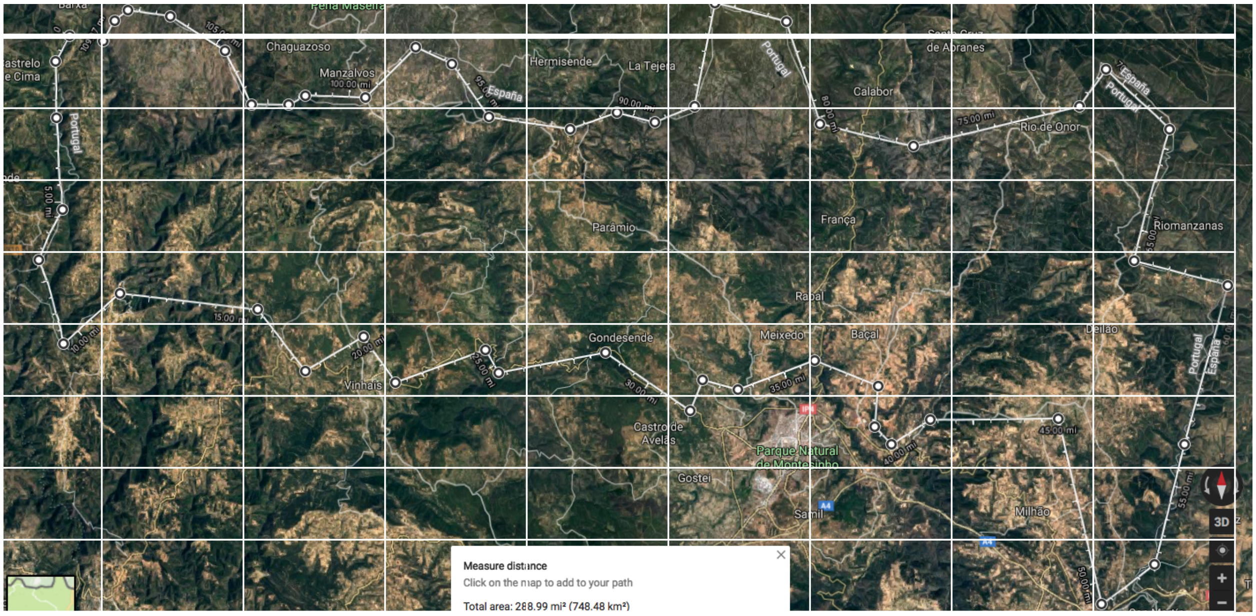
Appendix II: Final Modeling Procedure

```
#--------------------------------------------
#       Section 3.1-3.2
#--------------------------------------------
# create the indicators for weekend and summer
wkd <- rep(0, nrow(fires))
wkd[fires$day %in% c("fri", "sat", "sun")] <- 1
wkdM <- rep(0,nrow(fires))
wkdM[fires$day %in% c("fri", "sat", "sun", "mon")] <- 1
summer <- rep(0, nrow(fires))
summer[fires$month %in% c("jun", "jul", "aug", "sep")] <- 1

fires$wkd <- wkd
fires$wkdM <- wkdM
fires$summer <- summer

# transform the area
areaTrans <- log(fires$area + 1)
fires$areaTrans <- areaTrans

# rain indicator
rainvnorain <- rep(0, nrow(fires))
rainvnorain[fires$rain != 0] <- 1
fires$rainvnorain <- rainvnorain

# wetness metric
rel_humid100_temp <- c(0, 5, 10, 15, 17, 19, 20, 22, 24, 26, 29, 32, 35)
rel_humid100_water <- c(4.2, 5.74, 7.84, 10.7, 12.12, 13.73, 14.62, 16.56
                        , 18.76, 21.25, 25.62, 30.89, 37.24)

water_at_full <- approxfun(x = rel_humid100_temp, y = rel_humid100_water )
est_wetness_metric <- function(Temp, rh){
  return(water_at_full(Temp) * rh)
}
wetness <- est_wetness_metric(fires$temp, fires$RH)

fires$wetness <- wetness
```




```
#---------------------------------------------
#       Section 3.2
#---------------------------------------------

# simplified FFMC
fires$tFFMC <- ifelse(fires$FFMC < 80, 0, 1)

# indicator for forest
forest_coords <- c(1, 1, 1, 1, 1, 1, 1, 1, 1
                   , 0, 0, 1, 1, 1, 0, 1, 1, 1
                   , 0, 0, 1, 0, 0, 0, 1, 1, 0
                   , 0, 0, 1, 0, 0, 0, 1, 1, 0
                   , 0, 1, 0, 0, 1, 1, 1, 1, 1
                   , 0, 0, 0, 1, 0, 0, 0, 0, 1
                   , 1, 1, 1, 1, 0, 0, 0, 0, 1
                   , 1, 1, 1, 1, 1, 0, 0, 0, 0
                   , 1, 1, 1, 1, 1, 0, 0, 0, 0)
forest_coords <- matrix(forest_coords, nrow = 9, ncol = 9)
for(i in 1:nrow(fires)){
  fires[i, "forest_ind"] <- forest_coords[fires[i, "X"], fires[i, "Y"]]
}

# geo-spatial grid
fires[, "grid_group"] <- "other"                        # default (other)
```

```r
fires[fires$X %in% c(1, 2, 3) &
        fires$Y %in% c(2, 3, 4), "grid_group"] <- "tl" # top left mountain
fires[fires$X %in% c(3, 4, 5) &
        fires$Y %in% c(3, 4, 5), "grid_group"] <- "ml" # middle left mountain
fires[fires$X %in% c(5, 6, 7) &
        fires$Y %in% c(3, 4, 5), "grid_group"] <- "mr" # middle right mountain
fires[fires$X %in% c(7, 8) &
        fires$Y %in% c(6, 7), "grid_group"] <- "br"    # bottom right mountain

# transform response variable (ISI)
fires$sqISI <- sqrt(fires$ISI)

#  create train/test split
set.seed(575)
train.ind <- sample.int(n = nrow(fires), size = floor(nrow(fires) * 0.7), replace = FALSE)
train<- fires[train.ind, ]
test <- fires[-train.ind, ]
```

```r
#-------------------------------------------
#       Section 3.2
#-------------------------------------------
# calculate the groupings of FFMC by training quantile
ffmcQuant <- quantile(train$FFMC, probs = seq(0, 1, .1))
FFMCQuantile_train <- rep(0, nrow(train))
FFMCQuantile_test <- rep(0, nrow(test))
for (i in 10) {
  FFMCQuantile_train[ffmcQuant[i] < train$FFMC &
