ECON882 Assignment 2

Mehtab Hanzroh

2025-03-03

ECON882 Assignment 1

Importing Libraries

Loading Data

```
df = read.csv("../../Datasets/loan-defs.csv")
```

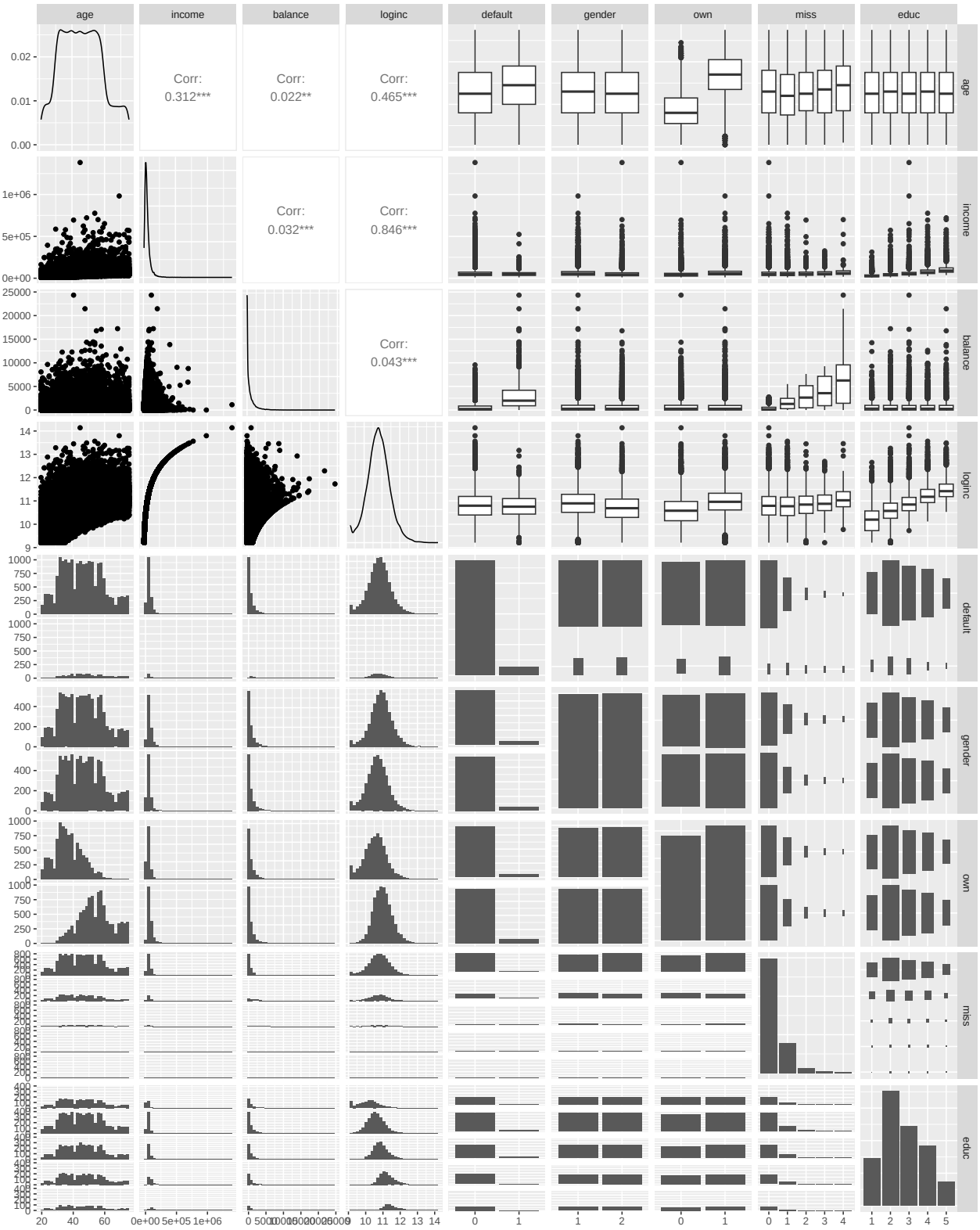
Question 1

```
set.seed(2)
df$loginc = log(df$income)
df_split = initial_split(df,prop=0.75)
df_train = training(df_split)
df_test = testing(df_split)
```

Question 2

Let's start with a pairs plot.

