# week 5 homework

**Question 11.1**

Using the crime data set uscrime.txt from Questions 8.2, 9.1, and 10.1, build a regression model using:

1. Stepwise regression
2. Lasso
3. Elastic net

For Parts 2 and 3, remember to scale the data first – otherwise, the regression coefficients will be on different scales and the constraint won't have the desired effect.

For Parts 2 and 3, use the `glmnet` function in R.

Notes on R:

- For the elastic net model, what we called  in the videos, glmnet calls "alpha"; you can get a range of results by varying alpha from 1 (lasso) to 0 (ridge regression) [and, of course, other values of alpha in between].
- In a function call like glmnet(x,y,family="mgaussian",alpha=1) the predictors x need to be in R's matrix format, rather than data frame format. You can convert a data frame to a matrix using as.matrix – for example, x <- as.matrix(data[,1:n-1])
- Rather than specifying a value of T, glmnet returns models for a variety of values of T.

```r
library(stats)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```r
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-8
```

```r
df <- read.table("../week 5 data-summer/data 11.1/uscrime.txt", header=T)
head(df)
```

```
##      M So   Ed  Po1  Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq     Prob
## 1 15.1  1  9.1  5.8  5.6 0.510  95.0  33 30.1 0.108 4.1   3940 26.1 0.084602
## 2 14.3  0 11.3 10.3  9.5 0.583 101.2  13 10.2 0.096 3.6   5570 19.4 0.029599
## 3 14.2  1  8.9  4.5  4.4 0.533  96.9  18 21.9 0.094 3.3   3180 25.0 0.083401
## 4 13.6  0 12.1 14.9 14.1 0.577  99.4 157  8.0 0.102 3.9   6730 16.7 0.015801
## 5 14.1  0 12.1 10.9 10.1 0.591  98.5  18  3.0 0.091 2.0   5780 17.4 0.041399
## 6 12.1  0 11.0 11.8 11.5 0.547  96.4  25  4.4 0.084 2.9   6890 12.6 0.034201
##      Time Crime
## 1 26.2011   791
## 2 25.2999  1635
## 3 24.3006   578
## 4 29.9012  1969
## 5 21.2998  1234
## 6 20.9995   682
```

```r
set.seed(42)

# train test split
random_row <- sample(1:nrow(df), as.integer(0.9*nrow(df),replace=F))

traindata = df[random_row,]
testdata = df[-random_row,]

# setup k-fold cross-validation
train.control <- trainControl(method="cv", number = 10)
```

```r
# train stepwise model
step_model <- train(Crime~.,data=traindata, method="lmStepAIC",
                    trControl=train.control, trace=F)
# model accuracy
step_model$results
```

```
##   parameter     RMSE  Rsquared      MAE   RMSESD RsquaredSD    MAESD
## 1      none 278.8677 0.5780877 238.7969 96.74656  0.2066704 74.29544
```

```r
# model coefficients
step_model$finalModel
```

```
##
## Call:
## lm(formula = .outcome ~ M + Ed + Po1 + M.F + U1 + U2 + Ineq +
##     Prob, data = dat)
##
## Coefficients:
## (Intercept)            M           Ed          Po1          M.F           U1
##    -6584.18        87.77       194.24        97.73        23.79     -6952.26
##          U2         Ineq         Prob
##      205.84        61.73     -3895.11
```

```r
# model summary
summary(step_model$finalModel)
```

```
##
## Call:
## lm(formula = .outcome ~ M + Ed + Po1 + M.F + U1 + U2 + Ineq +
##     Prob, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -449.86 -125.91   18.22  128.34  477.86
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6584.18    1283.35  -5.130 1.26e-05 ***
## M              87.77      36.37   2.413  0.02154 *
## Ed            194.24      58.20   3.338  0.00210 **
## Po1            97.73      17.36   5.630 2.87e-06 ***
## M.F            23.79      14.53   1.637  0.11104
## U1          -6952.26    3718.62  -1.870  0.07044 .
## U2            205.84      79.28   2.596  0.01397 *
## Ineq           61.73      15.32   4.029  0.00031 ***
## Prob        -3895.11    1623.38  -2.399  0.02223 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 206.1 on 33 degrees of freedom
## Multiple R-squared:  0.7918, Adjusted R-squared:  0.7414
## F-statistic: 15.69 on 8 and 33 DF,  p-value: 3.104e-09
```

```r
# train full stepwise model
full_model <- lm(Crime~M+Ed+Po1+U2+M.F+U1+U2+Ineq+Prob,
                 data=traindata)

# Stepwise regression model- in both directions
```

```r
stepfinal_model <- stepAIC(full_model, direction="both", trace=FALSE, k=2)

# model accuracy
summary(stepfinal_model)
```

```
##
## Call:
## lm(formula = Crime ~ M + Ed + Po1 + U2 + M.F + U1 + U2 + Ineq +
##     Prob, data = traindata)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -449.86 -125.91   18.22  128.34  477.86
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6584.18    1283.35  -5.130 1.26e-05 ***
## M              87.77      36.37   2.413  0.02154 *
## Ed            194.24      58.20   3.338  0.00210 **
## Po1            97.73      17.36   5.630 2.87e-06 ***
## U2            205.84      79.28   2.596  0.01397 *
## M.F            23.79      14.53   1.637  0.11104
## U1          -6952.26    3718.62  -1.870  0.07044 .
## Ineq           61.73      15.32   4.029  0.00031 ***
## Prob        -3895.11    1623.38  -2.399  0.02223 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 206.1 on 33 degrees of freedom
## Multiple R-squared:  0.7918, Adjusted R-squared:  0.7414
## F-statistic: 15.69 on 8 and 33 DF,  p-value: 3.104e-09
```

```r
# train full model with less predictors
full_model2 <- lm(Crime~M+Ed+Po1+Ineq+Prob,
                  data=traindata)

# Stepwise regression model- in both directions
stepfinal_model2 <- stepAIC(full_model2, direction="both", trace=FALSE, k=2)

# model accuracy
summary(stepfinal_model2)
```

```
##
## Call:
## lm(formula = Crime ~ M + Ed + Po1 + Ineq + Prob, data = traindata)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -523.51 -109.02   11.75  142.06  499.66
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept) -4061.23     886.66  -4.580 5.37e-05 ***
## M              74.58      35.99   2.072  0.04546 *
## Ed            166.60      48.40   3.442  0.00148 **
## Po1           119.26      15.45   7.718 3.85e-09 ***
## Ineq           69.65      16.04   4.343  0.00011 ***
## Prob        -4043.93    1749.60  -2.311  0.02665 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 222.6 on 36 degrees of freedom
## Multiple R-squared:  0.7349, Adjusted R-squared:  0.6981
## F-statistic: 19.96 on 5 and 36 DF,  p-value: 1.718e-09
```

```r
#create the evaluation metrics function
eval_metrics = function(model, df, predictions, target){
    resids = df[,target] - predictions
    resids2 = resids**2
    N = length(predictions)
    r2 = as.character(round(summary(model)$r.squared, 2))
    adj_r2 = as.character(round(summary(model)$adj.r.squared, 2))
    print(sprintf("adjusted r-squared: %s", adj_r2))
    print(sprintf("rmse: %s", as.character(round(sqrt(sum(resids2)/N), 2))))
}
pred_train = predict(stepfinal_model2, newdata = (traindata))
pred_test = predict(stepfinal_model2, newdata = testdata)
```

```r
# model accuracy on train data
print("metrics for training data")
```

```
## [1] "metrics for training data"
```

```r
eval_metrics(stepfinal_model2, traindata, pred_train, target='Crime')
```

```
## [1] "adjusted r-squared: 0.7"
## [1] "rmse: 206.11"
```

```r
# model accuracy on test data
print("metrics for test data")
```

```
## [1] "metrics for test data"
```

```r
eval_metrics(stepfinal_model2, testdata, pred_test, target='Crime')
```

```
## [1] "adjusted r-squared: 0.7"
## [1] "rmse: 67.55"
```

