

# The Drought Dataset

Presenters: Jay Barber and Dan Pagendam

# The Drought Dataset

- Exercise developed by Chris Wikle and Dan Pagendam (2019).
- The data for this exercise consists of:
  - monthly grids (2 degree x 2 degree) of sea surface temperature (SST) anomaly.
  - monthly rainfall anomaly in mm for the Murray Darling Basin (MDB).
- The data was obtained from two sources:
  - <http://www.bom.gov.au/climate/change/>
  - <http://iridl.ldeo.columbia.edu/>
- We'll attempt to use a **Long Short-Term Memory (LSTM)** Model to obtain 3 month out forecasts of rainfall anomaly from SST grids.

# Required Packages

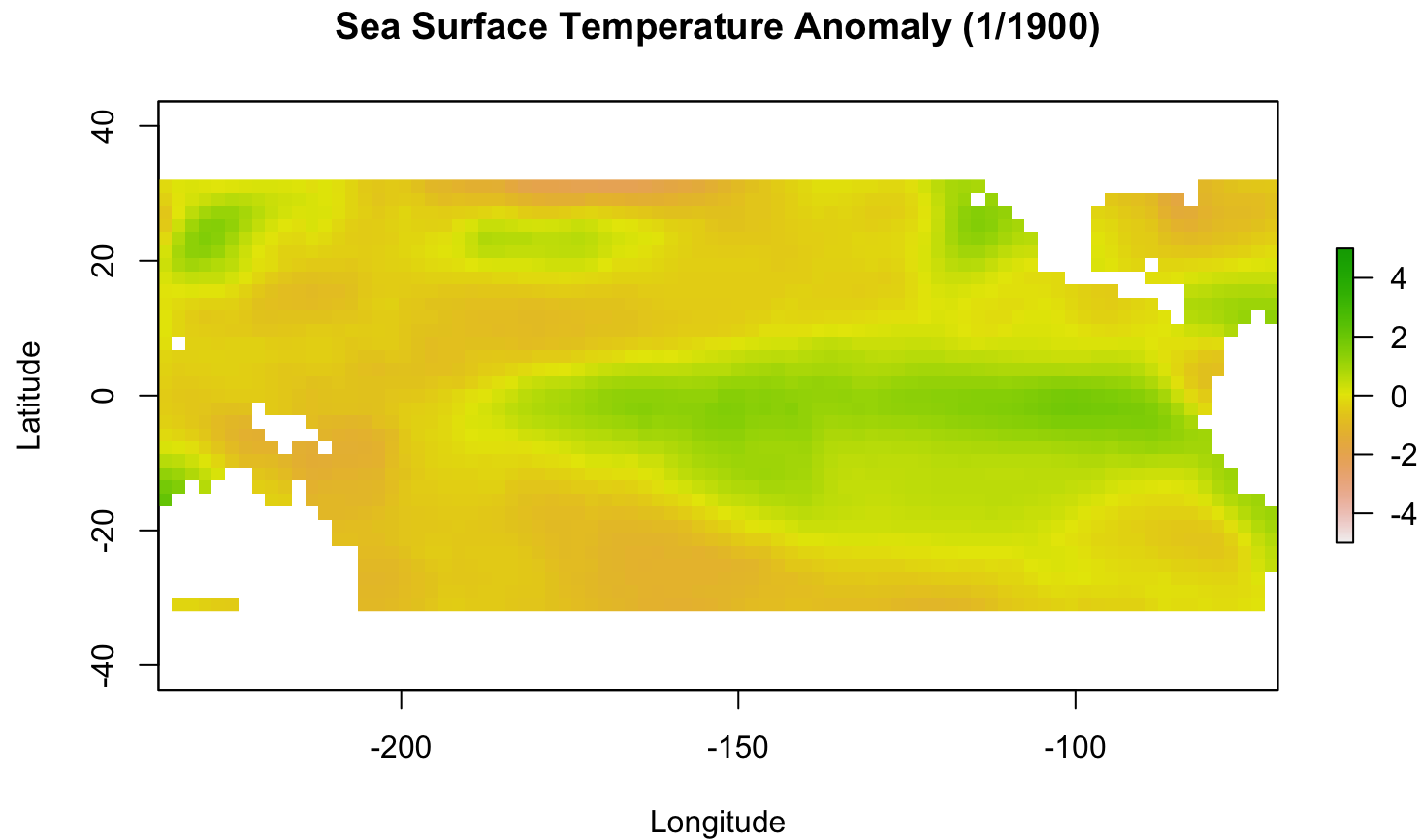
- For this exercise you will require the following packages:
  - raster
- You can install and load these as follows:

```
install.packages("raster", repos = "http://cran.us.r-project.org",  
                quiet = TRUE)  
library(raster)  
library(deepLearningRshort)  
library(keras)
```

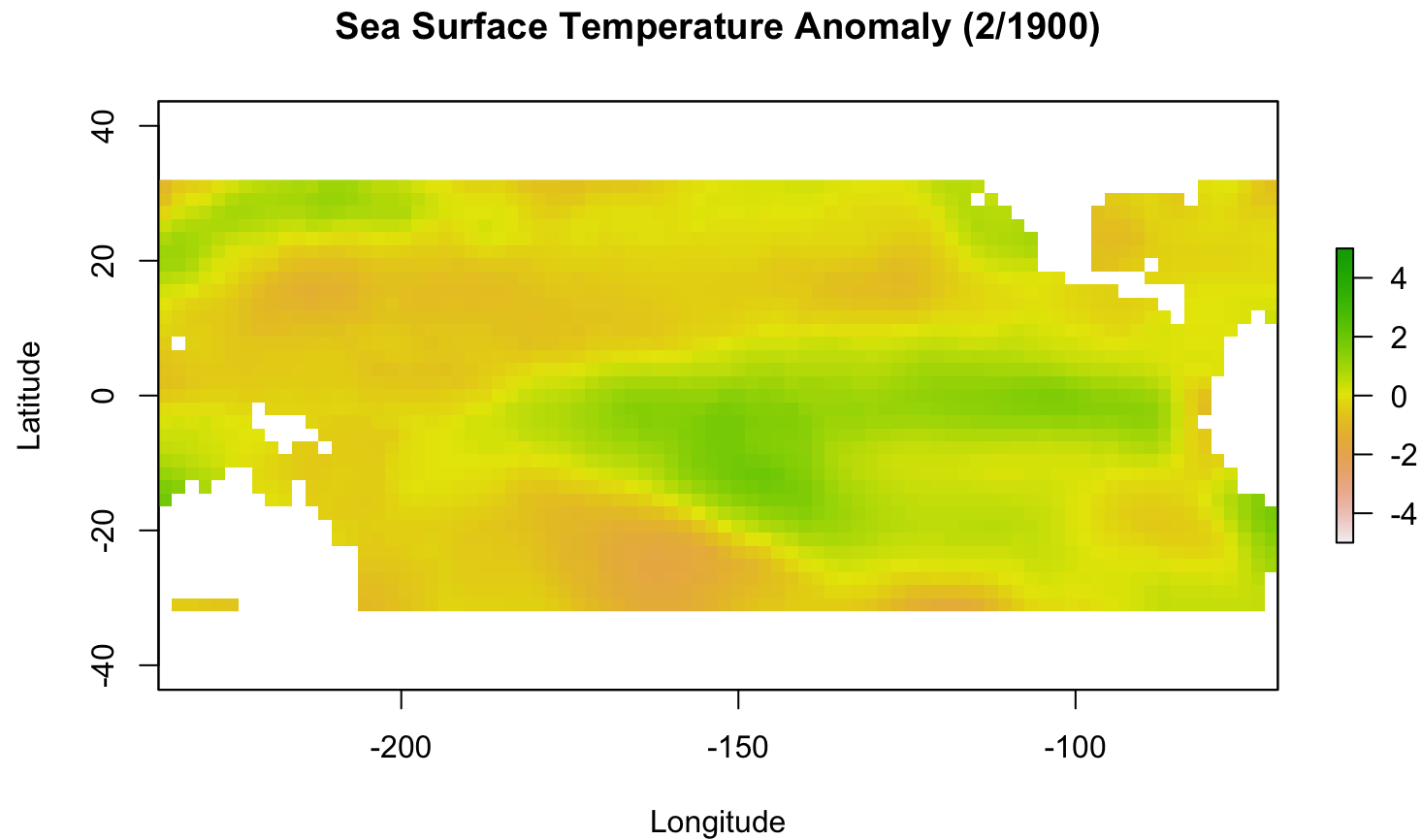
# Load the Drought Data

```
data(drought)
```