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

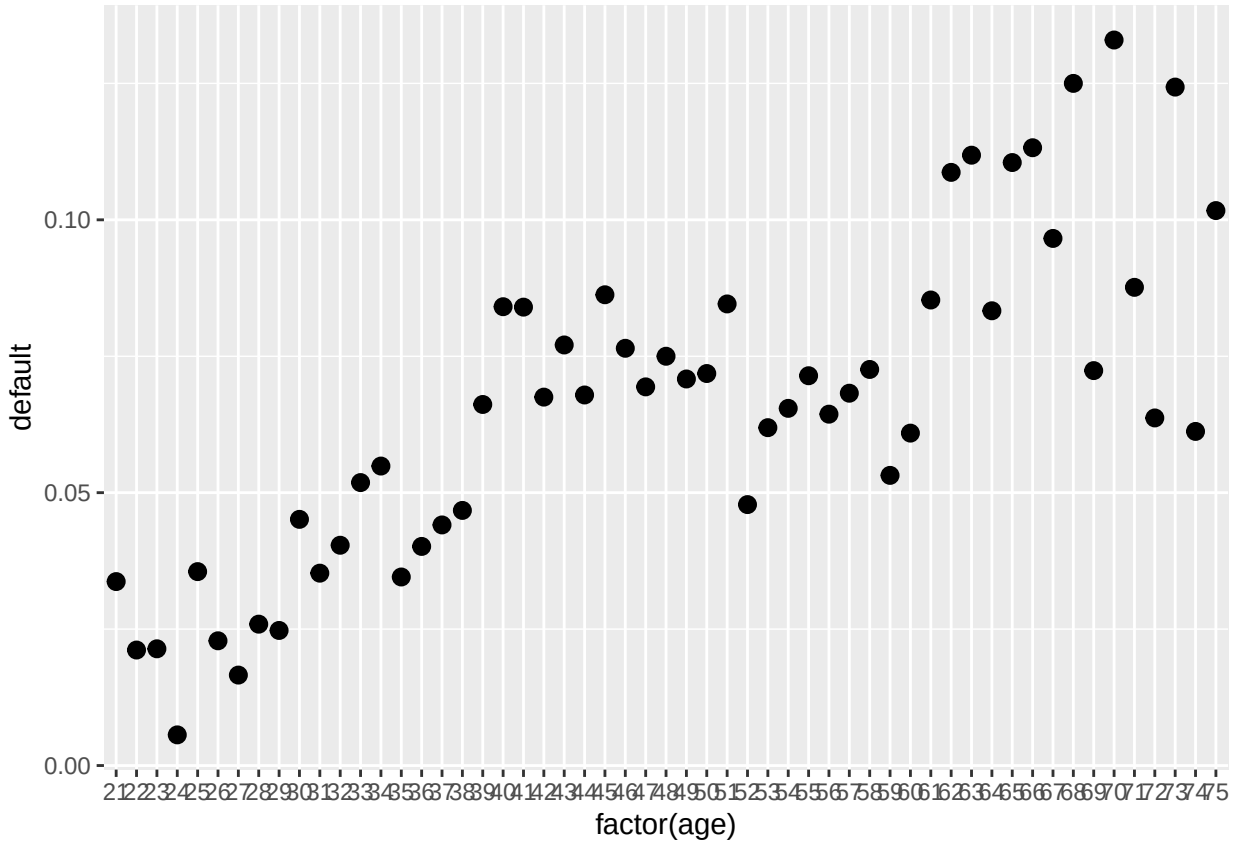


Based on the pairs plot, it is clear that gender and own are dummies, and thus will be included as-is in the linear model. The number of missed payments does not seem to have a strong non-linear relation with default, so we will include it as a continuous variable, although using dummies is certainly possible. Education does seem to have a non-linear relation, so we will use education dummies. The relation between default and age

is harder to see, so let's plot the proportion of defaulters at each age.

```
ggplot(df, aes(x=factor(age),y=default)) + stat_summary(fun=mean)
```

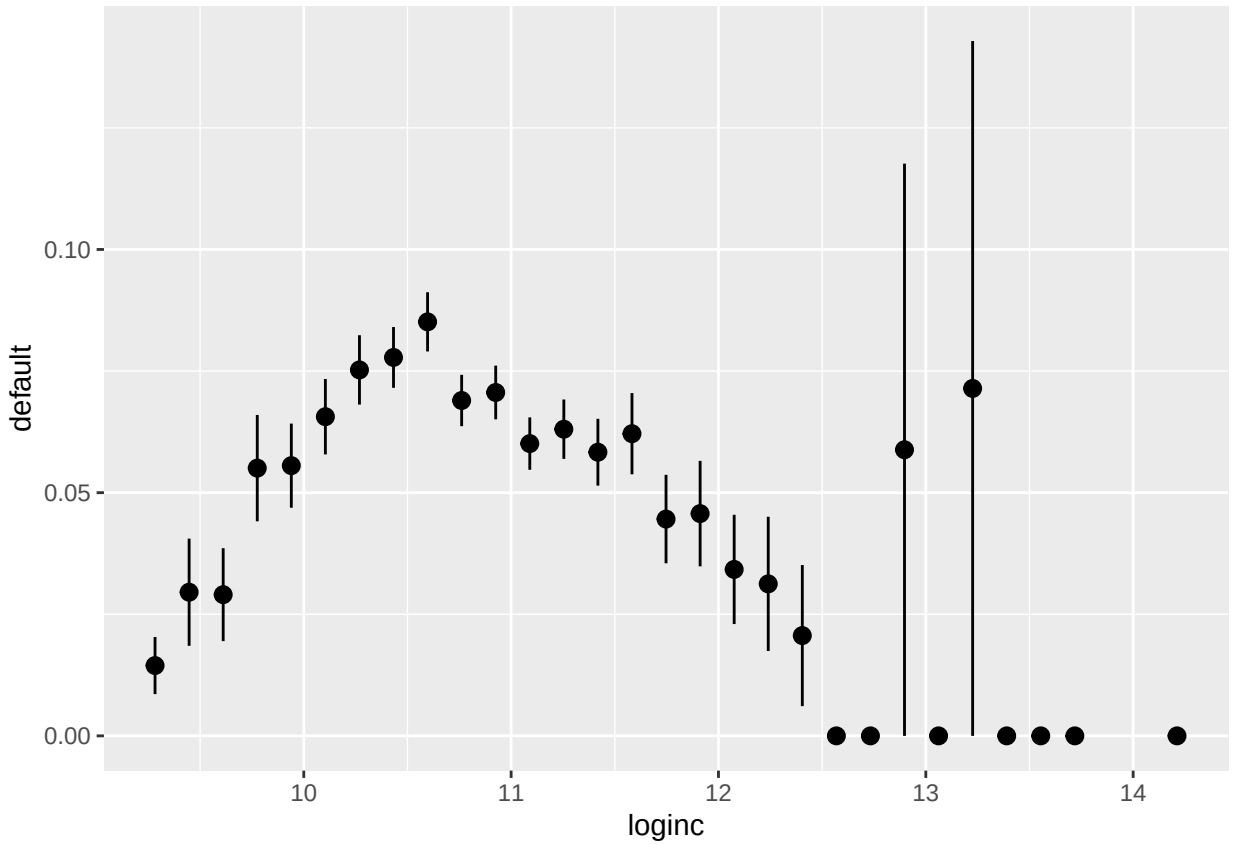
```
## Warning: Removed 55 rows containing missing values or values outside the scale range
## (`geom_segment()`).
```



