213, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

```
kpss.test(birth.ts)
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  birth.ts
## KPSS Level = 2.5676, Truncation lag parameter = 4, p-value = 0.01
```

ADF: -6.1 with a significant p value so stationary KPSS: 2.56 wuth a significant p value

If the results are mixed, the series can be made stationary if required. I will not perform any transformations on the time series at this point since the models I will use can manage differencing themselves if needed.

1.2 Modeling Time Series

Next, I'll select models and fit them to the data using training and testing sets.

1.2.1 Create Train and Test Time Series

The size of the test set is typically about 20-25% of the total sample, although this value depends on how long the sample is and how far ahead you want to forecast. The test set should ideally be at least as large as the maximum forecast horizon, the h parameter, required.

We have data for 12 years. It seems that due to the seasonality, I should keep data for each year together. If we split on whole years, we have 2 years for the test set, and 10 years for the training set. The test then is 24 months, and we'll use a forecast horizon of 12 months.

The split into training and testing time series is done using the window function.

```
train.ts <- window(birth.ts, start=1948, end=c(1957,12))
test.ts <- window(birth.ts, start=1958)
```

1.2.2 Benchmark Models: avg, rwf

In this section, I will fit simple models to be compared with more complex models such as HoltWinters and ARIMA models. The benchmark "models" are not really models in the sense that they do not learn any parameters, rather they simply execute simple calculations to produce their forecasts.

1.2.2.1 The Average Model The average model is really simple (that's why it's a great benchmark model): it just computes the average value of the series.

1.2.2.1.1 Fit Model on Training Data This code fits an average model to the training data with a forecast horizon of 12.

```
avg.fit <- meanf(train.ts, 12)
```

This code generates three plots that help evaluate the way the model fit the training data- specifically by examining the residuals.

```
checkresiduals(avg.fit)
```

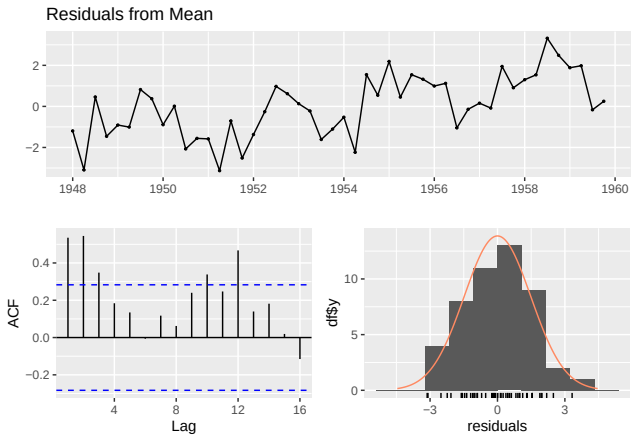

Figure 4: Residuals Plots for Avg Model.

```
##
##  Ljung-Box test
##
## data:  Residuals from Mean
## Q* = 40.502, df = 8, p-value = 2.583e-06
##
## Model df: 0.   Total lags used: 8
```

The lag diagram shows that the residuals are correlated which isn't a good thing. This means the pattern in the data was missed. THe residuals are pretty much normally distributed with a mean more or less around 0, but it's not super convincing.

This is a plot of the average model against the training series. It's an example of a model that is very ridgid, and does not pick up on the components of the series at all and is therefore underfitting the data.

```
train.comp <- decompose(train.ts)
birth.model.data <- cbind(train.comp$x, avg.fit$fitted)
autoplot(birth.model.data, facet = FALSE) +
  ylab("Birth Rate") + xlab("Year") +
  scale_color_manual(labels = c("TS Data", "The Average Model"),
                     values=c("darkgrey", "purple"))
```

1.2.2.1.2 Accuracy on Test Data This code calls the accuracy function that tests the model's forecasts on the test set, obtains the error metrics, and displays the RMSE results for RMSE and MAE for both the training and test errors in a formatted table.

```
avg.results <- accuracy(avg.fit, test.ts)
kable(avg.results[,2:3], caption='RMSE for the Average Model')
```