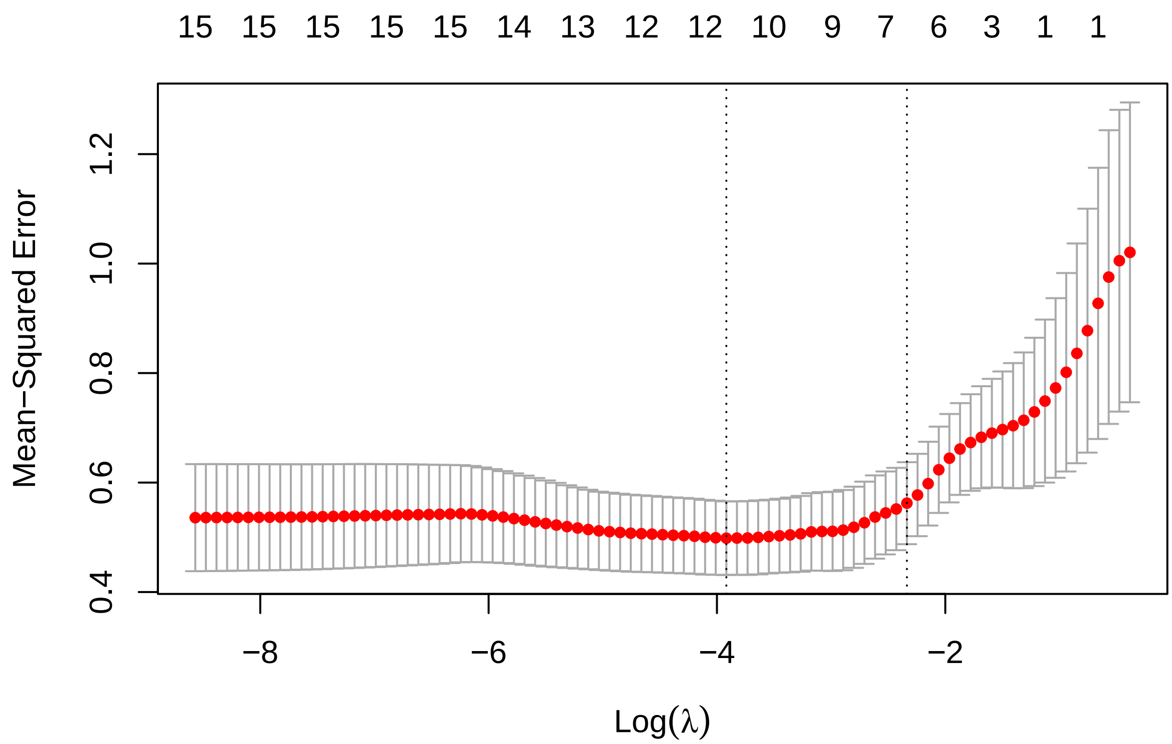
In part 1, we used cross-validation to select the most relevant predictors for our regression model. Then we used these relevant predictors to create a simple regression model. Finally we evaluate this simpler model on the training and test datasets.

