Exercise 2

a) Generate a simulated data set as follows:

In [1]:

```
srand(1)
x = randn(100)
y = x - 2x.^2 + randn(100)
;
```

In this data set, what is n and what is p? Write out the model used to generate the data in equation form

In this regression model $n = 100$ which is the length of the variables x and y. It seems reasonable to choose $p = 2$, since y is generated with from the two predictors x and x^2, plus a random error.

(c) Do 5-fold cross-validation to fit polynomial models using least squares, we compute the generalization error

Step 1: Create the folds

We will create 5 random equally sized subsamples, each one will be used as the test set.

In [2]:

```
n = 100
k = 5
s = convert(Integer, n / k)
seq = randperm(n)
fold = [seq[((i-1)*s + 1):(i*s)] for i in 1:k]
```

Out[2]:

```
5-element Array{Array{Int64,1},1}:
 [45, 92, 34, 86, 76, 30, 94, 88, 97, 79, 66, 100, 21, 80, 4, 72, 59, 42, 6
9, 31]
 [14, 85, 5, 19, 74, 6, 48, 89, 60, 75, 87, 83, 81, 15, 71, 23, 27, 40, 41,
 29]
 [82, 49, 22, 44, 17, 9, 73, 13, 28, 56, 11, 1, 7, 36, 20, 33, 47, 35, 68, 6
3]
 [84, 95, 99, 98, 77, 61, 24, 38, 53, 16, 62, 52, 43, 3, 54, 91, 25, 39, 26,
32]
 [65, 93, 2, 70, 67, 46, 64, 90, 57, 12, 78, 10, 37, 18, 51, 55, 96, 8, 50,
 58]
```

Step 2: fit models with cross-validation

In [3]:

```
# design matrix with polynomial entries
X = [ones(length(y)) x x.^2 x.^3 x.^4]
# store space for generalization error
gen_error = Array{Float64}(4, k)
train_error = Array{Float64}(4, k)
βhat_all = [[] for i in 1:k]
# k-fold cross validation
for i in 1:k # outer loop is the cross-validation loop
    # test and train indices
    test_idx = fold[i]
    train_idx = [l for l in 1:n if l ∉ fold[i]]
    # y data
    y_train = y[train_idx]
    y_test = y[test_idx]
    # fit model
    for deg in 1:4 # inner loop fits different polynomial models
        # X data
        X_train = X[train_idx, 1:(1 + deg)]
        X_test = X[test_idx, 1:(1 + deg)]
        # regression
        βhat_train = Symmetric(X_train' * X_train) \ (X_train' * y_train)
        push!(βhat_all[i], βhat_train)
        yhat_test = X_test * βhat_train
        yhat_train = X_train * βhat_train
        gen_error[deg, i] = mean((yhat_test - y_test).^2) # average cross validation error
        train_error[deg, i] = mean((yhat_train - y_train).^2) # average cross validation e
    end
end
```

We now print the mean squared error (MSE) per degree of fitted polynomial. We see that on average higher degrees performed better than degree one. Unfortunately, with my data, not even cross validation can detect that the true degree is two, since higher degrees perform as well. I was expecting cross validation to reduce overfitting, but it did not. However, we do see that the performance is much worst in the test set that in the train set.

In [4]:

```
mse_train = mapslices(mean, train_error, 2)
mse_test = mapslices(mean, gen_error, 2)
;
```

In [5]:

```
for deg in 1:4
    @printf("Degree: %i, MSE Train: %0.2f, MSE Test: %0.2f, SE test: %s \n",
        deg, mse_train[deg],  mse_test[deg], round.(gen_error[deg,:], 2))
end
```

```
Degree: 1, MSE Train: 6.45, MSE Test: 7.52, SE test: [9.03, 4.0, 7.25, 5.21,
12.13]
Degree: 2, MSE Train: 0.99, MSE Test: 1.08, SE test: [1.44, 1.11, 1.04, 1.0
2, 0.81]
Degree: 3, MSE Train: 0.97, MSE Test: 1.07, SE test: [1.4, 1.05, 1.06, 1.1,
 0.72]
Degree: 4, MSE Train: 0.96, MSE Test: 1.04, SE test: [1.4, 1.01, 1.08, 1.07,
0.65]
```

We now recover the averaged parameters for each degree across all folds.