Table 1: RMSE for the Average Model

	RMSE	MAE
Training set	1.469829	1.203020
Test set	1.723638	1.475986

RMSE and MAE from the Training set are smaller. The training error are fit errors, which means the model

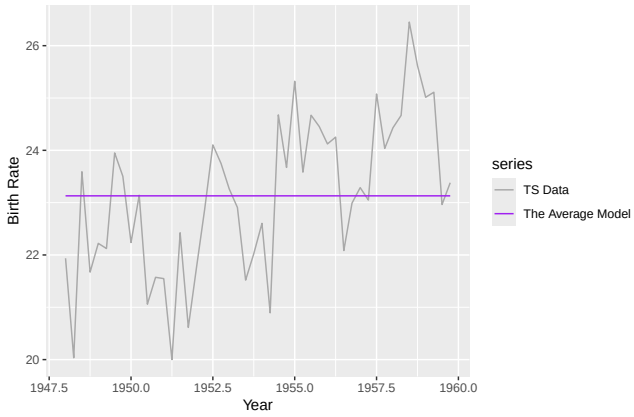

Figure 5: The Average Model.

is able to use parameter estimation as it refines its "forecasts". Test set errors are true forecasts, as the model's parameters are fixed (In the average model, the disparity is simply a result of the difference in the mean of the training set vs the test set).

1.2.2.1.3 Evaluate Model with Cross Validation I'll use the entire data set as the cross validation will create training and testing subsets at each fold.

This code calls the tsCV function to fit the average model using rolling cross validation with a 12 step forecast horizon. The function returns a matrix with 12 columns, one for horizons 1-12. I extracted the column for h=12 and use it to calculate the forecast error for that horizon in terms of RMSE (we'll not include MAE for simplicity).

```
avg.fit.cv <- tsCV(birth.ts, forecastfunction=meanf, h=12)
# get the error vector from the fit matrix at horizon 12
err.vec <- avg.fit.cv[,12]
# calculate RMSE on the test data
avg.rmse.cv <- sqrt(mean(err.vec^2, na.rm=TRUE))
```

It performed better than the single test set.

1.2.2.2 The Random Walk Model This type of model fits the data in the series by using the current value of the series as its fitted value for the next time step.

1.2.2.2.1 Fit Model on Training Data, Accuracy on Test Data The code below fits an rwf model and evaluates it with the accuracy function. Since the birth rate time series has a seasonal component, I used the snaive function to implement the seasonal version of rwf.

```
rwf.fit <- snaive(train.ts, 12)
rwf.results <- accuracy(rwf.fit, test.ts)
kable(rwf.results[,2:3], caption='RMSE for the Random Walk Model')
```

Table 2: RMSE for the Random Walk Model

	RMSE	MAE
Training set	1.836143	1.555659
Test set	1.665793	1.296417

```
rwf.fit.cv <- tsCV(birth.ts, snaive, h=12)
err.vec <- rwf.fit.cv[,12]
#RMSE on the test data
rwf.rmse.cv <- sqrt(mean(err.vec^2, na.rm=TRUE))
```

1.2.2.2.2 Evaluate Model with Cross Validation

1.2.3 ETS Models

There are many ETS models to choose from. The choice is based on the components of the time series you are modeling. We'll choose the holt winters model as it has the most flexibility and a parameter to deal with the seasonal component of the birth time series.

```
hw.fit <- hw(train.ts, h=12)
```

1.2.3.1 Fit Model on Training Data

1.2.3.2 Evaluate Fit Checking the fit of this model by calling checkresiduals on the hw model:

```
checkresiduals(hw.fit)
```

