AML_HW6

Vishal Dalmiya (Dalmiya2); Himanshu Shah (Hs8); Deepak Nagarajan (deepakn2)

Mar 23, 2018

Problem 1 - Linear regression with various regularizers The UCI Machine Learning dataset repository hosts a dataset giving features of music, and the latitude and longitude from which that music originates here. Investigate methods to predict latitude and longitude from these features, as below. There are actually two versions of this dataset. Either one is OK by me, but I think you'll find the one with more independent variables more interesting. You should ignore outliers (by this I mean you should ignore the whole question; do not try to deal with them). You should regard latitude and longitude as entirely independent.

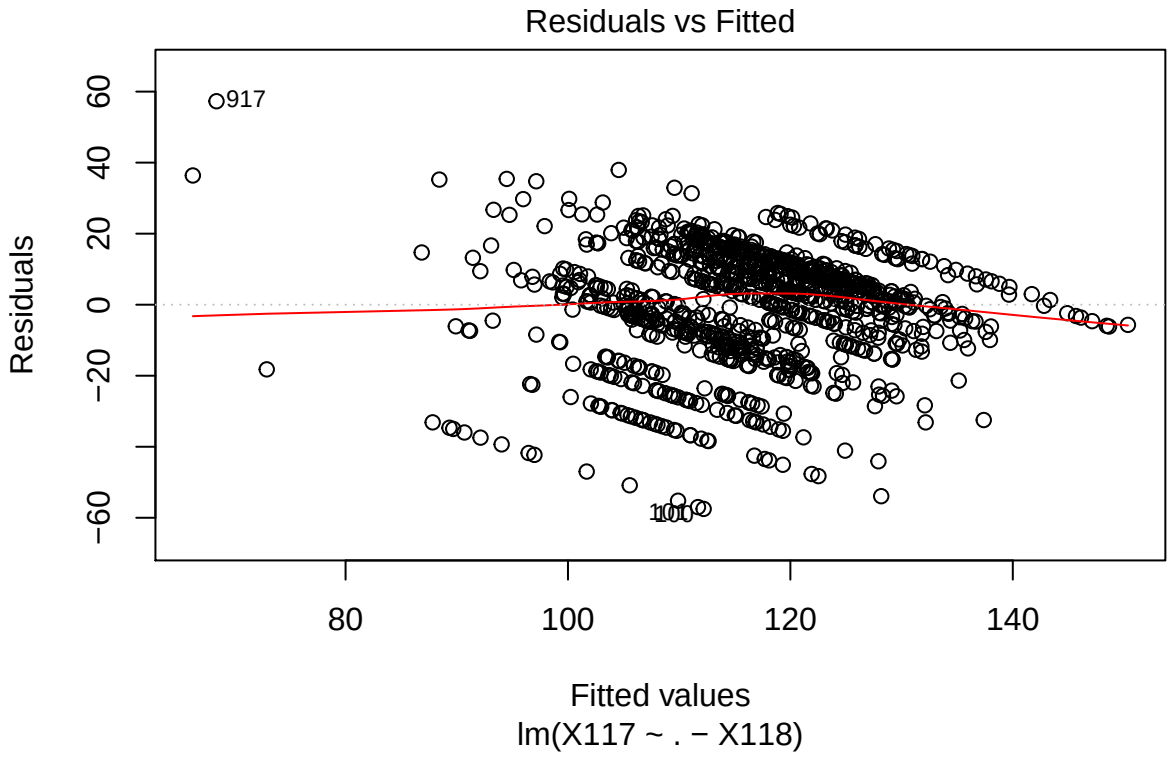
First, build a straightforward linear regression of latitude (resp. longitude) against features. What is the R-squared? Plot a graph evaluating each regression.

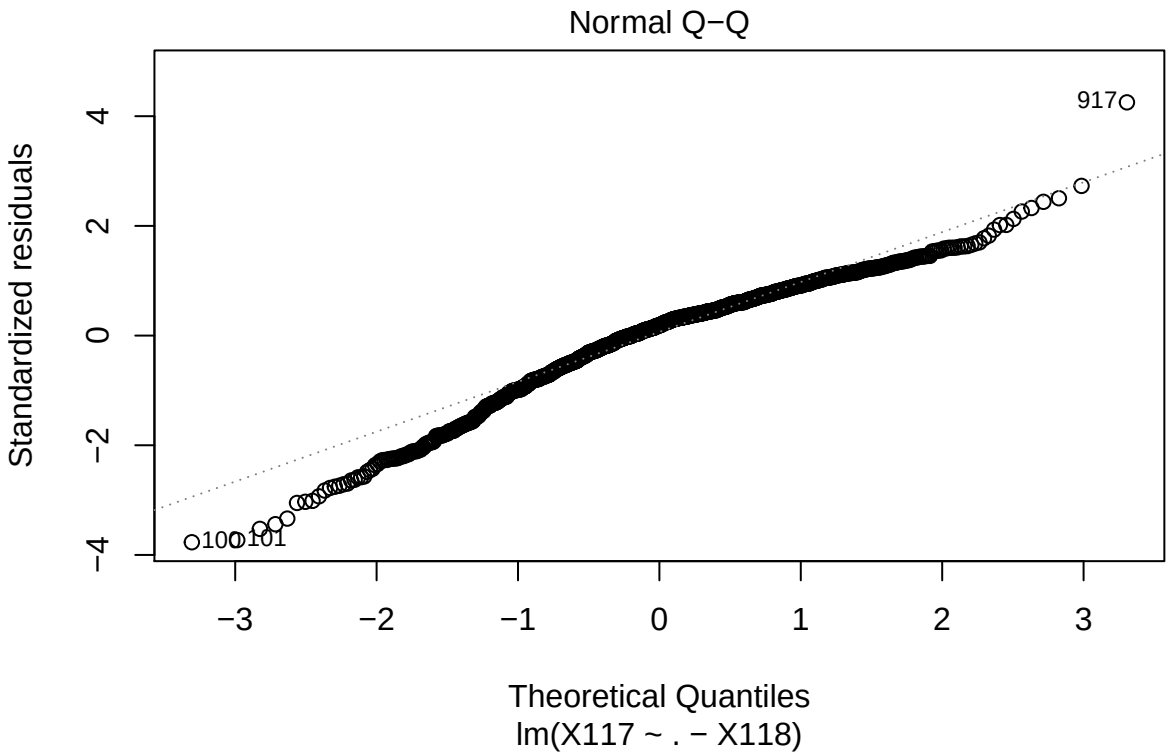
```
# RMSE
get_rmse = function(model, data, actual)
{
  sqrt(mean((actual - predict(model, newdata = data)) ^ 2))
}

# Compute R^2 glm
get_r2_glm = function(model, data, actual)
{
  expected = predict(model, data , s = model$lambda.min)

  sst = sum((actual - mean(actual)) ^ 2)
  sse = sum((expected  - actual) ^ 2)

  # R squared
  r2 = 1 - (sse / sst)
  r2
}

# RMSE glm
get_rmse_glm = function(model, data, actual)
{
  sqrt(mean((actual - predict(model, data , s = model$lambda.min)) ^ 2))
}

get_num_params_glm = function(model)
{
  Coefs = coef(model, s="lambda.min")
  UsedCoefs = Coefs[which(Coefs != 0 ) ]
  length(UsedCoefs)
}
```

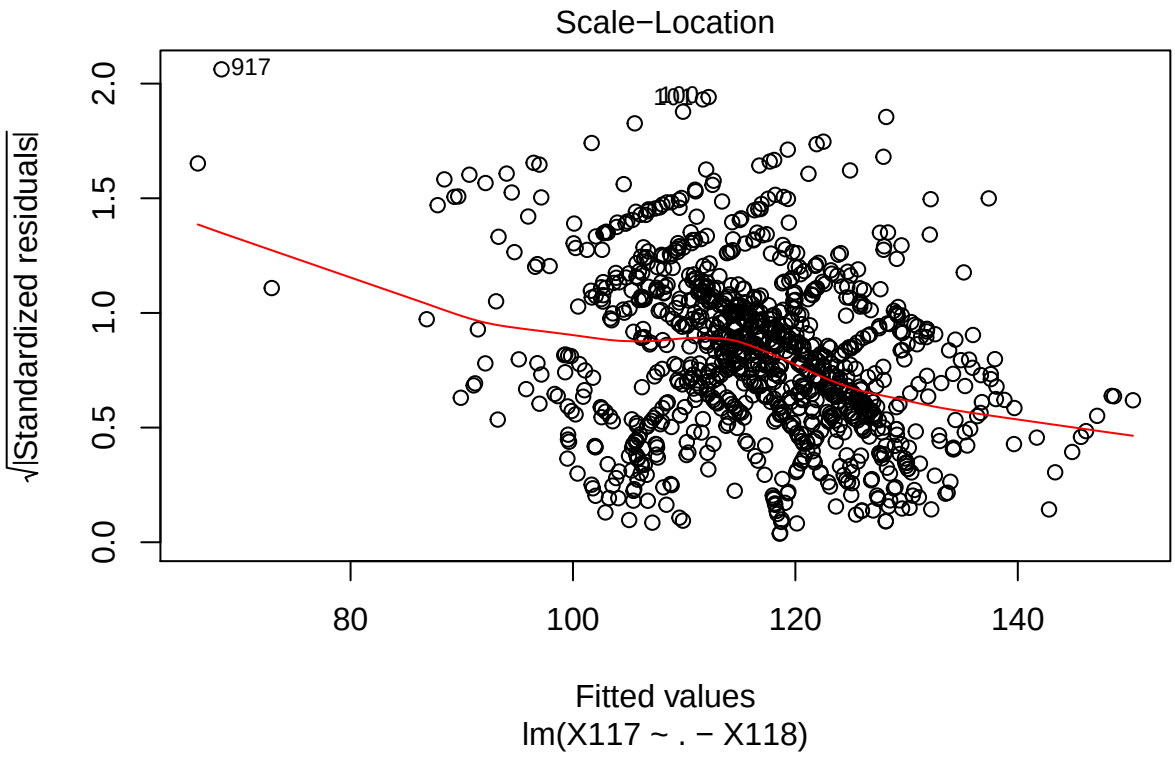
```
set.seed(20180321)
library(readr)
library(caret)
music_tracks <-
  read_csv(
  "Geographical Original of Music/default_plus_chromatic_features_1059_tracks.txt",
  col_names = FALSE
  )

music_tracks$X117 = music_tracks$X117 + 90
music_tracks$X118 = music_tracks$X118 + 180

model_lat = lm(X117 ~ . - X118, data = music_tracks)
model_lon = lm(X118 ~ . - X117, data = music_tracks)
```

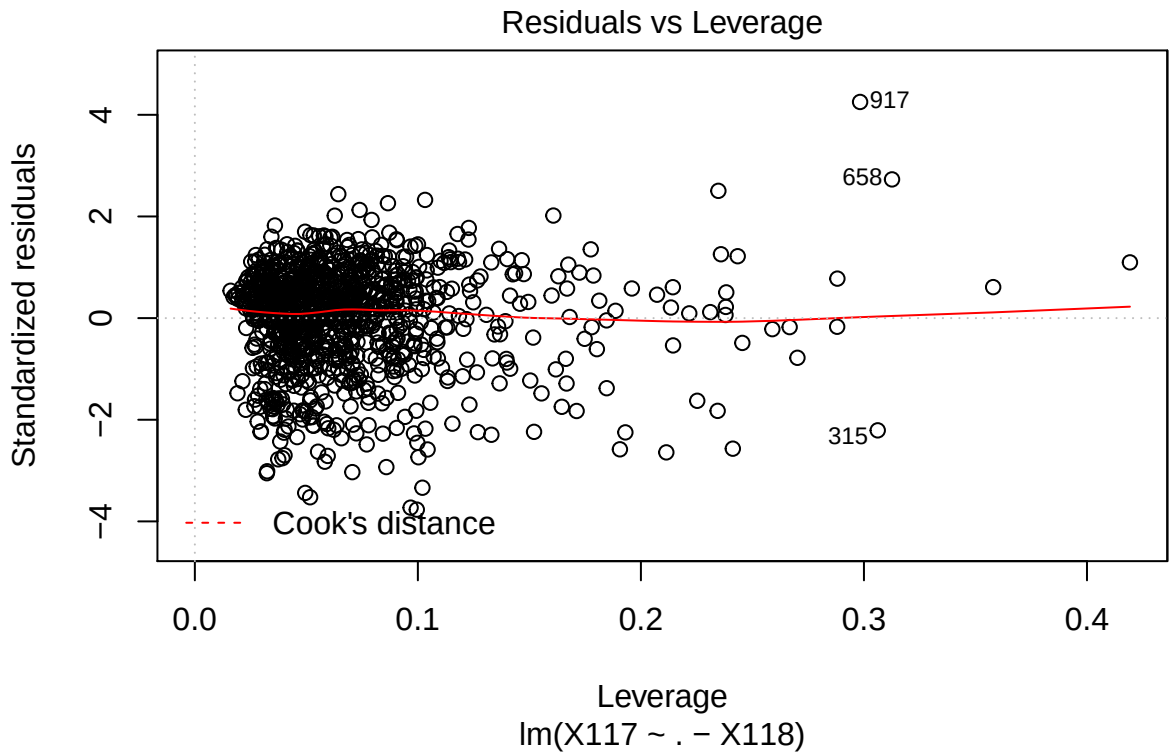
- The value of r^2 for latitude model is 0.2928092
- The value of r^2 for longitude model is 0.3645767

```
plot(model_lat)
```

