GeneData Analysis

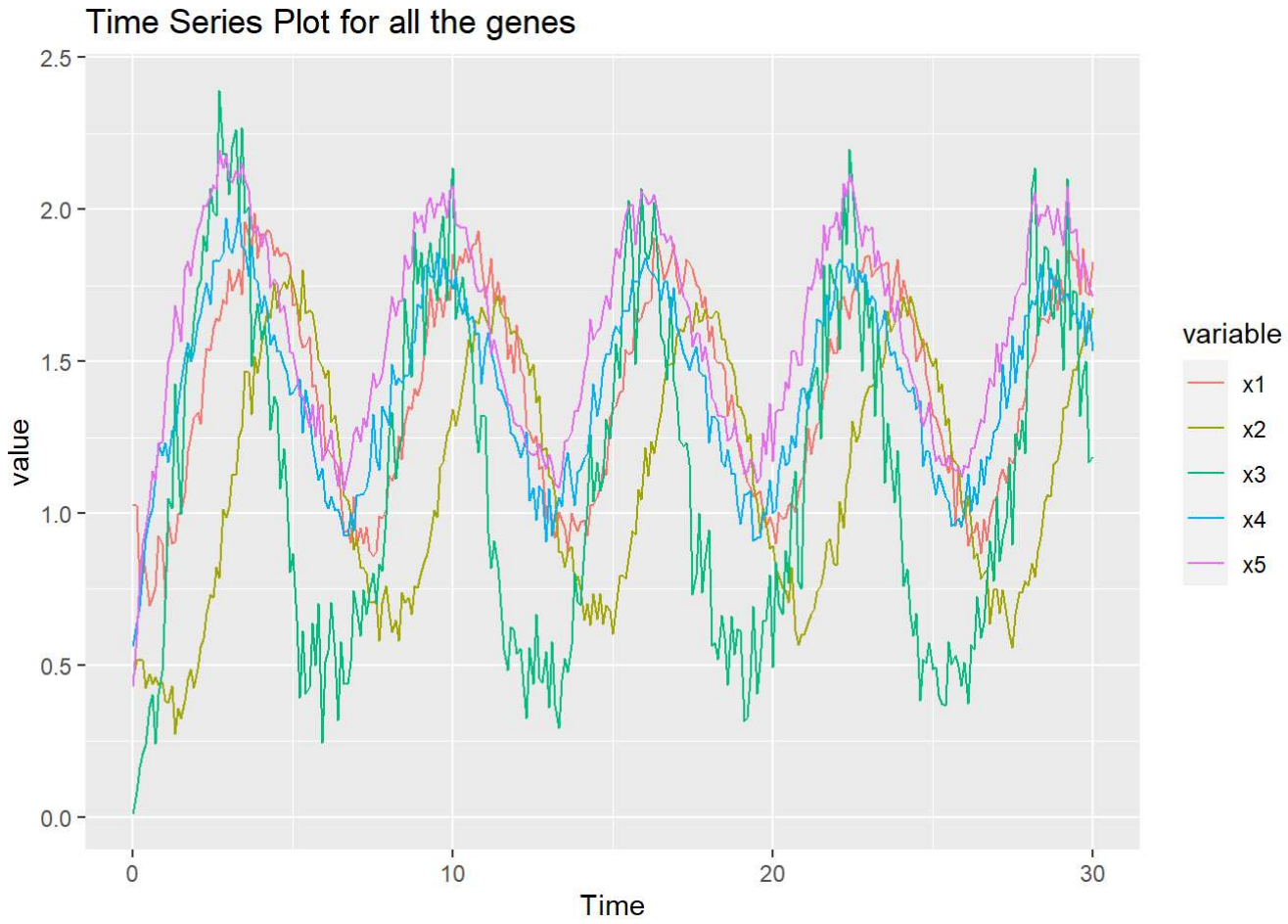
Exploratory Data Analysis

In this section we will explore the data set to summarize the main characteristics of the genes. We will look at different aspects including distribution and correlation of genes.

Time Series plot

First, we would like to see how the genes behaves with respect to time. Is there any trend for gene values with respect to time.

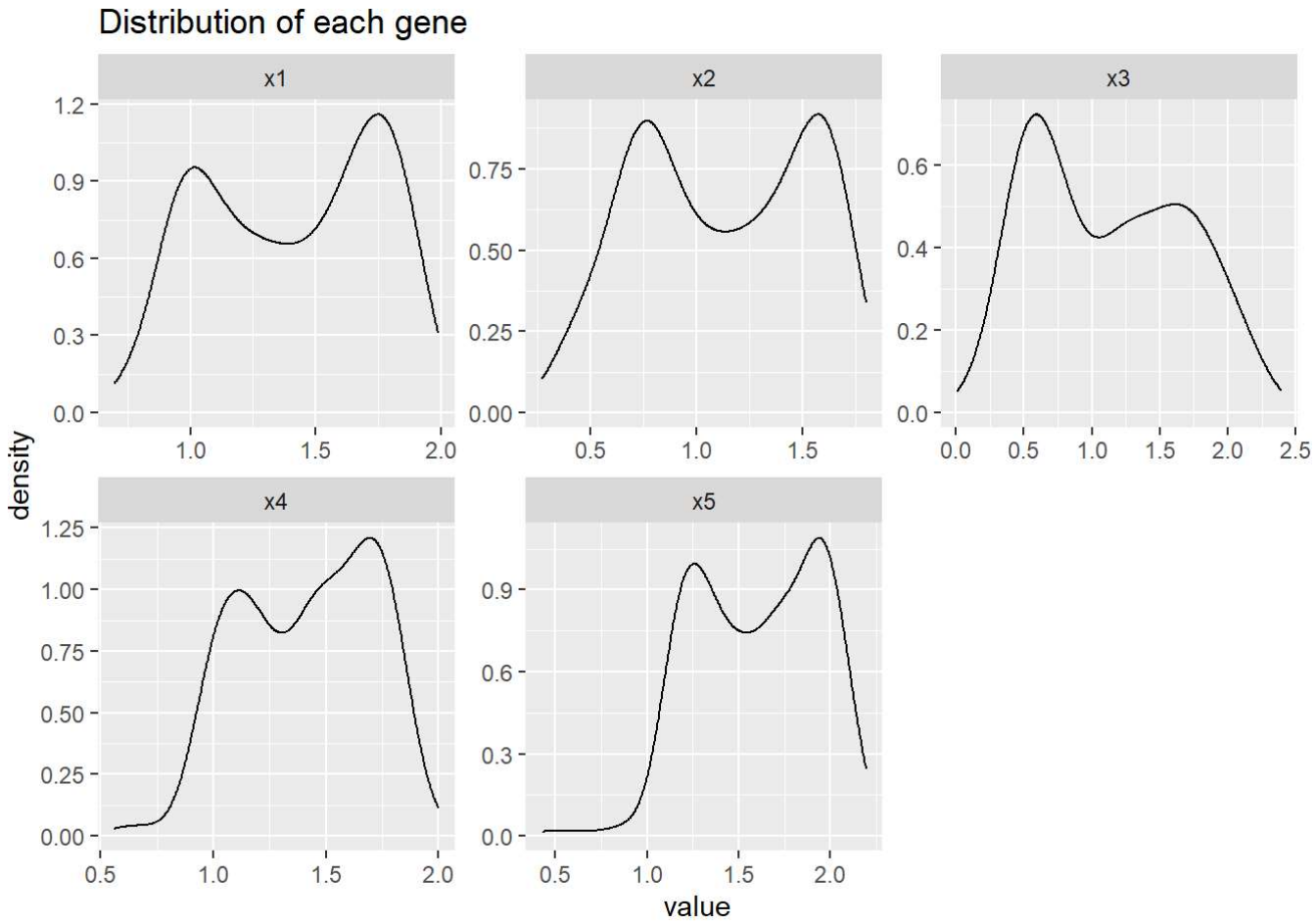
```
data <- read.csv("genedata.csv")
colnames(data)[1]<-c('Time')  #column name change

meltedData <- melt(data,id='Time')
ggplot(data = meltedData, aes(x=Time, y=value)) + geom_line(aes(colour=variable)) +
  ggtitle('Time Series Plot for all the genes')
```



