Homework 07

Brown University

DATA 1010

Fall 2020

In [1]:
```
using Plots, Distributions
using Pkg; Pkg.add("Statistics")
```

```
   Updating registry at `~/.julia/registries/General`
######################################################################## 100.0%
  Resolving package versions...
  Installed MbedTLS_jll ─ v2.16.8+1
No Changes to `~/Graduate/Data-1010/problem-sets/homework/hw07/Project.toml`
Updating `~/Graduate/Data-1010/problem-sets/homework/hw07/Manifest.toml`
  [c8ffd9c3] ↑ MbedTLS_jll v2.16.8+0 ⇒ v2.16.8+1
```

Problem 1

The p-value of a hypothesis test is defined to be the smallest significance level α with the property that we would have rejected the null hypothesis with the critical region chosen in the same way.

(a) Generate 10 observations from a normal distribution with mean 1 and standard deviation 1. These values are to be treated as unknown parameters μ and σ (imagine someone else generated the data and didn't tell you the two values 1 and 1).

Test the hypothesis that the mean of the distribution is positive (null hypothesis: $\mu = 0$, alternative hypothesis: $\mu > 0$). Be sure to use the most appropriate test, given that you know the data are normal but don't know the value of σ. Find the p-value for this hypothesis test. (You'll want to seed the random number generator, so that your results will be reproducible).

In [3]:
```
using Random; Random.seed!(123);
using Statistics
mean = 1
sd = 1
observations = rand(Normal(mean, sd), 10)
u = 0
t = (Statistics.mean(observations) - u) / (std(observations)/sqrt(length(observations)))
p_value = 1 - cdf(TDist(length(observations)-1),t)
p_value
```

Out[3]:
```
0.00019569888228898602
```

(b) Explain why the p-value is **not** the conditional probability that the null hypothesis is true given the observed data. Explain why you cannot even answer the question "what is the conditional probability that the null hypothesis is true given the observed data" in part (a).

(solution b) The conditional probability that the null hypothesis is true given the observed data can be represented by:

$$\mathrm{P}(H_0 \text{ is true} \mid Data) = \frac{\mathrm{P}(Data \mid H_0 \text{ is true})\,\mathrm{P}(H_0 \text{ is true})}{\mathrm{P}(Data \mid H_1 \text{ is true})\,\mathrm{P}(H_1 \text{ is true}) + \mathrm{P}(Data \mid H_0 \text{ is true})\,\mathrm{P}(H_0 \text{ is true})}$$

The p-value is $\mathrm{P}(Data \mid H_0 \text{ is true})$ which is the conditionaly probability that having oberserved data given the null hypothesis is true. We are not be able to get the result of this equation because the lack of information. We do not know $\mathrm{P}(H_1 \text{ is true})$ in the denominator of this equation.

p-value means the minimum α-value which would result in rejceting the null hypothesis.

(c) In part (a), the **effect size** (the difference between μ's actual value and its value under the null hypothesis) is reasonably large compared to the standard deviation. Would it have been possible to have a comparably small p-value even with a much small effect size? Illustrate by example, and conclude your solution with a comment on with a takeaway summary.

(solution c) From the example, we can see that we are not able to get a comparably small p-value with a much small effect size. In this example, we get none.

In [28]:
```julia
using Random; Random.seed!(123);
using Statistics
function simulate()
    mean = rand(Uniform(0,0.1), 1)[1]
    sd = 1
    observations = rand(Normal(mean, sd), 10)
    u = 0
    t = (Statistics.mean(observations) - u) / (std(observations)/sqrt(length(observations)))
    p_value = 1 - cdf(TDist(length(observations)-1),t)
    p_value
end
p_values = [simulate() for _ in 1:1000]
percent = sum(p_values.<p_value)/1000
```

Out[28]:
```
0.0
```

(d) **Challenge question; optional**. For each ordered pair (n, μ), where n is a positive integer and μ is a positive real number, we can consider the probability P of rejecting the null hypothesis that $\mu = 0$ given n observations from a normal distribution with mean μ (at the 95% confidence level, let's say).