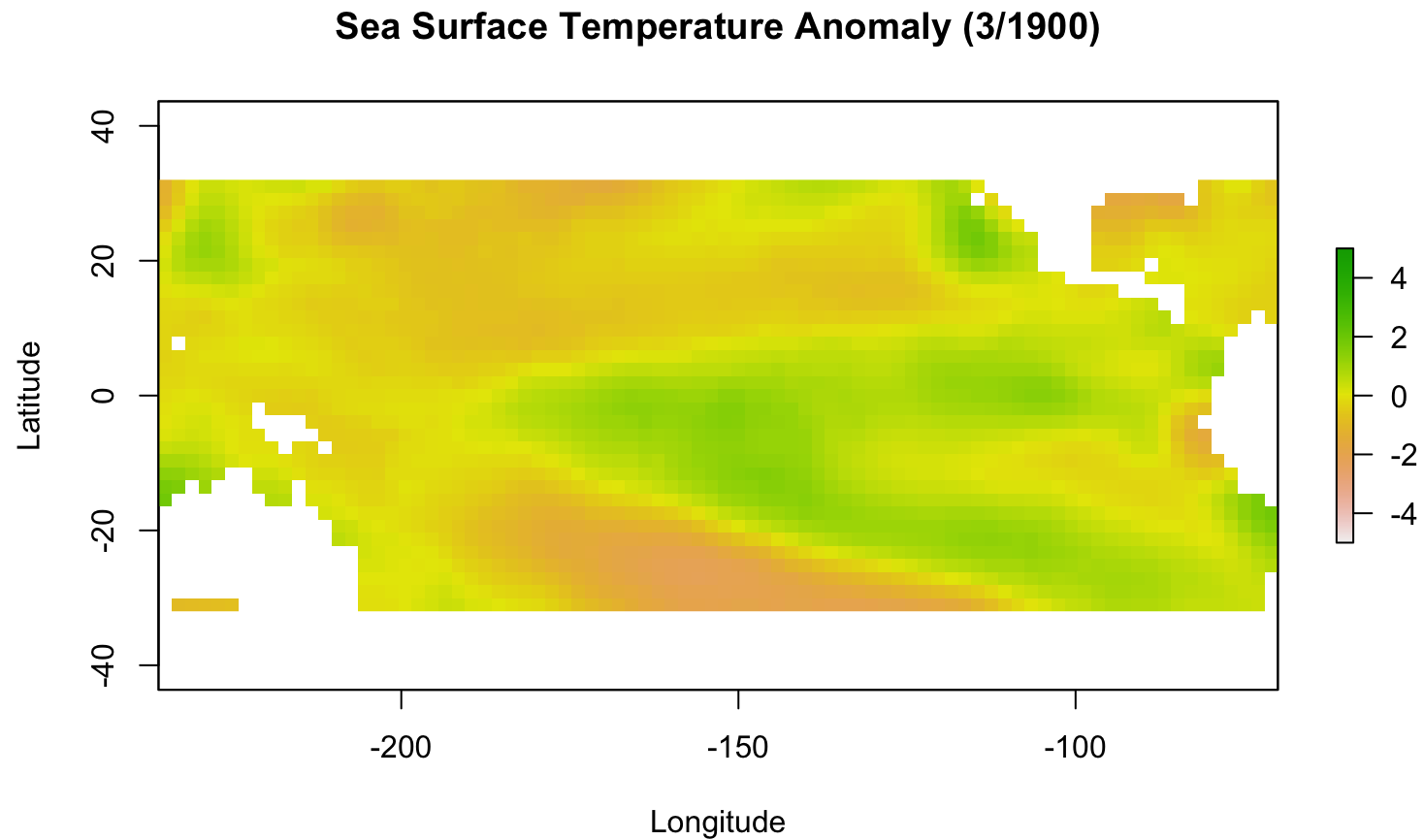
# Visualising the SST Anomaly Data



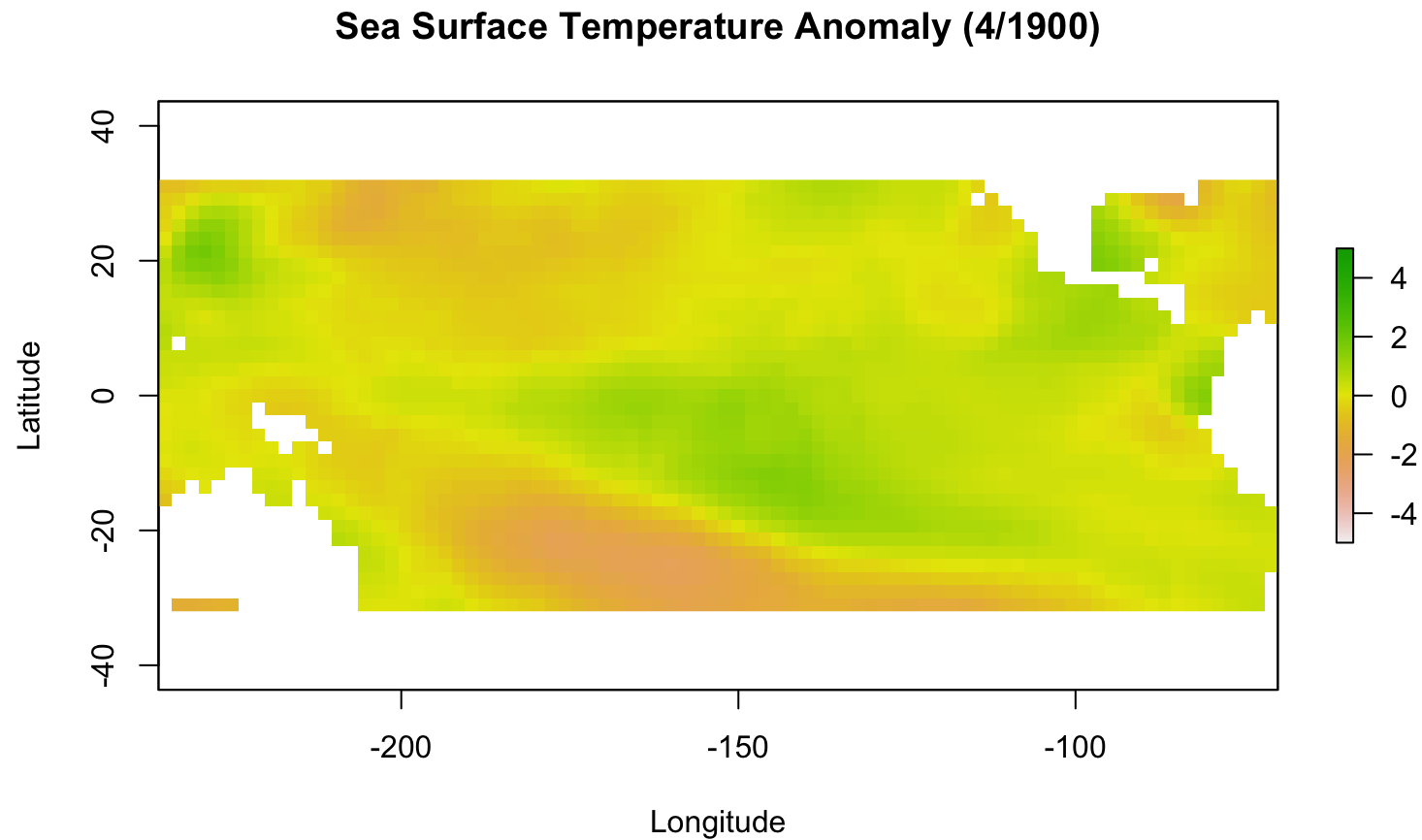
# Visualising the SST Anomaly Data



# Visualising the SST Anomaly Data

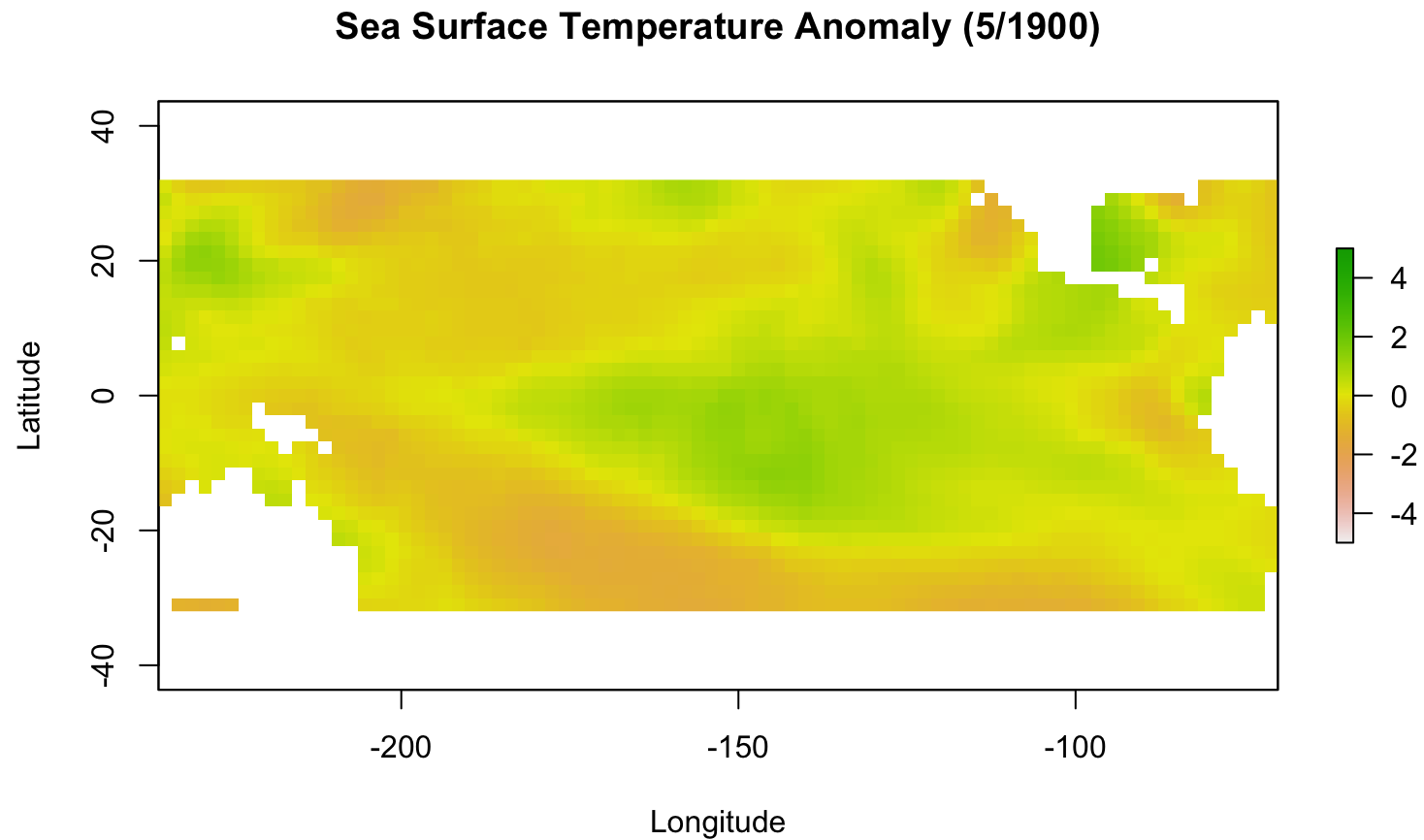


# Visualising the SST Anomaly Data

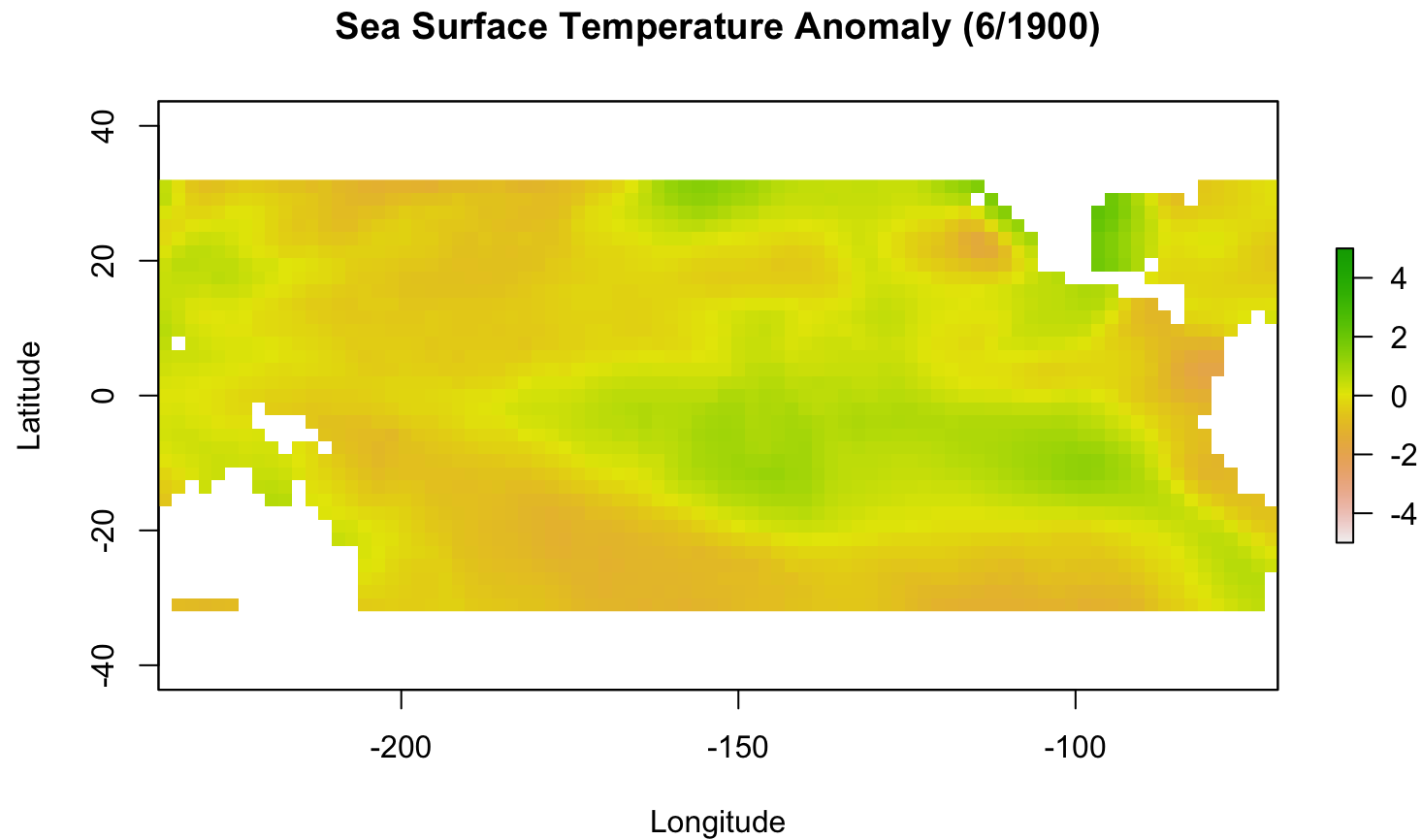