Distribution of genes

Lets look at how the genes are distributed.

```
data[,2:6] %>%
  keep(is.numeric) %>%                     # Keep only numeric columns
  gather() %>%                             # Convert to key-value pairs
  ggplot(aes(value)) +                     # Plot the values
    facet_wrap(~ key, scales = "free") +   # In separate panels
    geom_density()+ggtitle('Distribution of each gene')
```

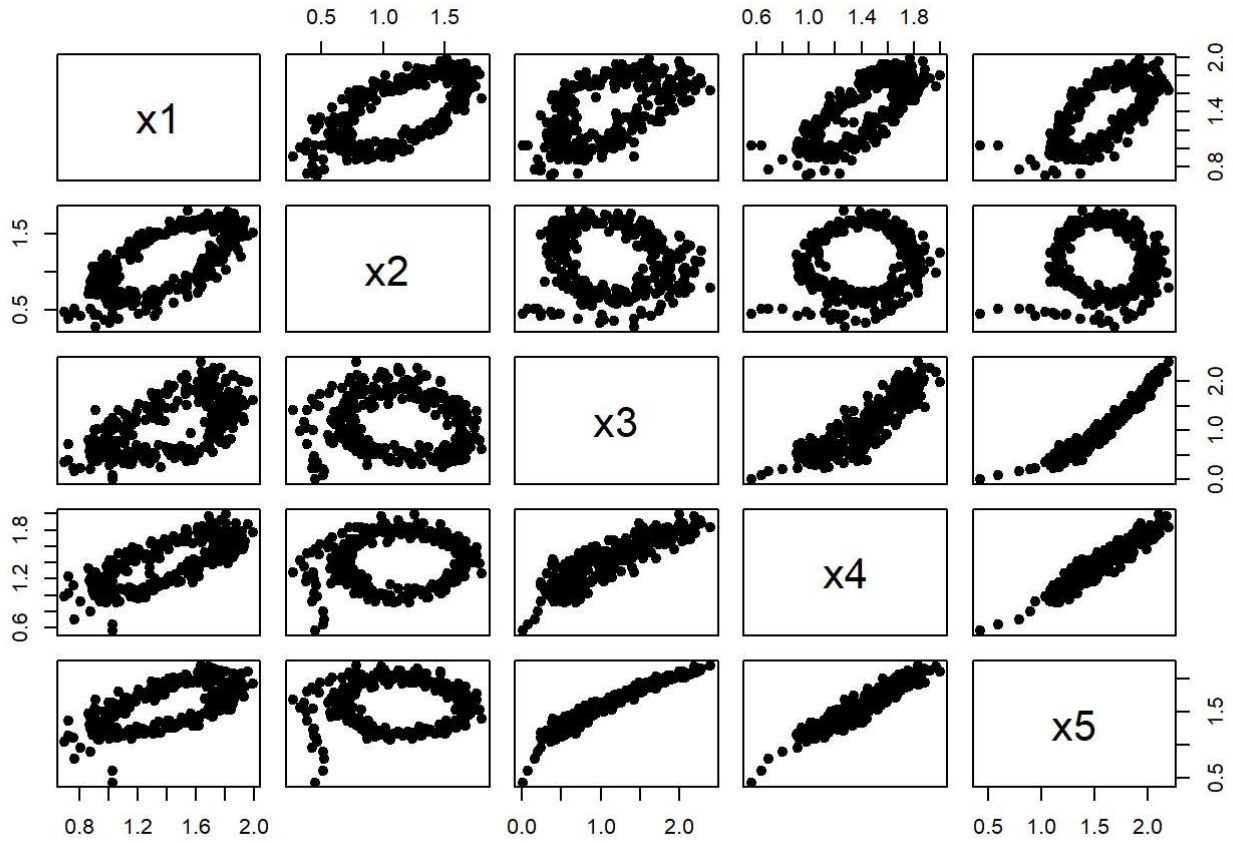



```
head(data)
```

```
##   Time        x1        x2         x3        x4        x5
## 1  0.0 1.0268834 0.4525760 0.01006418 0.5598125 0.4274130
## 2  0.1 1.0262801 0.5158073 0.08392307 0.6389631 0.5986789
## 3  0.2 0.7642321 0.5197737 0.16395952 0.6934493 0.7904950
## 4  0.3 0.8717433 0.5167898 0.20829830 0.7982659 0.8988766
## 5  0.4 0.8058304 0.4259886 0.23927017 0.9151582 0.9474713
## 6  0.5 0.6965125 0.4693196 0.36445269 0.9857926 1.0403834
```