A linear fit seems appropriate. We can create a binned scatter plot for income and balance.

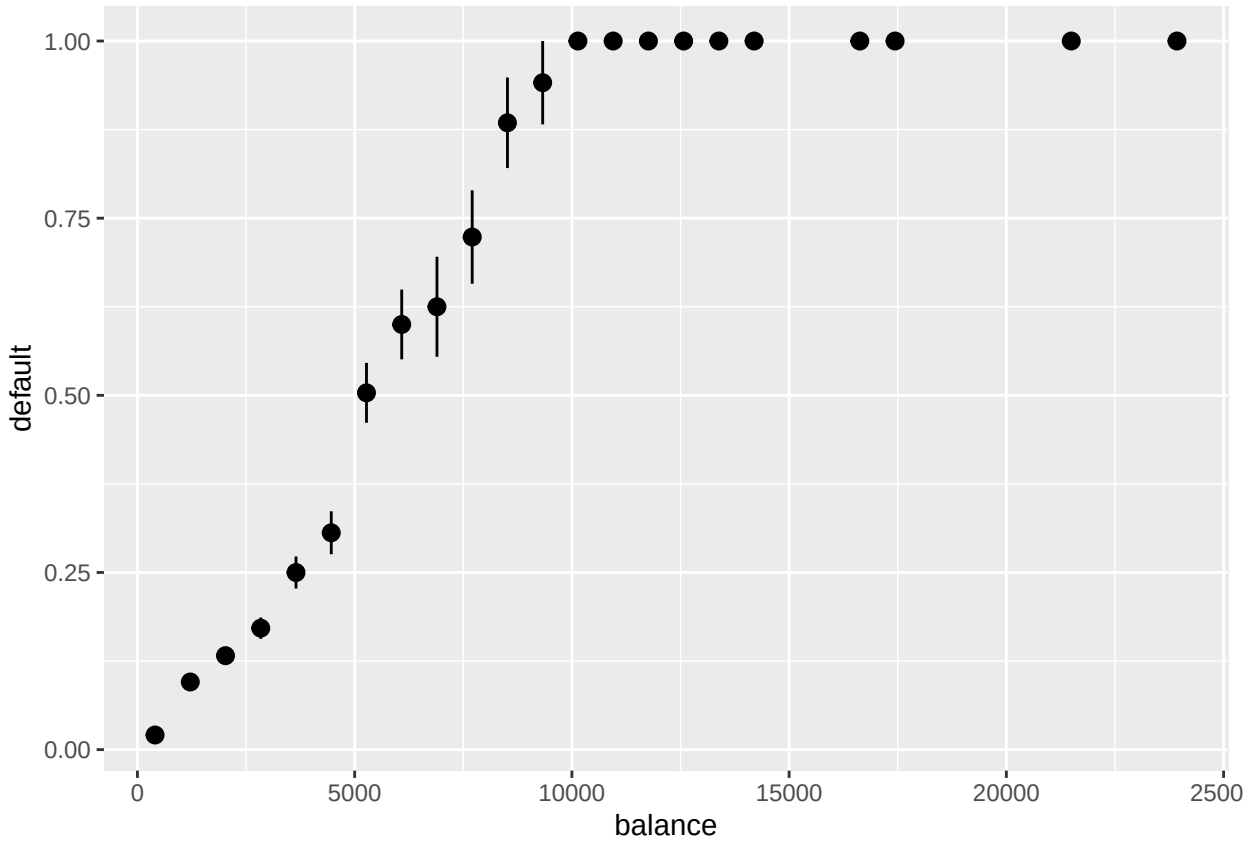
```
ggplot(df) + stat_summary_bin(aes(x=loginc,y=default), fun.data=mean_se)
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_segment()`).
```




```
ggplot(df) + stat_summary_bin(aes(x=balance,y=default), fun.data=mean_se)
```

```
## Warning: Removed 4 rows containing missing values or values outside the scale range
## (`geom_segment()`).
```



