# 042 Rainfall forecast for England

Wenxin Xu (uceswx0)

2022-05-08

## Load the required libraries

```
library(maptools)
library(lattice)
library(spdep)
library(sp)
library(rgdal)
library(ggplot2)
library(gridExtra)
library(gstat)
library(spacetime)
library(forecast)
library(nnet)
library(caret)
library(kernlab)
```

# 1 data processing and analysis

## 1.1 load data

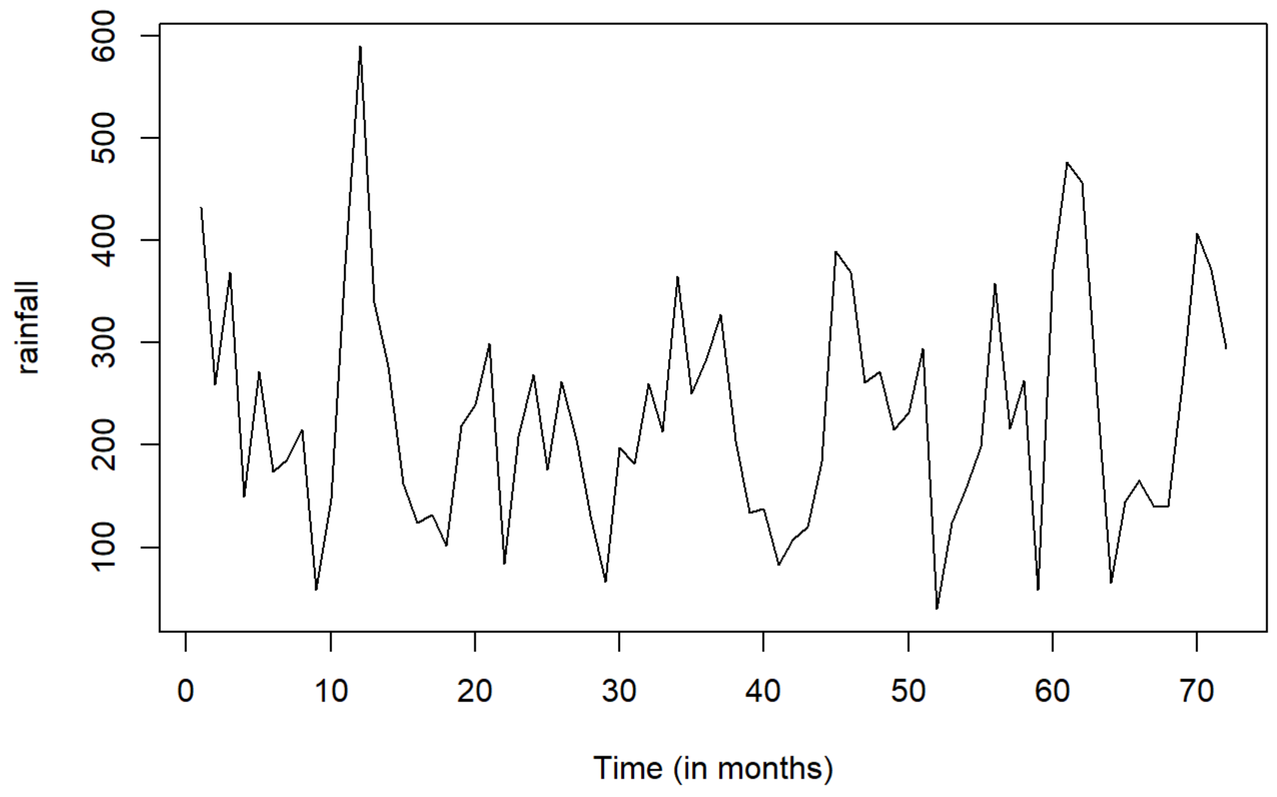
Here we import the data set and do some preliminary processing to generate the matrix.

```
# read csv data
rainfall <- read.csv("data/rainfall.csv")

# transform to matrix
rainfall_matrix <- data.matrix(rainfall[, 7:ncol(rainfall)])

# add rowname
rownames(rainfall_matrix) <- rainfall[, "name"]
```