# Visualising the SST Anomaly Data



# Visualising the SST Anomaly Data



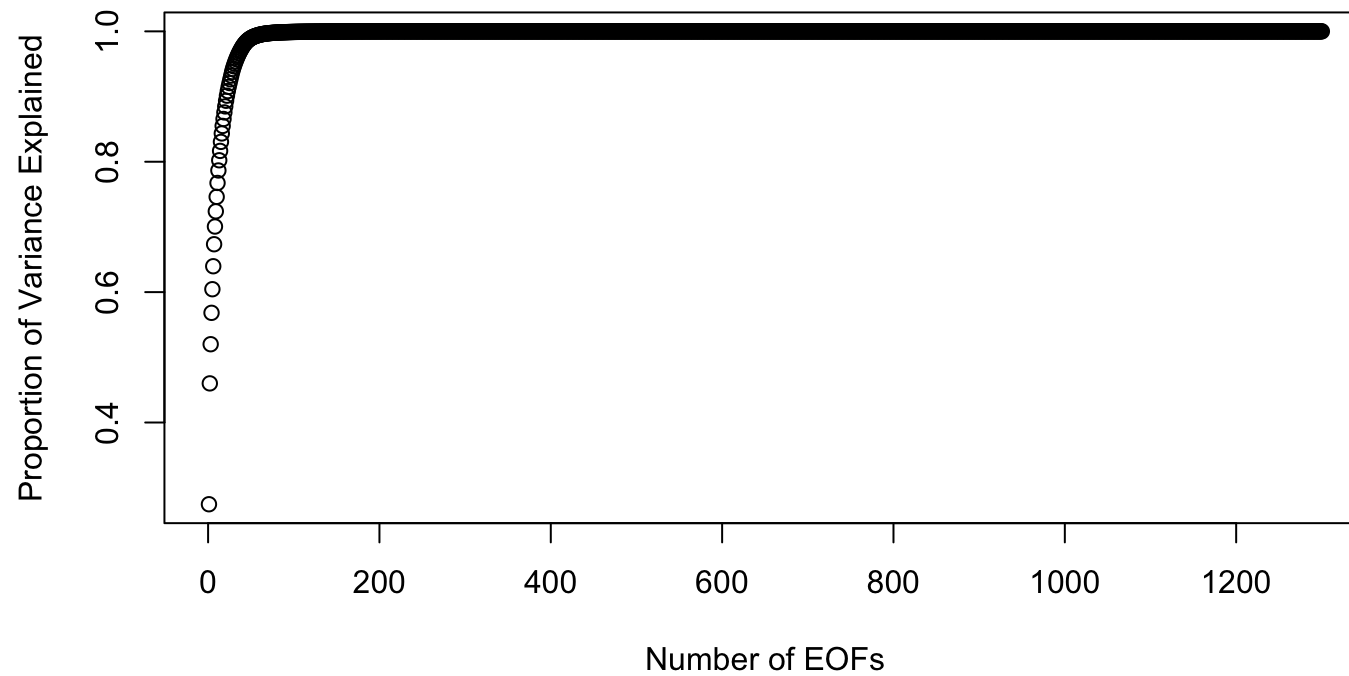
# Data Manipulation

```
batchSize <- 32
forecastMonthsAhead <- 3
timestepsPerSample <- 24
trainingInds <- 1:1300
testInds <- 1301:1434
```

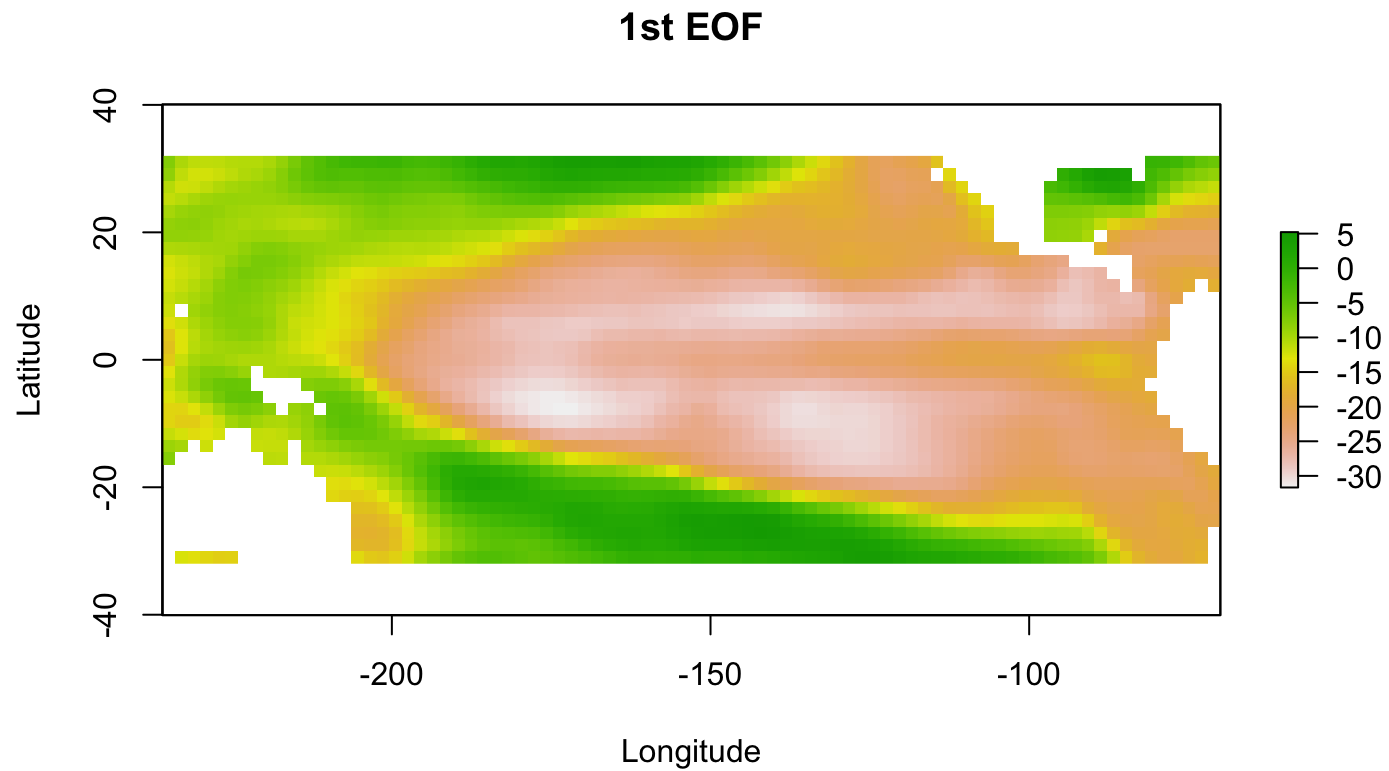
- We will reduce the dimensionality of the rasters using **singular value decomposition**.
- We will project the 2772 pixels onto 100 **Empirical Orthogonal Functions (EOFs)**.

```
numComponents <- 100
EOFList <- rasterToEOFs(anomalyRasterList[trainingInds],
                        numComponents = numComponents, plot = FALSE)
v.train <- EOFList[["rasterEOFs"]][["v.dim.red"]]
```