Make a heatmap which shows a plot of P with respect to $\log n$ and μ.

(e) (not optional) The quantity P in part (d) has a standard name, which we learned in Data Gymansia. What is it? (Answer only; no explanation required.)

(solution e)

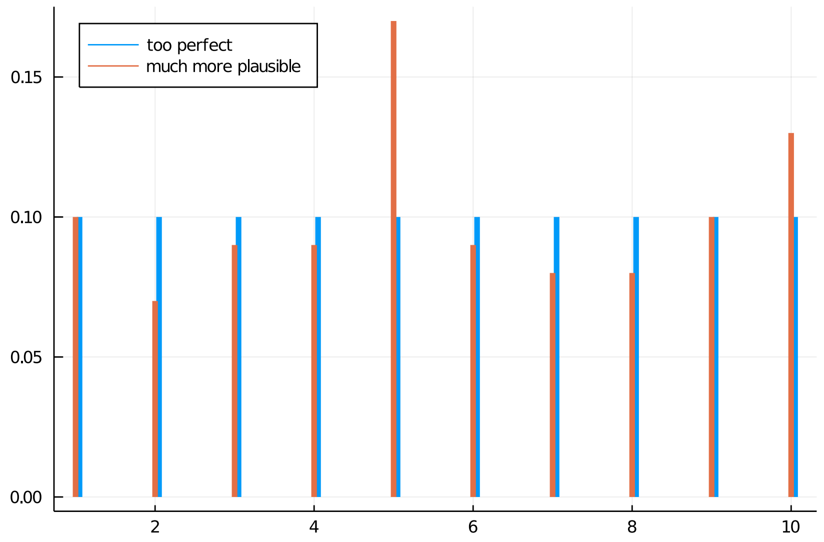
Power.

Problem 2

When we talked about pseudorandom number generators, we talked about how a random number generator which produces exactly equidistributed data is suspect:

In [19]:
```
using StatsBase
Random.seed!(123)
sticks((1:10) .+ 0.05, x -> 1/10, label = "too perfect", linewidth = 4, legend = :topleft)
tallies = countmap(rand(1:10, 100))
sticks!(1:10, x -> tallies[x]/100, label = "much more plausible", linewidth = 4)
```

Out[19]:



At the time, we had to appeal to common sense, but we're now in a position to test the hypothesis that a collection of data are drawn independently from a discrete uniform distribution.

Suppose the null hypothesis is that the data are drawn from $\{1, 2, \ldots, k\}$, with probability masses $p_1, \ldots, p_k$, respectively. Suppose that we have n observations, and for each j from 1 to k, suppose that X_j is the number of j's that we saw among our observations (these are the heights in the stick plot above). Consider the test statistic

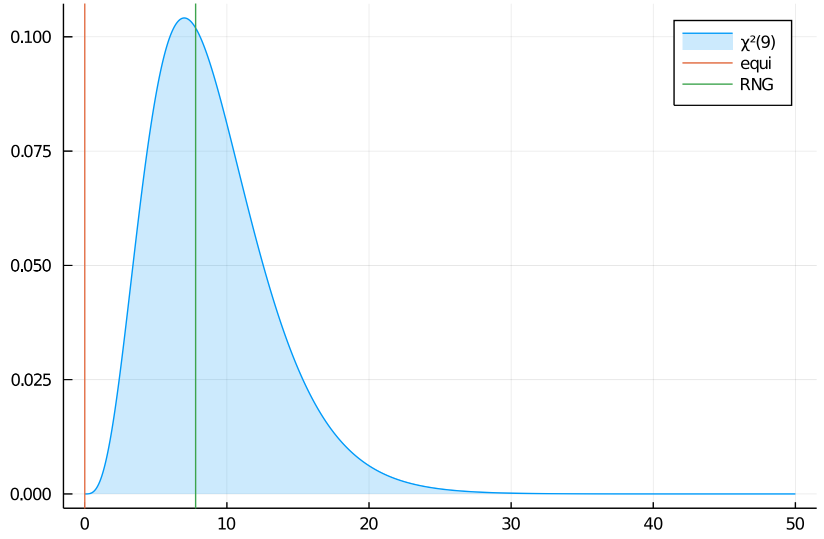
$$T = \frac{(X_1 - np_1)^2}{np_1} + \frac{(X_2 - np_2)^2}{np_2} + \cdots + \frac{(X_k - np_k)^2}{np_k}$$

.