## 1.2 Time series decomposition

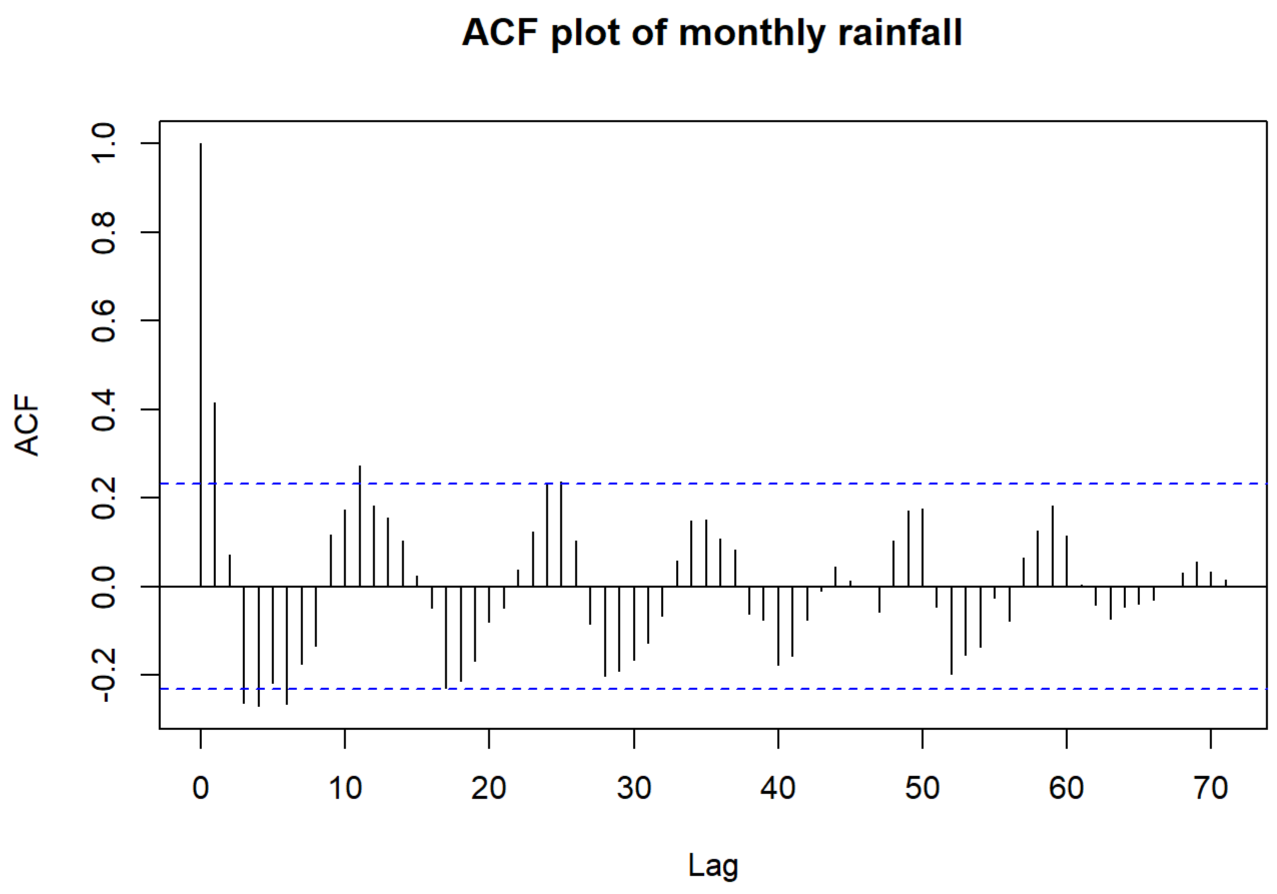
Here we use a line graph to visualise 72 months of time data for the Berkshire region.

```
# temporal data visulization
plot(rainfall_matrix["Berkshire",], ylab="rainfall", xlab="Time (in months)", type="l")
```