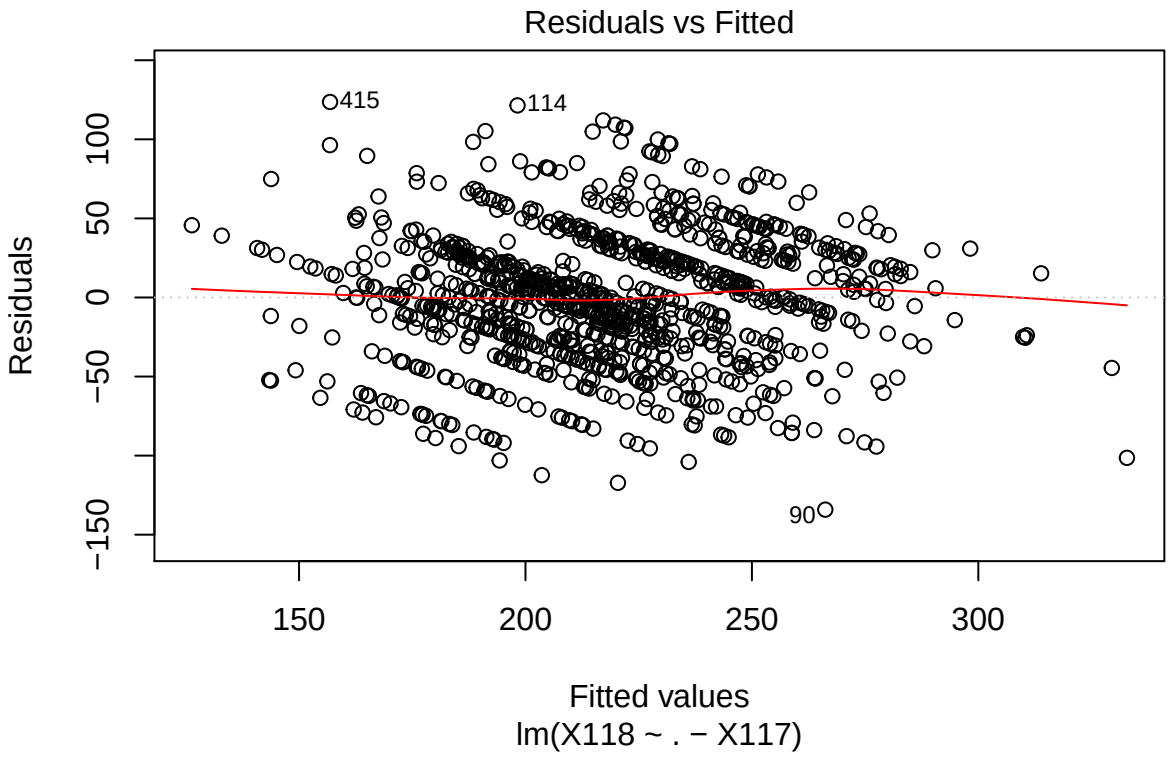


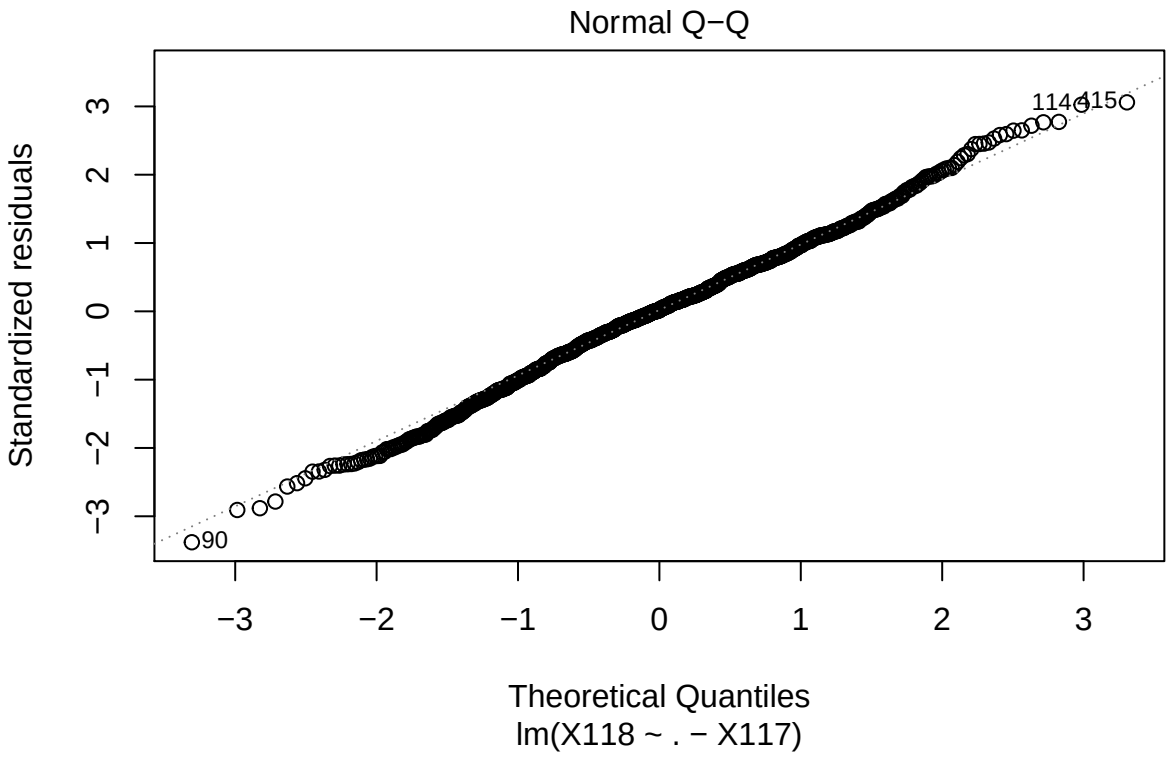

Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(X117 ~ . − X118)
917
100
101
−4
−2
0
2
4
−3
−2
−1
0
1
2
3

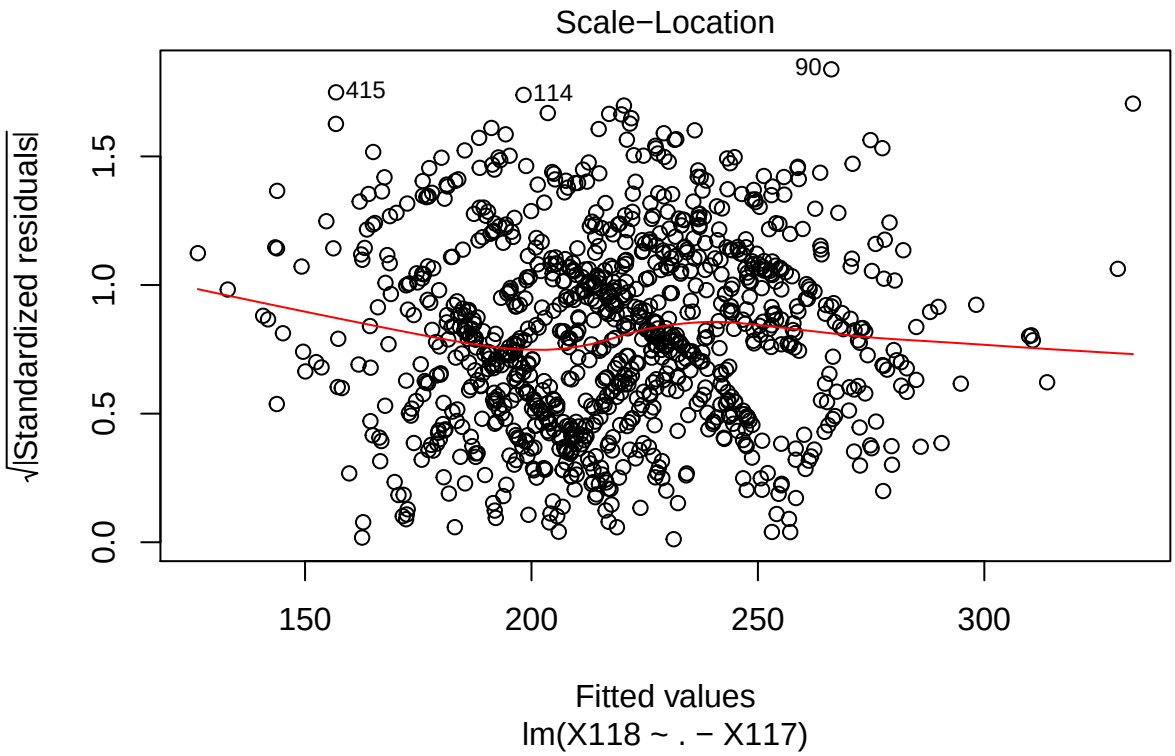

Scale−Location
917
100
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
80
100
120
140
Fitted values
lm(X117 ~ . − X118)

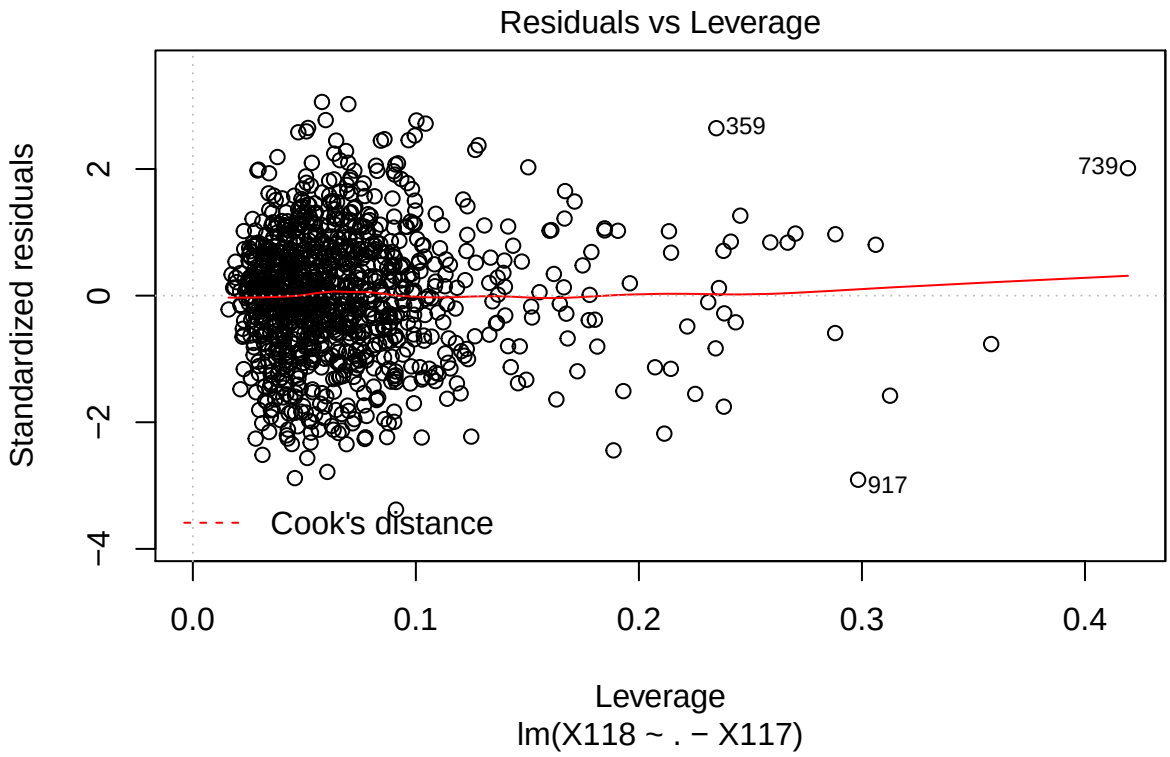



```
plot(model_lon)
```


Residuals vs Fitted
Residuals
Fitted values
lm(X118 ~ . − X117)
415
114
90
−150
−50
0
50
100
150
200
250
300

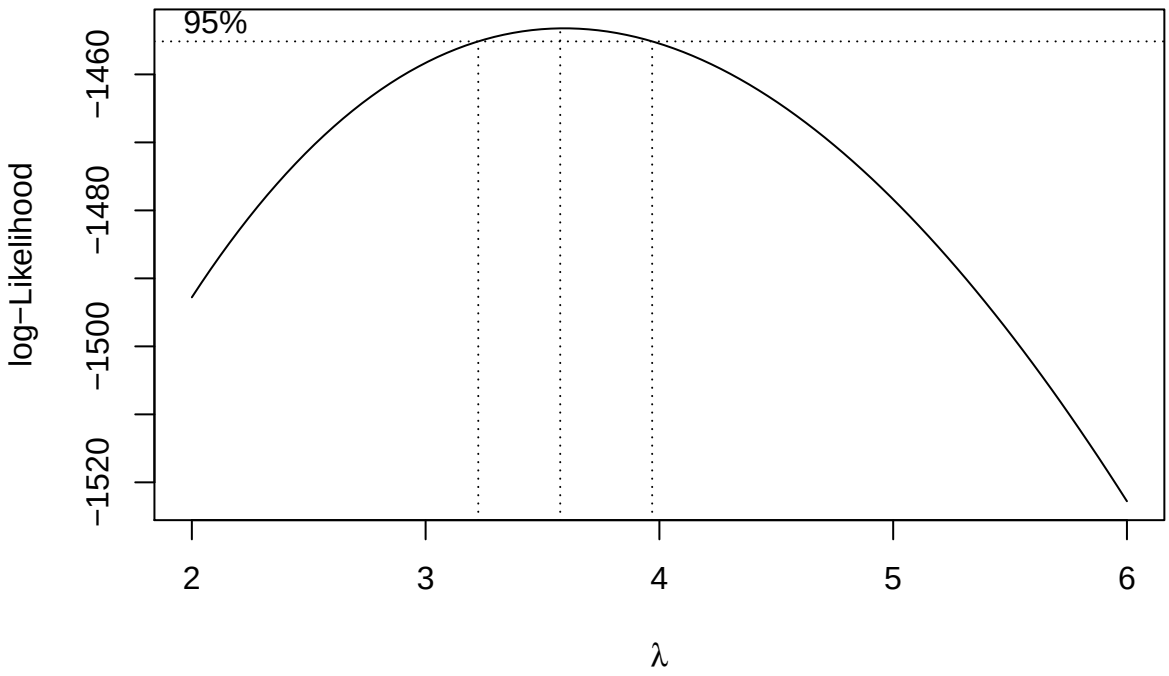

Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(X118 ~ . − X117)
90
114
415
−3
−2
−1
0
1
2
3