Pearson's χ^2 test says that the distribution of T is approximately χ^2_{k-1}.

In [34]:
```
χ²_plot = plot(0:0.01:50, x -> pdf(Chisq(9), x),
               fillrange = 0, fillopacity = 0.2, label = "χ²(9)")
vline!([0], label="equi")
vline!([7.800000000000001], label="RNG")
```

Out[34]:



Draw two vertical lines showing the values of T for the two datasets plotted above (the uniform one, and the one generated by Julia's PRNG).

(a) Show that you reject the null hypothesis (at 95% confidence, let's say) in the case of the equidistributed data, but not in the case of the data produced by Julia's RNG. Determine the critical region in a *two-sided* way, particularly so that the critical region includes the 2.5% quantile and below, together with the 97.5% quantile and above.

In [30]:
```
using StatsBase
Random.seed!(123)
n = 100
eqi = 0 # This because x_i - n *p_i is 0 for every i since data is equidistributed
rng = sum([(tallies[i]-n*0.1)^2/(n*0.1) for i in 1:10])
c_low = quantile(Chisq(k-1), [0, 0.025])
c_above = quantile(Chisq(k-1), [0.975, 1])
println("Critical Region: ", c_low, c_above)
println("equidistributed data: ", eqi)
println("Julia's RNG: ", rng)
```

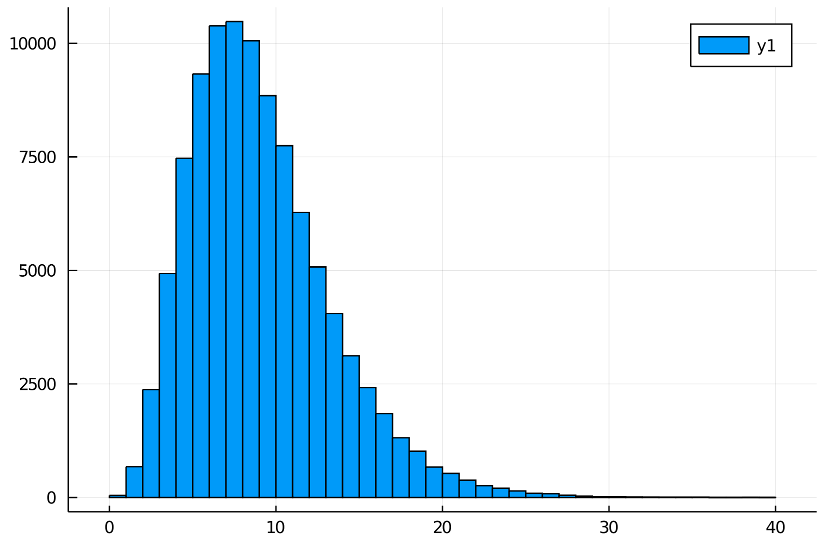
```
Critical Region: [0.0, 2.7003894999803575][19.022767798641635, Inf]
equidistributed data: 0
Julia's RNG: 7.800000000000001
```

(solution a) The null hypothesis is that the data are drawn from discrete random uniform distribution. Since the test statistics of equidistributed data is in the critical region and julia's RNG is not, we can reject null hypothesis in the former case but not in julia's RNG case.

(b) Provide empirical support for the Pearson χ^2 t-test in this instance by comparing the distribution plotted above with histogram showing the distribution of the value of T over 100,000 trials (where each trail involves drawing 100 independent random values from `1:10`).