A quadratic seems sensible for both, but they may not work well due to the fact that this is a linear probability model. Let's check the model with and without using these quadratic terms.

```
def_lpm = lm(default ~ age + balance + loginc + gender + own + miss +
               factor(educ), data=df_train)
summary(def_lpm)
```

```
##
## Call:
## lm(formula = default ~ age + balance + loginc + gender + own +
##     miss + factor(educ), data = df_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.71719 -0.06986 -0.02186  0.00839  1.02139
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)    3.964e-02  4.270e-02   0.928 0.353268
## age            1.451e-03  2.084e-04   6.962 3.49e-12 ***
## balance        5.191e-05  1.715e-06  30.275  < 2e-16 ***
## loginc        -1.176e-02  4.392e-03  -2.677 0.007442 **
## gender         1.315e-02  3.697e-03   3.557 0.000376 ***
## own            4.655e-03  4.744e-03   0.981 0.326468
## miss           8.375e-02  3.363e-03  24.905  < 2e-16 ***
## factor(educ)2  1.395e-03  5.815e-03   0.240 0.810387
## factor(educ)3  1.280e-02  6.654e-03   1.924 0.054394 .
## factor(educ)4 -3.772e-02  7.815e-03  -4.827 1.40e-06 ***
```

```
## factor(educ)5 -3.216e-02  9.890e-03  -3.252 0.001148 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2174 on 14989 degrees of freedom
## Multiple R-squared:  0.2299, Adjusted R-squared:  0.2294
## F-statistic: 447.5 on 10 and 14989 DF,  p-value: < 2.2e-16
```

```
lpm_pred = predict(def_lpm,df_test)
rmse(lpm_pred,df_test[,'default'])
```

```
## [1] 0.20849
```

```
def_lpm2 = lm(default ~ age + poly(balance,2) + poly(loginc,2) + gender + own + miss +
                factor(educ), data=df_train)
summary(def_lpm2)
```

```
##
## Call:
## lm(formula = default ~ age + poly(balance, 2) + poly(loginc,
##     2) + gender + own + miss + factor(educ), data = df_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.72388 -0.06922 -0.02291  0.00944  1.02725
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -0.0311608  0.0122121  -2.552  0.01073 *
## age                0.0012443  0.0002139   5.818 6.07e-09 ***
## poly(balance, 2)1  8.5593659  0.2835053  30.191  < 2e-16 ***
## poly(balance, 2)2  0.1739579  0.2176801   0.799  0.42422
## poly(loginc, 2)1  -0.5890666  0.3541217  -1.663  0.09624 .
## poly(loginc, 2)2  -1.0061510  0.2370110  -4.245 2.20e-05 ***
## gender             0.0141760  0.0037029   3.828  0.00013 ***
## own                0.0045944  0.0047410   0.969  0.33252
## miss               0.0838670  0.0033611  24.952  < 2e-16 ***
## factor(educ)2     -0.0061966  0.0060809  -1.019  0.30821
## factor(educ)3      0.0026050  0.0070685   0.369  0.71248
## factor(educ)4     -0.0474651  0.0081443  -5.828 5.72e-09 ***
## factor(educ)5     -0.0395318  0.0100330  -3.940 8.18e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2173 on 14987 degrees of freedom
## Multiple R-squared:  0.2309, Adjusted R-squared:  0.2303
## F-statistic: 374.9 on 12 and 14987 DF,  p-value: < 2.2e-16
```

```
lpm2_pred = predict(def_lpm2,df_test)
rmse(lpm2_pred,df_test[,'default'])
```

```
## [1] 0.2083876
```

The model with quadratics leads to a negligible improvement in fit. To be parsimonious we will proceed with the model without quadratic terms.