Scatter Plot

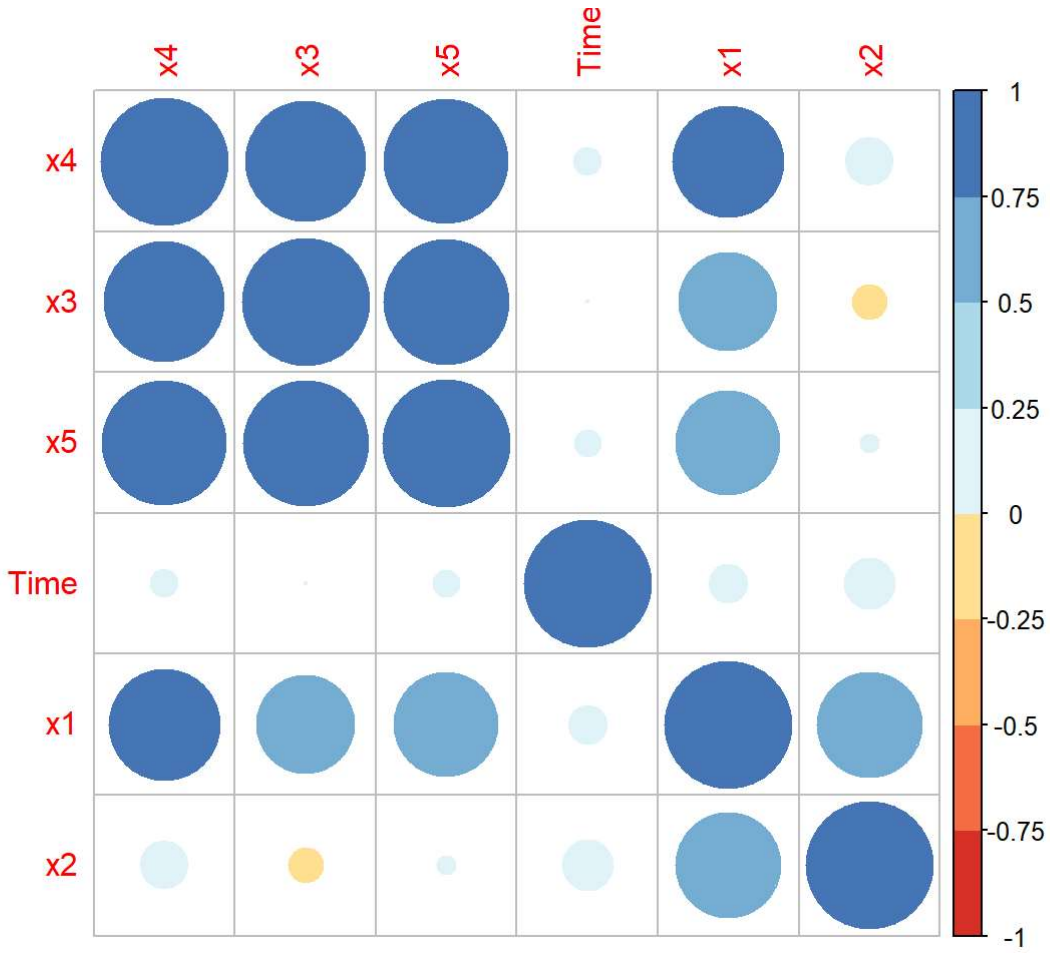
```
pairs(data[,2:6], pch = 19)
```



Correlation Plot

We would see at the correlation plot for all the genes to check if these genes have significant correlation with time or other genes.

```
corr <-cor(data)
corrplot(corr, order="hclust",
         col=brewer.pal(n=8, name="RdYlBu"))
```