In [113]:
```
function sample(n=100)
    observations = [rand(1:10) for _ in 1:n]
    cnt = counts(observations)
    T = sum([(i-n*0.1)^2/(n*0.1) for i in cnt])
    return T
end
res = [sample() for _ in 1:100000]
histogram(res, bins=70)
```

Out[113]:



The emperical plots shows that the distribution of T follows chi-squared distribution.

Problem 3

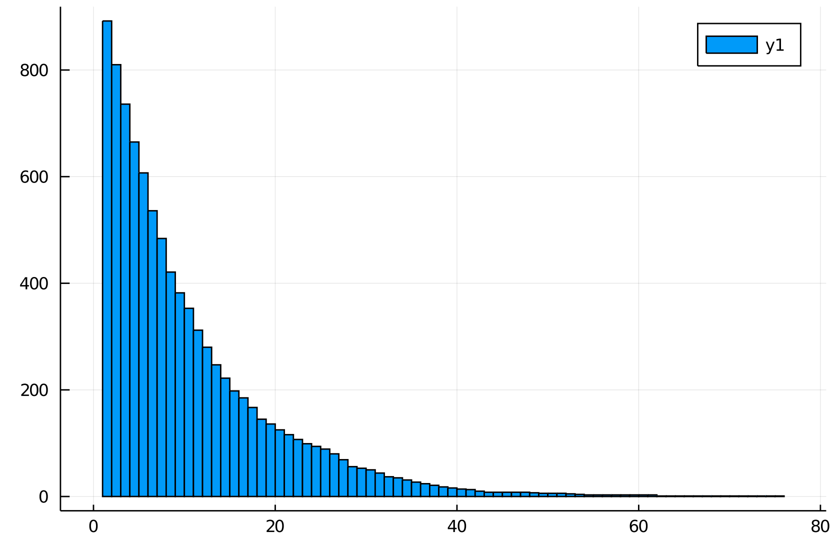
In this probability question, we'll use the permutation test to explore the role of baseline participation rates on elite representation.

Consider a population of size 1000 which is divided into two groups, one of size 900 and the other of size 100. Suppose that each individual has a skill level which is represented by a random variable distributed uniformly on $[0, 1]$ (with these random variables independent). Let's define a random variable N so that the N most skilled individuals in the overall population are from the larger subgroup.

(a) Estimate the distribution of N by simulation.

In [5]:
```
function simulation(n=1000)
    result = []
    for i in 1:n
        skills = rand(Uniform(0,1), 1000)
        maxes = partialsortperm(skills, 1:1000)
        for i in 1:length(maxes)
            if maxes[i] <= 900
                push!(result, i)
            else
                break
            end
        end
    end
    return result
end
res = simulation()
histogram(res)
```

Out[5]:



(b) Find the exact value of $\mathbb{P}(N > 10)$.

(solution b)

$$\mathbb{P}(N > 10) = 1 - \sum_{j=1}^{10} \prod_{i=1}^{j} \frac{900 - i + 1}{1000 - i + 1} * \frac{100}{1000 - i} - 100/1000$$

The answer to the above formula is calculated using julia, and it is 0.312.

In [24]:
```
p = zeros(10)
sum_p = 0
t = 1
for i in 1:10
    t *= (900-i+1)/(1000+1-i)
    p[i] = t
end
for i in 1:10
    p[i] = p[i] * 100/(1000-i)
end
sum_p = sum(p) + 100/1000
1 - sum_p
```

```
0.5881154887223382
```

Out[24]:
```
0.3118845112776618
```

Problem 4

Devise a hypothesis test which uses the DKW inequality to test the null hypothesis that a given collection of observations comes from a standard normal distribution. Be sure to identify your test statistic, critical region, etc. You should do this symbolically, but feel to show a numerical example if you feel that it would help clarify your explanation.