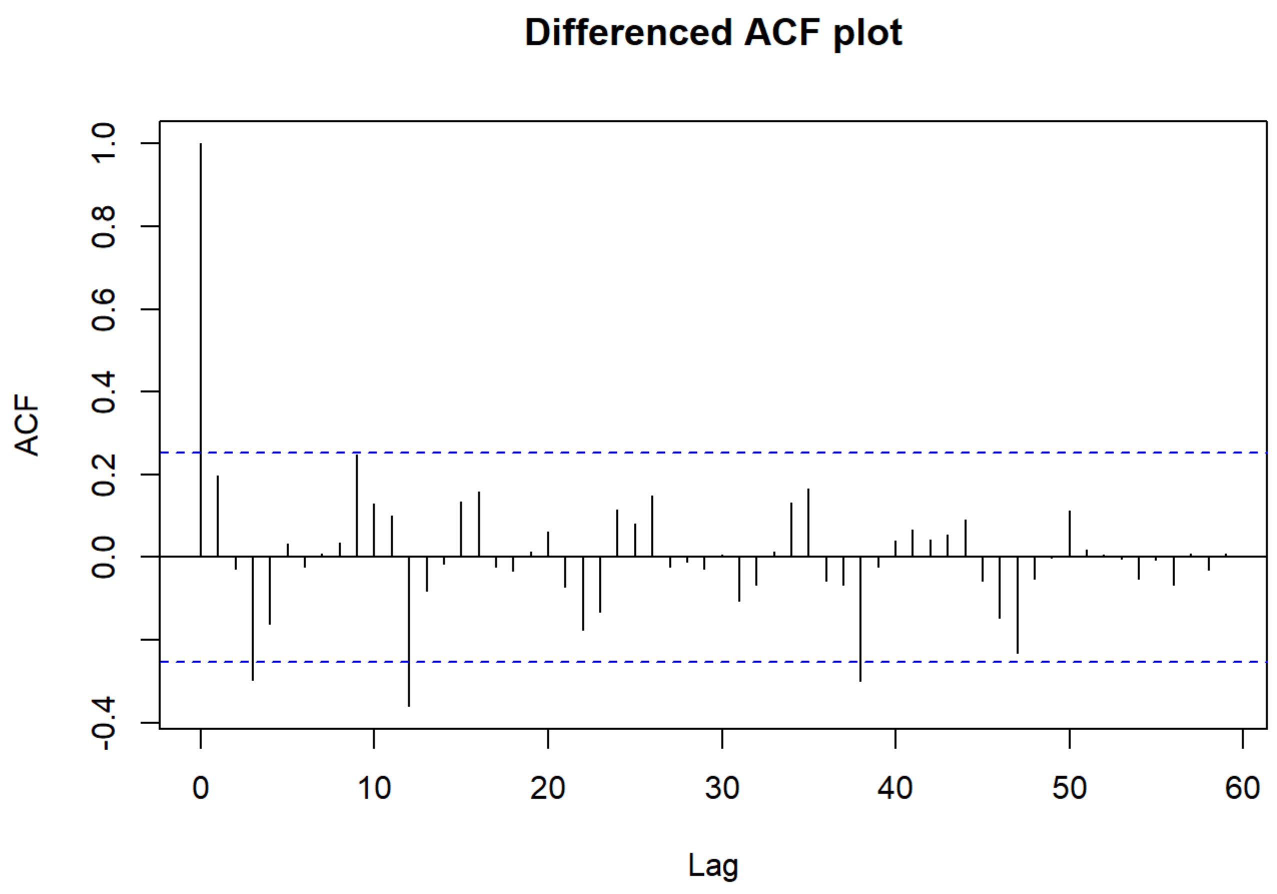


We need to examine the temporal autocorrelation of the data, first plotting the acf plot and finding that the data cycles in cycles of 12, i.e. the seasonal component has an order of 12, corresponding to 12 months of the year. The time lag of 12 needs to be differenced to eliminate the cyclical pattern and then the differenced acf plot needs to be analysed.

```
# load function package
source("data/starima_package.R")

# acf
acf(rainfall_matrix["Berkshire",], lag.max=72, xlab="Lag", ylab="ACF", main="ACF plot of monthl
y rainfall")
```



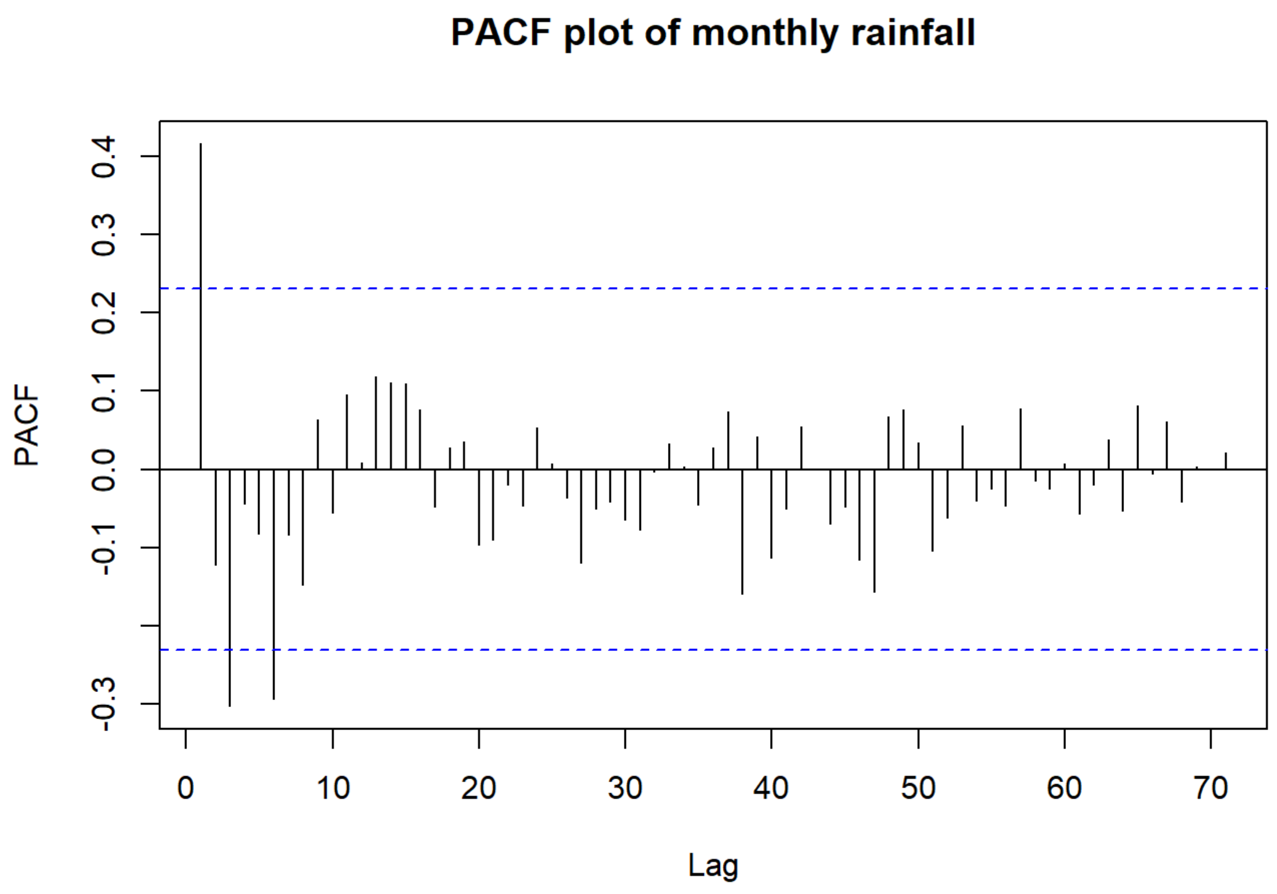
```
# differenced acf
rainfall_diff <- diff(rainfall_matrix["Berkshire",], lag=12, differences=1)
acf(rainfall_diff, lag.max=72, xlab="Lag", ylab="ACF", main="Differenced ACF plot")
```