                       train$FFMC <= ffmcQuant[i + 1]] <- i
  FFMCQuantile_test[ffmcQuant[i] < test$FFMC &
                      test$FFMC <= ffmcQuant[i + 1]] <- i
}

train$FFMCQuantile <- FFMCQuantile_train
test$FFMCQuantile <- FFMCQuantile_test

# condense X-Y grid
train$X2 <- train$X
train$Y2 <- train$Y
i <- 0
while (i < 1) {
  m <- as.matrix(table(train$Y2, train$X2))
  top <- sum(m[rownames(m) == min(rownames(m)), ])
  bottom <- sum(m[rownames(m) == max(rownames(m)), ])
  left <- sum(m[, colnames(m) == min(colnames(m))])
  right <- sum(m[, colnames(m) == max(colnames(m))])

  if (top == min(top, bottom, left, right)) {
    train[train$Y2 == min(rownames(m)), "Y2"] <- as.integer(min(rownames(m))) + 1
    if (min(prop.table(table(train$Y2, train$X2))) < .01) {
      i = 0
    } else{
      i = 1
    }
  } else if (bottom == min(top, bottom, left, right)) {
```

```r
    train[train$Y2 == max(rownames(m)), "Y2"] <- as.integer(max(rownames(m))) - 1
    if (min(prop.table(table(train$Y2, train$X2))) < .01) {
      i = 0
    } else{
      i = 1
    }
  } else if (left == min(top, bottom, left, right)) {
    train[train$X2 == min(colnames(m)), "X2"] <- as.integer(min(colnames(m))) + 1
    if (min(prop.table(table(train$Y2, train$X2))) < .01) {
      i = 0
    } else{
      i = 1
    }
  } else {
    train[train$X2 == max(colnames(m)), "X2"] <- as.integer(max(colnames(m))) - 1
    if (min(prop.table(table(train$Y2, train$X2))) < .01) {
      i = 0
    } else{
      i = 1
    }
  }
}
```

```r
#-------------------------------------------
#       Section 3.3
#-------------------------------------------
# cast as factors
vars_factors <- c("wkd", "wkdM", "summer", "FFMCQuantile", "rainvnorain", "grid_group"
                  , "month", "day", "X2", "Y2")
for(var in vars_factors) {
  train[, var] <- as.factor(train[, var])
}

# construct regression equation
f <- formula(sqISI ~ -1
             + tFFMC
             + X2
             + Y2
             + temp
             + RH
             + wind
             + wkd
             + summer
             + rainvnorain
             + forest_ind
             + DMC
             + DC
             + areaTrans
             + wetness
)
# build model matrix
X <- model.matrix(f, train)
Y <- as.matrix(train$sqISI)
a <- 1
```

```r
# run lasso
cv = cv.glmnet(X, Y, alpha = a)
lambda_opt = cv$lambda.min

lasso <- glmnet(X, Y, alpha = a, lambda = lambda_opt)
tmp <- sort(abs(coef(lasso)[, 1]), decreasing = TRUE)
varImp <- data.frame(VarNames = names(tmp), Beta = round(as.vector(tmp), 3))
varImp

# fit lm with most important variables
m <- lm(sqISI ~
          tFFMC
        + rainvnorain
        + summer
        + forest_ind
        + wind
        + temp
        + X2
        + wkd
        , data = train)

# diagnostics and added variable plots
par(mfrow = c(2, 2))
plot(m)
```

```r
summary(m)
avPlots(m)
```

```r
#-------------------------------------------
#       Section 4.1
#-------------------------------------------
# Weight based on tFFMC
train <- data.frame(train %>% group_by(tFFMC) %>% mutate(weight = n()))
m_weight <- lm(sqISI ~
                 rainvnorain
               + summer
               + wind
               + temp
               , weights = weight