Question 3

Let's check the performance of logit models using the same explanatory variables as above.

```
def_lgt = glm(default ~ age + balance + loginc + gender + own + miss +
                factor(educ), data=df_train, family='binomial')
summary(def_lgt)
```

```
##
## Call:
## glm(formula = default ~ age + balance + loginc + gender + own +
##     miss + factor(educ), family = "binomial", data = df_train)
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -2.289e+00  1.001e+00  -2.287 0.022221 *
## age            3.074e-02  4.576e-03   6.717 1.86e-11 ***
## balance        4.597e-04  2.564e-05  17.930  < 2e-16 ***
## loginc        -3.165e-01  1.036e-01  -3.056 0.002241 **
## gender         2.795e-01  8.156e-02   3.427 0.000609 ***
## own            1.003e-01  1.042e-01   0.963 0.335313
## miss           9.364e-01  4.900e-02  19.110  < 2e-16 ***
## factor(educ)2 -2.761e-02  1.168e-01  -0.236 0.813085
## factor(educ)3  2.222e-01  1.343e-01   1.655 0.098024 .
## factor(educ)4 -1.227e+00  1.943e-01  -6.312 2.75e-10 ***
## factor(educ)5 -1.301e+00  2.772e-01  -4.695 2.66e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7268.5  on 14999  degrees of freedom
## Residual deviance: 5129.6  on 14989  degrees of freedom
## AIC: 5151.6
##
## Number of Fisher Scoring iterations: 7
```

```
lgt_pred = predict(def_lgt,df_test,type='response')
rmse(lgt_pred,df_test[,'default'])
```

```
## [1] 0.2057566
```

```
def_lgt2 = glm(default ~ age + poly(balance,2) + poly(loginc,2) + gender + own + miss +
                 factor(educ), data=df_train, family='binomial')
summary(def_lgt2)
```

```
##
## Call:
## glm(formula = default ~ age + poly(balance, 2) + poly(loginc,
##     2) + gender + own + miss + factor(educ), family = "binomial",
##     data = df_train)
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -5.138851   0.273868 -18.764  < 2e-16 ***
## age                 0.026755   0.004682   5.714 1.10e-08 ***
## poly(balance, 2)1  72.871823   3.995854  18.237  < 2e-16 ***
```

```
## poly(balance, 2)2 -20.299304   2.771146  -7.325 2.38e-13 ***
## poly(loginc, 2)1  -19.525634   8.634512  -2.261 0.023738 *
## poly(loginc, 2)2  -26.784459   7.028432  -3.811 0.000138 ***
## gender              0.304324   0.081794   3.721 0.000199 ***
## own                 0.085752   0.103348   0.830 0.406683
## miss                0.938996   0.049149  19.105  < 2e-16 ***
## factor(educ)2      -0.160555   0.120745  -1.330 0.183617
## factor(educ)3       0.058149   0.139397   0.417 0.676570
## factor(educ)4      -1.355867   0.194613  -6.967 3.24e-12 ***
## factor(educ)5      -1.254709   0.266490  -4.708 2.50e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7268.5  on 14999  degrees of freedom
## Residual deviance: 5081.2  on 14987  degrees of freedom
## AIC: 5107.2
##
## Number of Fisher Scoring iterations: 7
```

```
lgt2_pred = predict(def_lgt2,df_test,type='response')
rmse(lgt2_pred,df_test[,'default'])
```

```
## [1] 0.2052287
```

The logit models do perform better than the LPM. The model with quadratic terms only performs marginally better so, again, we will use the simpler model.

Question 4

We first use model.matrix to get matrices that we can pass into glmnet, using our selected model's formula.

```
X_train = model.matrix(default ~ age + balance + loginc + gender + own + miss +
                         factor(educ) -1, data=df_train)
y_train = df_train$default
X_test = model.matrix(default ~ age + balance + loginc + gender + own + miss +
                        factor(educ) -1, data=df_test)
y_test = df_test$default
```

Let's now estimate the lasso and ridge models with cv.glmnet.

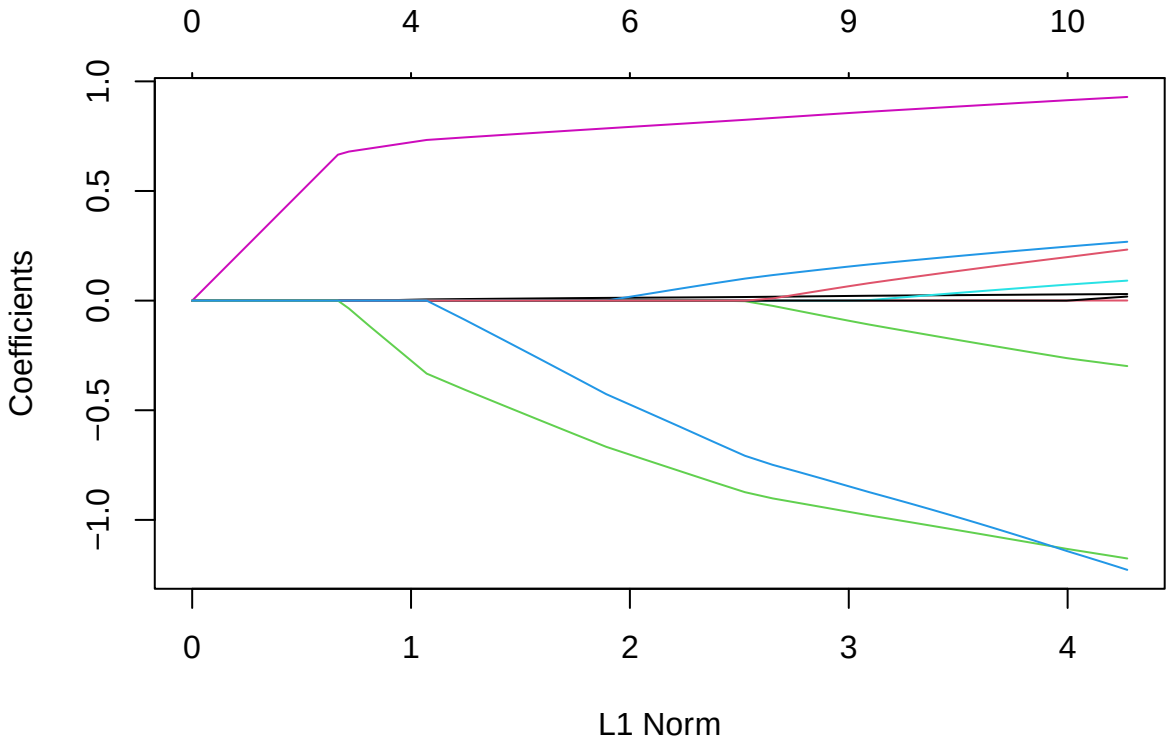
```
def_lasso = cv.glmnet(X_train,y_train,nfolds=10,alpha=1,family='binomial')
coef(def_lasso)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                            s1
## (Intercept)   -4.0903182491
## age            0.0113017124
## balance        0.0003990404
## loginc         .
## gender         .
## own            .
## miss           0.7761793883
## factor(educ)1  .
## factor(educ)2  .
## factor(educ)3  .
```

```
## factor(educ)4 -0.6069552805
## factor(educ)5 -0.3467844944
```