(solution)

Let $\hat{\mu}$ be the mean of observations and $\hat{\theta}$ be the standard deviation of the observations. The standard normal has $\mu = 0$ and $\theta = 1$. The DKW says that

$$\mathbb{P}\left(\max_x |F(x) - \hat{F}_n(x)| \leq \epsilon\right) \geq 1 - 2e^{-2n\epsilon^2}$$

Let $\alpha = 2e^{-2n\epsilon^2}$ and $\epsilon_n = \sqrt{\frac{1}{2n}\log(2/\alpha)}$, and the DKW gives a confidence band of ϵ_n with significance level α.

The test statistic is $\max_x |F(x) - \hat{F}_n(x)|$ and the critical region is $(-inf, -\epsilon_n)U(\epsilon_n, inf)$.

From the K-S test which is nonparametric, the test statistics follows Kolmogorov distribution. We could use certain alpha value (like 0.05) to find the critical value, or use pvalue() function to find the p value. If our test statustics falls into the critical region, or p value is less than alpha. Then we are able to decide whether we should reject the null hypothesis.

Problem 5

In class on 10-26, we talked about a simple example (predicting shoe size from height in cm and height in inches) demonstrating how a linear model can very badly overfit.

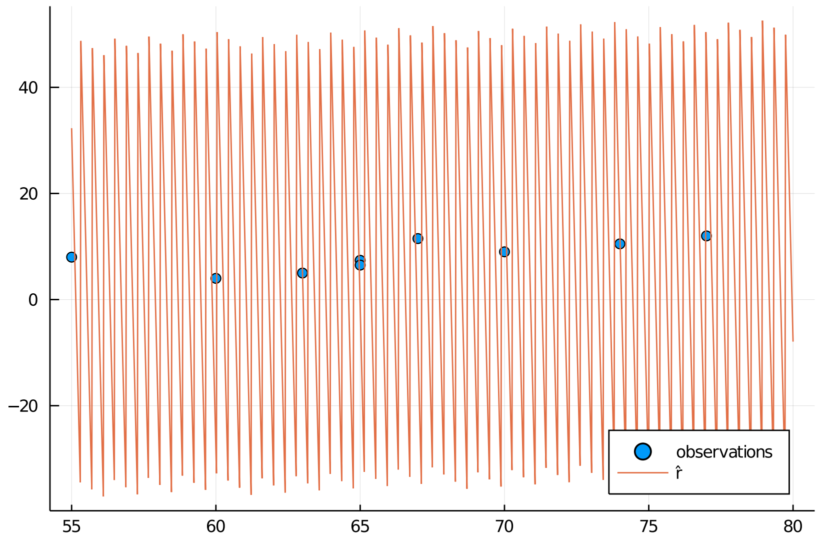
In class we visualized the overfitting by conceiving of the prediction function as a function of a single variable, namely the height in inches, with the other feature being derived from that one by converting and then rounding.

Visualize the prediction function by instead drawing a *heatmap* of the prediction for a given pair of height values. Plot a subset of the plane consisting of the *feasible* region (wherein the two height measures are compatible, taking rounding into account). Use your diagram to explain what the linear model is doing to overfit the data.

Note that you might need to change the centimeter column from nearest-millimeter rounding to nearest-centimeter rounding. Otherwise the detail is going to be hard to see in the diagram.