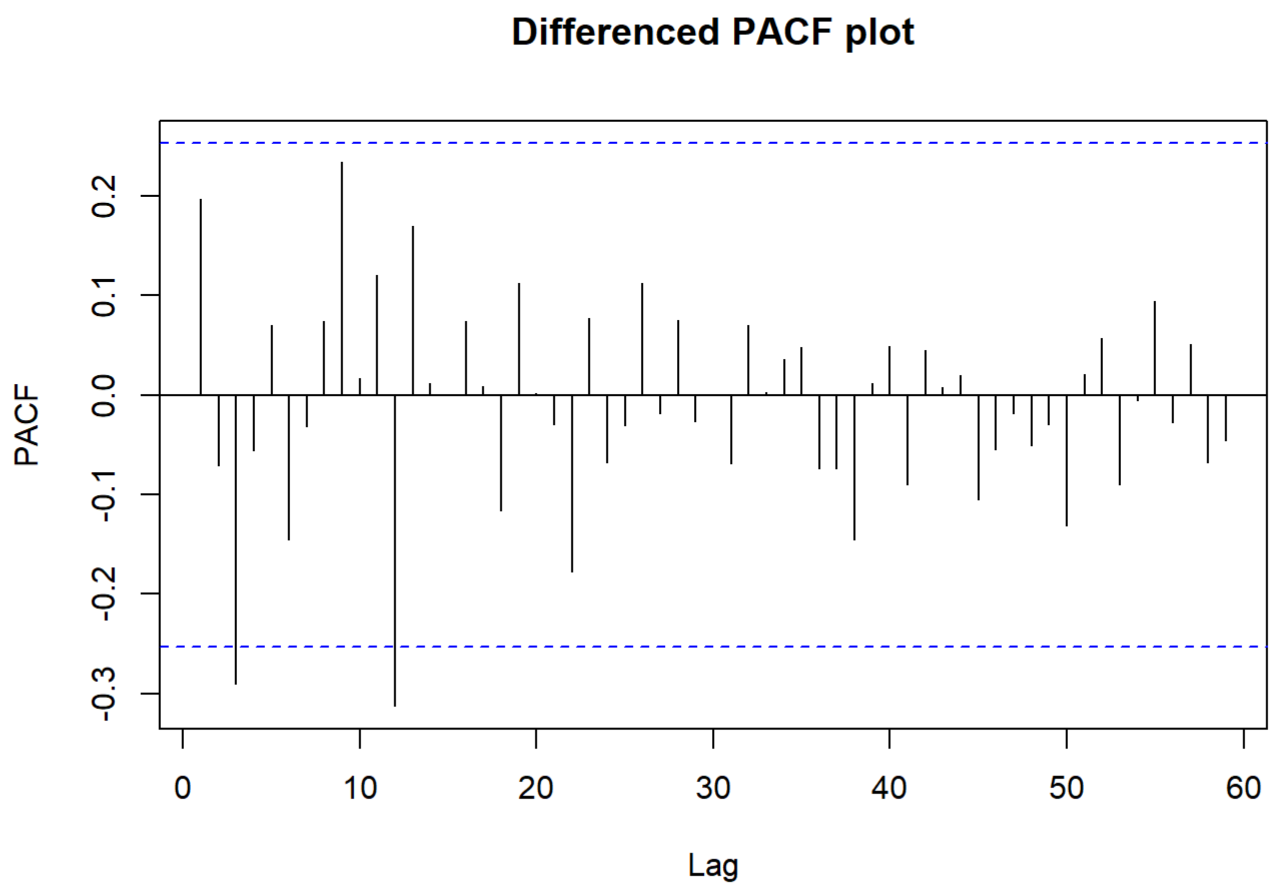
The differenced acf plot eliminates seasonal cycling in the data, with significant temporal autocorrelation at lags 3, 12 and 38.

We need to analyse the significance of the differenced pacf plot to determine the arima MA order.

```
# pacf
pacf(rainfall_matrix["Berkshire",], lag.max=72,xlab="Lag",ylab="PACF",main="PACF plot of monthl
y rainfall")
```



```
# differenced pacf
pacf(rainfall_diff, lag.max=72, xlab="Lag", ylab="PACF",main="Differenced PACF plot")
```



The PACF plot has a clear peak at lag 12.