We can see that the R-squares for both training and test dataset is the same while RSME is better in test dataset than training dataset

Part 2

```r
#scale data set
xtrain<-scale(as.matrix(traindata)[,-16], center=T, scale=T)
ytrain<-scale(as.matrix(traindata)[,16], center=T, scale=T)
xtest<-scale(as.matrix(testdata)[,-16], center=T, scale=T)
ytest<-scale(as.matrix(testdata)[,16], center=T, scale=T)
```

```r
# train lasso model
lasso_cv <- cv.glmnet(xtrain, ytrain, family="gaussian", alpha=1)

#plot lasso cv
plot(lasso_cv)
```




```r
coef(lasso_cv)
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)  1.254380e-16
## M            8.105801e-02
## So           .
## Ed           .
## Po1          6.731407e-01
## Po2          .
## LF           .
## M.F          1.285032e-01
```

```
## Pop           .
## NW            6.180575e-03
## U1            .
## U2            .
## Wealth        .
## Ineq          1.488520e-01
## Prob         -1.176118e-01
## Time          .
```

```
best_lambda <- lasso_cv$lambda.min
cat(best_lambda)
```

```
## 0.01988827
```

```
lasso_mod = glmnet(xtrain, ytrain, family="gaussian", alpha=1, lambda=best_lambda)
coef(lasso_mod)
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept) -4.442154e-17
## M            2.226763e-01
## So           2.939055e-02
## Ed           3.877426e-01
## Po1          7.569324e-01
## Po2          .
## LF           .
## M.F          1.430316e-01
## Pop         -1.725043e-02
## NW           2.206127e-02
## U1          -1.432053e-01
## U2           2.443221e-01
## Wealth       5.130037e-02
## Ineq         5.367606e-01
## Prob        -2.195127e-01
## Time         .
```