LASSO has aggressively shrunk a lot of the coefficients. Many have been set to zero, nearly including age and balance. It seems that the number of missed payments and the high education levels are strongly predictive of default, and they must be strongly correlated with the other variables. This seems to be supported by the pairs plot, apart from age and balance, which are not shrunk all the way to zero. Let's graph the regularization curve.

```
plot(glmnet(X_train,y_train,alpha=1,family='binomial'))
```



LASSO shrinks most coefficients to 0 even for larger values of lambda. At first glance, it does seem that LASSO is too aggressive. Let's compare to the results from a ridge penalty.

```
def_ridge = cv.glmnet(X_train,y_train,nfolds=10,alpha=0,lambda.min=0.000001,family='binomial')
coef(def_ridge)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                             s1
## (Intercept)   -3.2239627058
## age            0.0125989528
## balance        0.0003517127
## loginc        -0.1151953520
## gender         0.1392477132
## own            0.1535918908
## miss           0.7002098433
## factor(educ)1  0.1573018362
## factor(educ)2  0.1340170658
## factor(educ)3  0.2429207340
## factor(educ)4 -0.4489261587
## factor(educ)5 -0.4288765289
```

Ridge does not shrink the coefficients of the variables nearly as much as LASSO. Interestingly, age and

balance have still been shrunk quite a bit. Clearly, there must be a lot of co-linearity between these variables. Let's compare their RMSE.

```
lasso_pred = predict(def_lasso,X_test,type='response')
ridge_pred = predict(def_ridge,X_test,type='response')
rmse(lasso_pred,y_test)
```

```
## [1] 0.2080117
```

```
rmse(ridge_pred,y_test)
```

```
## [1] 0.2074996
```

The ridge model seems to fit marginally better, although neither model does better than the logistic regression model! Thus, even with a tiny amount of regularization strength (lambda.min is small), the model performs worse than if we had not used a penalty at all.

Question 5

We will just use table to get these confusion matrices.

```
lpm_class_50 = lpm_pred > 0.5
lpm_class_20 = lpm_pred > 0.2
logit_class_50 = lgt_pred > 0.5
logit_class_20 = lgt_pred > 0.2
lasso_class_50 = lasso_pred > 0.5
lasso_class_20 = lasso_pred > 0.2
ridge_class_50 = ridge_pred > 0.5
ridge_class_20 = ridge_pred > 0.2
table(lpm_class_50,y_test)
```

```
##             y_test
## lpm_class_50    0    1
##        FALSE 4693  259
##        TRUE     9   39
```

```
table(logit_class_50,y_test)
```

```
##               y_test
## logit_class_50    0    1
##          FALSE 4668  225
##          TRUE    34   73
```

```
table(lasso_class_50,y_test)
```

```
##               y_test
## lasso_class_50    0    1
##          FALSE 4673  233
##          TRUE    29   65
```

```
table(ridge_class_50,y_test)
```

```
##               y_test
## ridge_class_50    0    1
##          FALSE 4685  245
##          TRUE    17   53
```

```
table(lpm_class_20,y_test)
```

```
##             y_test
```

```
## lpm_class_20    0    1
##        FALSE 4371  140
##        TRUE   331  158
```

```
table(logit_class_20,y_test)
```

```
##               y_test
## logit_class_20    0    1
##          FALSE 4540  163
##          TRUE   162  135
```

```
table(lasso_class_20,y_test)
```

```
##               y_test
## lasso_class_20    0    1
##          FALSE 4571  178
##          TRUE   131  120
```

```
table(ridge_class_20,y_test)
```

```
##               y_test
## ridge_class_20    0    1
##          FALSE 4586  184
##          TRUE   116  114
```

The LPM certainly has the worst performance. Between un-penalized logit and the lasso/ridge models, logit has better overall performance with more entries on the diagonal, but lasso/ridge do a better job at classifying non-defaulters. Using the smaller threshold also seems to be a better choice here for overall classification performance, but ultimately this choice depends on the specific empirical use (i.e. loss function) of the model.