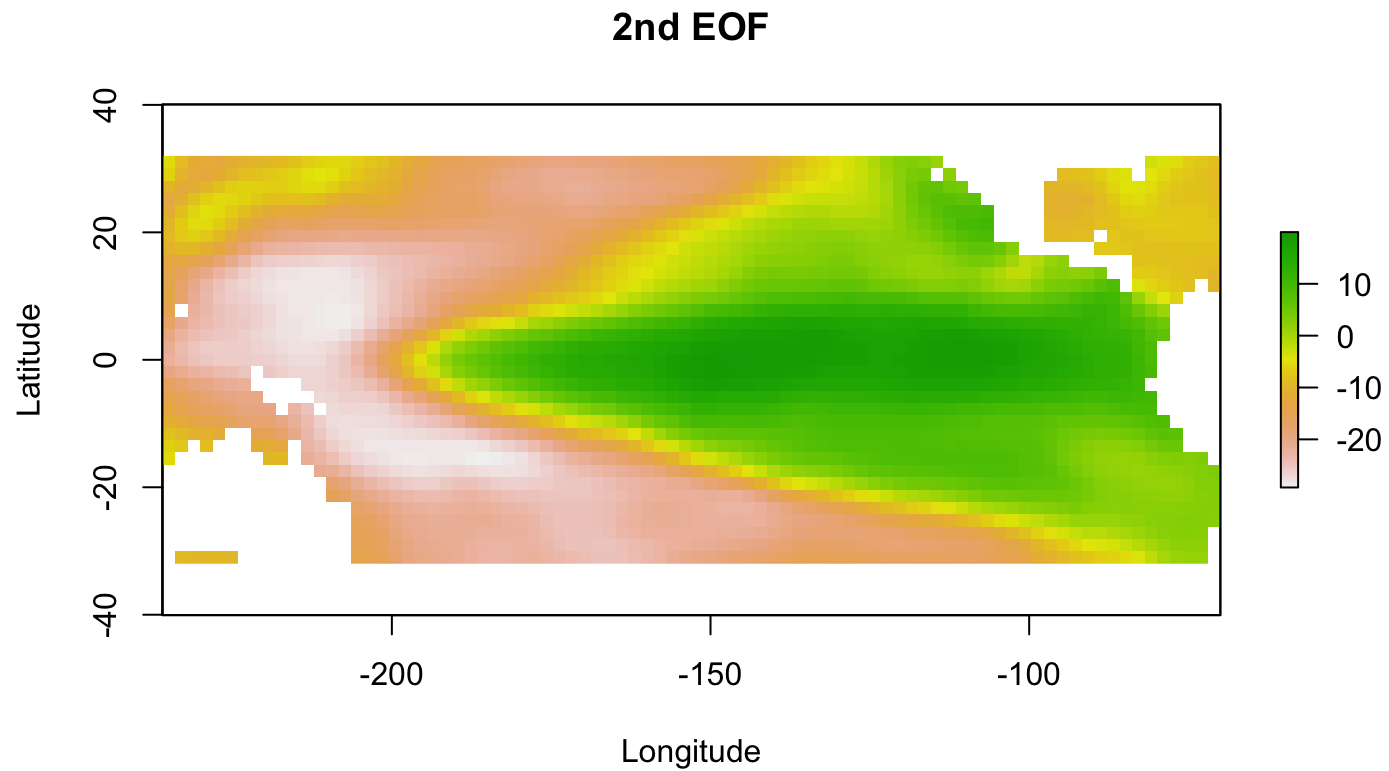
# Dimension Reduction



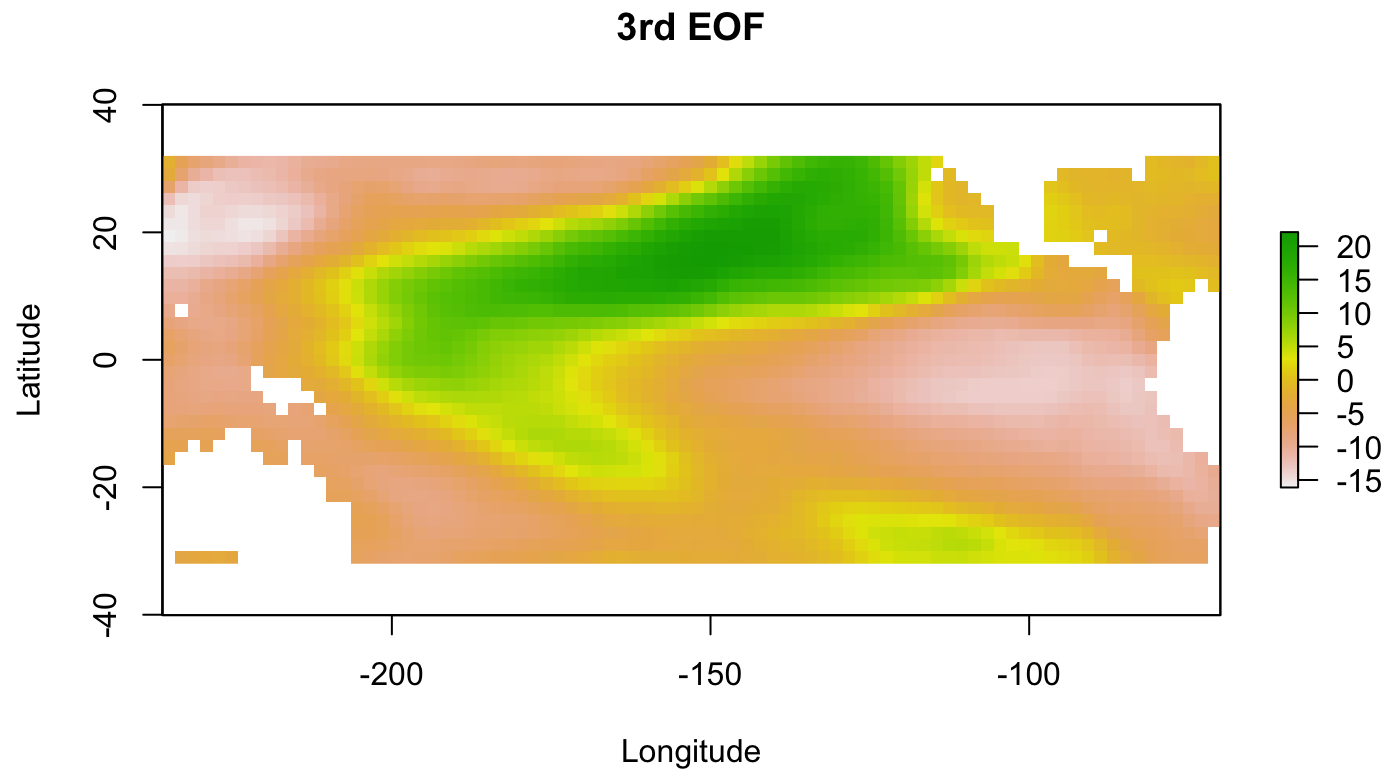
# Dimension Reduction



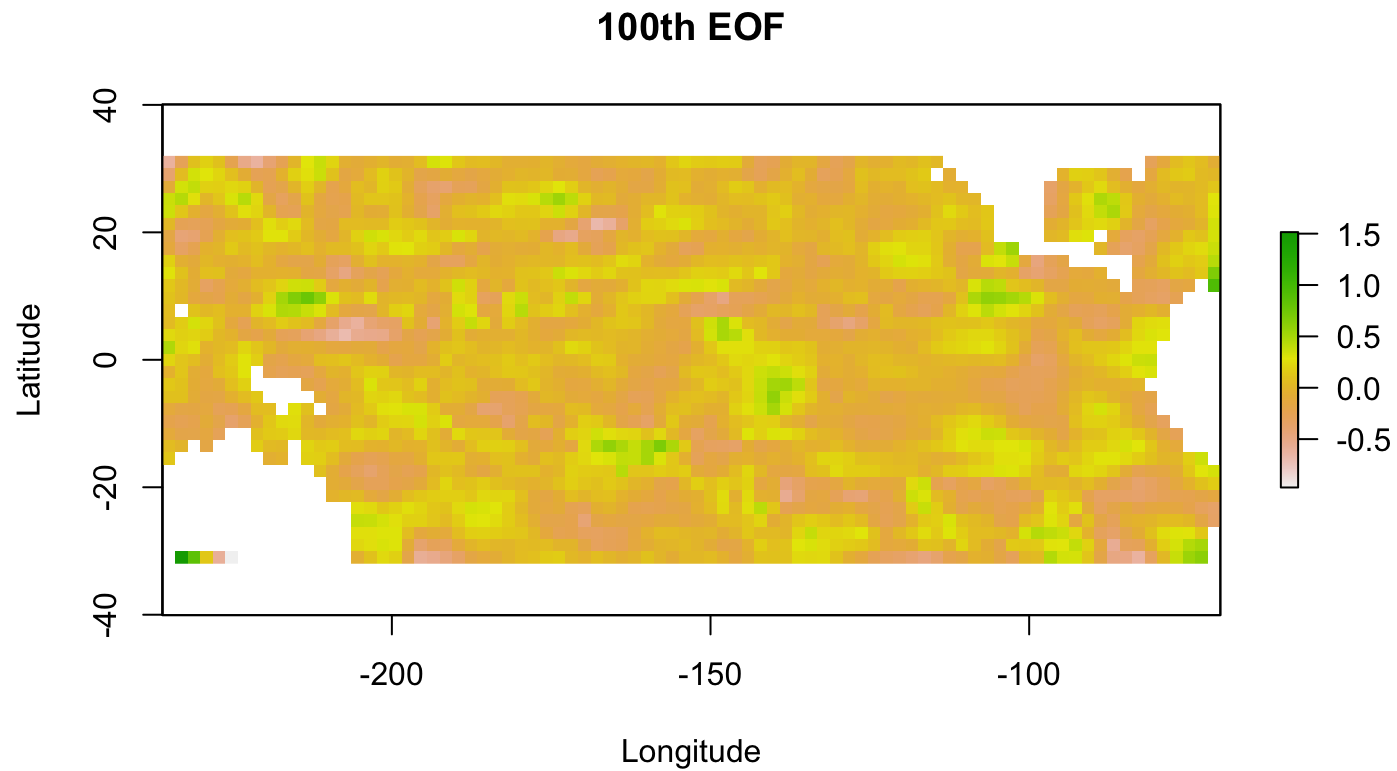
# Dimension Reduction



# Dimension Reduction



# Dimension Reduction





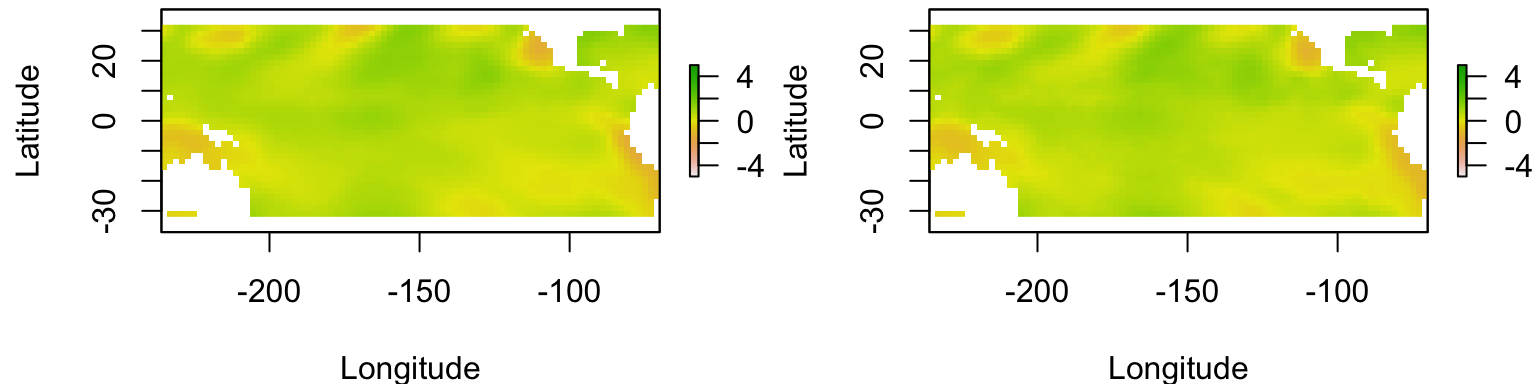
# Checking the Dimension Reduction

```
testSample <- 1434
X <- EOFList$rasterEOFs$EOFs
r1 <- anomalyRasterList[[testSample]]
validPixels <- EOFList[["raster.validPixels"]]
Y <- getValues(r1)
Y <- Y[validPixels]
lm1 <- lm(Y~X)
intercept <- coefficients(lm1)[1]
alpha <- coefficients(lm1)[1]
beta <- coefficients(lm1)[2:(numComponents + 1)]
r2 <- alpha + EOFsToRaster(X, matrix(beta, nrow = 1),
    c(33, 84), validPixels)[[1]]
extent(r2) <- extent(r1)
```

# Checking the Dimension Reduction

- We can accurately reproduce the SST anomaly grids from the 100 EOFs.

```
par(mfrow = c(1, 2))  
plot(r1, xlab = "Longitude", ylab = "Latitude", zlim = c(-5, 5))  
plot(r2, xlab = "Longitude", ylab = "Latitude", zlim = c(-5, 5))
```