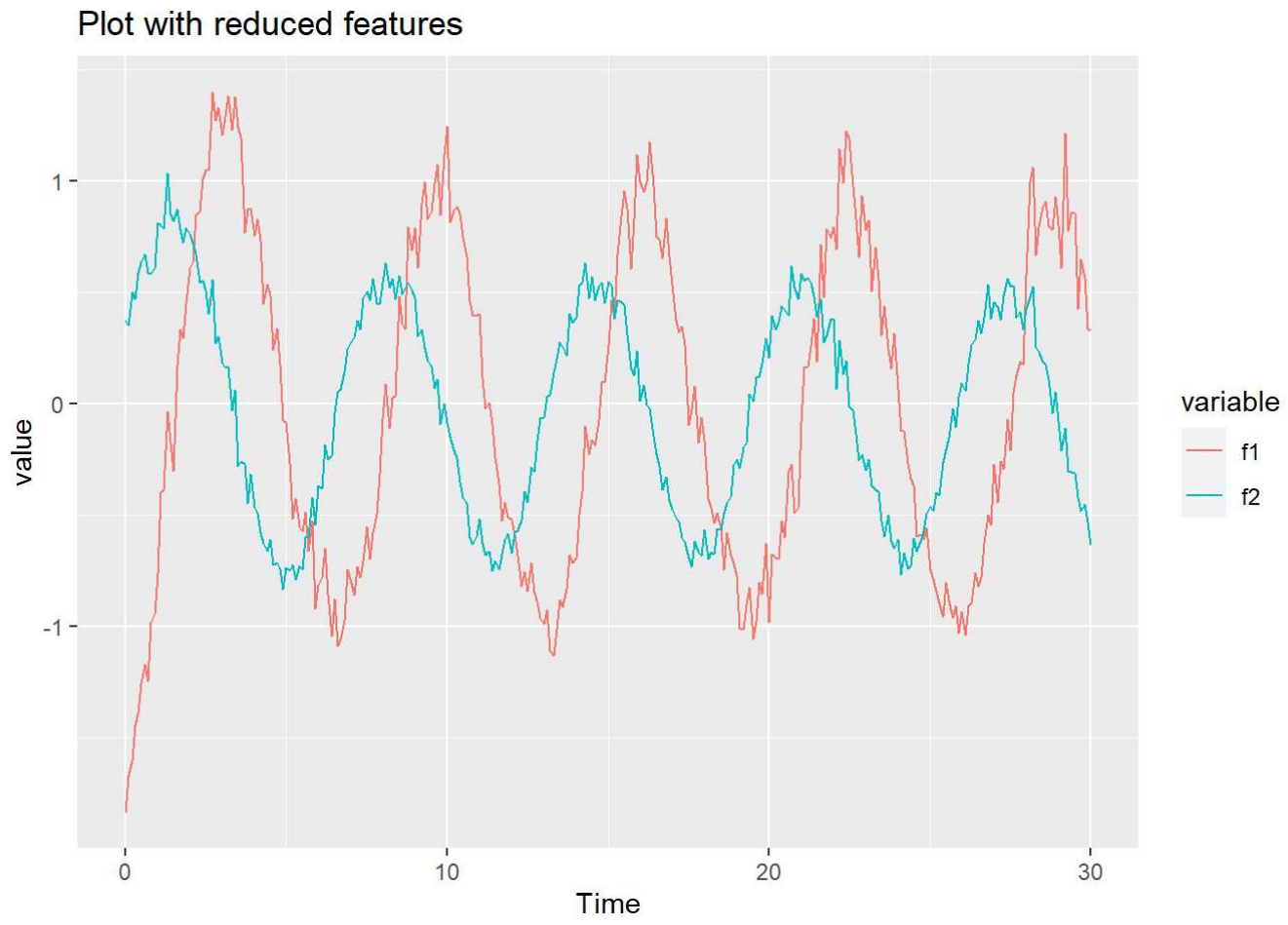
Dimensionality Reduction

In this section we will reduce the dimensions of the data i.e from 5 dimensions to 2 dimensions. Here we have used single value decomposition as a method of choice. In R, prcomp uses single value decomposition for dimensionality reduction.

```
data.pca<- prcomp(data[,2:6])
summary(data.pca)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5
## Standard deviation     0.7498 0.4704 0.11320 0.07515 0.03175
## Proportion of Variance 0.7002 0.2756 0.01596 0.00703 0.00126
## Cumulative Proportion  0.7002 0.9758 0.99171 0.99874 1.00000
```

```
red.data <- as.data.frame(data.pca$x[,1:2]) #Reduced features extraction
red.data$Time = data$Time # Time column addition
colnames(red.data)<-c('f1','f2','Time')

red.melt <- melt(red.data,id='Time')  #Display, reshape
ggplot(data = red.melt, aes(x=Time, y=value)) + geom_line(aes(colour=variable)) +ggtitle('Plot w
ith reduced features')
```



Modelling

In this section, we will do modeling using non linear regression. We will divide the data in two parts i.e train and test.

```
set.seed(123)

sampSize <- floor(0.80 * nrow(data)) #sampling
train_ind <- sample(seq_len(nrow(data)), size = sampSize)
train <- data[train_ind, ] # test train split
test <- data[-train_ind, ]

cat('Shape of train: ',dim(train))
```