Question 6

We will use pROC for this.

```
lpm_roc = roc(df_test$default ~ lpm_pred)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
lgt_roc = roc(df_test$default ~ lgt_pred)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
lasso_roc = roc(df_test$default ~ lasso_pred)
```

```
## Setting levels: control = 0, case = 1
```

```
## Warning in roc.default(response, predictors[, 1], ...): Deprecated use a matrix
## as predictor. Unexpected results may be produced, please pass a numeric vector.
```

```
## Setting direction: controls < cases
```

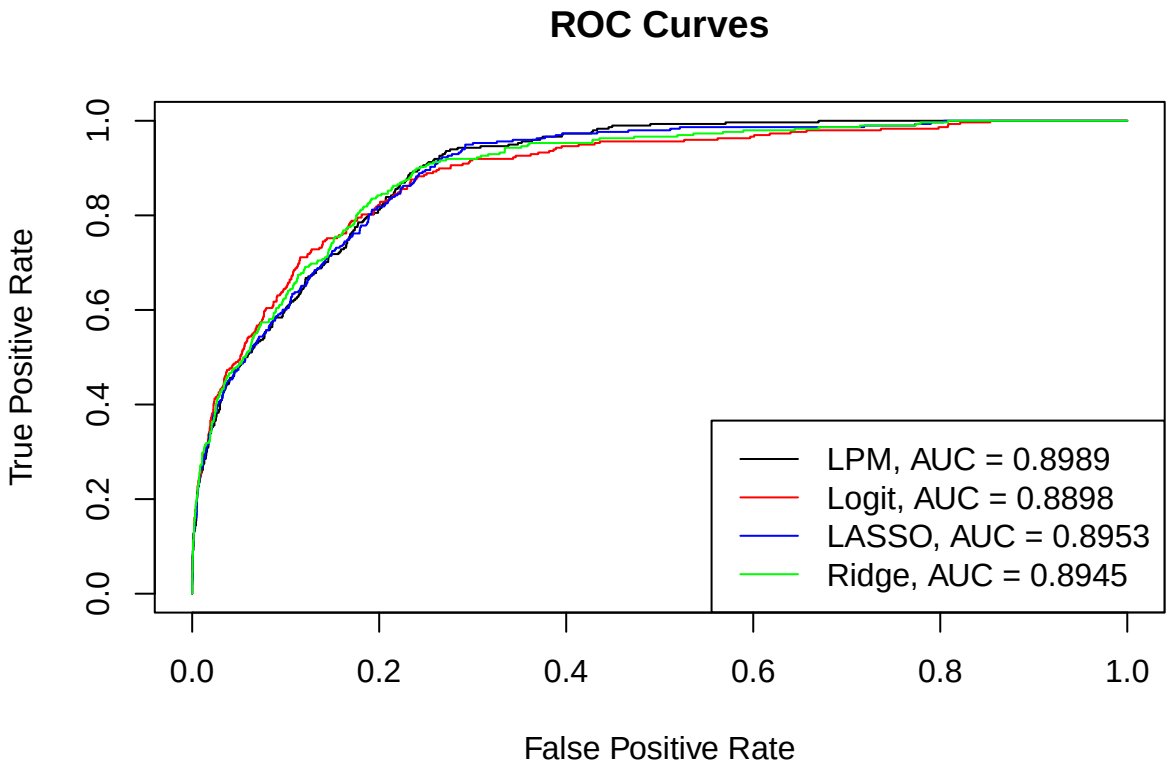
```
ridge_roc = roc(df_test$default ~ ridge_pred)
```

```
## Setting levels: control = 0, case = 1
```

```
## Warning in roc.default(response, predictors[, 1], ...): Deprecated use a matrix
## as predictor. Unexpected results may be produced, please pass a numeric vector.
```

```
## Setting direction: controls < cases
```

```r
plot(1-lpm_roc$specificities,lpm_roc$sensitivities,type='l',
     xlab="False Positive Rate",ylab="True Positive Rate",main="ROC Curves")
lines(1-lgt_roc$specificities,lgt_roc$sensitivities,col='red')
lines(1-lasso_roc$specificities,lasso_roc$sensitivities,col='blue')
lines(1-ridge_roc$specificities,ridge_roc$sensitivities,col='green')
legend("bottomright",legend = c(paste0('LPM, AUC = ',round(lpm_roc$auc,4)),
                                paste0('Logit, AUC = ',round(lgt_roc$auc,4)),
                                paste0('LASSO, AUC = ',round(lasso_roc$auc,4)),
                                paste0('Ridge, AUC = ',round(ridge_roc$auc,4))),
                                col=c('black','red','blue','green'),lty=1)
```