               , data = train)
par(mfrow = c(2, 2))
plot(m_weight)
```

```r
summary(m_weight)
avPlots(m_weight)
```

```r
# Random intercepts with tFFMC
train[train$X2 == 2 & train$Y2 == 4, "region"] = 1
train[train$X2 == 3 & train$Y2 == 4, "region"] = 2
train[train$X2 == 4 & train$Y2 == 4, "region"] = 3
train[train$X2 == 5 & train$Y2 == 4, "region"] = 4
train[train$X2 == 6 & train$Y2 == 4, "region"] = 5
train[train$X2 == 7 & train$Y2 == 4, "region"] = 6
```

```r
train[train$X2 == 2 & train$Y2 == 5, "region"] = 7
train[train$X2 == 3 & train$Y2 == 5, "region"] = 8
train[train$X2 == 4 & train$Y2 == 5, "region"] = 9
train[train$X2 == 5 & train$Y2 == 5, "region"] = 10
train[train$X2 == 6 & train$Y2 == 5, "region"] = 11
train[train$X2 == 7 & train$Y2 == 5, "region"] = 12

train$region = as.factor(train$region)

m_rand <- lme(sqISI ~ summer + wind + temp + rainvnorain + tFFMC
              , random = ~1|region
              , data =  train[-89, ]
              , method = "REML")
summary(m_rand)
plot(m_rand)
```

```r
par(mfrow = c(1, 1))
qqnorm(m_rand$residuals)
```

```r
#-------------------------------------------
#       Section 4.1-4.2
#-------------------------------------------
# build final model
m1 = lm(sqISI ~ summer + wind + temp + rainvnorain
        , data = train)

# MSE for raw ISI
1 / nrow(train) * sum((train$sqISI^2 - m1$fitted.values^2)^2)

# 1000 boostrap samples for each coefficient
B <- 1000
ResidualBootstrapM1 <- t(replicate(B, {
  yb <- fitted(m1) + resid(m1)[sample.int(nrow(train), replace = TRUE)]
  boot <- model.matrix(m1)
  coef(lm(yb ~ boot - 1))
}))

# look at distributions of coefficients
par(mfrow = c(2, 2))
hist(ResidualBootstrapM1[,1], xlab = "summer", main = "")
hist(ResidualBootstrapM1[,2], xlab = "wind", main = "")
hist(ResidualBootstrapM1[,3], xlab = "temp", main = "")
hist(ResidualBootstrapM1[,4], xlab = "rainvnorain", main = "")
```

```r
# empirical CI
t(apply(ResidualBootstrapM1, 2, quantile, c(.025, .975)))
```

```r
#-------------------------------------------
#       Section 5
#-------------------------------------------
# cast as factors
vars_factors <- c("summer", "rainvnorain")
for(var in vars_factors) {
  test[, var] <- as.factor(test[, var])
```

```r
}

# fit candidate model on test data
m1 <- lm(sqISI ~ summer + wind + temp + rainvnorain
         , data = test)
par(mfrow = c(2, 2))
summary(m1)
plot(m1)
```

```r
avPlots(m1)
```

```r
# plot fitted values
par(mfrow = c(1, 1))
plotdf <- data.frame(cbind(test$sqISI^2, m1$fitted.values^2))
names(plotdf) <- c("ActualValues", "FittedValues")
ggplot(plotdf, aes(x = FittedValues, y = ActualValues)) +
  geom_point() +
  geom_segment(aes(x = 0, y = 0, xend = 15, yend = 15), col = "blue")
```

```r
# MSE for response ISI
1 / nrow(test) * sum((test$sqISI - m1$fitted.values)^2)
# MSE for raw ISI
1 / nrow(test) * sum((test$sqISI^2 - m1$fitted.values^2)^2)

# 1000 bootstrap samples for each coefficient
bootCoefs  <- t(replicate(B, {
  yb <- fitted(m1) + resid(m1)[sample(nrow(test), replace = TRUE)]
  boot <- model.matrix(m1)
  coef(lm(yb ~ boot - 1))
}))

# empirical CI
t(apply(bootCoefs, 2, quantile, c(.025, .5, .975)))
# empirical p-values
apply(bootCoefs < 0, 2, sum) / B
```