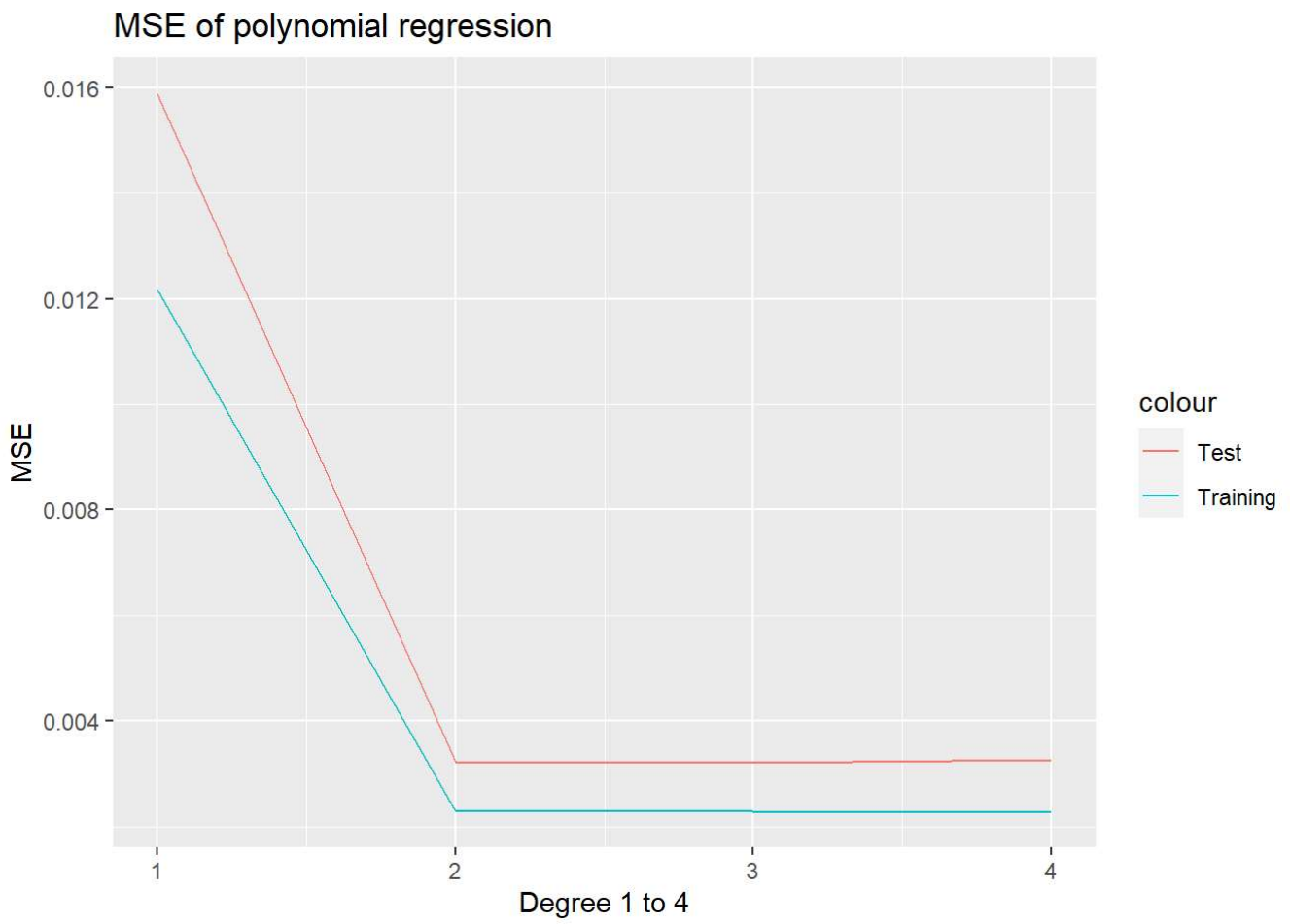
```
## Shape of train:  240 6
```

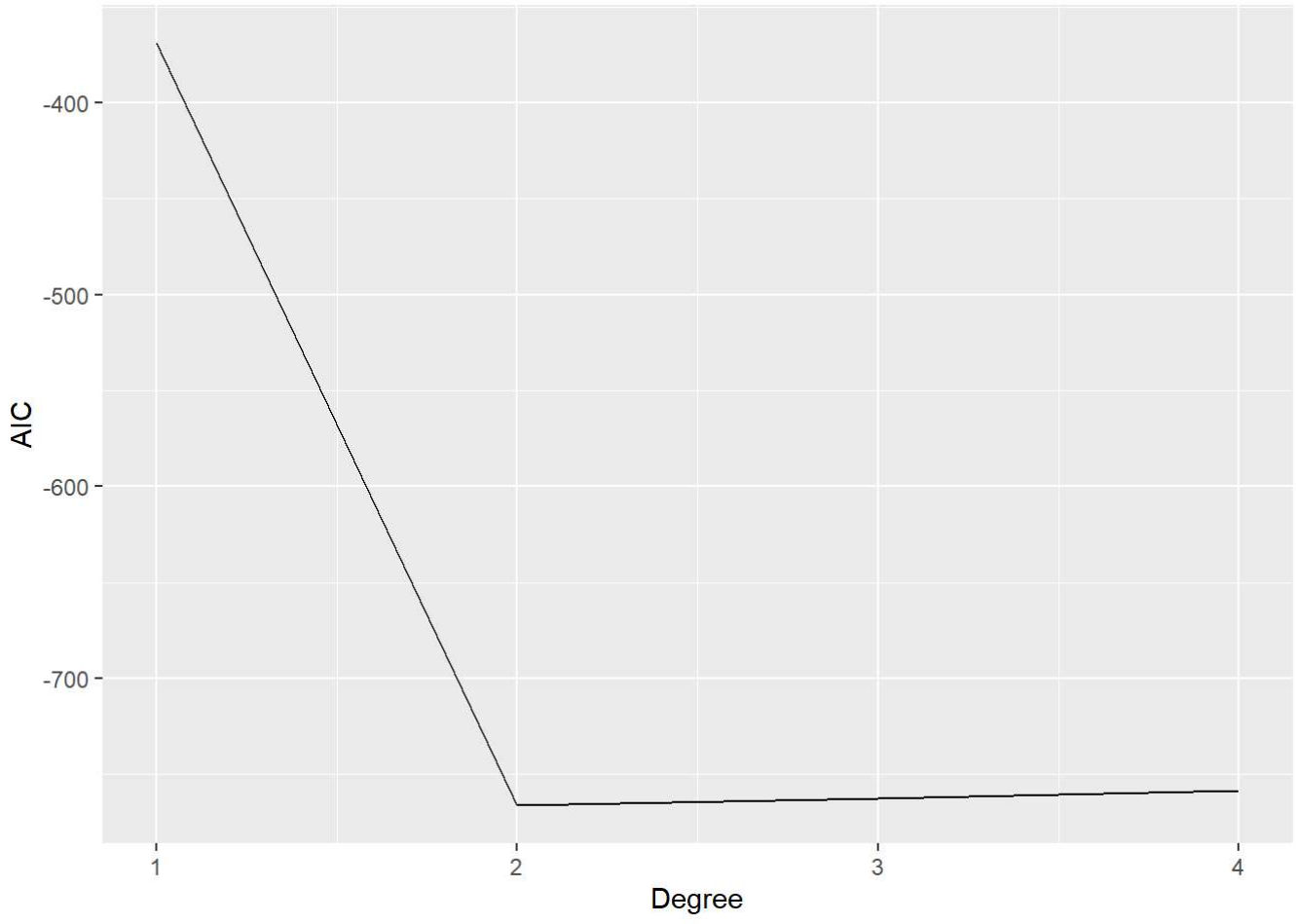
```
cat('\nShape of test: ',dim(test))
```

```
##
## Shape of test:  61 6
```

Iterative approach

```
polySelector <- function(train, test){

  trainMSE <- c()
  testMSE <- c()
  AIC.list <- c()

  for (i in 1:4){
    model = lm(data=train,formula=x3~poly(x4,degree = i)+poly(x5,degree = i))

    AIC.list[i] <- AIC(model)

    trainPreds = predict(model,train)
    testPreds = predict(model,test)

    trainMSE[i] <- mean( (train$x3-trainPreds)^2 )
    testMSE[i] <- mean( (test$x3-testPreds)^2 )
  }

  results <- data.frame(training=trainMSE,test=testMSE,ind=c(1:4))

  MSEPlot <- ggplot(results,aes(ind))+
    geom_line(aes(y=trainMSE,colour='Training'))+
    geom_line(aes(y=testMSE,colour='Test'))+
    ggtitle('MSE of polynomial regression')+
    xlab('Degree 1 to 4')+ylab('MSE')
  print(MSEPlot)

  aicBic <- data.frame(AIC=AIC.list,ind=c(1:4))
  ggplot(aicBic)+geom_line(aes(x=ind,y=AIC))+xlab('Degree')
}

polySelector(train,test)
```


MSE of polynomial regression
0.016
0.012
0.008
0.004
MSE
1
2
3
4
Degree 1 to 4
colour
Test
Training


-400
-500
-600
-700
AIC
1
2
3
4
Degree

Here we can see that the polynomial degree of two gives the lowest MSE on test and trained data. Therefore we will select the degree of two and the model structure would be:

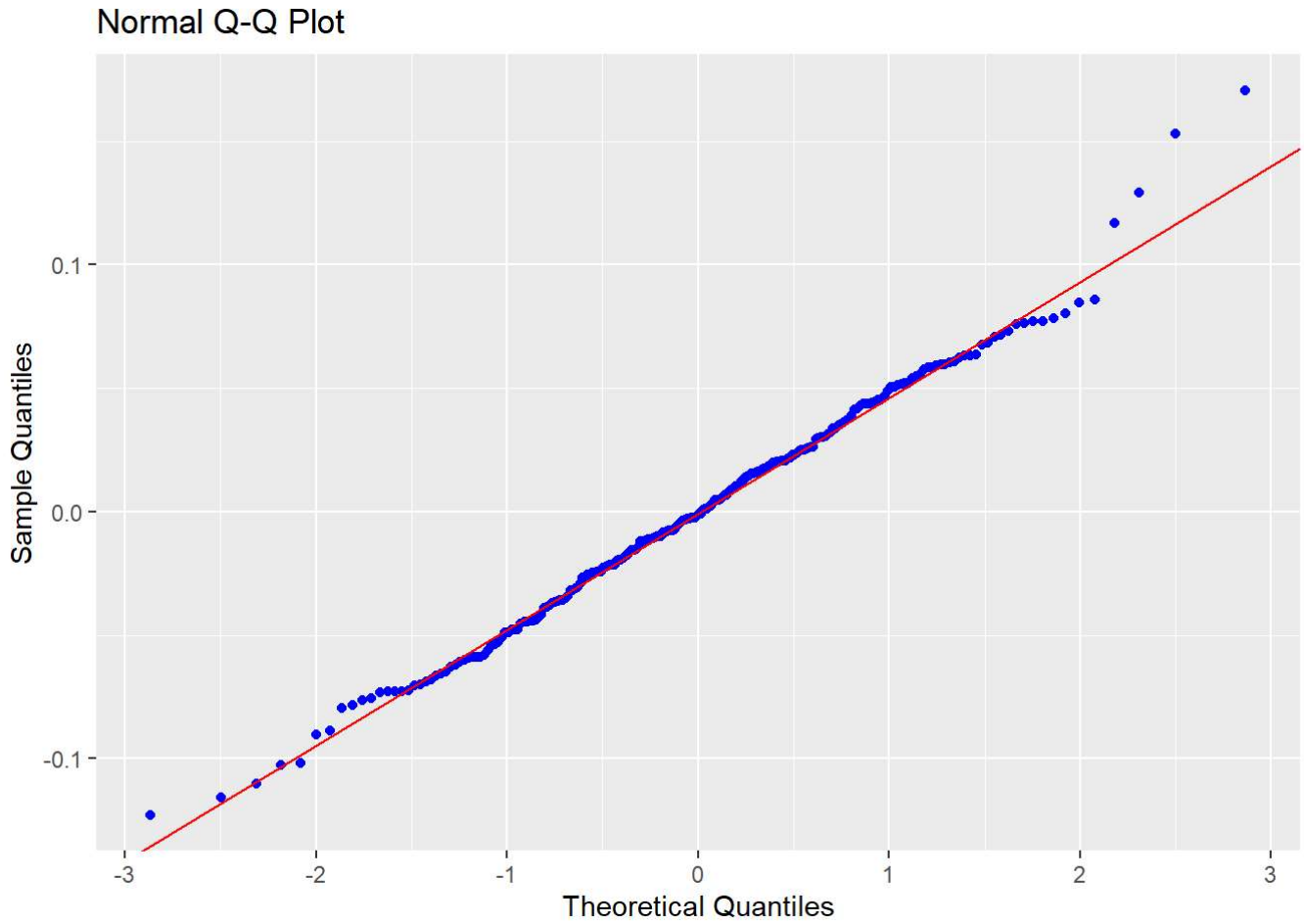
$x_3 = w_0 + a_1x_4 + a2x_4^2 + b_1x_5 + b_2x_5^2 + e$, where e is gaussian noise.