In [6]:

```
βpooled = [Array{Float64}(deg + 1) for deg in 1:4]
for deg in 1:4
    for i in 1:k
        βpooled[deg] += βhat_all[i][deg] / k
    end
end
for deg in 1:4
   @printf("Degree: %i %25.25s = %35.35s \n", deg,
        join(["β$i" for i in 0:deg], ", "), round.(βpooled[deg], 2))
end
```

```
Degree: 1                    β0, β1 =                         [-2.1, 1.35]
Degree: 2                β0, β1, β2 =                [-0.17, 1.13, -1.88]
Degree: 3            β0, β1, β2, β3 =           [-0.18, 0.87, -1.88, 0.1]
Degree: 4        β0, β1, β2, β3, β4 =    [-0.25, 0.87, -1.67, 0.1, -0.05]
```

We see above that for degree 2, the pooled coefficients $\hat{\beta}$ do a nice job recovering the true values

$$(\beta_0, \beta_1, \beta_2) = (0, 1, -2, 0, 0) \approx (-0.17, 1.13, -1.88) = (\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2)$$

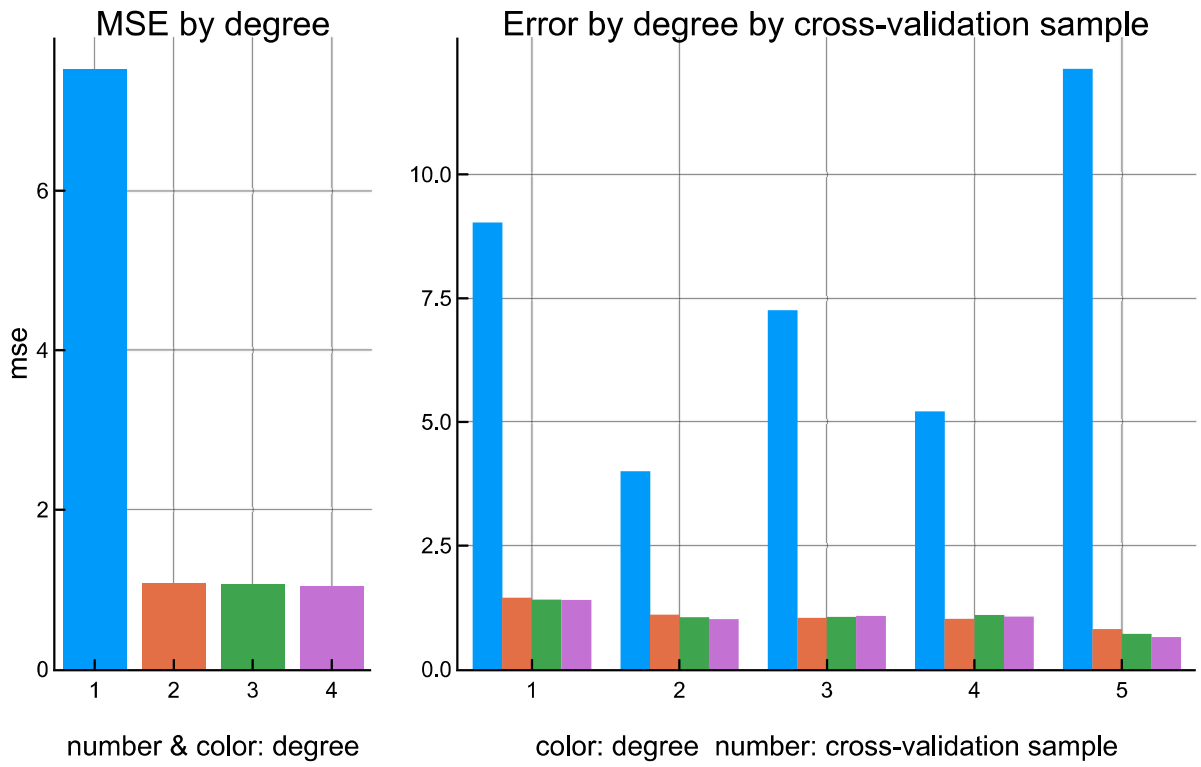
In [7]:

```
using Plots
```

In [8]:

```
plot(
    plot(mapslices(mean, gen_error, 2),
        seriestype = :bar, ylab = "mse", xlab = "number & color: degree",
        title = "MSE by degree",
        titlefont = font(12),
        guidefont = font(10),
        group = 1:4),
    plot(transpose(gen_error),
        st = :bar, legend = true, xlab = "color: degree  number: cross-validation sample",
        title = "Error by degree by cross-validation sample",
        titlefont = font(12),
        guidefont = font(10)),
    leg = false,
    layout = @layout [a{0.3w} b{0.7w}]
)
```

Out[8]:



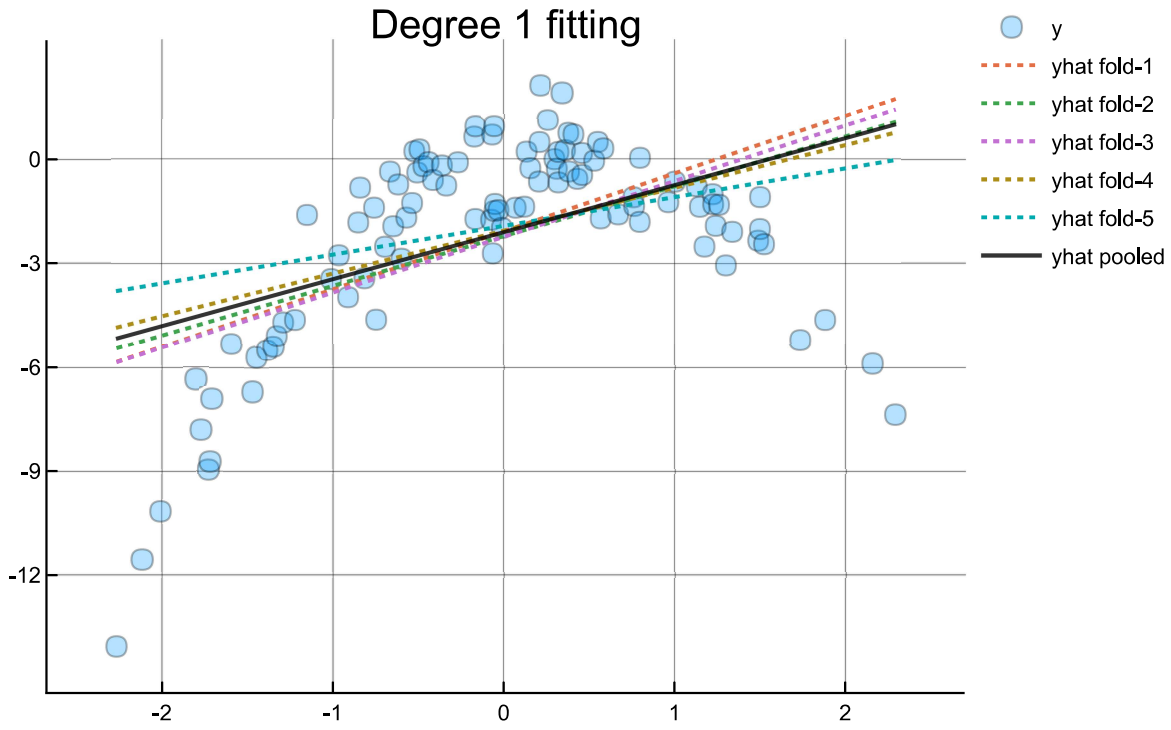
In [9]:

```
order = sortperm(x)
xo = x[order]
Xo = X[order,:]
yo = y[order]
;
```