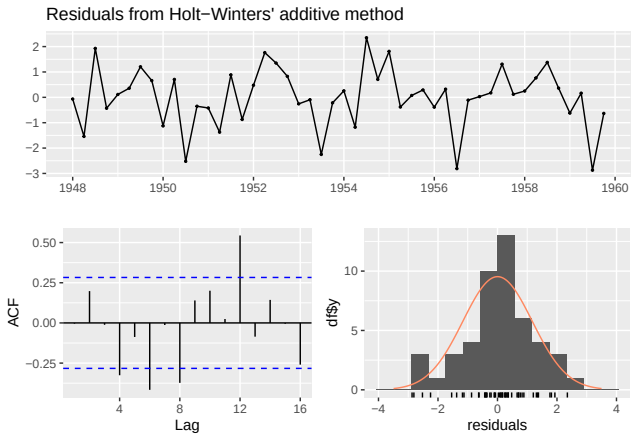

Figure 6: Residuals Plots for the Holt Winters Model.

```
##
##  Ljung-Box test
##
## data:  Residuals from Holt-Winters' additive method
## Q* = 26.515, df = 8, p-value = 0.0008568
##
## Model df: 0.   Total lags used: 8
```

The results of the fit are mixed. The plot of the residuals seems roughly centered on 0 (in fact, the mean is 0.05765806. By visual inspection, the variation seems pretty much even throughout the series. However, the ACF plot shows some correlation. The Ljung-Box test corroborates the ACF plot that there is correlation in the residuals.

1.2.3.3 Accuracy on Test Data Displaying the RMSE and MAE in a table as we have done for the other models.

```
hw.results <- accuracy(hw.fit, test.ts)
kable(hw.results[,2:3], caption='RMSE for the Holt Winters Model')
```

Table 3: RMSE for the Holt Winters Model

	RMSE	MAE
Training set	1.153202	0.8572247
Test set	1.157751	1.0095451

```
hwfxn <- function(x, h) {forecast(hw(x, h))}
hw.fit.cv <- tsCV(birth.ts, hwfxn, h=12)
err.vec <- hw.fit.cv[,12]
#RMSE on the cv
hw.rmse.cv <- sqrt(mean(err.vec^2, na.rm=TRUE))
```

1.2.3.4 Evaluate Model with Cross Validation

1.2.4 Autoregressive Moving Average Model: ARIMA

We saw that the holt winters model fit showed correlation in its fit residuals. A model that can better handle correlation Next, I will fit an ARIMA model to the training data and make forecasts. I will use a function that will find the best arima model by automation.

1.2.4.1 Fit Model on Training Data This code calls the auto.arima function and passes in the training data. This may take a minute to run.

ARIMA requires a stationary time series, but auto.arima automatically handles non-stationary series by applying differencing. It also finds values for the ARIMA model parameters p,d,q.

```
arima.fit <- auto.arima(train.ts)
```

1.2.4.2 Evaluate Fit Checking the fit of this model by calling checkresiduals:

```
checkresiduals(arima.fit)
```

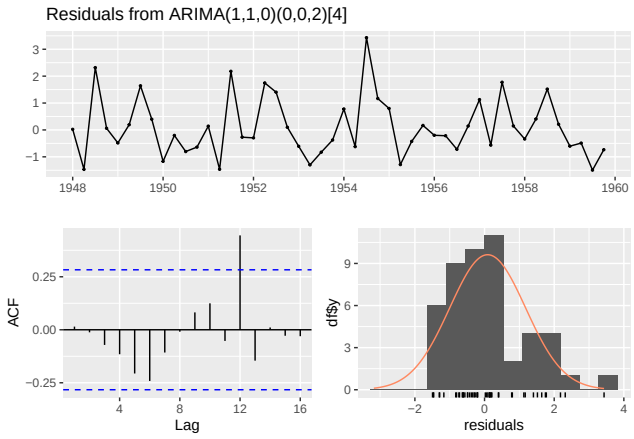

Figure 7: Residuals Plots for the ARIMA Winters Model.