# Dimension Reduction of the Test Data

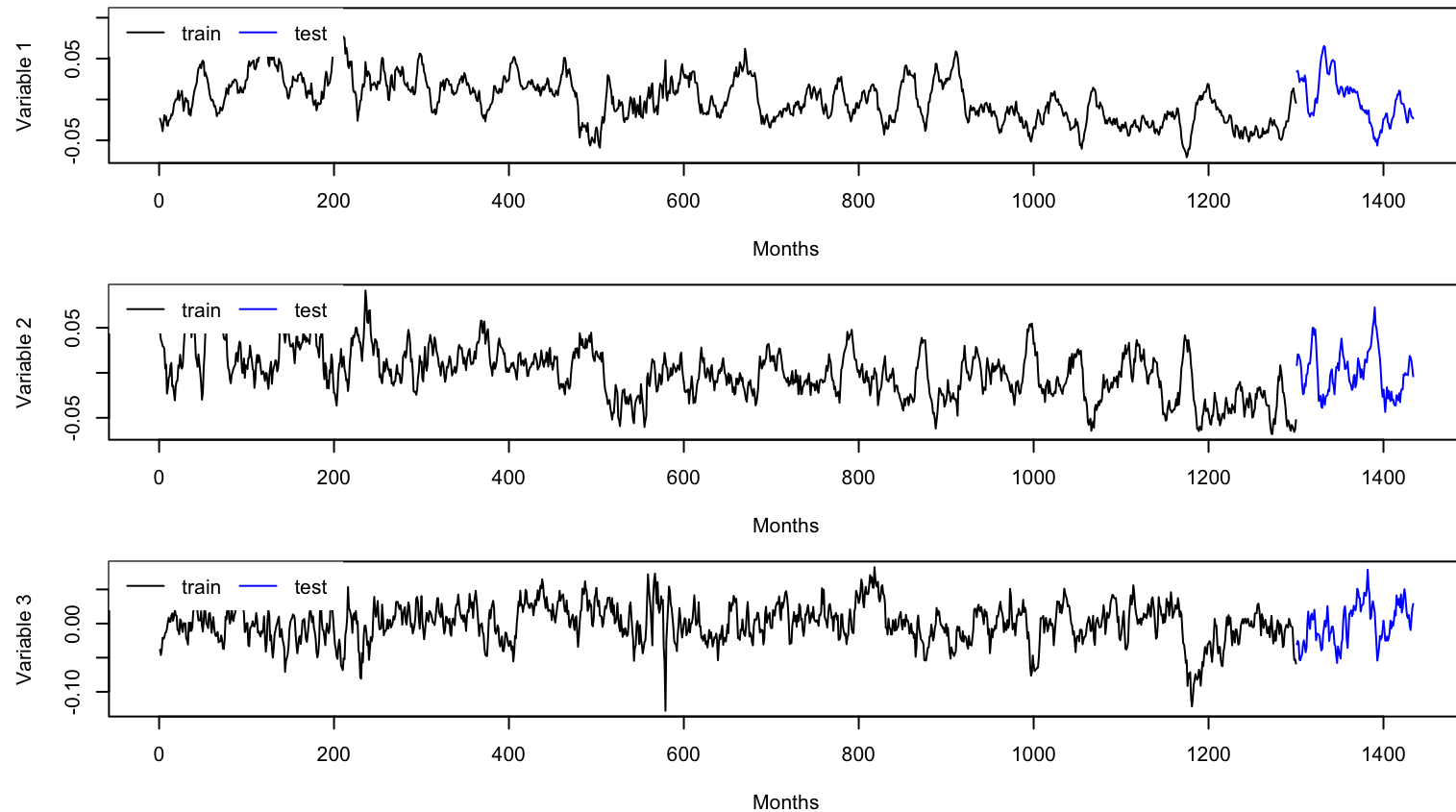
- Here we project the SST anomaly grids in the test set onto the EOFs that we generated from the training data.
- You can think of `v.test` as a multivariate time series of coefficients that we can use to reconstruct SST anomaly from the EOFs.
- We can think of these as latent features derived from the observed system.

```
v.test <- proj.raster.EOFs(anomalyRasterList[testInds],  
  EOFList[["rasterEOFs"]][["EOFs"]],  
  EOFList[["raster.validPixels"]])
```

# Dimension Reduction of the Test Data

```
par(mfrow = c(3, 1), mar = c(4,4,1,1))
plot(trainingInds, v.train[, 1], ty = "l", xlim = c(0, 1434),
      xlab = "Months", ylab = "Variable 1")
lines(testInds, v.test[, 1], col = "blue")
legend("topleft", legend = c("train", "test"), horiz = TRUE,
      box.lwd = 0, col = c("black", "blue"), lty = 1)
plot(trainingInds, v.train[, 2], ty = "l", xlim = c(0, 1434),
      xlab = "Months", ylab = "Variable 2")
lines(testInds, v.test[, 2], col = "blue")
legend("topleft", legend = c("train", "test"), horiz = TRUE,
      box.lwd = 0, col = c("black", "blue"), lty = 1)
plot(trainingInds, v.train[, 3], ty = "l", xlim = c(0, 1434),
      xlab = "Months", ylab = "Variable 3")
lines(testInds, v.test[, 3], col = "blue")
legend("topleft", legend = c("train", "test"), horiz = TRUE,
      box.lwd = 0, col = c("black", "blue"), lty = 1)
```

# Dimension Reduction of the Test Data



# Wrangling Data for a RNN in Keras

- The predictors are the **coefficients for the EOFs** over time.

```
v.combined <- rbind(v.train, v.test)
```

- Scale the predictors to the model.

```
v.scaling.train <- scaleCols.pos(v.combined[trainingInds, ])  
v.train.scaled <- v.scaling.train[["X.scaled"]]  
v.scaling.test <- scaleCols.pos(v.combined[testInds, ],  
                                colMaxsX = v.scaling.train[["colMaxsX"]],  
                                colMinsX = v.scaling.train[["colMinsX"]])  
v.test.scaled <- v.scaling.test[["X.scaled"]]  
  
v.scaled <- rbind(v.train.scaled, v.test.scaled)
```

# Formatting Tensors for a RNN in Keras

```
numDims <- ncol(v.scaled)
tensorData <- tensorfyData.rnn(v.scaled, forecastMonthsAhead,
                                timestepsPerSample, indicesX = 1:numDims,
                                indicesY = 1:numComponents, indicesTrain = trainingInds,
                                indicesTest = testInds)
str(tensorData)

## List of 8
## $ X.train.rnn      : num [1:1273, 1:24, 1:100] 0.268 0.248 0.222 0.182 0.237 ...
## $ Y.train.rnn      : num [1:1273, 1:100] 0.396 0.323 0.363 0.413 0.408 ...
## $ X.test.rnn       : num [1:107, 1:24, 1:100] 0.597 0.602 0.571 0.524 0.545 ...
## $ Y.test.rnn       : num [1:107, 1:100] 0.58 0.654 0.7 0.724 0.756 ...
## $ x.train.tsInds: int [1:1273] 24 25 26 27 28 29 30 31 32 33 ...
## $ x.test.tsInds  : int [1:107] 1324 1325 1326 1327 1328 1329 1330 1331 1332 1333 ...
## $ y.train.tsInds: int [1:1273] 27 28 29 30 31 32 33 34 35 36 ...
## $ y.test.tsInds  : int [1:107] 1327 1328 1329 1330 1331 1332 1333 1334 1335 1336 ...
```

# Scaling Data for a RNN in Keras

- Scale the Outputs of the Model

```
Y.train.inds <- tensorData$y.train.tsInds
Y.test.inds <- tensorData$y.test.tsInds
Y.train.rnn_MDB <- rainfallAnomaly[Y.train.inds, 3]
Y.test.rnn_MDB <- rainfallAnomaly[Y.test.inds, 3]
Y.train.min <- min(Y.train.rnn_MDB)
Y.train.max <- max(Y.train.rnn_MDB)

Y.train.rnn_MDB <- (Y.train.rnn_MDB - Y.train.min)/
                  (Y.train.max - Y.train.min)
Y.test.rnn_MDB <- (Y.test.rnn_MDB - Y.train.min)/
                  (Y.train.max - Y.train.min)
```