## 2 ARIMA

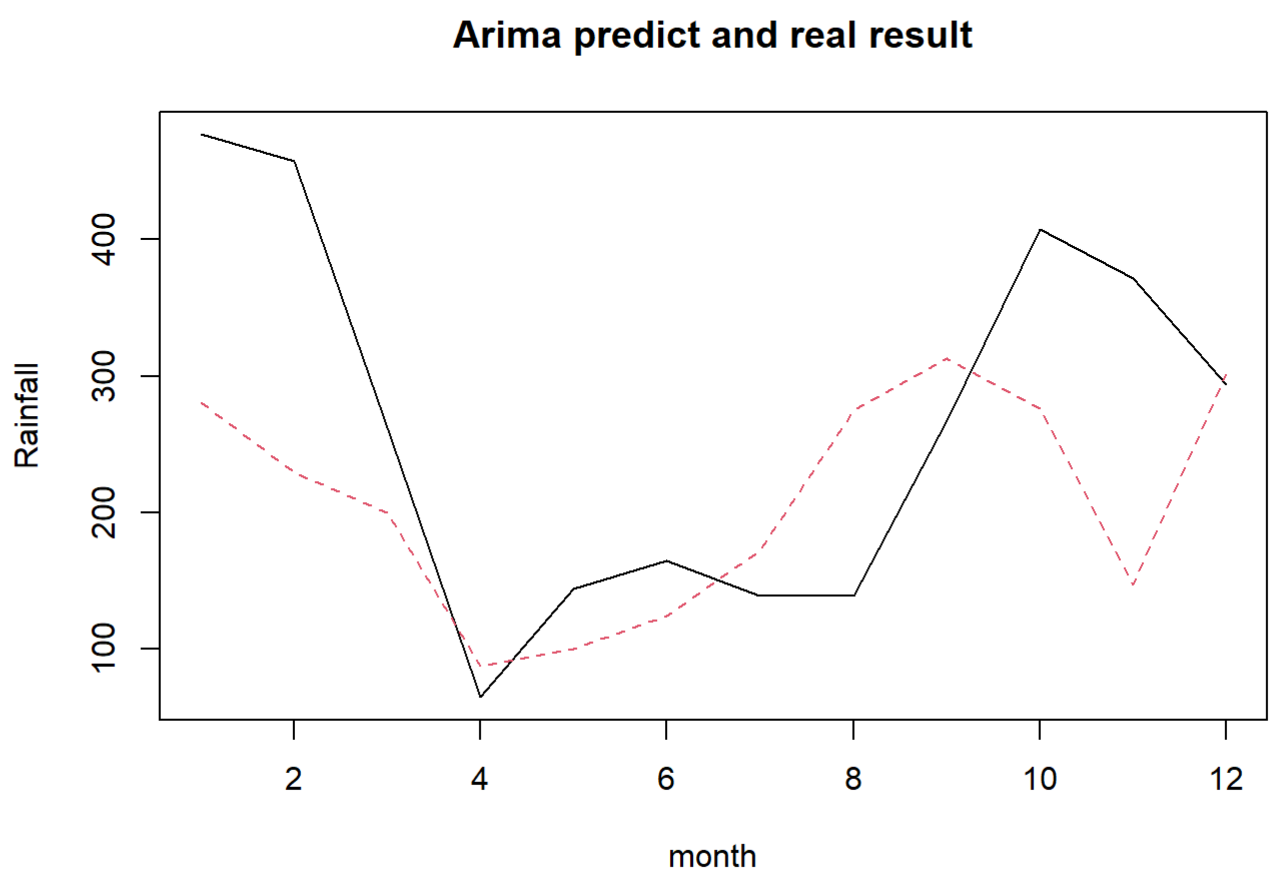
Using the ACF and PACF plots after differencing above, we can determine the parameters of the arima model as arima(1,0,1)(1,1,2)12. The following arima fit is performed using data from 1-60 as the training set.

```
# fit
# arima(1,0,1)(1,1,2)12
timestart<-Sys.time()

fit_arima <- arima(rainfall_matrix["Berkshire",1:60], order=c(1,0,1),
                   seasonal=list(order=c(1,1,2), period=12))

# time
timeend<-Sys.time()
runningtime_ar <-timeend-timestart
print(runningtime_ar)
```

```
## Time difference of 2.336737 secs
```

The modeling time was calculated by fitting the arima model above. Next, the model is projected for the next 12 months and plotted against the true values, with the true values in black and the predicted values in red dashed lines.

```
# predict
pre_arima <- predict(fit_arima, n.ahead=12)
matplot(1:12,cbind(rainfall_matrix["Berkshire",61:72],pre_arima$pred),type="l",main="Arima pred
ict and real result", xlab="month", ylab="Rainfall")
```



The predictions were generally in line with the trend. Next the arima model was evaluated parametrically, using the ormalized Root Mean Square Error, variance and running time as evaluation metrics for the calculations.

```
# Evaluation
# Normalized Root Mean Square Error
NRMSE_ar <- NRMSE(res=fit_arima$residuals,
                  obs=rainfall_matrix["Berkshire",1:60])
NRMSE_ar
```

```
## [1] 0.841011
```

```
# Variance
var_ar <- 0
count <- 0
for (fitted in pre_arima$pred){
  var_ar <- var_ar + (fitted-rainfall_matrix["Berkshire",61+count])^2
  count <- count+1
}
var_ar/count
```

```
## [1] 15643.72
```

```
# time
print(runningtime_ar)
```

```
## Time difference of 2.336737 secs
```