##

```
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,1,0)(0,0,2)[4]
## Q* = 7.4013, df = 5, p-value = 0.1925
##
## Model df: 3.   Total lags used: 8
```

1.2.4.3 Accuracy on Test Data

ARIMA models have three input parameters:
p= order of the autoregressive part
d= degree of first differencing involved
q= order of the moving average part

The auto.arima function discovered the values for p, d, and q when we fit it to the training data. This specifies the model to use to evaluate its accuracy on forecasting the test data.

```
arima.test <- Arima(test.ts, order=c(2,0,0))
arima.results <- accuracy(arima.test)
kable(arima.results[,2:3], caption='RMSE for the ARIMA Model')
```

Table 4: RMSE for the ARIMA Model

	x
RMSE	1.2615347
MAE	0.9838866

Note: the results produced from the accuracy() command are actually on the test set (despite the output saying "Training set").

1.2.4.4 Evaluate Model with Cross Validation

Because the model will be trained and then tested repeatedly in cross validation, I now have to pass the model specification to the tsCV function. I used the p,d,q params from the model that was trained.

```
# pass those values in to Arima to use this model in cv
arimafxn <- function(x, h) {forecast(Arima(x, order=c(2,0,0)))}
arima.fit.cv <- tsCV(birth.ts, arimafxn, h=12)
err.vec <- arima.fit.cv[,12]
#RMSE on the cv
arima.rmse.cv <- sqrt(mean(err.vec^2, na.rm=TRUE))
```

1.2.5 Summary of Benchmark, ETS, and ARIMA models

Table with the RMSE results on the test set for each model.

```
# gather the RMSE on test data for all models
avg.rmse.test <- avg.results[2,2]
rwf.rmse.test <- rwf.results[2,2]
hw.rmse.cv <- hw.results[2,2]
arima.rmse.cv <- arima.results[1,2]
# create a matrix for kable to render
test.mat = matrix(c(avg.rmse.test,rwf.rmse.test, hw.rmse.cv, arima.rmse.cv),
           nrow = 4,ncol = 1,)
rownames(test.mat) = c("avg", "rwf", "hw", "arima")
colnames(test.mat) = c("RMSE")

kable(test.mat, caption='Forecast RMSE on Test Data.')
```

Table 5: Forecast RMSE on Test Data.

	RMSE
avg	1.723638
rwf	1.665793
hw	1.157751
arima	1.261535

Table with the results for RMSE for each model on cross validation.

```
# gather the RMSE on cv for all models
cv.err.data <- c(avg.rmse.cv,rwf.rmse.cv,hw.rmse.cv,arima.rmse.cv)

# create a matrix for kable to render
cv.mat = matrix(cv.err.data,
           nrow = 4,ncol = 1,)
rownames(cv.mat) = c("avg", "rwf", "hw", "arima")
colnames(cv.mat) = c("RMSE")

kable(cv.mat, caption='Forecast RMSE on Cross Validation.')
```

Table 6: Forecast RMSE on Cross Validation.

	RMSE
avg	2.571464
rwf	1.166026
hw	1.157751
arima	1.261535

The rankings of these models differ. The rwf model is the best, then the hw model. The cross validation table should be chosen, cross validation is usually more robust, and within this table the hw model is the best, which is consistent.

1.3 Forecasting: a Note on Model Selection and Usage

The results of the model evaluation provide a way for me to select the model I think will do the best job forecasting future birth rates for some number of steps in the future.