The AIC score is also lowest for model structure of degree 2 and therefore is the most good fit.

Residula normality test

We need to see of the residuals are guassian and for that we do Q-Q plot and we can see from plot below the residuals are near guassian.

```
model = lm(data=train, x3~poly(x4,degree = 2)+poly(x5,degree = 2))
ols_plot_resid_qq(model)
```



Parameter estimation

lm() uses ordinary least sqaures for parameter estimation and hence the parameter estimated are:

$x_3 = 1.1390 - 4.62x_4 + 0.28x_4^2 + 12.95x_5 + 1.28x_5^2$

```
model = lm(data=train, x3~poly(x4,degree = 2)+poly(x5,degree = 2))
model
```

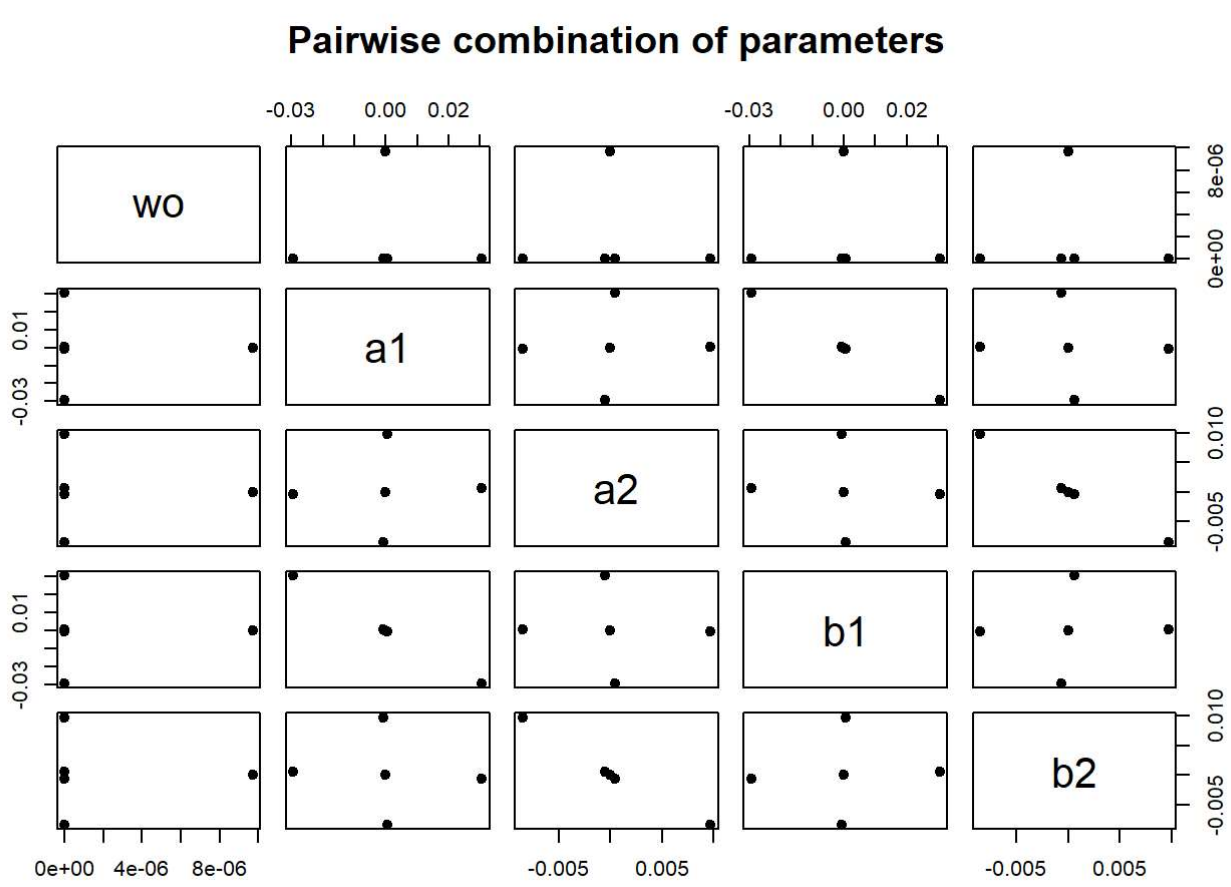
```
##
## Call:
## lm(formula = x3 ~ poly(x4, degree = 2) + poly(x5, degree = 2),
##     data = train)
##
## Coefficients:
##           (Intercept)  poly(x4, degree = 2)1  poly(x4, degree = 2)2
##                1.1390               -4.6230                 0.2822
## poly(x5, degree = 2)1  poly(x5, degree = 2)2
##               12.9514                1.2884
```

Covariance matrix

```
covMat <- vcov(model)

rownames(covMat)<-c('wo','a1','a2','b1','b2')
colnames(covMat) <- c('wo','a1','a2','b1','b2')
print(covMat)
```

```
##               wo            a1            a2            b1            b2
## wo  9.724522e-06  1.393151e-19  6.598815e-20 -1.359000e-19 -6.146860e-20
## a1  1.393151e-19  3.054132e-02  5.117957e-04 -2.934904e-02 -6.226435e-04
## a2  6.598815e-20  5.117957e-04  9.733619e-03 -4.815386e-04 -8.488595e-03
## b1 -1.359000e-19 -2.934904e-02 -4.815386e-04  3.053721e-02  5.893768e-04
## b2 -6.146860e-20 -6.226435e-04 -8.488595e-03  5.893768e-04  9.737725e-03
```

```
pairs(covMat, pch = 19,main='Pairwise combination of parameters')
```