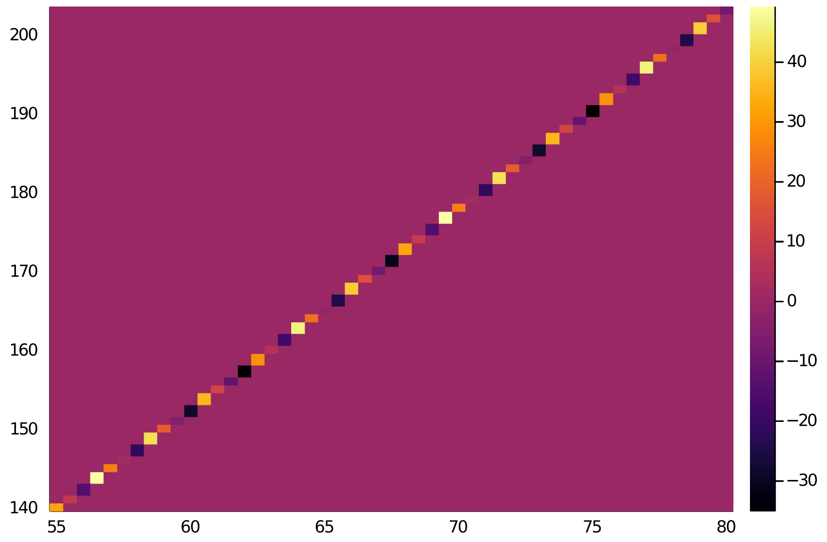
In [88]:
```
s = [5, 4, 12, 8, 9, 7.4, 6.5, 11.5, 10.5]
h = [63, 60, 77, 55, 70, 65, 65, 67, 74]
h_cm = round.(2.54 * h, digits=1)
β̂ = [ones(9) h h_cm] \ s
scatter(h, s, label = "observations", legend = :bottomright)
plot!(55:0.02:80, x -> [1, x, round(2.54 * x, digits=0)]' * β̂, label = "r̂")
```

Out[88]:



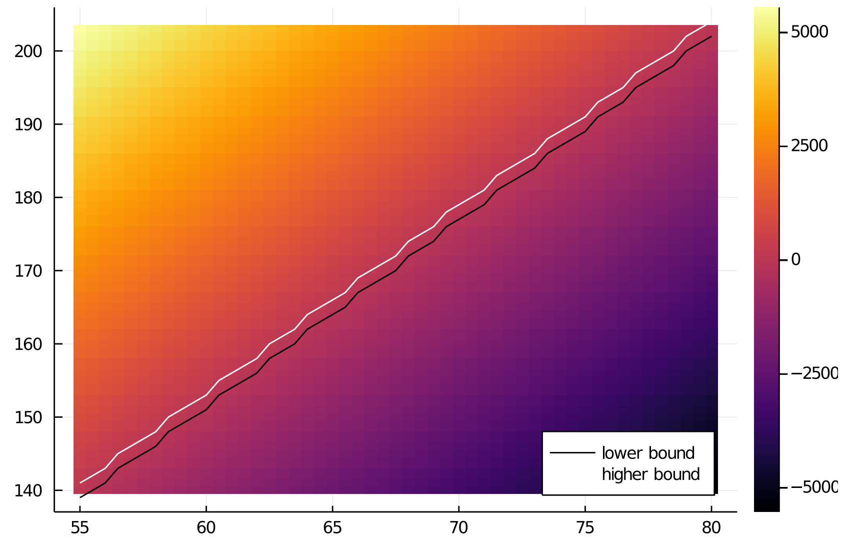
In [114]:
```
using LinearAlgebra
predict(x, y) = [1, x, y]'  * β̂
step = 0.5
predictions = [predict(x, round(2.54*x, digits=0)) for x in 55:step:80]
xs = 55:step:80
ys = [round(2.54 * x, digits=0) for x in xs]
heatmap(xs, ys, diagm(predictions))
```

Out[114]:



In [115]:
```
heatmap(xs, ys, (x,y)->predict(x,y))
plot!(55:step:80, x->round(2.54 * x, digits=0)-1, label = "lower bound", color = "black", legend = :bottomright)
plot!(55:step:80, x->round(2.54 * x, digits=0)+1, label = "higher bound", color = "white", legend = :bottomright)
```

Out[115]:



Since the prediction function is the input times Beta, which is calculated directly from the inverse of show size and the training data, the prediction function will fit every training data point accurately. However, this will cause overfitting as we might take the noise into account in Beta. Furthermore, because the feature height_cm is lineary proportional to height, which mean they are highly correlated, the prediction function will have a extremely infeasible output if these two features do not match in the input. From the smalle rectangles, we can still see some black boxes and yellow boxes which means the shoe size are not correct empiracally. That means the prediction function is overfitting because the small bias lead to high variance in the heatmap.