Scale−Location
415
114
90
√|Standardized residuals|
0.0
0.5
1.0
1.5
150
200
250
300
Fitted values
lm(X118 ~ . − X117)

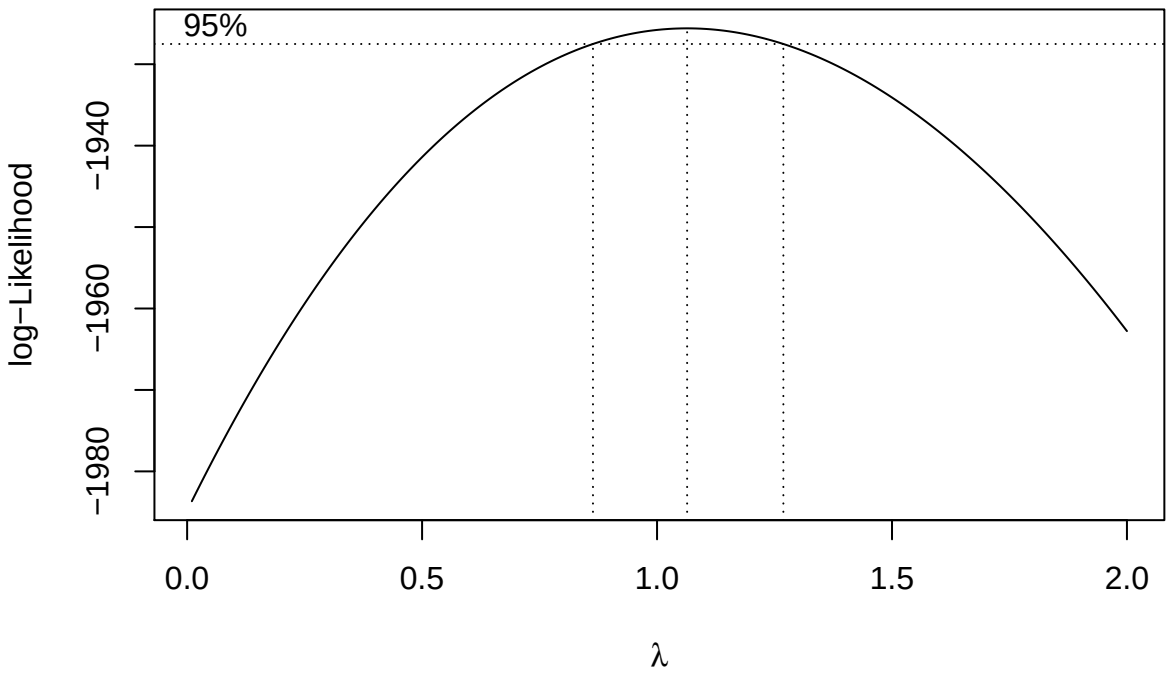


Does a Box-Cox transformation improve the regressions? Notice that the dependent variable has some negative values, which Box-Cox doesn't like. You can deal with this by remembering that these are angles, so you get to choose the origin. why do you say so? For the rest of the exercise, use the transformation if it does improve things, otherwise, use the raw data.

```
library(MASS)
boxcox(model_lat, lambda = seq(2, 6, 0.1))
```




```
boxcox(model_lon, lambda = seq(0.01, 2.0, 0.001))
```

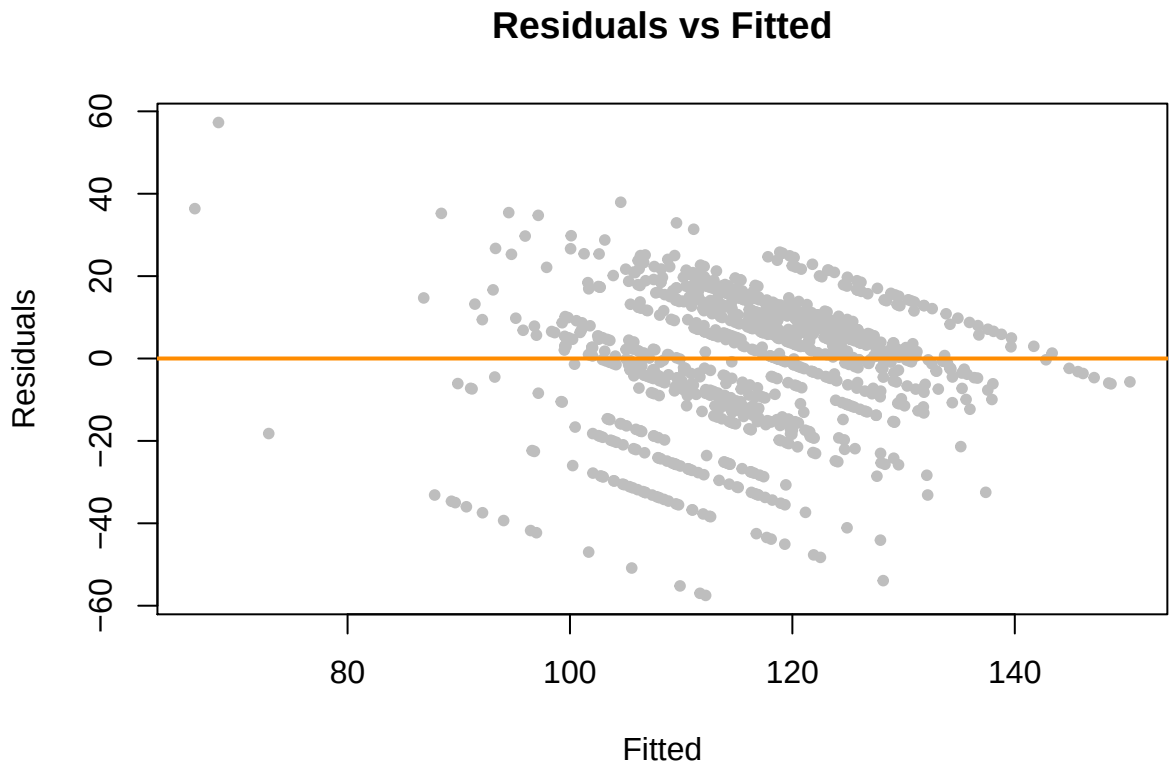


- We selected lambda = 3.5, based on the 95% log-likelihood for the latitude model.
- We selected lambda = 1, based on the 95% log-likelihood for the longitude model.

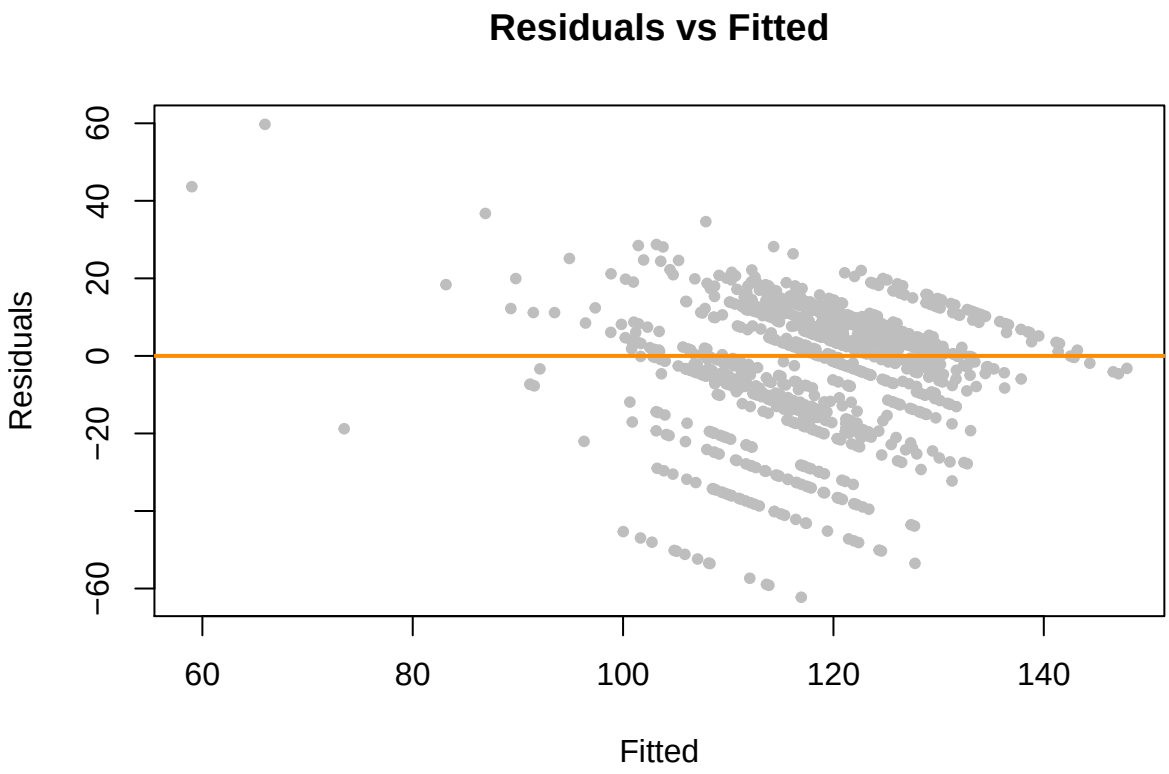
```
model_lat_transform = lm((((X117 ^ 3.5) - 1) / 3.5)  ~ . - X118, data = music_tracks)

fitted = (((predict(model_lat_transform, music_tracks) * 3.5) + 1) ^ (1/3.5))
res    = music_tracks$X117 - fitted

sst = sum((music_tracks$X117 - mean(music_tracks$X117)) ^ 2)
sse = sum((fitted  - music_tracks$X117) ^ 2, na.rm = TRUE)

# R squared
r2 = 1 - (sse / sst)
r2
```

```
## [1] 0.247779
```

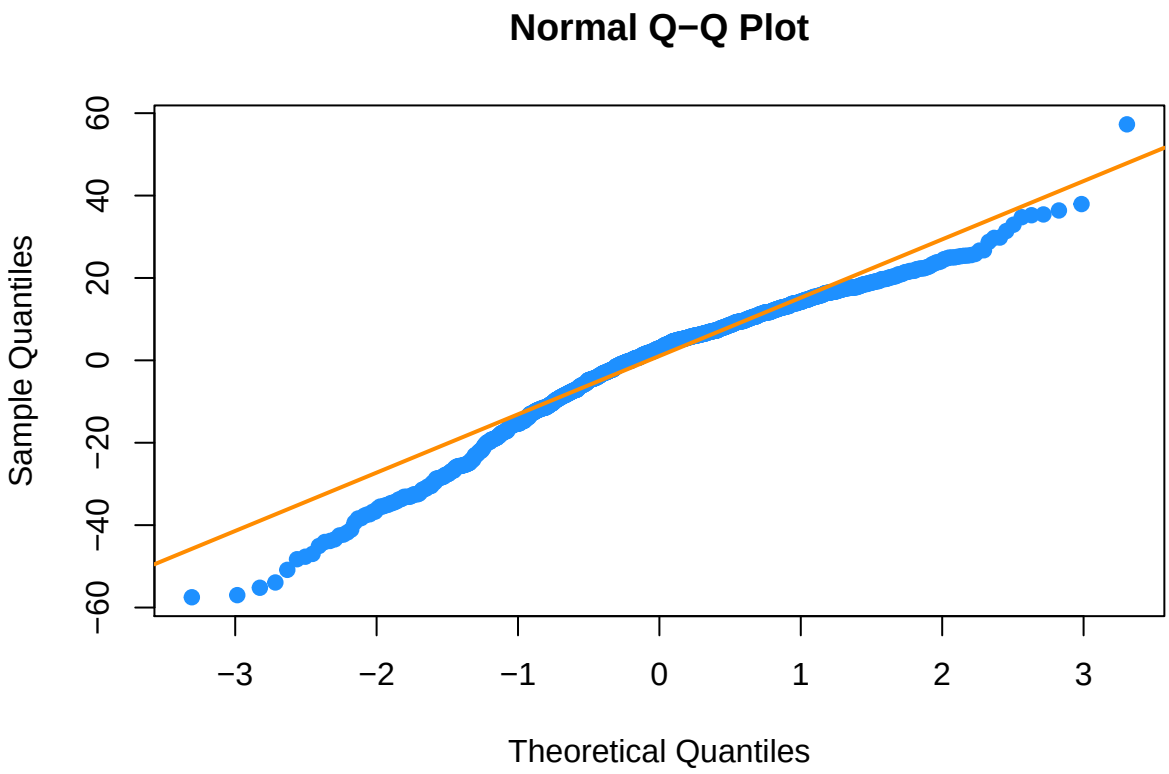
```
# fitted vs residuals for model latitude
plot(
  model_lat$fitted.values,
  model_lat$residuals,
  col = "grey",
  pch = 20,
  main = "Residuals vs Fitted",
  xlab = "Fitted",
  ylab = "Residuals"
)
abline(h = 0, col = "darkorange", lwd = 2)
```