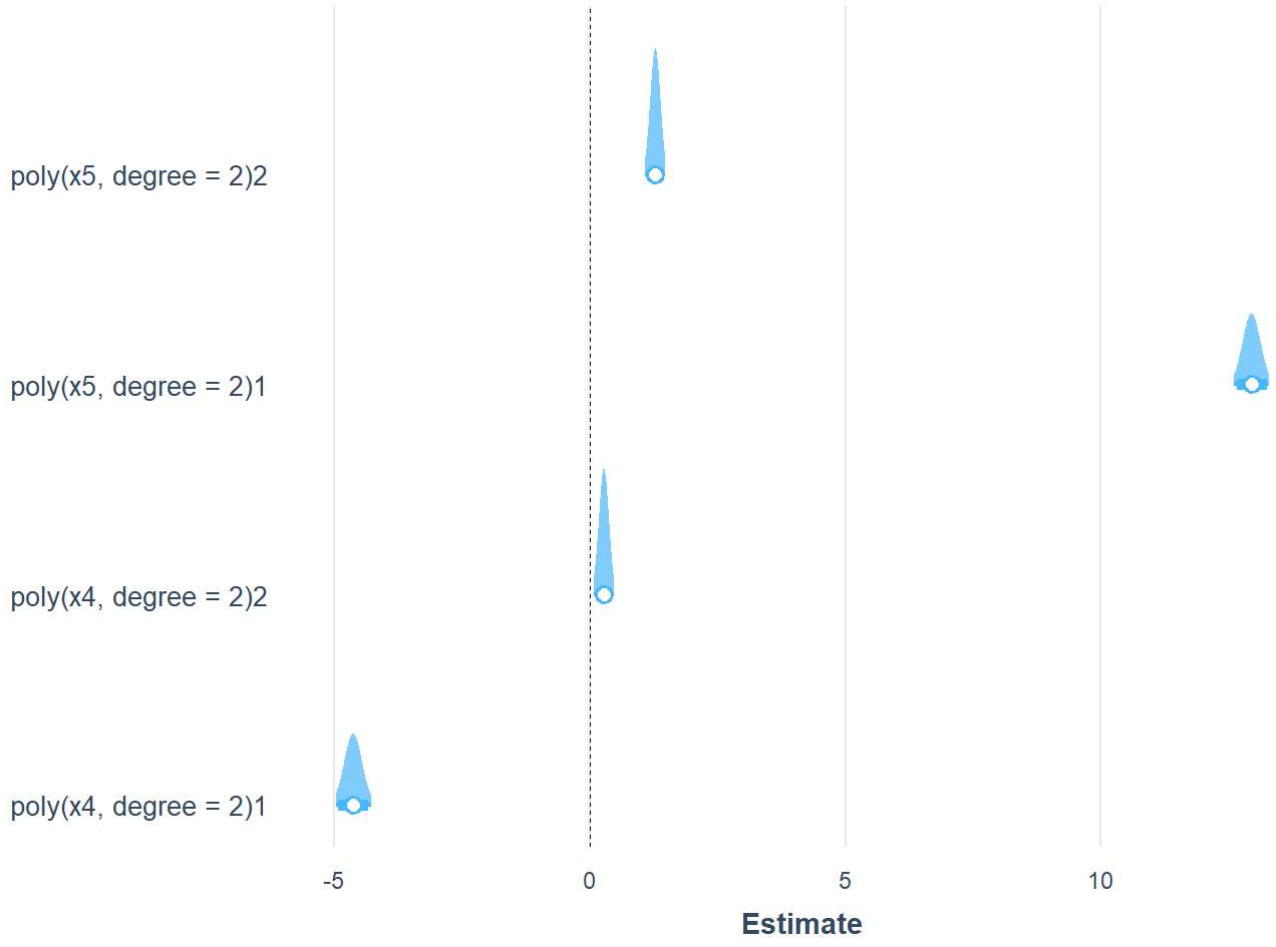



```
plot_summs(model, scale = FALSE, plot.distributions = TRUE, inner_ci_level = .9,main='Paramter u
ncertainity')
```

```
## Registered S3 methods overwritten by 'broom':
##   method            from
##   tidy.glht         jtools
##   tidy.summary.glht jtools
```

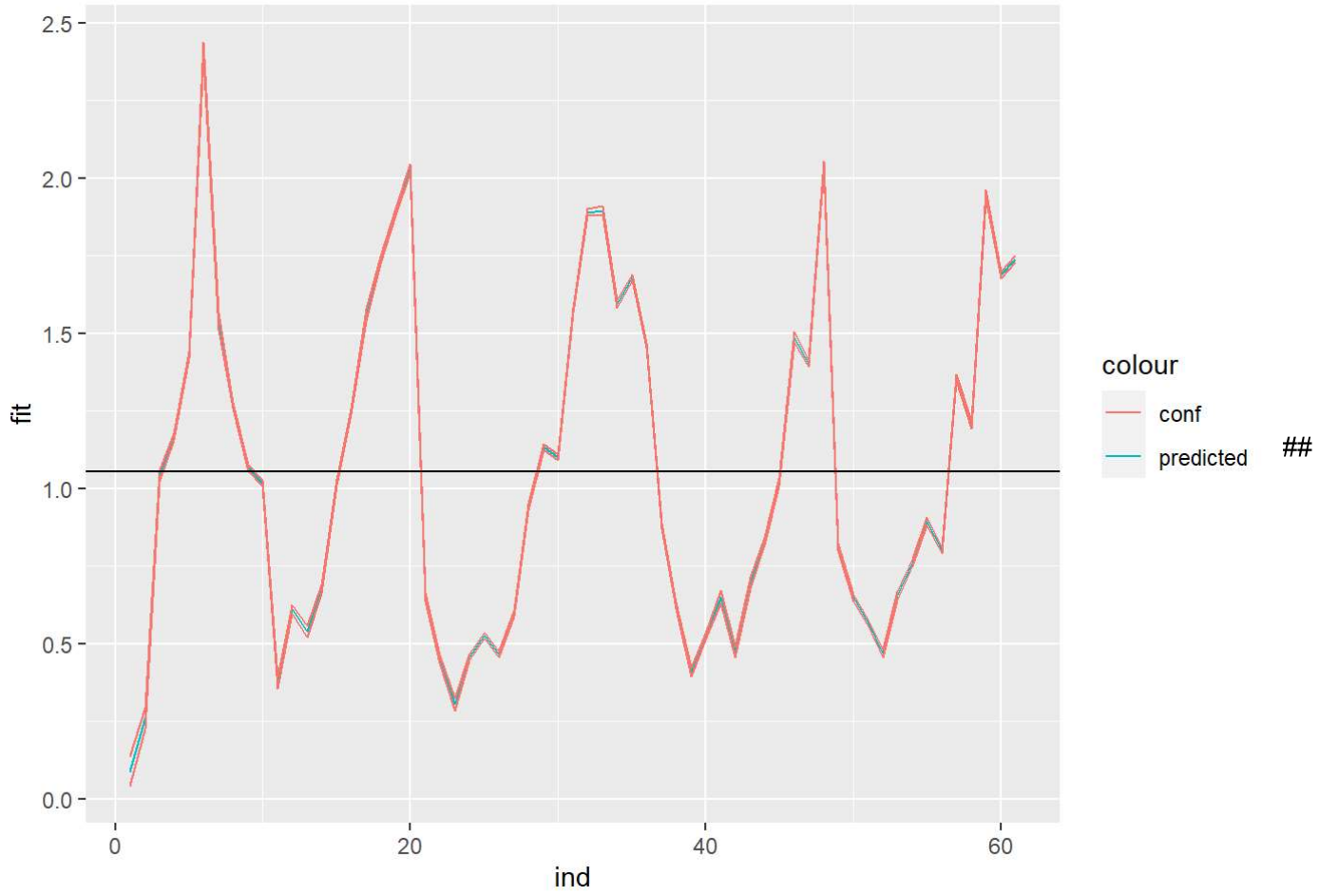
```
## Loading required namespace: broom.mixed
```

```
## Registered S3 methods overwritten by 'broom.mixed':
##   method         from
##   augment.lme    broom
##   augment.merMod broom
##   glance.lme     broom
##   glance.merMod  broom
##   glance.stanreg broom
##   tidy.brmsfit   broom
##   tidy.gamlss    broom
##   tidy.lme       broom
##   tidy.merMod    broom
##   tidy.rjags     broom
##   tidy.stanfit   broom
##   tidy.stanreg   broom
```