Problem 6

(a) Using the Boston Housing Dataset (https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html), perform a QDA on features RM and DIS, with labels being whether median value of owner-occupied homes in exceed 25,000 dollars. You should not use all your data points when you are building your model; test the accuracy of your model with the unused datapoints. Randomly assign 80% of the observations to use for training and the remaining 20% for testing.

You should do this directly, by computing the relevant μ and Σ estimates and computing the multivariate Gaussian densities. Do not look for a library to do it for you (that would probably be more work anyway).

(b) Build a QDA model using all of the columns provided. Does this new model yield better accuracy?

You can load this dataset from the MLDatasets package (https://github.com/JuliaML/MLDatasets.jl) in Julia.a

In [3]:
```
using MLDatasets
```

In [46]:
```
?BostonHousing
```

```
search: BostonHousing
```

Out[46]: Boston Housing Dataset.

Sources: (a) Origin: This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University. (b) Creator: Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978. (c) Date: July 7, 1993

Number of Instances: 506

Number of Attributes: 13 continuous attributes (including target attribute "MEDV"), 1 binary-valued attribute.

Features: 1. CRIM per capita crime rate by town 2. ZN proportion of residential land zoned for lots over 25,000 sq.ft. 3. INDUS proportion of non-retail business acres per town 4. CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise) 5. NOX nitric oxides concentration (parts per 10 million) 6. RM average number of rooms per dwelling 7. AGE proportion of owner-occupied units built prior to 1940 8. DIS weighted distances to five Boston employment centres 9. RAD index of accessibility to radial highways 10. TAX full-value property-tax rate per 10,000 dollars 11. PTRATIO pupil-teacher ratio by town 12. B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town 13. LSTAT % lower status of the population

Target: 14. MEDV Median value of owner-occupied homes in 1000's of dollars

Note: Variable #14 seems to be censored at 50.00 (corresponding to a median price of 50,000); Censoring is suggested by the fact that the highest median price of exactly 50,000 is reported in 16 cases, while 15 cases have prices between 40,000 and 50,000, with prices rounded to the nearest hundred. Harrison and Rubinfeld do not mention any censoring.

The data file stored in this repo is a copy of the This is a copy of UCI ML housing dataset. https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (https://archive.ics.uci.edu/ml/machine-learning-databases/housing/)

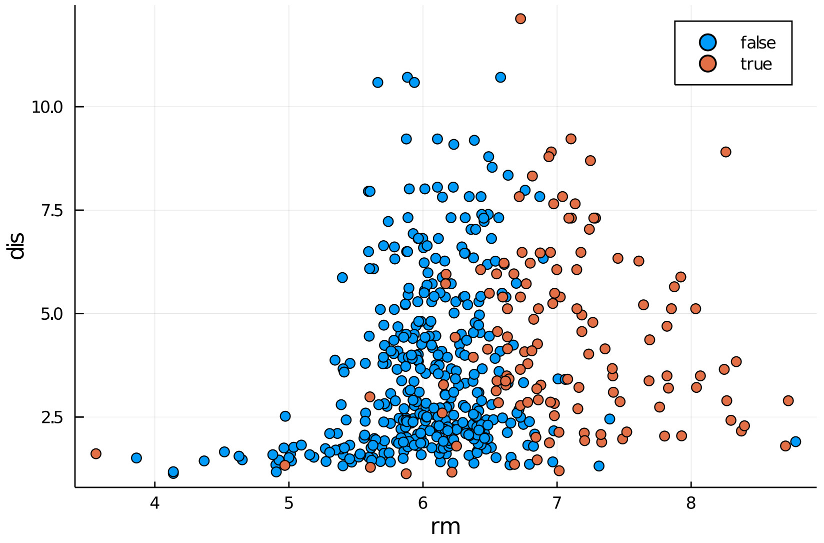
Interface

- `BostonHousing.features` (@ref)
- `BostonHousing.targets` (@ref)
- `feature_names` (@ref)