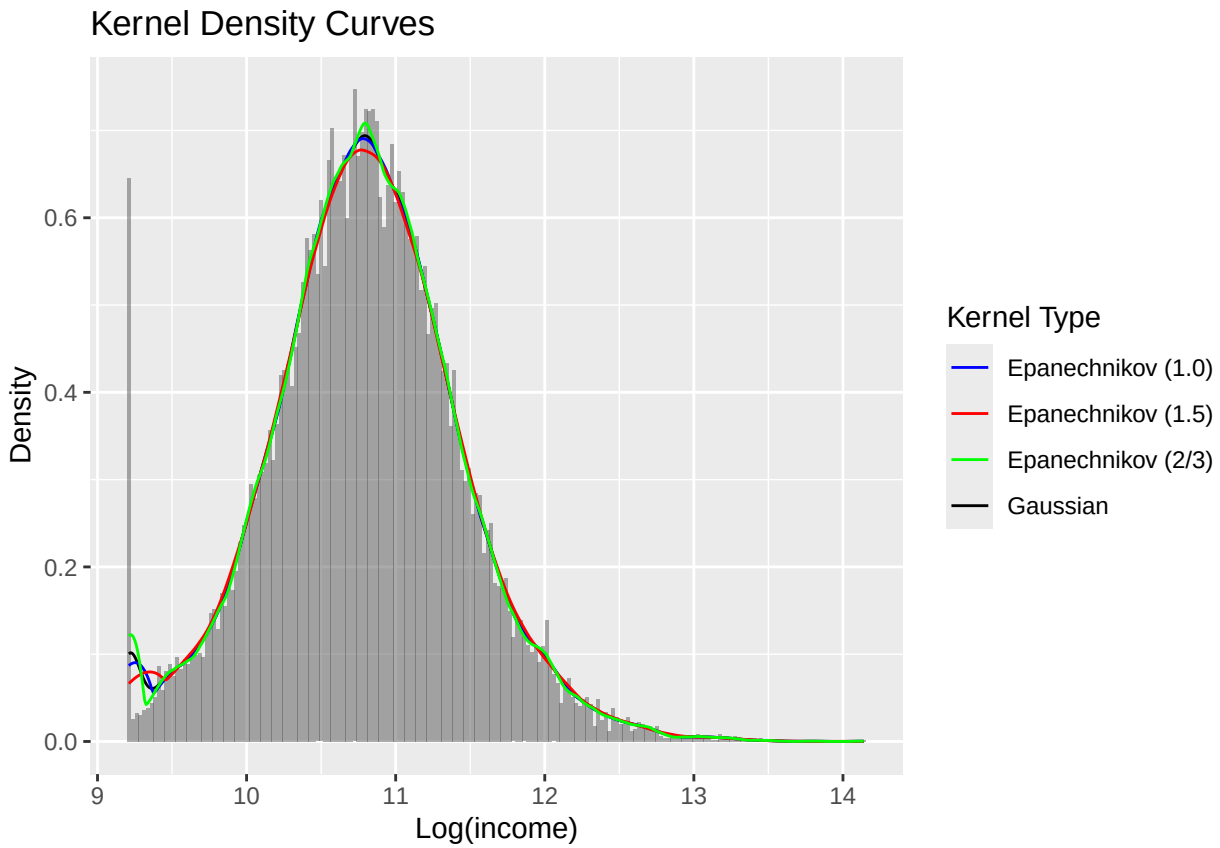
All models seem to perform similarly. Logit and Ridge do best for larger thresholds, and trade places with the LPM and LASSO for smaller thresholds. LPM actually seems to have the highest AUC, but all their AUC's are similar.

Question 7

We will use ggplot for this.

```r
ggplot(data=df,mapping=aes(x=loginc)) +
  geom_histogram(aes(y = after_stat(density)),bins=200, alpha = 0.5) +
  geom_density(kernel='gaussian',aes(color = "Gaussian"), key_glyph = "path") +
  geom_density(kernel='epanechnikov',aes(color = "Epanechnikov (1.0)"), key_glyph = "path") +
  geom_density(kernel='epanechnikov',adjust=1.5,aes(color = "Epanechnikov (1.5)"), key_glyph = "path") +
  geom_density(kernel='epanechnikov',adjust=2/3,aes(color = "Epanechnikov (2/3)"), key_glyph = "path") +
  scale_color_manual(name = "Kernel Type",
                     values = c("Gaussian" = "black",
                                "Epanechnikov (1.0)" = "blue",
                                "Epanechnikov (1.5)" = "red",
                                "Epanechnikov (2/3)" = "green")) +
```

```
labs(x='Log(income)',y='Density',title='Kernel Density Curves')
```



The Gaussian and Epanechnikov kernels seem to produce very similar density curves, which is to be expected since the empirical distribution seems to be quite smooth. Clearly, the DGP has a large number of individuals at the minimum income (10,000), which produces a large bin at the left tail of the distribution. Only the curve with the larger bandwidth level in red seems to not be affected by this. For all curves, there is a large right tail which follows the empirical distribution reasonably well. The density curve with the smaller bandwidth level in green is certainly too rough. The red curve is probably a good choice among these four density curves, although a bandwidth level between that of the red and green curves may be more optimal.