# 3 ANN

For the ANN model the values from the first three months are used as input to predict the next month’s oracle, so the dataset needs to be reconstructed into data values for the four time points. The dataset is then sliced into a training set and a test set (12 months) and fitted using the ANN model for training. After several sets of tests, a decay factor of decay=5e-2, a neuron size=14 and a number of iterations maxit=1000 are selected.

```
# Reconstructing the data matrix
m <- 3
x <- t(as.matrix(rainfall_matrix))
svm_data <- embed(x =x[,"Berkshire"], dimension = m+1)
colnames(svm_data) <- c("y_t-3","y_t-2","y_t-1","y_t")

# split dataset into train and test dataset
yTrain <- svm_data[1:57,1]
xTrain <- svm_data[1:57,-1]
yTest <- svm_data[58:nrow(svm_data),1]
xTest <- svm_data[58:nrow(svm_data),-1]

# train model
timestart<-Sys.time()
fit_ann <- nnet(xTrain, yTrain, decay=5e-2, linout = TRUE, size=14, maxit=1000)
```

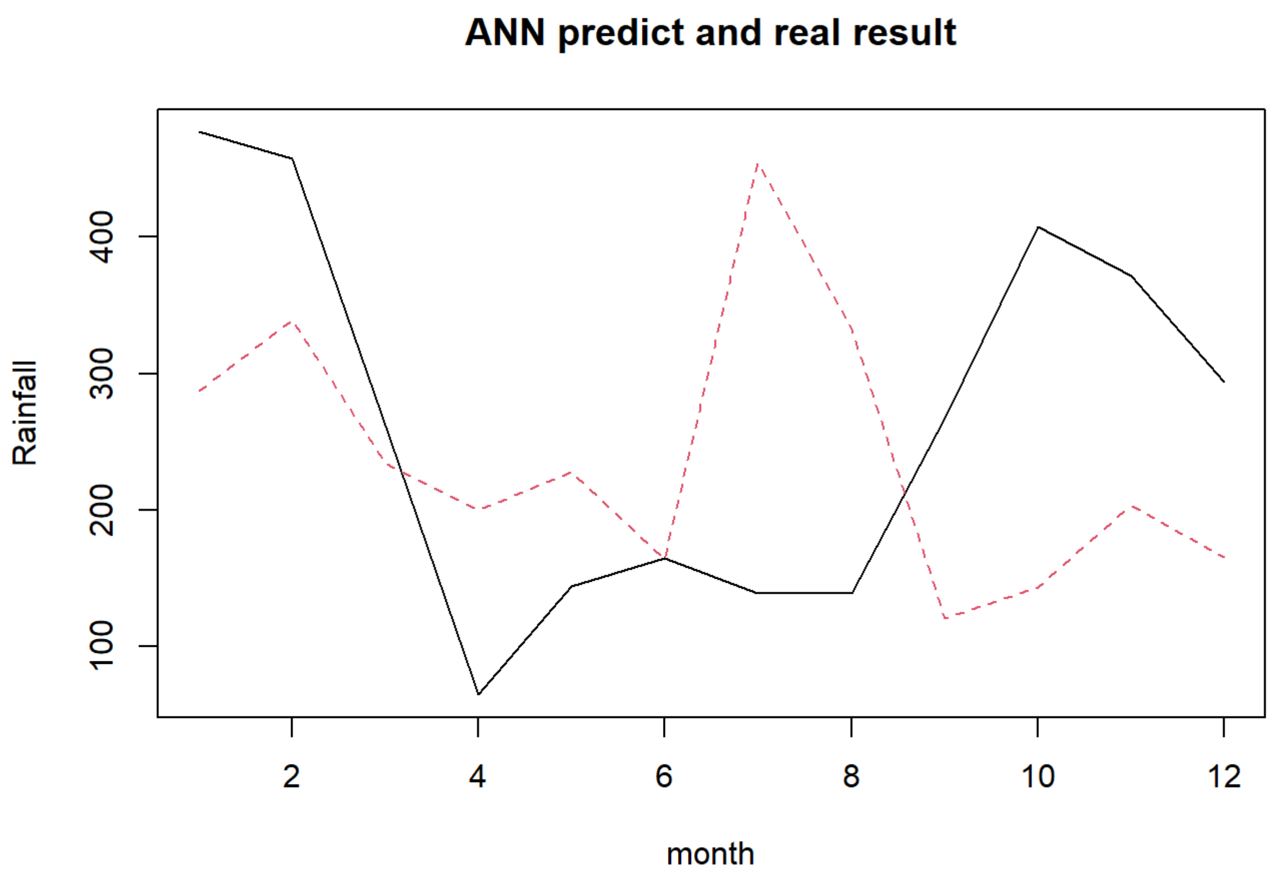
```
## # weights:  71
## initial  value 3207788.079867
## iter  10 value 600724.953621
## iter  20 value 593843.709399
## iter  30 value 431040.453831
## iter  40 value 398419.596377
## iter  50 value 391799.347425
## iter  60 value 390415.349598
## iter  70 value 373189.715374
## iter  80 value 362386.815927
## iter  90 value 358890.043185
## iter 100 value 356389.690736
## iter 110 value 347530.717935
## iter 120 value 326224.991500
## iter 130 value 318329.862085
## iter 140 value 305676.106716
## iter 150 value 305011.406947
## iter 160 value 304236.816239
## iter 170 value 301700.089494
## iter 180 value 300986.663194
## iter 190 value 300608.590532
## iter 200 value 299915.937527
## iter 210 value 285409.106216
## iter 220 value 279717.015313
## iter 230 value 274559.238756
## iter 240 value 273553.416627
## iter 250 value 254366.057019
## iter 260 value 253748.710318
## iter 270 value 236438.267999
## iter 280 value 236309.282218
## iter 290 value 236183.169333
## iter 300 value 234953.368225
## iter 310 value 233827.907429
## iter 320 value 233160.144962
## iter 330 value 215345.369061
## iter 340 value 205315.440465
## iter 350 value 205020.141270
## iter 360 value 194811.706358
## iter 370 value 194498.856656
## iter 380 value 194117.434213
## iter 390 value 193987.294424
## iter 400 value 186942.785253
## iter 410 value 185677.656104
## iter 420 value 184860.262739
## iter 430 value 176978.755605
## iter 440 value 176605.274063
## iter 450 value 176270.677102
## iter 460 value 175313.844298
## iter 470 value 170035.691355
## iter 480 value 164544.851023
## iter 490 value 149244.663339
## iter 500 value 145904.003363
## iter 510 value 144799.052507
## iter 520 value 144112.099591
## iter 530 value 142255.464762
## iter 540 value 141829.726952
## iter 550 value 141395.568110
## iter 560 value 141232.450283
## iter 570 value 140987.248438
## iter 580 value 140239.050458
## iter 590 value 136538.001544
## iter 600 value 135803.755841
## iter 610 value 132946.499922
## iter 620 value 131621.939290
## iter 630 value 128039.568136
## iter 640 value 127497.221730
## iter 650 value 127484.529042
## iter 660 value 127435.394405
## iter 670 value 127343.445827
## iter 680 value 126824.902645
## iter 690 value 126273.773620
## iter 700 value 123576.318815
## iter 710 value 123073.850807
## iter 720 value 122695.874657
## iter 730 value 119981.584169
## iter 740 value 116056.679196
## iter 750 value 115578.890745
## iter 760 value 115324.794820
## iter 770 value 113830.210706
## iter 780 value 111423.818840
## iter 790 value 109973.453554
## iter 800 value 108426.006958
## iter 810 value 108272.320073
## iter 820 value 108083.417540
## iter 830 value 107987.011819
## iter 840 value 107149.635013
## iter 850 value 104991.849367
## iter 860 value 104465.056284
## iter 870 value 102133.277208
## iter 880 value 80464.908843
## iter 890 value 77021.372332
## iter 900 value 76758.308182
## iter 910 value 76358.425999
## iter 920 value 75677.074826
## iter 930 value 75602.099039
## iter 940 value 73059.482477
## iter 950 value 71626.449896
## iter 960 value 70948.163303
## iter 970 value 70899.908596
## iter 980 value 70673.137445
## iter 990 value 68179.079036
## iter1000 value 68056.750440
## final  value 68056.750440
## stopped after 1000 iterations
```

```
# train time
timeend<-Sys.time()
runningtime_ann <-timeend-timestart
print(runningtime_ann)
```

```
## Time difference of 0.1154699 secs
```