The chosen model can then be fit to the entire data set and then be used for forecasting. This code will fit an arima model to the entire data set with a forecast window of 2 years.

```
hw.fit.all <- hw(birth.ts, h=24)
hw.forecasts <- forecast(hw.fit.all, h=24)
```

The following code plots the forecasts for the ARIMA model.

```
autoplot(hw.forecasts)
```

A negative spike in the birthrates followed by the start of a sharp increase back up towards the higher 20's

2 Part 2: Linear Regression and Mulivariate TS

Multiple linear regression is a model in the form:

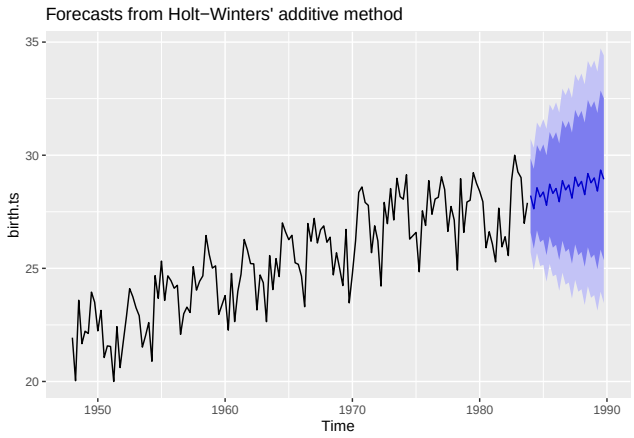

Figure 8: ARIMA Forecasts for One Year.

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

where y is the predicted variable, the coefficients, betas, are learned parameters, and x1 to xn are the predictors. If y and each of the predictors are time series, we can fit a linear regression model to this data.

The data set contains the quarterly percentage changes of the following for the US from 1970 Q1 to 2016 Q3: - Consumption: personal consumption expenditure. - Income: personal disposable income. - Production: the national industrial productionrate. - Savings: the national personal savings rate. - Unemployment: the national unemployment rate.

These statements read the data from a csv file, checks for outliers and missing data by displaying a summary. The first column is removed as it is a row index.

```
pcons.df<- read.csv("persconsumption.csv")
summary(pcons.df)
```

```
##       Index        Consumption          Income          Production
##  Min.   :  1.0   Min.   :-2.2741   Min.   :-4.2652   Min.   :-6.85104
##  1st Qu.: 47.5   1st Qu.: 0.4198   1st Qu.: 0.3378   1st Qu.: 0.05568
##  Median : 94.0   Median : 0.7721   Median : 0.7237   Median : 0.65793
##  Mean   : 94.0   Mean   : 0.7465   Mean   : 0.7176   Mean   : 0.50806
##  3rd Qu.:140.5   3rd Qu.: 1.0898   3rd Qu.: 1.1650   3rd Qu.: 1.30572
##  Max.   :187.0   Max.   : 2.3183   Max.   : 4.5365   Max.   : 4.14957
##     Savings         Unemployment
##  Min.   :-68.788   Min.   :-0.900000
##  1st Qu.: -4.218   1st Qu.:-0.200000
##  Median :  1.280   Median : 0.000000
##  Mean   :  1.222   Mean   : 0.007487
##  3rd Qu.:  6.651   3rd Qu.: 0.100000
##  Max.   : 50.758   Max.   : 1.400000
```

```
pcons.df<- pcons.df[,-1]
```

This statement makes a ts object that consists of a separate time series for each variable in the data set.

```
cons.all.ts <- ts(pcons.df, frequency=4, start=c(1970,1))
```

2.1 Data Exploration

This code extracts the Consumption time series and plots it.

```
consumption.ts <- cons.all.ts[,1]
autoplot(consumption.ts) + ylab("% Change in Consumption") + xlab("Year")
```

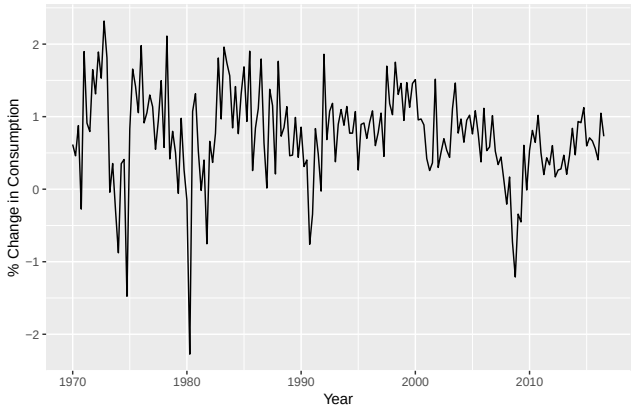

Figure 9: Personal Consumption Expenditure per Year.