In [4]:
```
X, y = BostonHousing.features(), BostonHousing.targets()
# MLDatasets returns data which are transposed relative to the usual
# "observations by features" format. So we permute the dimensions:
X = permutedims(X, (2, 1))
y = y[:]; # same for y as well
```

In [5]:
```
using Plots
scatter(X[:, 6], X[:, 8], group = y .> 25, xlabel = "rm", ylabel = "dis")
```

Out[5]:



In [81]:
```
using Random
using Statistics, Distributions
Random.seed!(42)
indices = shuffle([i for i in 1:506])
train_size = 404
train_indices = indices[1:train_size]
test_indices = indices[train_size+1:506]
X_part = X[:,[6,8]]
y_label = y .> 25
X_train, X_test = X_part[train_indices,:], X_part[test_indices,:]
y_train, y_test = y_label[train_indices,:], y_label[test_indices,:]
cnt = countmap(y_train)
proportion = (cnt[0]/train_size, cnt[1]/train_size)
```

Out[81]:
```
(0.7797029702970297, 0.2202970297029703)
```

In [82]:
```
using Statistics, Distributions
mu_above = Statistics.mean([[X_train[i,1], X_train[i,2]] for i = 1:train_size if y_train[i]])
mu_below = Statistics.mean([[X_train[i,1], X_train[i,2]] for i = 1:train_size if !y_train[i]])
std_above = cov([[X_train[i,1], X_train[i,2]] for i = 1:train_size if y_train[i]])
std_below = cov([[X_train[i,1], X_train[i,2]] for i = 1:train_size if !y_train[i]])
ns_above = MvNormal(mu_above, std_above)
ns_below = MvNormal(mu_below, std_below)
dists = [ns_below, ns_above]
predict(x) = argmax([proportion[i]*pdf(dists[i],x) for i in 1:2])
test_x = [[X_test[i,1], X_test[i,2]] for i = 1:506-train_size]
predict_output = [predict(i) - 1 for i in test_x]
y_true = convert.(Int, y_label)
accuracy = count([predict_output[i] == y_true[i] for i = 1:length(predict_output)]) / length(predict_output)
```

Out[82]:
```
0.7450980392156863
```

(solution a) Using only RM and DIS, the QDA gives the prediction model with 0.745 accuracy

In [83]:
```
using Random
using Statistics, Distributions
Random.seed!(42)
indices = shuffle([i for i in 1:506])
train_size = 404
train_indices = indices[1:train_size]
test_indices = indices[train_size+1:506]
y_label = y .> 25
X_train, X_test = X[train_indices,:], X[test_indices,:]
y_train, y_test = y_label[train_indices,:], y_label[test_indices,:]
cnt = countmap(y_train)
proportion = (cnt[0]/train_size, cnt[1]/train_size)
```

Out[83]:
```
(0.7797029702970297, 0.22029702970297030703)
```

In [84]:
```
using Statistics, Distributions
mu_above = Statistics.mean([[X_train[i,j] for j in 1:13] for i = 1:train_size if y_train[i]])
mu_below = Statistics.mean([[X_train[i,j] for j in 1:13] for i = 1:train_size if !y_train[i]])
std_above = cov([[X_train[i,j] for j in 1:13] for i = 1:train_size if y_train[i]])
std_below = cov([[X_train[i,j] for j in 1:13] for i = 1:train_size if !y_train[i]])
ns_above = MvNormal(mu_above, std_above)
ns_below = MvNormal(mu_below, std_below)
dists = [ns_below, ns_above]
predict(x) = argmax([proportion[i]*pdf(dists[i],x) for i in 1:2])
test_x = [[X_test[i,j] for j in 1:13] for i = 1:506-train_size]
predict_output = [predict(i) - 1 for i in test_x]
y_true = convert.(Int, y_label)
accuracy = count([predict_output[i] == y_true[i] for i = 1:length(predict_output)]) / length(predict_output)
```

Out[84]:
```
0.5686274509803921
```

(solution b) Using all the features, the QDQ gives the prediction model with 0.569 accuracy. The result is not better.