Prediction

```
testpreds = as.data.frame( predict(model, newdata = test, interval = "confidence") )
testpreds$ind= c(1:dim(testpreds)[1])  # by default 95 confidence

ggplot(testpreds)+geom_line(aes(x=ind,y=fit,colour='predicted'))+
  geom_line(aes(x=ind,y=upr,colour='conf'))+
  geom_line(aes(x=ind,y=lwr,colour='conf'))+
  geom_hline(yintercept = mean(testpreds$fit))
```



Model Validation

This section uses K-fold cross validation (with 10 iterations) technique for validating the model

```
set.seed(123)
train.control <- trainControl(method = "cv", number = 10)

model <- train(x3~poly(x4,degree = 2)+poly(x5,degree = 2), data = data, method = "lm",
               trControl = train.control)
# Summarize the results
print(model)
```

```
## Linear Regression
##
## 301 samples
##   2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 272, 269, 270, 272, 269, 271, ...
## Resampling results:
##
##   RMSE        Rsquared   MAE
##   0.04997768  0.9926197  0.03966265
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

RMSE of 0.049 was found by 10-fold cross-validation.

Approximate Baysian Computation

```
library(rstanarm)
```

```
## Loading required package: Rcpp
```

```
## Registered S3 methods overwritten by 'lme4':
##   method                          from
##   cooks.distance.influence.merMod car
##   influence.merMod                car
##   dfbeta.influence.merMod         car
##   dfbetas.influence.merMod        car
```

```
## rstanarm (Version 2.19.3, packaged: 2020-02-11 05:16:41 UTC)
```

```
## - Do not expect the default priors to remain the same in future rstanarm versions.
```

```
## Thus, R scripts should specify priors explicitly, even if they are just the defaults.
```

```
## - For execution on a local, multicore CPU with excess RAM we recommend calling
```

```
## options(mc.cores = parallel::detectCores())
```

```
## - bayesplot theme set to bayesplot::theme_default()
```

```
##    * Does _not_ affect other ggplot2 plots
```

```
##    * See ?bayesplot_theme_set for details on theme setting
```

```
##
## Attaching package: 'rstanarm'
```

```
## The following objects are masked from 'package:caret':
##
##     compare_models, R2
```

```
#install.packages("bayestestR")
library(latticeExtra)
```

```
##
## Attaching package: 'latticeExtra'
```

```
## The following object is masked from 'package:ggplot2':
##
##     layer
```

```
library(bayestestR)
model <- stan_glm(x3~poly(x4,degree = 2)+poly(x5,degree = 2), data=data)
```

```
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.287 seconds (Warm-up)
## Chain 1:                0.306 seconds (Sampling)
## Chain 1:                0.593 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.287 seconds (Warm-up)
## Chain 2:                0.476 seconds (Sampling)
## Chain 2:                0.763 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
```

```
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.297 seconds (Warm-up)
## Chain 3:                0.442 seconds (Sampling)
## Chain 3:                0.739 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.276 seconds (Warm-up)
## Chain 4:                0.439 seconds (Sampling)
## Chain 4:                0.715 seconds (Total)
## Chain 4:
```

```
describe_posterior(model)
```

```
## Possible multicollinearity between poly(x5, degree = 2)2 and poly(x4, degree = 2)2 (r = 0.8
8). This might lead to inappropriate results. See 'Details' in '?rope'.
```

```
## # Description of Posterior Distributions
##
## Parameter             | Median | CI | CI_low | CI_high |    pd | ROPE_CI | ROPE_low | ROPE_hi
gh | ROPE_Percentage |  Rhat |  ESS
## ------------------------------------------------------------------------------------------------
---------------------------------
## (Intercept)           |  1.121 | 89 |  1.117 |   1.126 | 1.000 |      89 |   -0.056 |     0.0
56 |           0.000 | 1.000 | 4155
## poly(x4, degree = 2)1 | -5.182 | 89 | -5.474 |  -4.908 | 1.000 |      89 |   -0.056 |     0.0
56 |           0.000 | 1.002 | 2137
## poly(x4, degree = 2)2 |  0.167 | 89 | -0.004 |   0.335 | 0.939 |      89 |   -0.056 |     0.0
56 |           0.105 | 1.002 | 2578
## poly(x5, degree = 2)1 | 14.462 | 89 | 14.165 |  14.732 | 1.000 |      89 |   -0.056 |     0.0
56 |           0.000 | 1.003 | 2133
## poly(x5, degree = 2)2 |  1.624 | 89 |  1.454 |   1.791 | 1.000 |      89 |   -0.056 |     0.0
56 |           0.000 | 1.002 | 2634
```

Marginal Distribution

```
library(latticeExtra)
marginal.plot(model)
```