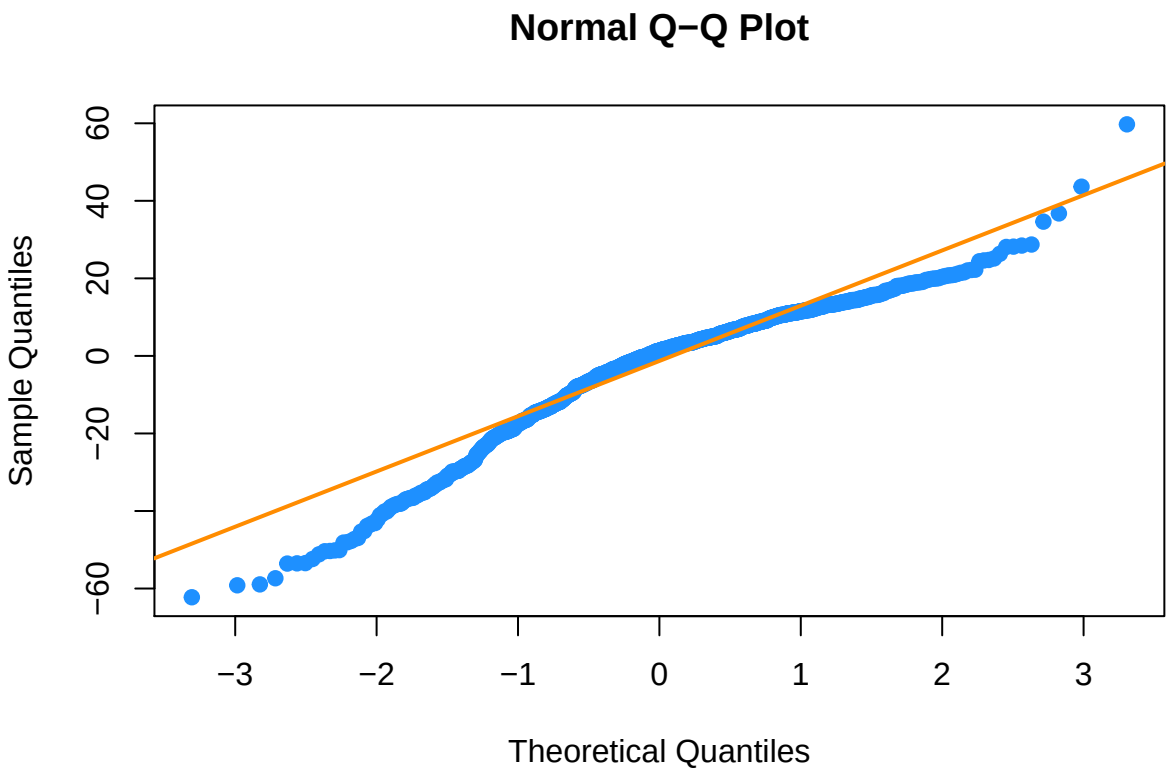

```
# fitted vs residuals for boxcox transformed model in original co-ordinates
plot(
  fitted,
  res,
  col = "grey",
  pch = 20,
  main = "Residuals vs Fitted",
  xlab = "Fitted",
  ylab = "Residuals"
)
abline(h = 0, col = "darkorange", lwd = 2)
```




```
## q-q plot for model latitude
qqnorm(model_lat$residuals, col = "dodgerblue", pch = 20, cex = 1.5)
qqline(model_lat$residuals, col = "darkorange", lwd = 2)
```




```
## q-q plot for boxcox transformed model
qqnorm(res, col = "dodgerblue", pch = 20, cex = 1.5)
qqline(res, col = "darkorange", lwd = 2)
```



- Model longitude does not need a transformation but Model latitude response needed a transformation as per the boxcox
- It seems like there is some slight degradation in the R^2 value for the transformed model (model_lat_transform) 0.247779 compared to original model 0.2928092
- The scale of the Residuals vs Fitted plots looks similar with & without transformation
- Based on the above we want to keep original models for both latitude & longitude (i.e. no transformation)

Use glmnet to produce:

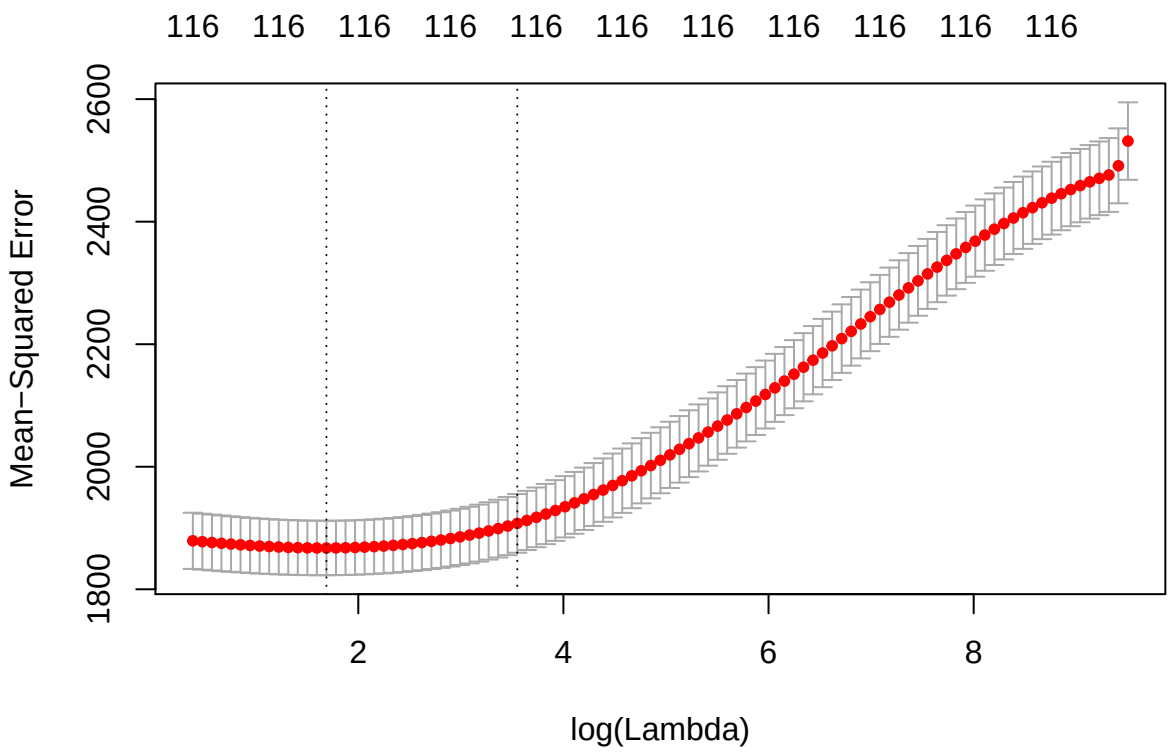
A regression regularized by L2 (equivalently, a ridge regression). You should estimate the regularization coefficient that produces the minimum error. Is the regularized regression better than the unregularized regression?

A regression regularized by L1 (equivalently, a lasso regression). You should estimate the regularization coefficient that produces the minimum error. How many variables are used by this regression? Is the regularized regression better than the unregularized regression?

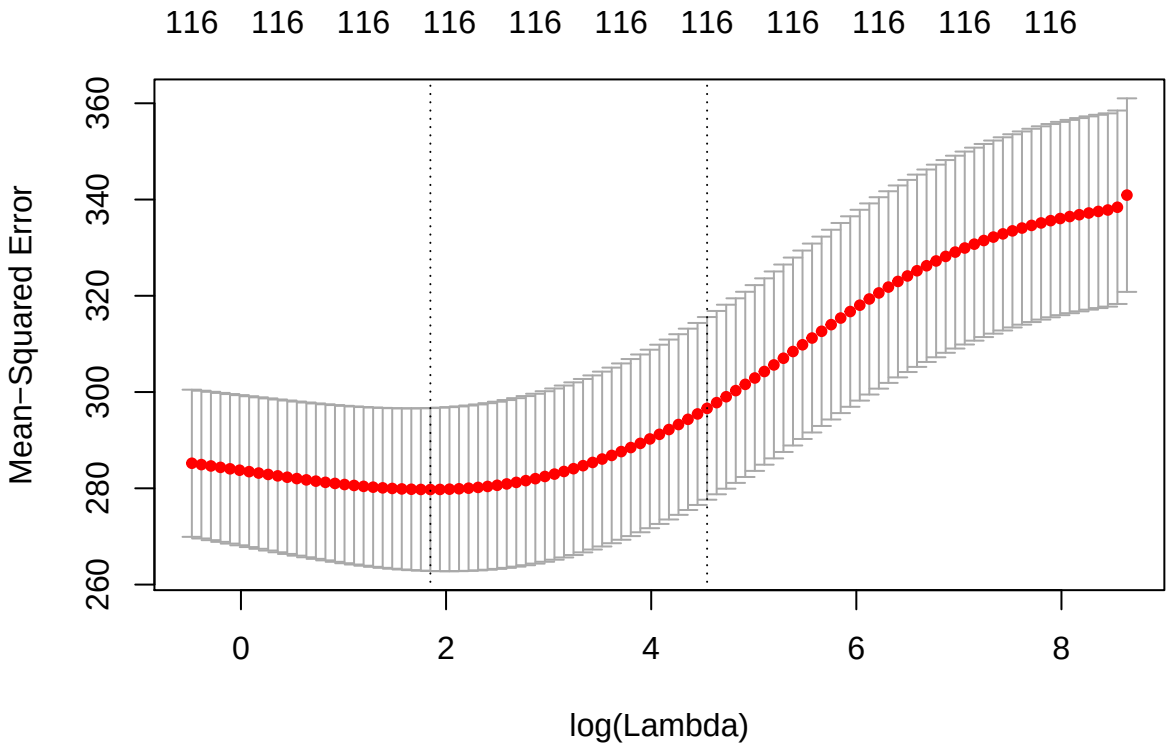
A regression regularized by elastic net (equivalently, a regression regularized by a convex combination of L1 and L2). Try three values of alpha, the weight setting how big L1 and L2 are. You should estimate the regularization coefficient that produces the minimum error. How many variables are used by this regression? Is the regularized regression better than the unregularized regression?

Q1-L2 Regularization

```
library(glmnet)

x       = as.matrix(music_tracks[, 1:(ncol(music_tracks) - 2)])
y_lat   = music_tracks$X117
y_lon   = music_tracks$X118

# Create Cross validation
folds = cut(seq(1, nrow(music_tracks)), breaks = 20, labels = FALSE)

Unreg_CVerr_lat = array(dim = 20)
Unreg_CVerr_lon = array(dim = 20)
# Perform 10 fold cross validation

for (i in 1:20) {
  # Segement data by folds
  test_idx = which(folds == i, arr.ind = TRUE)
  test_data = music_tracks[test_idx, ]
  train_data = music_tracks[-test_idx, ]

  model_lat_cv = lm(X117 ~ . - X118, data = train_data)
  model_lon_cv = lm(X118 ~ . - X117, data = train_data)


  fitted_results_lat = predict(model_lat_cv, newdata = test_data, type = 'response')
  fitted_results_lon = predict(model_lon_cv, newdata = test_data, type = 'response')
  #fitted_results = ifelse(fitted_results > 0.5, 1, 0)

  Unreg_CVerr_lat = (test_data$X117 - fitted_results_lat)^2
  Unreg_CVerr_lon = (test_data$X118 - fitted_results_lon)^2

}


### Regularized regression for longitudes
## Ridge regression
cv_fit_lon1 = cv.glmnet(x,
                        y_lon,
                        alpha = 0,
                        keep = TRUE)



plot(cv_fit_lon1)
```




```
cv_fit_lat1 = cv.glmnet(x,
                        y_lat,
                        alpha = 0,
                        keep = TRUE)
plot(cv_fit_lat1)
```




```
print(paste('CV Error,unregularized, latitude:- ', mean(Unreg_CVerr_lat)))
```

```
## [1] "CV Error,unregularized, latitude:-  319.576894152802"
```

```
print(paste('CV Error,regularized, latitude:- ', cv_fit_lat1$cvm[which.min(cv_fit_lat1$lambda)]))
```

```
## [1] "CV Error,regularized, latitude:-  285.207518345127"
```

```
print(paste('CV Error,unregularized, longtitude:- ', mean(Unreg_CVerr_lon)))
```

```
## [1] "CV Error,unregularized, longtitude:-  1819.42907549398"
```

```
print(paste('CV Error,regularized, longtitude:- ', cv_fit_lon1$cvm[which.min(cv_fit_lon1$lambda)]))
```

```
## [1] "CV Error,regularized, longtitude:-  1879.09305168759"
```

```
print(paste(
'Best Regularization Coeff, lattitude:- ',
cv_fit_lat1$lambda.min
))
```

```
## [1] "Best Regularization Coeff, lattitude:-  6.34133731800637"
```

```
print(paste(
'Best Regularization Coeff, longitude:- ',
cv_fit_lon1$lambda.min
))
```

```
## [1] "Best Regularization Coeff, longitude:-  5.41632046141742"
```

```
(get_num_params_glm(cv_fit_lat1))
```

```
## [1] 117
```

Q1-L2 Regularization

Longitude

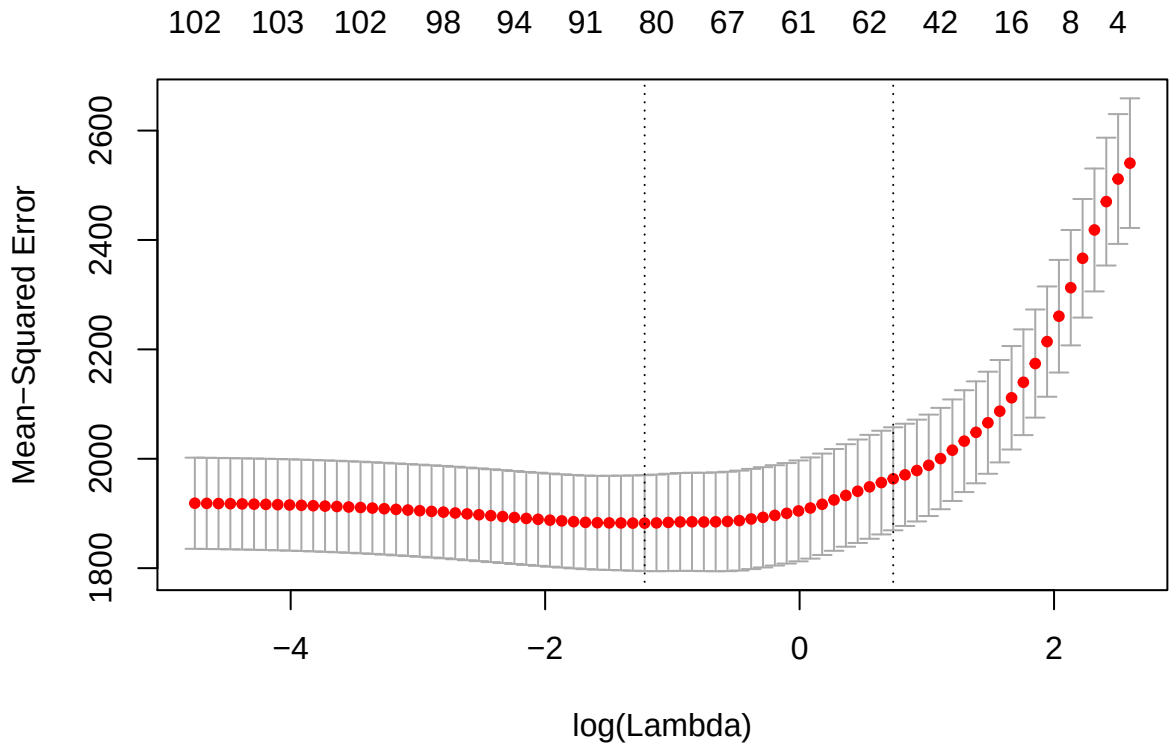
- The regularization coefficient which produces minimum error is 5.4163205
- The number of parameters is 117 compared to 117 which is without regularization
- From the cross validated error it is evident that unregularized model has a lower error than the regularized model.
- So, based on the CV error we chose the unregularized model for longitudes.