Looks stationary.

The code below applies the adf and kpss tests to the Consumption series.

```
adf.test(cons.all.ts[,5])
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  cons.all.ts[, 5]
## Dickey-Fuller = -4.5287, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

```
kpss.test(cons.all.ts[,5])
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  cons.all.ts[, 5]
## KPSS Level = 0.11007, Truncation lag parameter = 4, p-value = 0.1
```

adf: -4.5287 with a very significant p value , kpss: 0.11007 with an insignificant p value

I know it's possible that applying linear regression to a non-stationary time series can lead to erroneous regressions. So the code below plots all of the time series in the data set.

```
autoplot(cons.all.ts, facets = TRUE)
```

Now I will proceed to fit two linear regression models on training data and test their forecasting performance on test data,

2.2 Create training and test sets

I will hold out 20% of the data for testing. Then I will use the row indexes to create the subsets. We want to break the series up with respect to the period of the series, which is quarterly, four quarters per year. The calculation is as follows: There are 187 rows in the series where each row is one quarter, so four rows is one year. The data starts at Q1 1970, and ends at Q3 in 2016 (We note that the last year is only three quarters). We want to break the data on a whole year. The years 1970-2015 represent 46 whole years * 4 quarters = 184 rows. That leaves the 3 quarters from 2016 to make 187 total. So, 20% of the data (whole years) is about 9.3

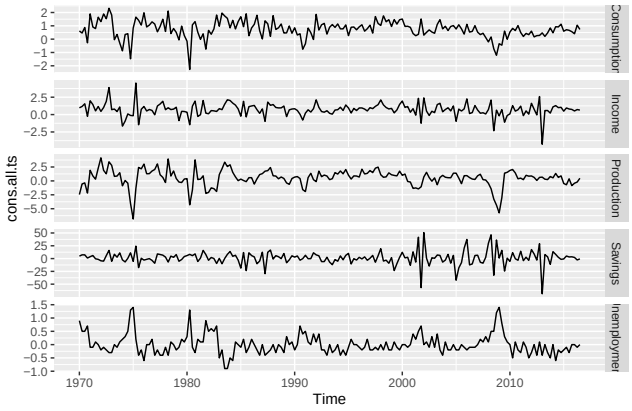

Figure 10: Plots of all Time Series in the Data Set.

years, so we will take 9 whole years, 36 rows plus 3 rows for 2016 = 39 rows to make up the test set. That means 148 rows for the training set. The head and tail functions will help create these subsets of the data.

```
reg.train.ts = head(cons.all.ts, 148)
reg.test.ts = tail(cons.all.ts, 39)
```

2.3 Fit Models on Training Data

Now, I will fit two linear regression models: a simple model with one predictor, and a more complex model that includes all predictors.

```
reg.mod.simple <- tslm(formula = Consumption ~ Income, data = reg.train.ts)

reg.mod.multi <- tslm( Consumption ~ Income + Production + Unemployment + Savings,
  data = reg.train.ts)
```

As with linear regression on non-time series data, we interpret the coefficients in the same way. Execute the summary below to see the fit results.