The trained ANN model is predicted and the prediction results are plotted against each other.

```
#predict
pred_ann<-predict(fit_ann, xTest)

matplot(cbind(yTest, pred_ann[,]), type="l", main="ANN predict and real result", xlab="month",y
lab="Rainfall")
```



For the ANN model fit evaluation, the same NRMSE, variance and training time were chosen as for Arima.

```
# evaluation
# NRMSE
NRMSE_ann <- NRMSE(fit_ann$residuals,obs=yTest)
NRMSE_ann
```

```
## [1] 0.2184084
```

```
# variance
var_ar <- 0
count <- 0
for (fitted in pred_ann[,]){
  var_ar <- var_ar + (fitted-yTest[1+count])^2
  count <- count+1
}
var_ar/count
```

```
## [1] 28990.21
```

```
# time
print(runningtime_ann)
```

```
## Time difference of 0.1154699 secs
```

## 4 SVM

The SVM model is similar to the ANN model in that it uses the first three months to predict the next month. However, here a five-fold cross-validation is performed to find the optimal parameters for the selected SVM hyperparameters, which are sigma = 0.01 and C = 100

```
# set cross_validation
cv <- trainControl(method = "cv", number=5)

# training model
# set dataset to find the most parameterized
SVR_grid <- expand.grid(.sigma=c(0.001, 0.01, 0.1,1), .C=c(1,10,100,1000))
# Cross-validation to find optimal parameters
timestart<-Sys.time()

SVR_fit <- train(xTrain, yTrain, method="svmRadial",
                      tuneGrid=SVR_grid, trControl=cv, type="eps-svr")
SVR_fit
```