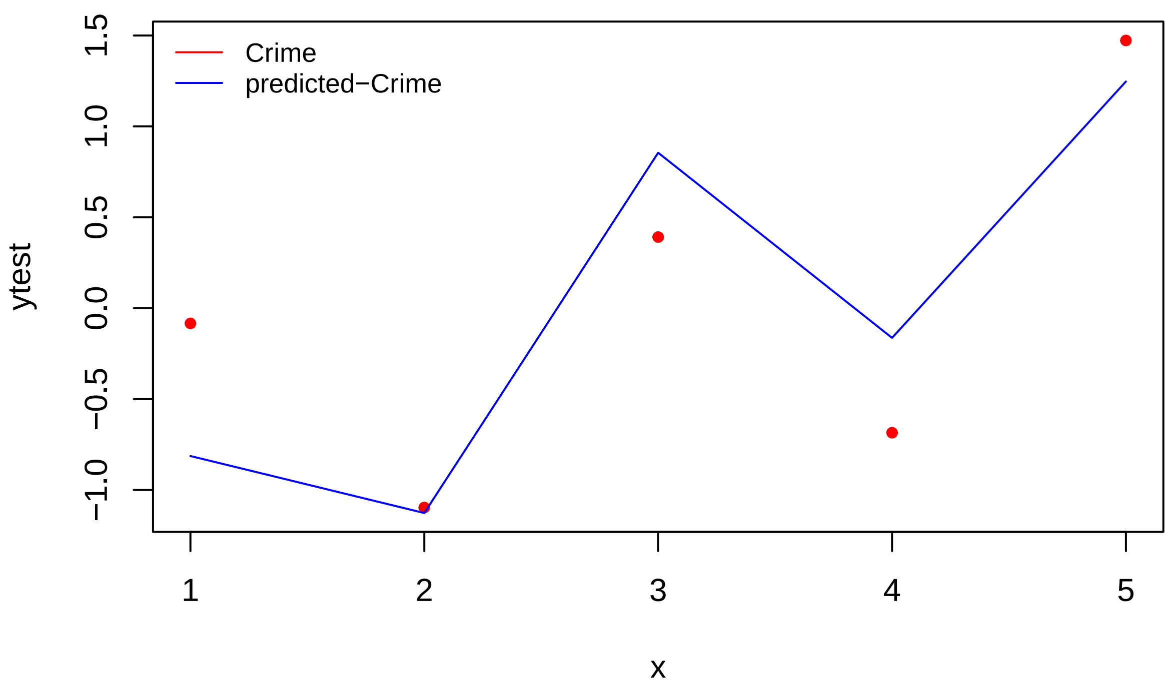
```
# Compute R^2 from true and predicted values
eval_results <- function(true, predicted, df) {
  SSE <- sum((predicted - true)^2)
  SST <- sum((true - mean(true))^2)
  R_square <- 1 - SSE / SST
  RMSE = sqrt(SSE/nrow(df))


  # Model performance metrics
data.frame(
  RMSE = RMSE,
  Rsquare = R_square)
}

# Prediction and evaluation on train data
yhat.train = predict(lasso_mod, xtrain)
eval_results(ytrain, yhat.train, traindata)
```

```
##          RMSE  Rsquare
## 1 0.4604497 0.782815
```

```r
# Prediction and evaluation on test data
yhat.test = predict(lasso_mod, xtest)
eval_results(ytest, yhat.test, testdata)
```

```
##          RMSE  Rsquare
## 1 0.4629885 0.732052
```

```r
x = 1:length(ytest)
plot(x, ytest, ylim=c(min(yhat.test), max(ytest)), pch=20, col="red")
lines(x, yhat.test, lwd="1", col="blue")
legend("topleft", legend=c("Crime", "predicted-Crime"), col=c("red", "blue"), lty=1, cex=0.8, lwd=1, bty
```




For part 2, we found the optimal lambda value to be 0.01988827. Using the optimal lambda value, we trained a lasso regression model and obtained metrics for both train and test datasets. We found that the RSME and R squares of the model were fairly similar for both datasets.

The Lasso model RMSE metric however cannot be compared with the RMSE of the Stepwise model, since the Stepwise model was trained on unscaled data. Therefore, moving forwards, we will only use the R-squared value to compare models.