```
summary(reg.mod.simple)
```

```
##
## Call:
## tslm(formula = Consumption ~ Income, data = reg.train.ts)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.38354 -0.28726  0.02884  0.29735  1.42717
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.58415    0.06726   8.685 6.97e-15 ***
## Income       0.32429    0.05742   5.647 8.24e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6038 on 146 degrees of freedom
## Multiple R-squared:  0.1793, Adjusted R-squared:  0.1737
## F-statistic: 31.89 on 1 and 146 DF,  p-value: 8.243e-08
```

The Income coefficient is positive, so it has a positive effect on Consumption rate. The value of the coefficient shows that a one unit increase in Income (a 1 percentage point increase in personal disposable income) results on average in 0.32 units increase in Consumption (an average increase of 0.32 percentage points in personal consumption expenditure).

2.4 Evaluate Fit

This code to displays plots about fit residuals.

```
checkresiduals(reg.mod.simple)
```

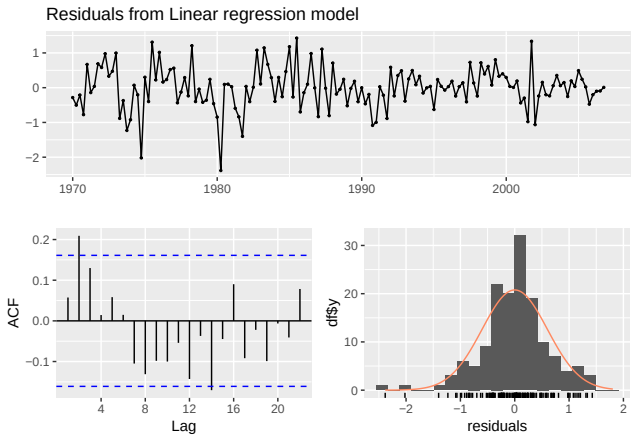

Figure 11: Residuals Plots for the Simple Regression Model.

```
##
##  Breusch-Godfrey test for serial correlation of order up to 8
##
## data:  Residuals from Linear regression model
## LM test = 14.359, df = 8, p-value = 0.07289
```

The residuals plot look fairly evenly distributed around 0. There is some skew in the histogram, and some spikes in the ACF plot where correlation in the residuals is occurring.

2.5 Accuracy on Test Data

Now I will obtain forecasts from both models on the test data. The forecast function needs the newdata parameter to be in the form of a data frame.

The following code obtians the predictions and calculates the forecast errors for both models.

```
# have to convert ts to data frame
test.df <- as.data.frame(reg.test.ts)
# obtain predictions
reg.mod.simple.pred <- forecast(reg.mod.simple, newdata=test.df)
reg.mod.multi.pred <- forecast(reg.mod.multi, newdata=test.df)
#RMSE: Root mean squared error and MAE: Mean absolute error
# simple reg model
rmse.simple<- sqrt(mean((reg.mod.simple.pred$mean-test.df$Consumption) ^2))
mae.simple<- mean(abs(reg.mod.simple.pred$mean-test.df$Consumption))
# multiple reg model
rmse.multi<- sqrt(mean((reg.mod.multi.pred$mean-test.df$Consumption) ^2))
mae.multi<- mean(abs(reg.mod.multi.pred$mean-test.df$Consumption))
```

2.6 Summary of Model Forecast Errors

This code produces a table that summarizes the results of the models' performance.

```
# gather the RMSE and MAE on test data for both models
cv.err.data <- c(rmse.simple,mae.simple,rmse.multi,mae.multi)
# create a matrix for kable to render
cv.mat = matrix(cv.err.data,
           nrow = 2,ncol = 2,)
rownames(cv.mat) = c("simple", "multi")
colnames(cv.mat) = c("RMSE", "MAE")

kable(cv.mat, caption='Forecast RMSE on Cross Validation for Regression Models.')
```

Table 7: Forecast RMSE on Cross Validation for Regression Models.

	RMSE	MAE
simple	0.6241018	0.2617616
multi	0.4647381	0.1958865

The multi performed better. It had slightly lower MAE and RMSE values that prove it to be slightly more accurate.

The better model can be fit to all of the data and would be available for use in forecasting future Consumption rates. Researchers and analysts can input values for Income, Production, and Unemployment, and Savings to see what those values would forecast in terms of Consumption.