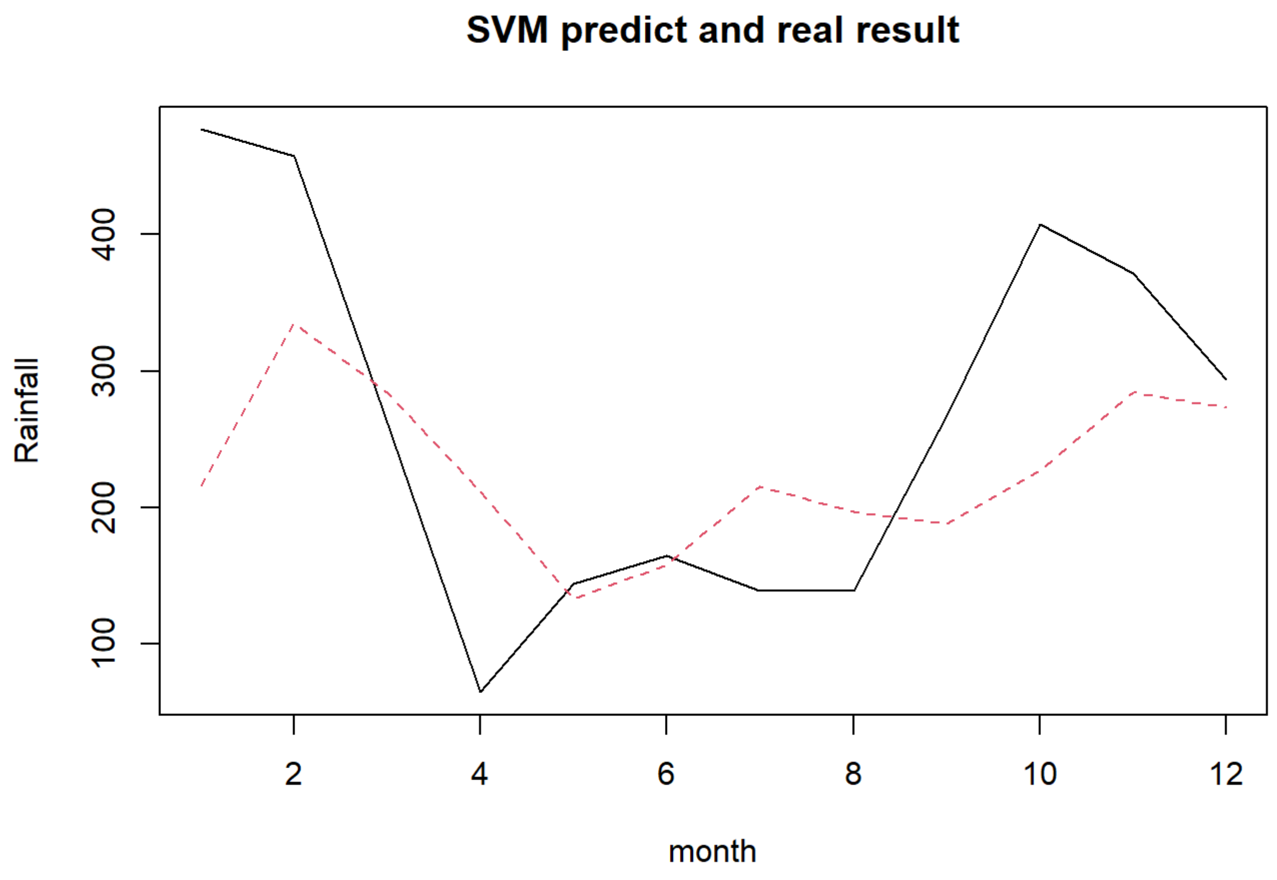
```
## Support Vector Machines with Radial Basis Function Kernel
##
## 57 samples
##  3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 45, 46, 45, 45, 47
## Resampling results across tuning parameters:
##
##   sigma  C     RMSE       Rsquared    MAE
##   0.001     1  100.71906  0.15493522   78.74308
##   0.001    10   96.77398  0.17564907   74.14171
##   0.001   100   94.98782  0.20414737   72.97360
##   0.001  1000   95.99730  0.20139297   73.88089
##   0.010     1   97.35771  0.16280731   75.02361
##   0.010    10   96.29004  0.18469153   74.51962
##   0.010   100   96.15448  0.18930243   76.02260
##   0.010  1000   97.60750  0.17540313   77.20511
##   0.100     1  100.22796  0.11598275   78.87330
##   0.100    10  101.21728  0.11900290   78.90662
##   0.100   100  120.99178  0.14040588   97.70057
##   0.100  1000  197.42393  0.13945102  146.40495
##   1.000     1  107.84332  0.06159354   87.54232
##   1.000    10  140.24922  0.07576462  113.78515
##   1.000   100  165.45211  0.02907350  134.21795
##   1.000  1000  256.95049  0.03527911  198.24266
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.001 and C = 100.
```

```
# time
timeend<-Sys.time()
runningtime_svm <-timeend-timestart
print(runningtime_svm)
```

```
## Time difference of 2.036645 secs
```

Predictions are made and plotted using the trained model.

```
# predict
# sigma = 0.01 and C = 100
svm_pred <- predict(SVR_fit, xTest)

matplot(cbind(yTest,svm_pred),ylab="Rainfall", xlab="month", main="SVM predict and real result"
, type="l")
```



The SVM results were evaluated. The SVM was unable to output residuals, so it was evaluated only for variance and training time.

```
# evaluation
# Variance
var_ar <- 0
count <- 0
for (fitted in svm_pred){
  var_ar <- var_ar + (fitted-yTest[1+count])^2
  count <- count+1
}
var_ar/count
```

```
## [1] 13418.38
```

```
# time
print(runningtime_svm)
```

```
## Time difference of 2.036645 secs
```