# Editing the Tensors for the RNN

- By default, `tensorfyData.rnn` will assume that you are trying to predict the same time series that you are using as inputs.
- In our case, the `EOF coefficients are the inputs`, but we are attempting to `forecast rainfall anomaly`.
- So let's replace the output tensors in the list called `tensorData`.

```
tensorData[["Y.train.rnn"]] <- Y.train.rnn_MDB  
tensorData[["Y.test.rnn"]] <- Y.test.rnn_MDB
```

# Editing the Tensors for the RNN

- Let's extract the important tensors from the tensor list.

```
X.rnn.train <- tensorData[["X.train.rnn"]]
```

```
X.rnn.test <- tensorData[["X.test.rnn"]]
```

```
Y.rnn.train <- tensorData[["Y.train.rnn"]]
```

```
Y.rnn.test <- tensorData[["Y.test.rnn"]]
```

# A Custom Loss Function

- We will use a **Gaussian likelihood function** and use the negative log-likelihood as our loss function.

```
Gaussian_logLikelihood <- function(y_true, y_pred)
{
  K <- backend()
  muMask <- K$constant(c(1, 0), shape = c(2, 1))
  sigmaMask <- K$constant(c(0, 1), shape = c(2, 1))

  sigma <- K$exp(K$dot(y_pred, sigmaMask))
  mu <- K$dot(y_pred, muMask)

  ll <- -0.5*K$square((mu - y_true)/(sigma)) - K$log(sigma)
  -1*K$sum(ll, axis = 1L)
}
```

# Building an LSTM Model

```
library(keras)
model <- keras_model_sequential()

## Loaded TensorFlow version 2.6.0

model %>%
  layer_lstm(units = 128,
             input_shape = c(timestepsPerSample, numDims),
             return_sequences = FALSE,
             stateful = FALSE) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dense(units = 2)
```

# Compiling the Model

- We compile the model with our custom **negative log-likelihood loss**.
- We will use the **Adam** optimisation algorithm with default parameters.

```
model %>% compile(loss = Gaussian_logLikelihood,  
                  optimizer = optimizer_rmsprop())
```

# Training and Early Stopping

```
history <- model %>% fit(x = X.rnn.train, y = Y.rnn.train,  
  batch_size = batchSize, epochs = 200, shuffle = TRUE,  
  validation_data = list(X.rnn.test, Y.rnn.test),  
  callbacks = list(callback_early_stopping(  
    monitor = "val_loss", min_delta = 0, patience = 20)))
```

```
bestModel <- load_model_hdf5(filepath = "MDB_Gaussian.hd5",  
  custom_objects = list(Gaussian_logLikelihood =  
    Gaussian_logLikelihood))
```

# Assessing the Predictions

- Calculate the mean and standard deviations of the 3 month out (Gaussian) predictive distributions.
- Then create 50% and 95% prediction intervals.

```
lstmPredictions <- bestModel %>% predict(X.rnn.test)

mu <- Y.train.min + lstmPredictions[, 1]*(Y.train.max - Y.train.min)
sigma <- exp(lstmPredictions[, 2])*(Y.train.max - Y.train.min)
n <- length(mu)
upper95 <- mu + 1.96*sigma
lower95 <- mu - 1.96*sigma
upper50 <- mu + 0.674*sigma
lower50 <- mu - 0.674*sigma
```

# Assessing the Predictions

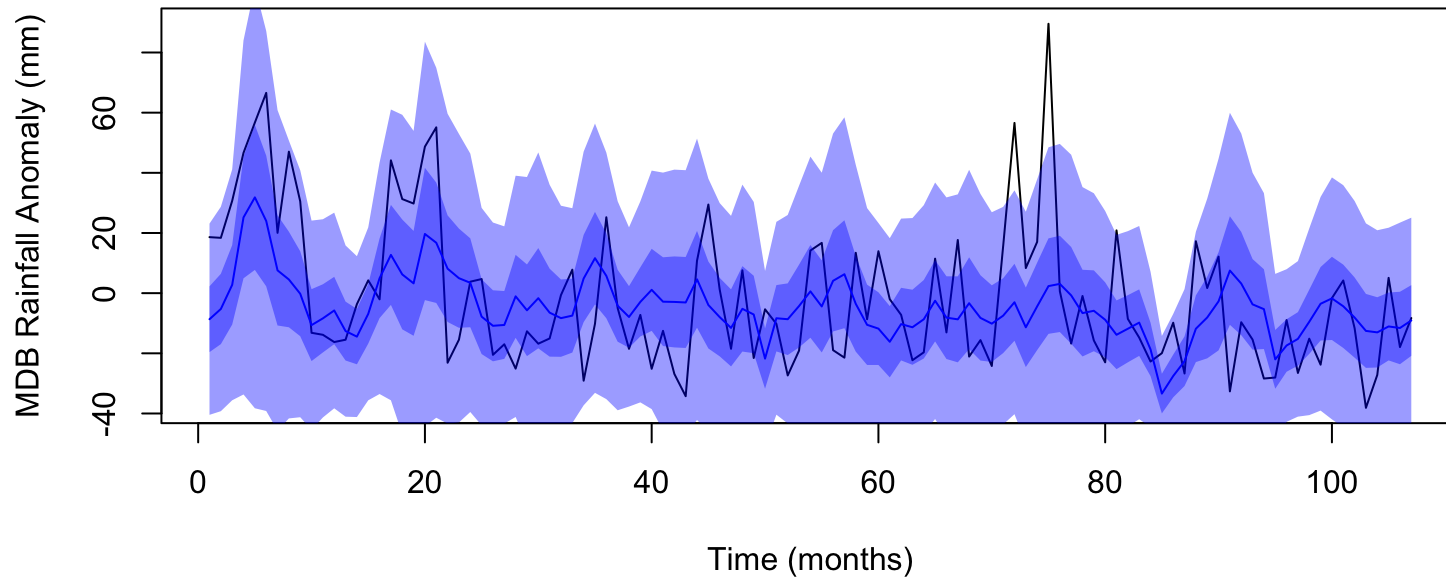
- Plot the true time series with 3-month-out forecast and 50% and 95% prediction intervals.

```
plot(rainfallAnomaly[Y.test.inds, 3], ty = "l",  
     xlab = "Time (months)",  
     ylab = "MDB Rainfall Anomaly (mm)")  
lines(mu, col = "blue")  
polygon(x = c(1:n, rev(1:n), 1),  
        y = c(lower95, rev(upper95), lower95[1]),  
        col = fade("blue", 100), border = NA)  
polygon(x = c(1:n, rev(1:n), 1),  
        y = c(lower50, rev(upper50), lower50[1]),  
        col = fade("blue", 100), border = NA)
```



# Assessing the Predictions

- Plot the true time series with 3-month-out forecast and 50% and 95% prediction intervals.



# Assessing the Predictions

- Calculate what percentage of the time the true rainfall anomaly was within the 50% and 95% prediction intervals.

```
n <- (length(Y.test.inds))
coverage50 <- length(which(rainfallAnomaly[Y.test.inds, 3] > lower50
                           & rainfallAnomaly[Y.test.inds, 3] < upper50))/n
coverage95 <- length(which(rainfallAnomaly[Y.test.inds, 3] > lower95
                           & rainfallAnomaly[Y.test.inds, 3] < upper95))/n
print(coverage50)
```

```
## [1] 0.4485981
```

```
print(coverage95)
```

```
## [1] 0.9719626
```

# Some things to try

- How does varying the number of units in the LSTM layer affect the predictions?
- How do the predictions change if you add three dense layers after the LSTM layer (instead of just 1)?
- How are the predictions if much fewer EOFs are used for prediction?
- How does fewer EOFs affect the number of parameters in the model?