Comparing the R-squared value of the Stepwise model (0.700) to that of the Lasso model (0.732), we can see that the R-squared value of the Lasso model is slightly better, as expected.

Part 3

```r
# Set training control
train_cont <- trainControl(method="repeatedcv",
                           number=10,
                           repeats=5,
                           search="random",
                           verboseIter=F)


# Train the model
elastic_reg <- train(Crime~.,data=as.matrix(scale(traindata)), method="glmnet",
                     preProcess=c("center", "scale"),
                     tuneLength=10, trControl=train_cont)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```r
# Best tuning parameter
elastic_reg$bestTune
```

```
##       alpha     lambda
## 9 0.8205803 0.02098472
```

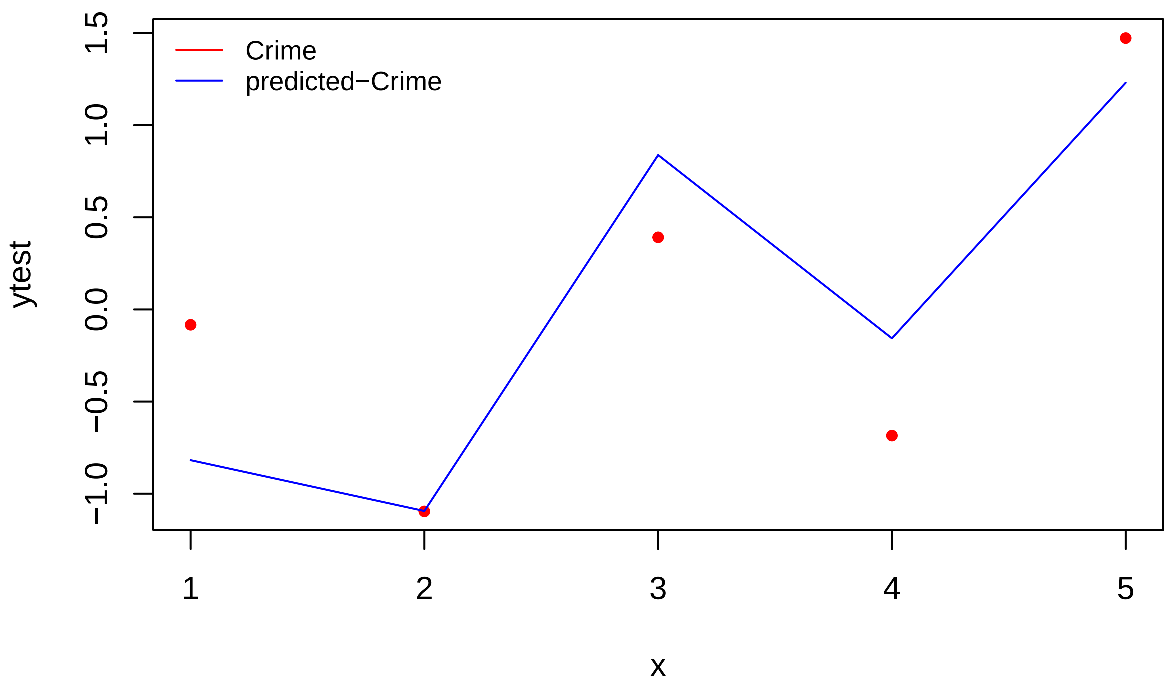
```r
# Make predictions on training set
pred_train <- predict(elastic_reg, xtrain)
eval_results(ytrain, pred_train, as.matrix(traindata))
```

```
##        RMSE   Rsquare
## 1 0.4575133 0.7855762
```

```r
# Make predictions on test set
pred_test <- predict(elastic_reg, xtest)
eval_results(ytest, pred_test, as.matrix(testdata))
```

```
##        RMSE   Rsquare
## 1 0.4641148 0.7307468
```

```r
x = 1:length(ytest)
plot(x, ytest, ylim=c(min(pred_test), max(ytest)), pch=20, col="red")
lines(x, pred_test, lwd="1", col="blue")
legend("topleft", legend=c("Crime", "predicted-Crime"), col=c("red", "blue"), lty=1,cex = 0.8, lwd=1, b
```




There is no set alpha for Elastic Net regression, but by using the parameter `tuneLength = 10`, we can generate 10 combinations of values for alpha and lambda to test. Our best alpha and lambda values are then given from `elastic_reg$bestTune` to be alpha=0.8205803 and lambda=0.02098472.