Latitude

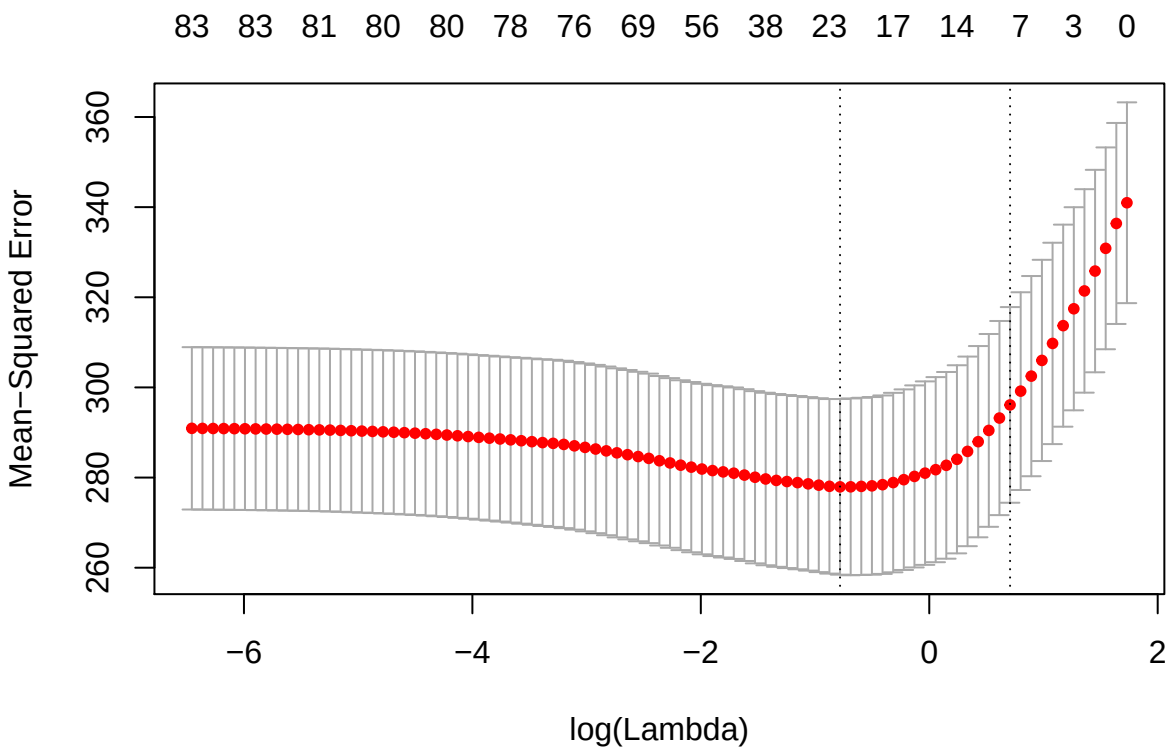
- The regularization coefficient which produces minimum error is 6.3413373
- The number of parameters is 117 compared to 117 which is without regularization
- From the cross validated error it is evident that regularized model has a lower error than the unregularized model.
- So, based on the CV error we chose the regularized model for latitudes.

Q1-L1 Regularization

```
## lasso - seems to be best
cv_fit_lon2 = cv.glmnet(x,
                        y_lon,
                        alpha = 1,
                        keep = TRUE)
plot(cv_fit_lon2)
```




```
## lasso
cv_fit_lat2 = cv.glmnet(x,
                        y_lat,
                        alpha = 1,
                        keep = TRUE)
plot(cv_fit_lat2)
```




```
print(paste('CV Error,unregularized, latitude:- ', mean(Unreg_CVerr_lat)))
```

```
## [1] "CV Error,unregularized, latitude:-  319.576894152802"
```

```
print(paste('CV Error,regularized, latitude:- ', cv_fit_lat2$cvm[which.min(cv_fit_lat2$lambda)]))
```

```
## [1] "CV Error,regularized, latitude:-  290.919878615641"
```

```
print(paste('CV Error,unregularized, longtitude:- ', mean(Unreg_CVerr_lon)))
```

```
## [1] "CV Error,unregularized, longtitude:-  1819.42907549398"
```

```
print(paste('CV Error,regularized, longtitude:- ', cv_fit_lon2$cvm[which.min(cv_fit_lon2$lambda)]))
```

```
## [1] "CV Error,regularized, longtitude:-  1918.71659728719"
```

```
print(paste(
'Best Regularization Coeff, lattitude:- ',
cv_fit_lat2$lambda.min
))
```

```
## [1] "Best Regularization Coeff, lattitude:-  0.457895856944847"
```

```
print(paste(
'Best Regularization Coeff, longitude:- ',
cv_fit_lon2$lambda.min
))
```

```
## [1] "Best Regularization Coeff, longitude:-  0.295854438600779"
```

```
print(paste('# of Parameters, Unregularized, latitude:- ', length(coef(model_lat))))
```

```
## [1] "# of Parameters, Unregularized, latitude:-  117"
```

```
print(paste(
'# of Parameters, regularized, latitude:- ',
get_num_params_glm(cv_fit_lat2)
))
```

```
## [1] "# of Parameters, regularized, latitude:-  22"
```

```
print(paste('# of Parameters, Unregularized, longitude:- ', length(coef(model_lon))))
```

```
## [1] "# of Parameters, Unregularized, longitude:-  117"
```

```
print(paste(
'# of Parameters, regularized, longitude:- ',
get_num_params_glm(cv_fit_lon2)
))
```

```
## [1] "# of Parameters, regularized, longitude:-  83"
```

Q1-L1 Regularization

Longitude

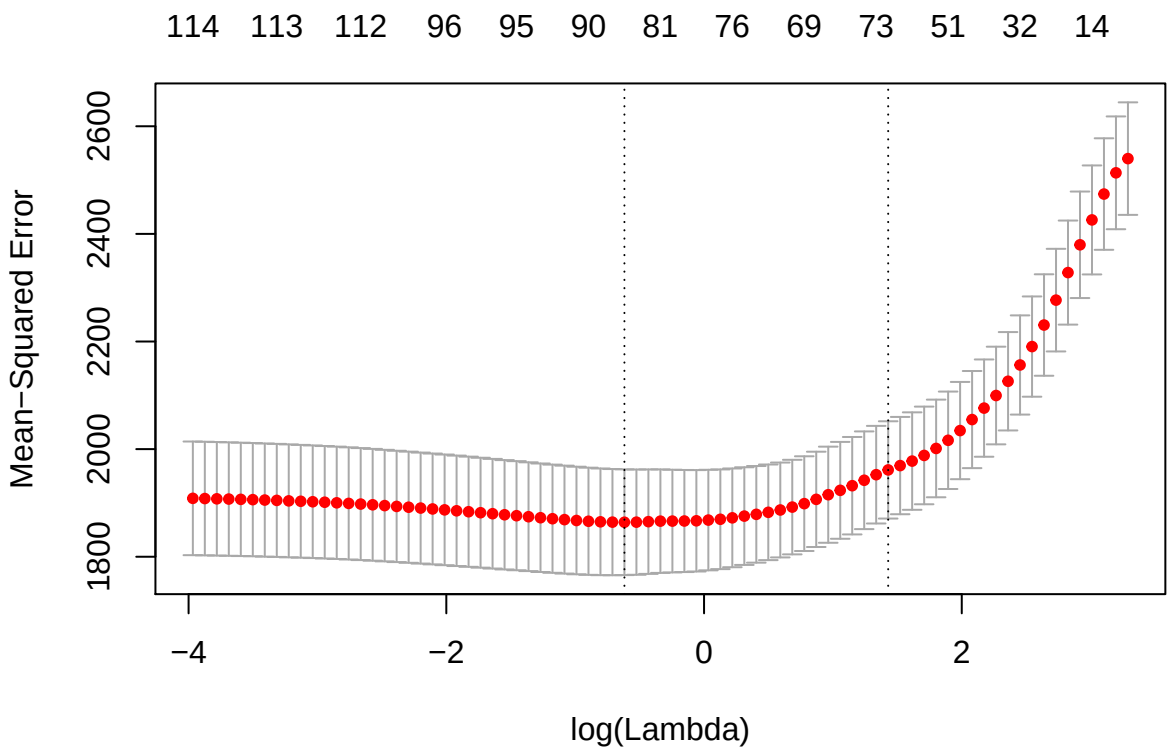
- The regularization coefficient which produces minimum error is 0.2958544
- The number of parameters are only 83 compared to 117 which is without regularization
- From the cross validated error it is evident that unregularized model has a lower error than the regularized model, but the no of parameters are much lower for regularized model.
- So, based on the CV error, and no of parameters we chose the regularized model for longitudes.

Latitude

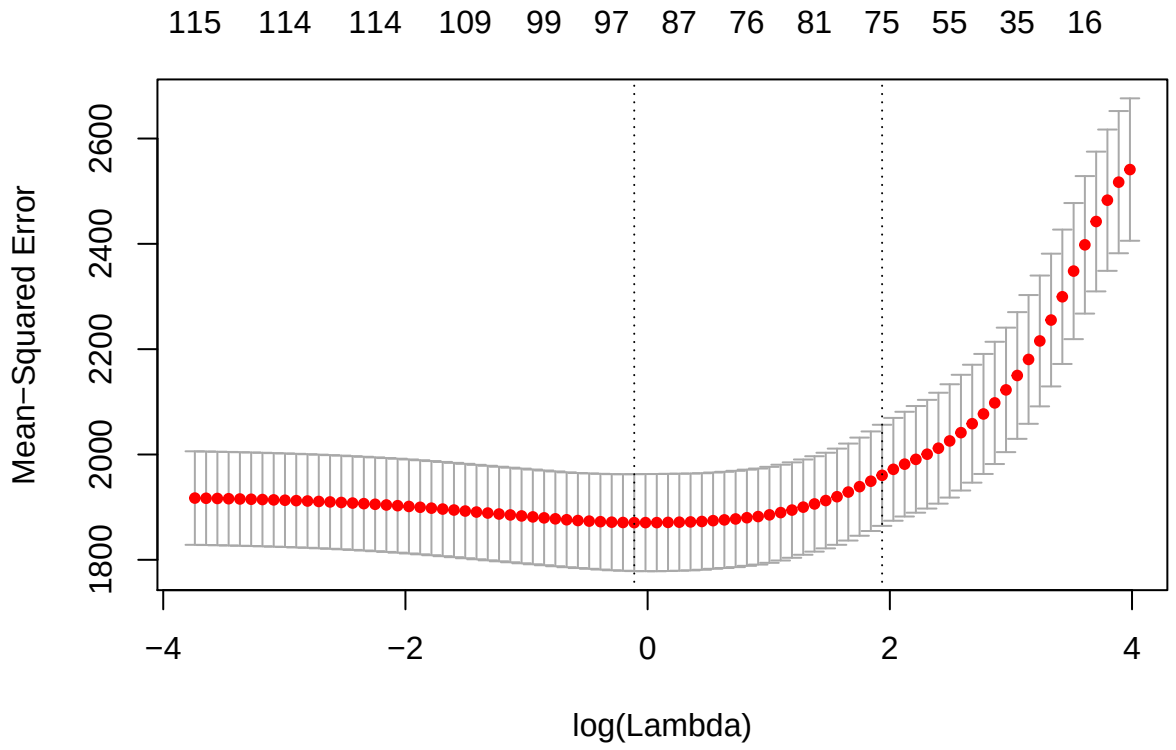
- The regularization coefficient which produces minimum error is 0.4578959
- The number of parameters are only 22 compared to 117 which is without regularization
- From the cross validated error it is evident that regularized model has a lower error than the unregularized model, also the no of parameters are much lower for regularized model.
- So, based on the CV error, and no of parameters we chose the regularized model for latitudes.

Q1-Elastic Net Regularization

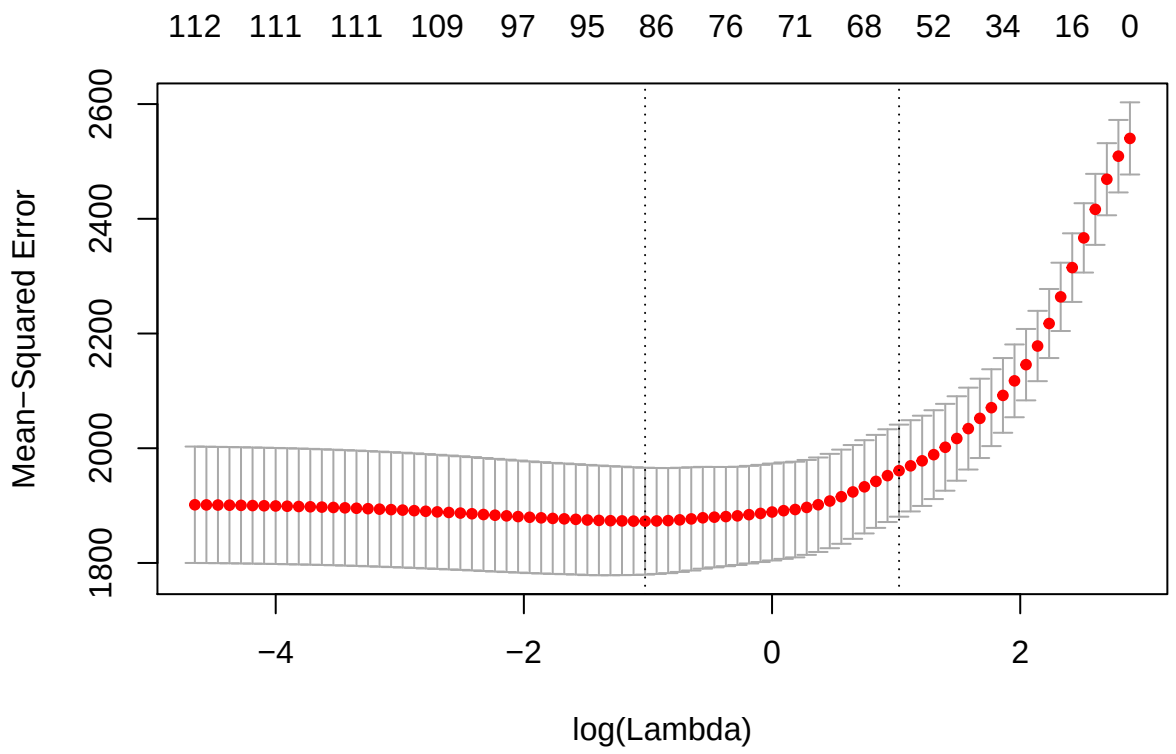
```
## ElasticNet - Longitudes
cv_fit_lon3 = cv.glmnet(x,
                        y_lon,
                        alpha = 0.5,
                        keep = TRUE)
plot(cv_fit_lon3)
```



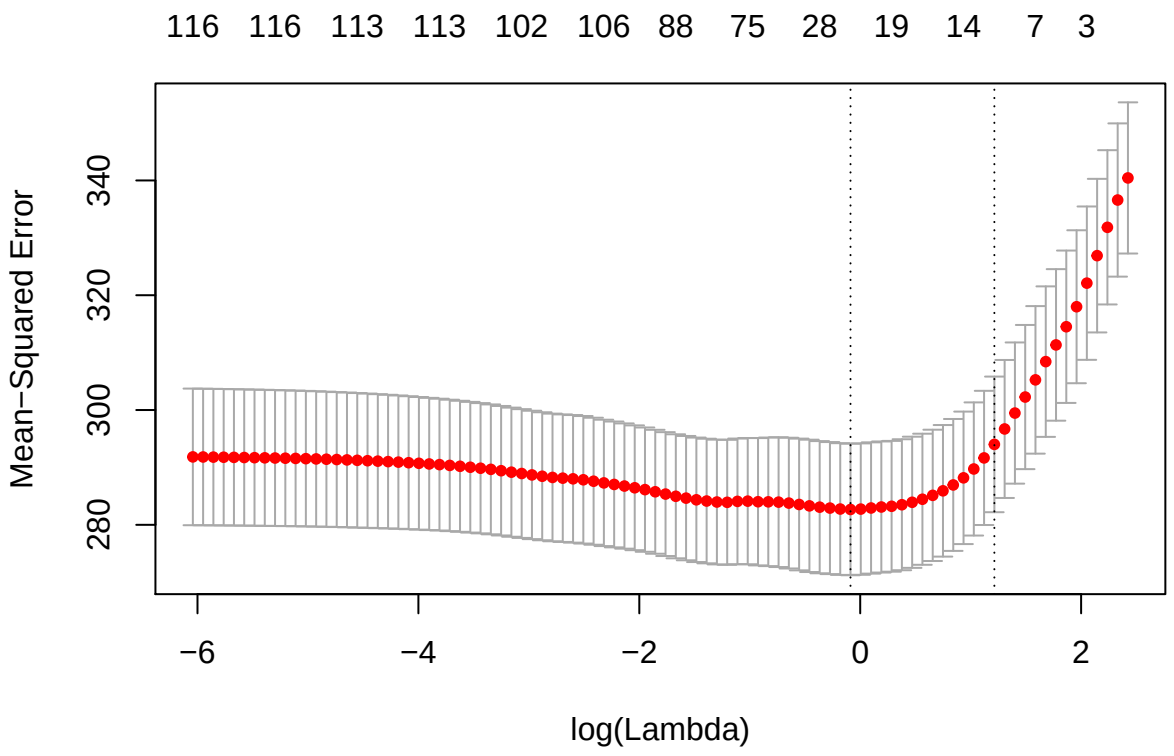

```
# (get_r2_glm(cv_fit_lon3, x, y_lon))
# (get_rmse_glm(cv_fit_lon3, x, y_lon))
# (get_num_params_glm(cv_fit_lon3))

cv_fit_lon4 = cv.glmnet(x,
                        y_lon,
                        alpha = 0.25,
                        keep = TRUE)
plot(cv_fit_lon4)
```



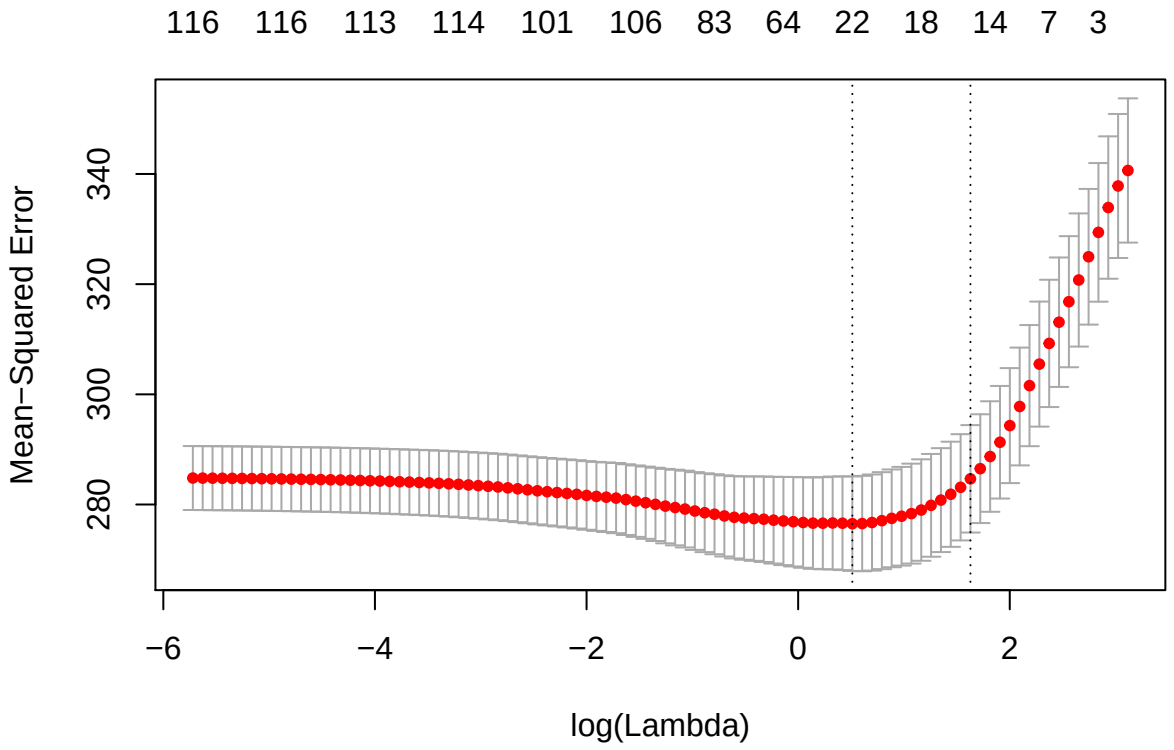

```
# (get_r2_glm(cv_fit_lon4, x, y_lon))
# (get_rmse_glm(cv_fit_lon4, x, y_lon))
# (get_num_params_glm(cv_fit_lon4))

cv_fit_lon5 = cv.glmnet(x,
                        y_lon,
                        alpha = 0.75,
                        keep = TRUE)
plot(cv_fit_lon5)
```



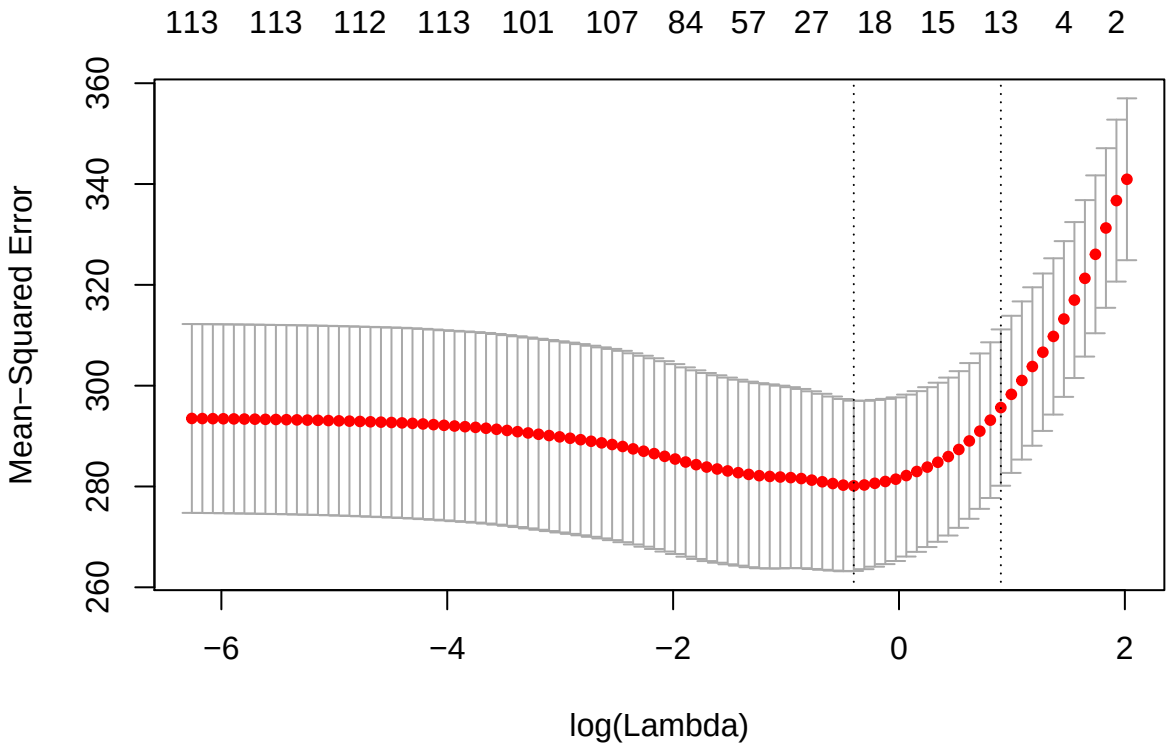

```
# (get_r2_glm(cv_fit_lon5, x, y_lon))
# (get_rmse_glm(cv_fit_lon5, x, y_lon))
# (get_num_params_glm(cv_fit_lon5))


## ElasticNet - Latitudes
cv_fit_lat3 = cv.glmnet(x,
                        y_lat,
                        alpha = 0.5,
                        keep = TRUE)
plot(cv_fit_lat3)
```




```
# (get_r2_glm(cv_fit_lat3, x, y_lat))
# (get_rmse_glm(cv_fit_lat3, x, y_lat))
# get_num_params_glm(cv_fit_lat3)

cv_fit_lat4 = cv.glmnet(x,
                        y_lat,
                        alpha = 0.25,
                        keep = TRUE)
plot(cv_fit_lat4)
```




```
# (get_r2_glm(cv_fit_lat4, x, y_lat))
# (get_rmse_glm(cv_fit_lat4, x, y_lat))
# get_num_params_glm(cv_fit_lat4)

cv_fit_lat5 = cv.glmnet(x,
                        y_lat,
                        alpha = 0.75,
                        keep = TRUE)
plot(cv_fit_lat5)
```




```
# (get_r2_glm(cv_fit_lat5, x, y_lat))
# (get_rmse_glm(cv_fit_lat5, x, y_lat))
# (get_num_params_glm(cv_fit_lat5))

#CV Error
print(paste('CV Error,unregularized, latitude:- ', mean(Unreg_CVerr_lat)))
```

```
## [1] "CV Error,unregularized, latitude:-  319.576894152802"
```

```
print(paste(
'CV Error,regularized, latitude, alpha = 0.5:- ',
cv_fit_lat3$cvm[which.min(cv_fit_lat3$lambda)]
))
```

```
## [1] "CV Error,regularized, latitude, alpha = 0.5:-  291.832428509647"
```

```
print(paste(
'CV Error,regularized, latitude, alpha = 0.25:- ',
cv_fit_lat4$cvm[which.min(cv_fit_lat4$lambda)]
))
```

```
## [1] "CV Error,regularized, latitude, alpha = 0.25:-  284.807799627452"
```

```
print(paste(
'CV Error,regularized, latitude, alpha = 0.75:- ',
cv_fit_lat5$cvm[which.min(cv_fit_lat5$lambda)]
))
```

```
## [1] "CV Error,regularized, latitude, alpha = 0.75:-  293.496392173714"
```

```r
print(paste('CV Error,unregularized, longtitude', mean(Unreg_CVerr_lon)))
```

```
## [1] "CV Error,unregularized, longtitude 1819.42907549398"
```

```r
print(paste(
'CV Error,regularized, longtitude, alpha = 0.5:- ',
cv_fit_lon3$cvm[which.min(cv_fit_lon3$lambda)]
))
```

```
## [1] "CV Error,regularized, longtitude, alpha = 0.5:-  1908.51060474701"
```

```r
print(paste(
'CV Error,regularized, longtitude, alpha = 0.25:- ',
cv_fit_lon4$cvm[which.min(cv_fit_lon4$lambda)]
))
```

```
## [1] "CV Error,regularized, longtitude, alpha = 0.25:-  1917.26748168344"
```

```r
print(paste(
'CV Error,regularized, longtitude, alpha = 0.75:- ',
cv_fit_lon5$cvm[which.min(cv_fit_lon5$lambda)]
))
```

```
## [1] "CV Error,regularized, longtitude, alpha = 0.75:-  1901.47210762182"
```

```r
# Optimum value for regularization constant
print(paste(
'Best Regularization Coeff, lattitude, alpha = 0.5:- ',
cv_fit_lat3$lambda.min
))
```

```
## [1] "Best Regularization Coeff, lattitude, alpha = 0.5:-  0.915791713889693"
```

```r
print(paste(
'Best Regularization Coeff, lattitude, alpha = 0.25:- ',
cv_fit_lat4$lambda.min
))
```

```
## [1] "Best Regularization Coeff, lattitude, alpha = 0.25:-  1.66887060411092"
```

```r
print(paste(
'Best Regularization Coeff, lattitude, alpha = 0.75:- ',
cv_fit_lat5$lambda.min
))
```

```
## [1] "Best Regularization Coeff, lattitude, alpha = 0.75:-  0.670053516961806"
```

```r
print(paste(
'Best Regularization Coeff, longitude, alpha = 0.5:- ',
cv_fit_lon3$lambda.min
))
```

```
## [1] "Best Regularization Coeff, longitude, alpha = 0.5:-  0.539143091368973"
```

```r
print(paste(
'Best Regularization Coeff, longitude, alpha = 0.25:- ',
cv_fit_lon4$lambda.min
))
```

```
## [1] "Best Regularization Coeff, longitude, alpha = 0.25:-  0.895212132383008"
```

```
print(paste(
'Best Regularization Coeff, longitude, alpha = 0.75:- ',
cv_fit_lon5$lambda.min
))
```

```
## [1] "Best Regularization Coeff, longitude, alpha = 0.75:-  0.359428727579316"
```

```
# no of Paramets after regularization
print(paste(
'# of Parameters, regularized, latitude, alpha = 0.5:- ',
get_num_params_glm(cv_fit_lat3)
))
```

```
## [1] "# of Parameters, regularized, latitude, alpha = 0.5:-  23"
```

```
print(paste(
'# of Parameters, regularized, latitude, alpha = 0.25:- ',
get_num_params_glm(cv_fit_lat4)
))
```

```
## [1] "# of Parameters, regularized, latitude, alpha = 0.25:-  23"
```

```
print(paste(
'# of Parameters, regularized, latitude, alpha = 0.75:- ',
get_num_params_glm(cv_fit_lat5)
))
```

```
## [1] "# of Parameters, regularized, latitude, alpha = 0.75:-  22"
```

```
print(paste(
'# of Parameters, regularized, longitude, alpha = 0.5:- ',
get_num_params_glm(cv_fit_lon3)
))
```

```
## [1] "# of Parameters, regularized, longitude, alpha = 0.5:-  89"
```

```
print(paste(
'# of Parameters, regularized, longitude, alpha = 0.25:- ',
get_num_params_glm(cv_fit_lon4)
))
```

```
## [1] "# of Parameters, regularized, longitude, alpha = 0.25:-  94"
```

```
print(paste(
'# of Parameters, regularized, longitude, alpha = 0.75:- ',
get_num_params_glm(cv_fit_lon5)
))
```

```
## [1] "# of Parameters, regularized, longitude, alpha = 0.75:-  89"
```