Comparing our models, we can see that our R-squared values for all three models are actually fairly similar, with the highest being the Lasso regression model (0.732), followed by the Elastic Net regression (0.731) and then the base Stepwise model (0.700).

**Question 12.1**

Describe a situation or problem from your job, everyday life, current events, etc., for which a design of experiments approach would be appropriate.

One application of a design of experiments approach is in process optimization/improvement in engineering. The engineers can identify key process variables and interactions to optimize them for desired outcomes.

**Question 12.2**

To determine the value of 10 different yes/no features to the market value of a house (large yard, solar roof, etc.), a real estate agent plans to survey 50 potential buyers, showing a fictitious house with different combinations of features. To reduce the survey size, the agent wants to show just 16 fictitious houses. Use R's `FrF2` function (in the `FrF2` package) to find a fractional factorial design for this experiment: what set of features should each of the 16 fictitious houses have? Note: the output of `FrF2` is "1" (include) or "-1" (don't include) for each feature.

```
library(FrF2)
```

```
## Loading required package: DoE.base
```

```
## Loading required package: grid

## Loading required package: conf.design

## Registered S3 method overwritten by 'partitions':
##   method            from
##   print.equivalence lava

## Registered S3 method overwritten by 'DoE.base':
##   method           from
##   factorize.factor conf.design

##
## Attaching package: 'DoE.base'

## The following objects are masked from 'package:stats':
##
##     aov, lm

## The following object is masked from 'package:graphics':
##
##     plot.design

## The following object is masked from 'package:base':
##
##     lengths
```

```r
house <- FrF2(nruns=16, nfactors=10, default.levels=c("Yes","No"))
data.frame(house)
```

```
##      A   B   C   D   E   F   G   H   J   K
## 1   No Yes Yes  No Yes Yes  No  No  No  No
## 2  Yes Yes Yes Yes  No  No  No  No Yes  No
## 3  Yes  No Yes Yes Yes  No Yes  No  No Yes
## 4   No Yes Yes Yes Yes Yes  No Yes Yes Yes
## 5   No Yes  No  No Yes  No Yes  No Yes Yes
## 6   No  No Yes Yes  No Yes Yes Yes  No  No
## 7  Yes  No  No Yes Yes Yes  No  No Yes  No
## 8  Yes  No Yes  No Yes  No Yes Yes Yes  No
## 9   No  No Yes  No  No Yes Yes  No Yes Yes
## 10 Yes Yes Yes  No  No  No  No Yes  No Yes
## 11  No Yes  No Yes Yes  No Yes Yes  No  No
## 12 Yes Yes  No  No  No Yes Yes Yes Yes  No
## 13 Yes  No  No  No Yes Yes  No Yes  No Yes
## 14  No  No  No Yes  No  No  No Yes Yes Yes
## 15 Yes Yes  No Yes  No Yes Yes  No  No Yes
## 16  No  No  No  No  No  No  No  No  No  No
```

Based on the implementation of the fractional factorial design, for 10 features to be shown for 16 houses, we can create a matrix that gives us an idea of which houses have which features (A-K)

**Question 13.1**

For each of the following distributions, give an example of data that you would expect to follow this distribution (besides the examples already discussed in class).

*Binomial:* A binomial distribution is used for observations where there can only be 2 outcomes, for example, if a person receives 20 emails per day and an email can be spam or not spam, and a binomial probability distribution can be used to determine the probability that a certain number of emails are spam.

*Geometric:* A geometric distribution is a discrete probability distribution that describes the number of failures you observe before a success in a series of independent trials. One example of this would be how many times you roll a die before you get the number '6'.

*Poisson:* A poisson distribution is used to model the probability of a certain number of events occurring independently within a fixed time interval. This could be the probability that a call center receives n calls per hour.

*Exponential:* Exponential probability distributions model the time until a certain event occurs, for example, the time between phone calls in a call center.

*Weibull:* The Weibull probability distribution is used to model the time-to-failure or the failure rate proportional to a power of time. For example, the amount of time a user spends on a webpage before they click away.