Q1-Elastic Net Regularization

Longitude

- The regularization coefficient with $\alpha = 0.75$ is 0.3594287
- The number of parameters is 89 compared to 117 which is without regularization

- From the cross validated error it is evident that unregularized model has a lower error than the regularized model, but the no of parameters are much lower for regularized models.
- So, based on the CV error, and no of parameters we chose the regularized model for longitude with α=0.75, since it has the lowest no of parameters and CV error comparable to other regularized models.

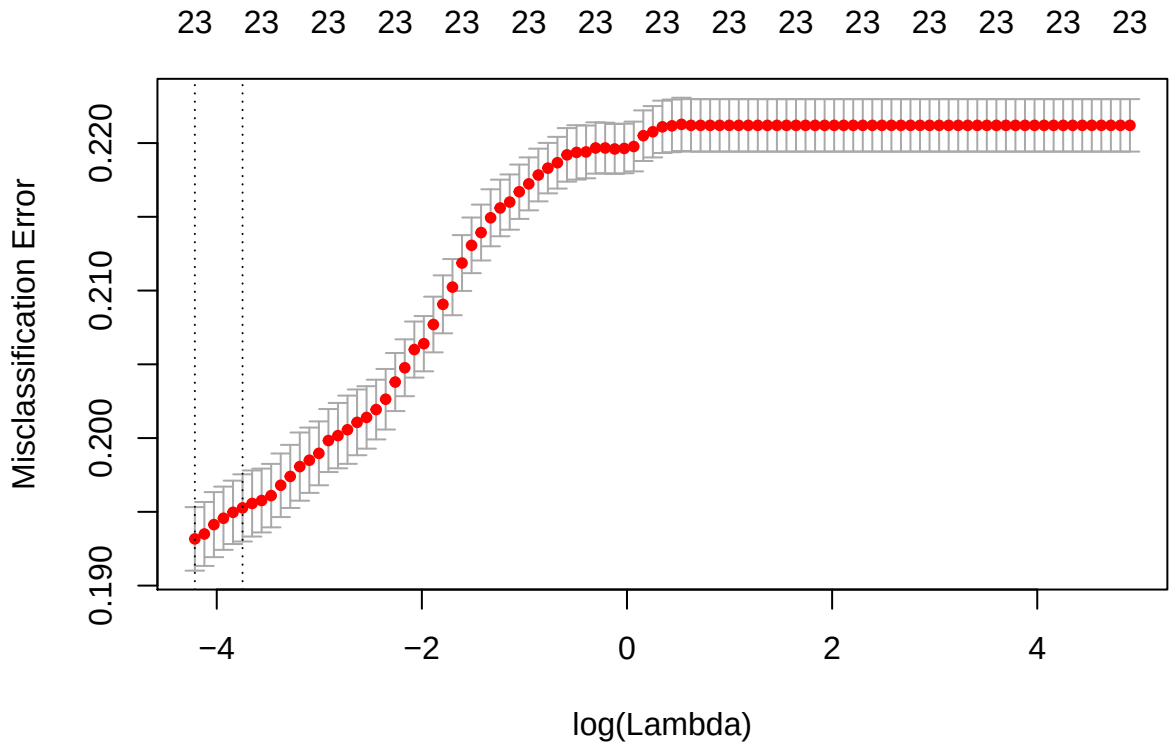
Latitude

- The regularization coefficient with $\alpha = 0.75$ is 0.6700535
- The number of parameters is 22 compared to 117 which is without regularization
- From the cross validated error it is evident that all regularized models have lower CV error than the unregularized model, also the no of parameters are much lower for regularized model.
- So, based on the CV error, and no of parameters we chose the regularized model for latitude with α=0.75, since it has the lowest no of parameters and CV error comparable to other regularized models.

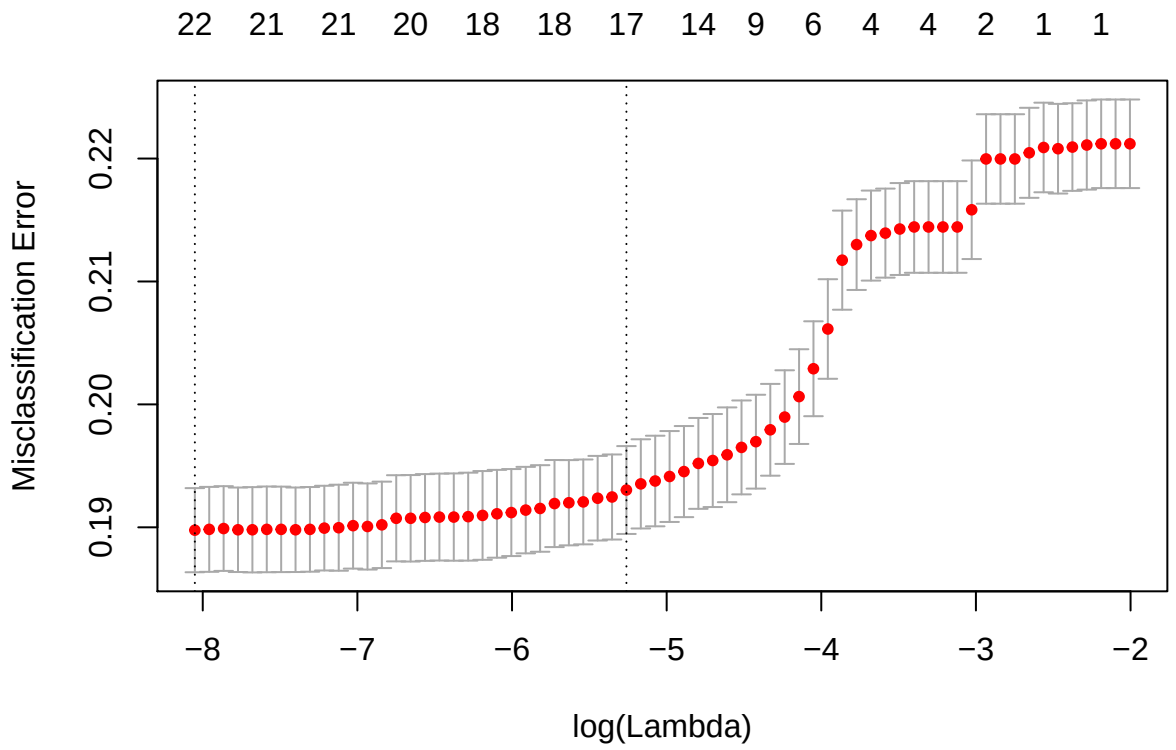
Problem 2 - Logistic regression The UCI Machine Learning dataset repository hosts a dataset giving whether a Taiwanese credit card user defaults against a variety of features here. Use logistic regression to predict whether the user defaults. You should ignore outliers, but you should try the various regularization schemes we have discussed.

```
default_credit_card_clients =
  read_csv("default_credit_card_clients.csv")

## Logistic regression using glm
# Randomly shuffle the data
credit_card_data = default_credit_card_clients[sample(nrow(default_credit_card_clients)), -1]

# Create 10 equally size folds
folds = cut(seq(1, nrow(credit_card_data)), breaks = 20, labels = FALSE)

mis_classification_err = array(dim = 20)
# Perform 10 fold cross validation

for (i in 1:20) {
  # Segement data by folds
  test_idx = which(folds == i, arr.ind = TRUE)
  test_data = credit_card_data[test_idx, ]
  train_data = credit_card_data[-test_idx, ]

  model = glm(Default ~ .,
              data = train_data,
              family = binomial(link = "logit"))

  fitted_results = predict(model, newdata = test_data, type = 'response')
  fitted_results = ifelse(fitted_results > 0.5, 1, 0)

  mis_classification_err[i] = mean(data.frame(fitted_results) != data.frame(test_data$Default))
}
```

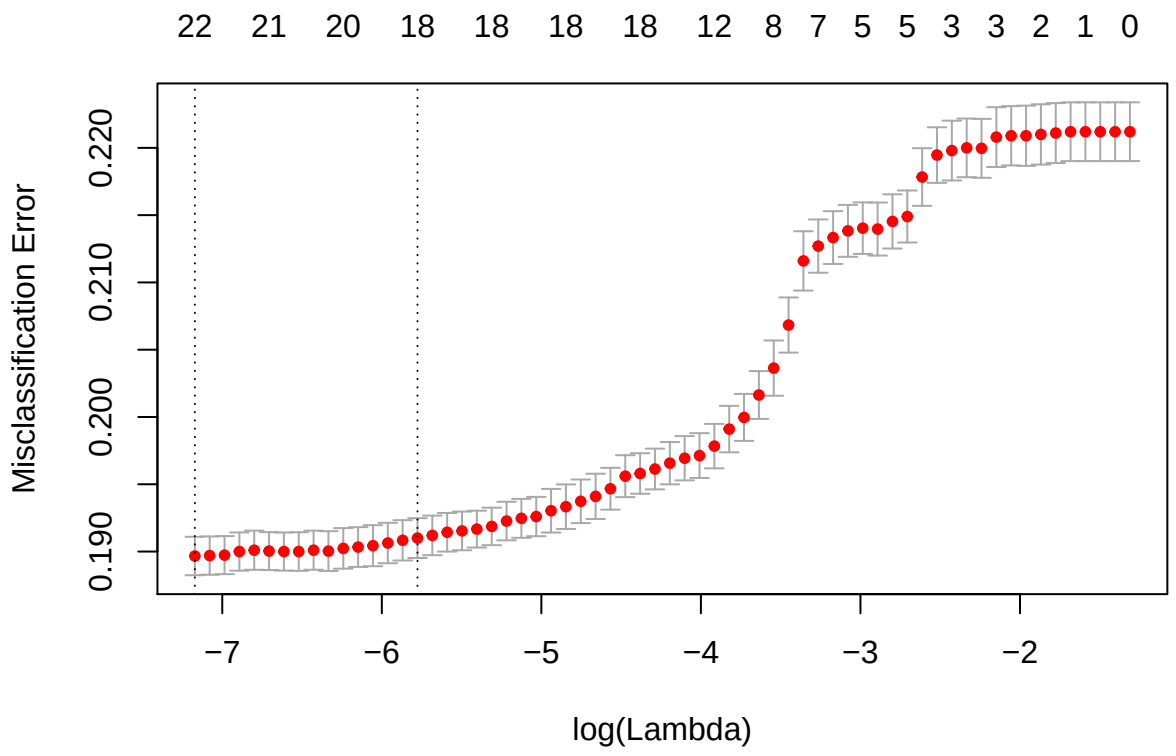
```
X = as.matrix(credit_card_data[, 1:(ncol(credit_card_data)) - 1])

Y = credit_card_data$Default

cv_fit_ridge = cv.glmnet(
  X,
  Y,
  alpha = 0,
  keep = TRUE,
  family = "binomial",
  type.measure = "class"
)

plot(cv_fit_ridge)
```



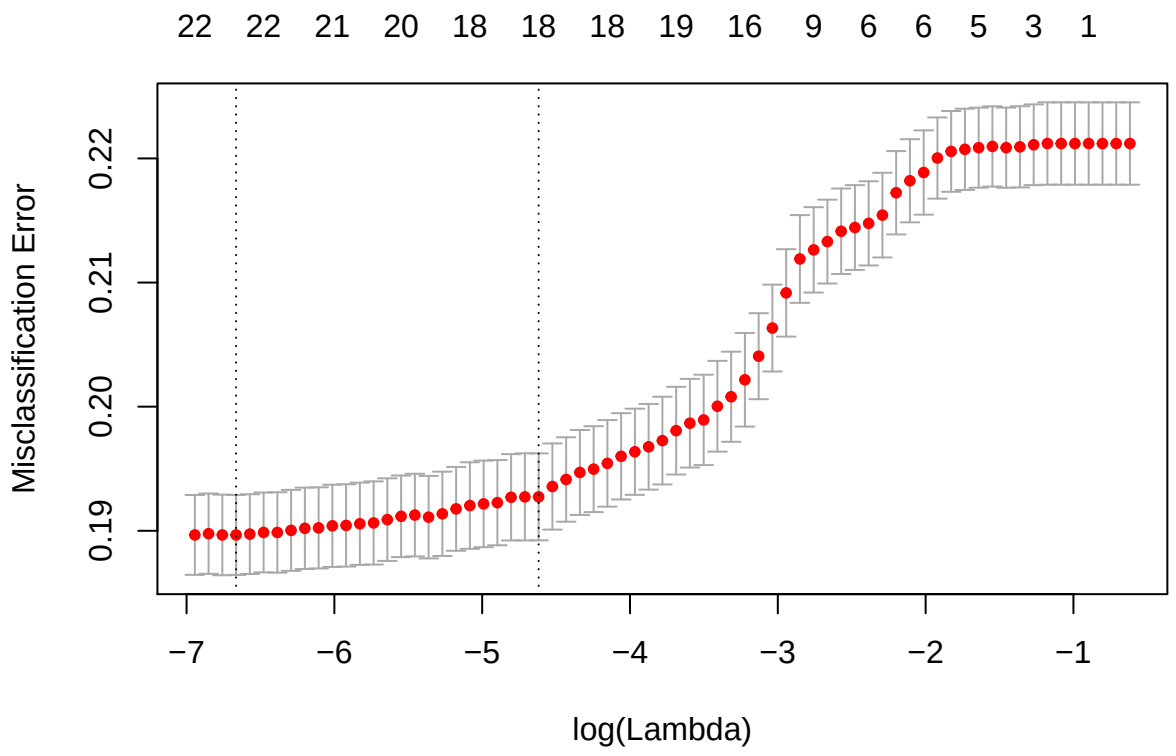

```
## For Lasso
cv_fit_lasso = cv.glmnet(
  X,
  Y,
  alpha = 1,
  keep = TRUE,
  family = "binomial",
  type.measure = "class"
)

plot(cv_fit_lasso)
```



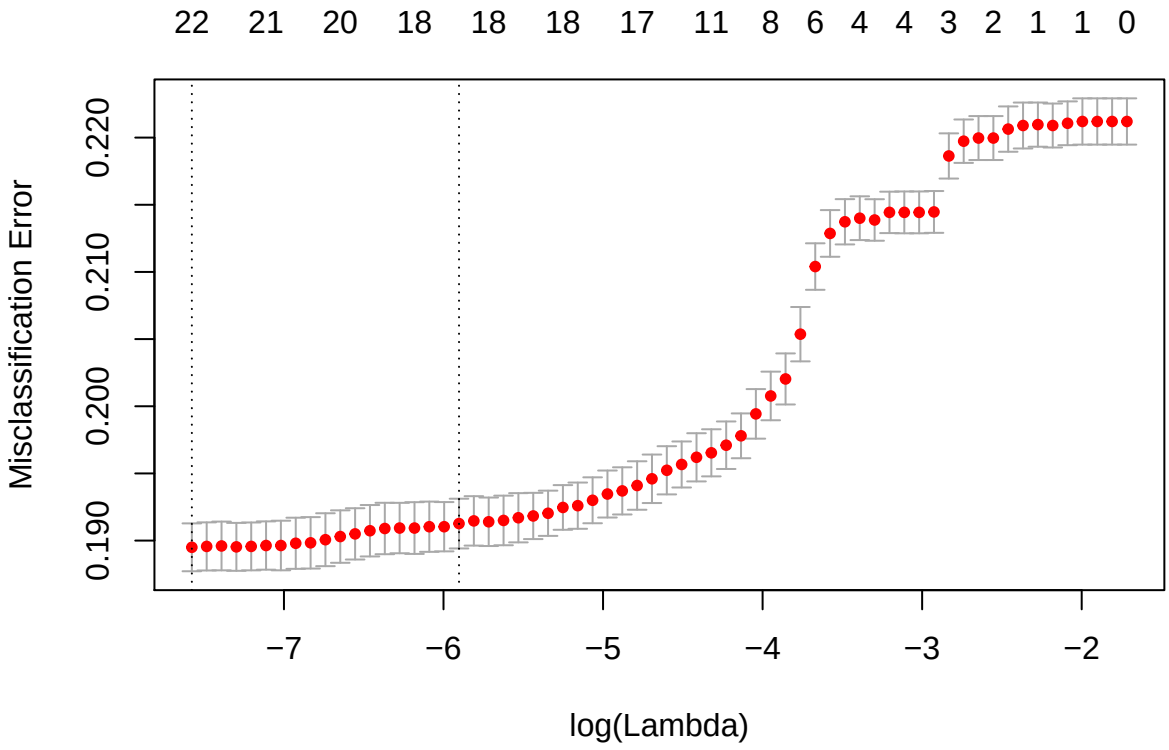

```
## For Elastic Net
cv_fit_enet1 = cv.glmnet(
  X,
  Y,
  alpha = 0.5,
  keep = TRUE,
  family = "binomial",
  type.measure = "class"
)

plot(cv_fit_enet1)
```




```
## For Elastic Net
cv_fit_enet2 = cv.glmnet(
  X,
  Y,
  alpha = 0.25,
  keep = TRUE,
  family = "binomial",
  type.measure = "class"
)

plot(cv_fit_enet2)
```




```
## For Elastic Net
cv_fit_enet3 = cv.glmnet(
  X,
  Y,
  alpha = 0.75,
  keep = TRUE,
  family = "binomial",
  type.measure = "class"
)

plot(cv_fit_enet3)
```




```
## Statistics
print(paste('Unregularized Accuracy:- ', (accuracy = 1 - mean(
  mis_classification_err
  ))))
```

```
## [1] "Unregularized Accuracy:-  0.810666666666667"
```

```
  print(paste('Unregularized Error:- ', (CVErr_glm = mean(
  mis_classification_err
  ))))
```

```
## [1] "Unregularized Error:-  0.189333333333333"
```

```
  print(paste('Unregularized # of params:- ', (num_params = length(coef(
  model
  )))))
```

```
## [1] "Unregularized # of params:-  24"
```

```
  print(paste(
  'Regularized Ridge Accuracy:- ',
  (accuracy_ridge =  1 - cv_fit_ridge$cvm[which.min(cv_fit_ridge$lambda)])
  ))
```

```
## [1] "Regularized Ridge Accuracy:-  0.806833333333333"
```

```
  print(paste('Regularized Ridge Error:- ', (CVErr_ridge = cv_fit_ridge$cvm[which.min(cv_fit_ridge$lambd
```

```
## [1] "Regularized Ridge Error:-  0.193166666666667"
```

```
  print(paste(
  'Regularized Ridge # of params:- ',
  (num_params_ridge = get_num_params_glm(cv_fit_ridge))
  ))
```

```
## [1] "Regularized Ridge # of params:-  24"
```

```
  print(paste(
  'Regularized Ridge Best lambda:- ',
  (lm_ridge = cv_fit_ridge$lambda.min)
  ))
```

```
## [1] "Regularized Ridge Best lambda:-  0.0147950762551908"
```

```
  print(paste(
  'Regularized Lasso Accuracy:- ',
  (accuracy_lasso =  1 - cv_fit_lasso$cvm[which.min(cv_fit_lasso$lambda)])
  ))
```

```
## [1] "Regularized Lasso Accuracy:-  0.810233333333333"
```

```
  print(paste('Regularized Lasso Error:- ', (CVErr_lasso = cv_fit_lasso$cvm[which.min(cv_fit_lasso$lambd
```

```
## [1] "Regularized Lasso Error:-  0.189766666666667"
```

```
  print(paste(
  'Regularized Lasso # of params:- ',
  (num_params_lasso = get_num_params_glm(cv_fit_lasso))
  ))
```

```
## [1] "Regularized Lasso # of params:-  23"
```

```
  print(paste(
  'Regularized Lasso Best lambda:- ',
  (lm_lasso = cv_fit_lasso$lambda.min)
  ))
```

```
## [1] "Regularized Lasso Best lambda:-  0.000318750255258521"
```

```
  print(paste(
  'Regularized EN Accuracy alpha = 0.5:- ',
  (accuracy_enet1 =  1 - cv_fit_enet1$cvm[which.min(cv_fit_enet1$lambda)])
  ))
```

```
## [1] "Regularized EN Accuracy alpha = 0.5:-  0.810333333333333"
```

```
  print(paste(
  'Regularized EN Error alpha = 0.5:- ',
  (CVErr_enet1 = cv_fit_enet1$cvm[which.min(cv_fit_enet1$lambda)])
  ))
```

```
## [1] "Regularized EN Error alpha = 0.5:-  0.189666666666667"
```

```
  print(paste(
  'Regularized EN # of params alpha = 0.5:- ',
  (num_params_enet1 = get_num_params_glm(cv_fit_enet1))
  ))
```

```
## [1] "Regularized EN # of params alpha = 0.5:-  23"
```

```
  print(paste(
  'Regularized EN Best lambda alpha = 0.5:- ',
  (lm_enet1 = cv_fit_enet1$lambda.min)
  ))
```

```
## [1] "Regularized EN Best lambda alpha = 0.5:-  0.000767871621834558"
```

```
  print(paste(
  'Regularized EN Accuracy alpha = 0.25:- ',
  (accuracy_enet2 =  1 - cv_fit_enet2$cvm[which.min(cv_fit_enet2$lambda)])
  ))
```

```
## [1] "Regularized EN Accuracy alpha = 0.25:-  0.810333333333333"
```

```
  print(paste(
  'Regularized EN Error alpha = 0.25:- ',
  (CVErr_enet2 = cv_fit_enet2$cvm[which.min(cv_fit_enet2$lambda)])
  ))
```

```
## [1] "Regularized EN Error alpha = 0.25:-  0.189666666666667"
```

```
  print(paste(
  'Regularized EN # of params alpha = 0.25:- ',
  (num_params_enet2 = get_num_params_glm(cv_fit_enet2))
  ))
```

```
## [1] "Regularized EN # of params alpha = 0.25:-  23"
```

```
  print(paste(
  'Regularized EN Best lambda alpha = 0.25:- ',
  (lm_enet2 = cv_fit_enet2$lambda.min)
  ))
```

```
## [1] "Regularized EN Best lambda alpha = 0.25:-  0.00127500102103408"
```

```
  print(paste(
  'Regularized EN Accuracy alpha = 0.75:- ',
  (accuracy_enet3 =  1 - cv_fit_enet3$cvm[which.min(cv_fit_enet3$lambda)])
  ))
```

```
## [1] "Regularized EN Accuracy alpha = 0.75:-  0.8105"
```

```
  print(paste(
  'Regularized EN Error alpha = 0.75:- ',
  (CVErr_enet3 = cv_fit_enet3$cvm[which.min(cv_fit_enet3$lambda)])
  ))
```

```
## [1] "Regularized EN Error alpha = 0.75:-  0.1895"
```

```
  print(paste(
  'Regularized EN # of params alpha = 0.75:- ',
  (num_params_enet3 = get_num_params_glm(cv_fit_enet3))
  ))
```

```
## [1] "Regularized EN # of params alpha = 0.75:-  23"
```

```
  print(paste(
  'Regularized EN Best lambda alpha = 0.75:- ',
  (lm_enet3 = cv_fit_enet3$lambda.min)
  ))
```

```
## [1] "Regularized EN Best lambda alpha = 0.75:-  0.000511914414556372"
```

- The CV error are pretty comparable between all the models. But the CV error for "Unregularized" and for the "Regularized ElasticNet ($\alpha = 0.75$)" was slightly lower than the rest.
- We would choose the model "Regularized ElasticNet ($\alpha = 0.75$)" over "Unregularized" since it has lower number of parameters required.