In [10]:

```
deg = 1
Xdeg = Xo[:,1:(deg + 1)]
yhat_folds = hcat([Xdeg*βhat_all[i][deg] for i in 1:k]...)
yhat_pooled = Xdeg*βpooled[deg]
plot(xo, yo, st = :scatter, alpha = 0.3, ms = 5,
    label = "y", title = "Degree 1 fitting")
plot!(xo, yhat_folds, st = :line, lw = 2,
    label = hcat(["yhat fold-$i" for i in 1:5]...), ls = :dot)
plot!(xo, yhat_pooled, st = :line, lw = 2,
    alpha = 0.8, label = "yhat pooled", color = :black)
```

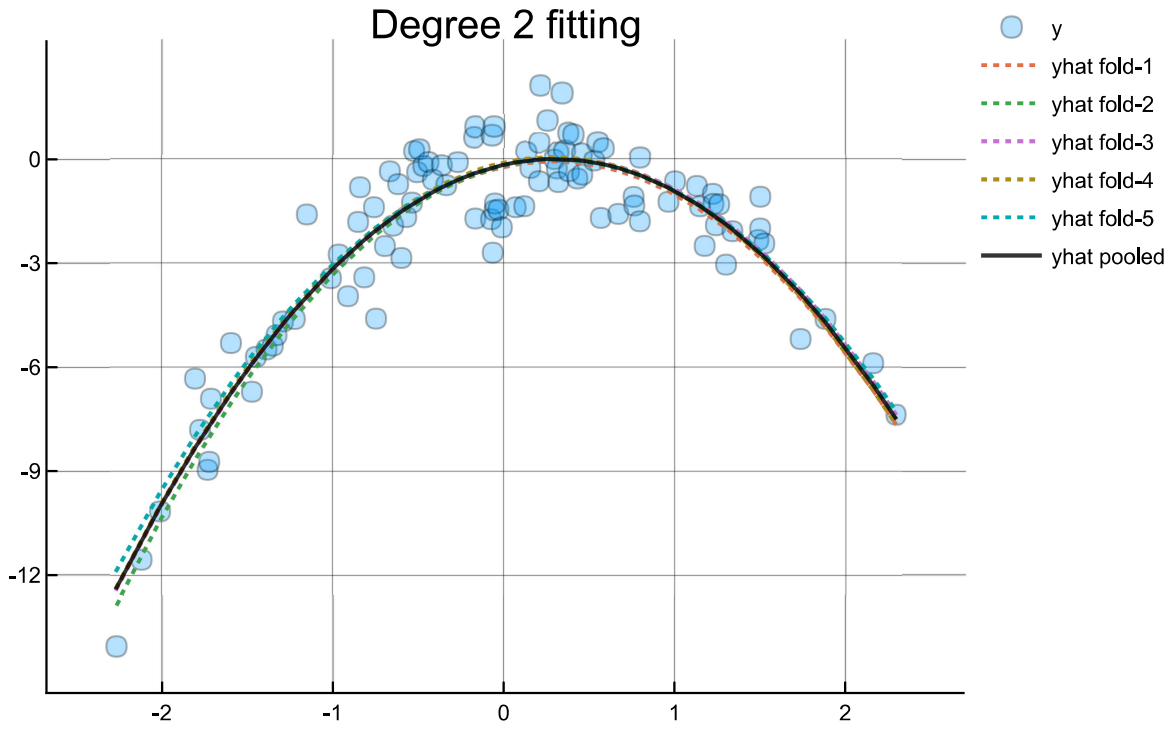
Out[10]:



In [11]:

```
deg = 2
Xdeg = Xo[:,1:(deg + 1)]
yhat_folds = hcat([Xdeg*βhat_all[i][deg] for i in 1:k]...)
yhat_pooled = Xdeg*βpooled[deg]
plot(xo, yo, st = :scatter, alpha = 0.3, ms = 5,
    label = "y", title = "Degree 2 fitting")
plot!(xo, yhat_folds, st = :line, lw = 2,
    label = hcat(["yhat fold-$i" for i in 1:5]...), ls = :dot)
plot!(xo, yhat_pooled, st = :line, lw = 2, alpha = 0.8,
    label = "yhat pooled", color = :black)
```

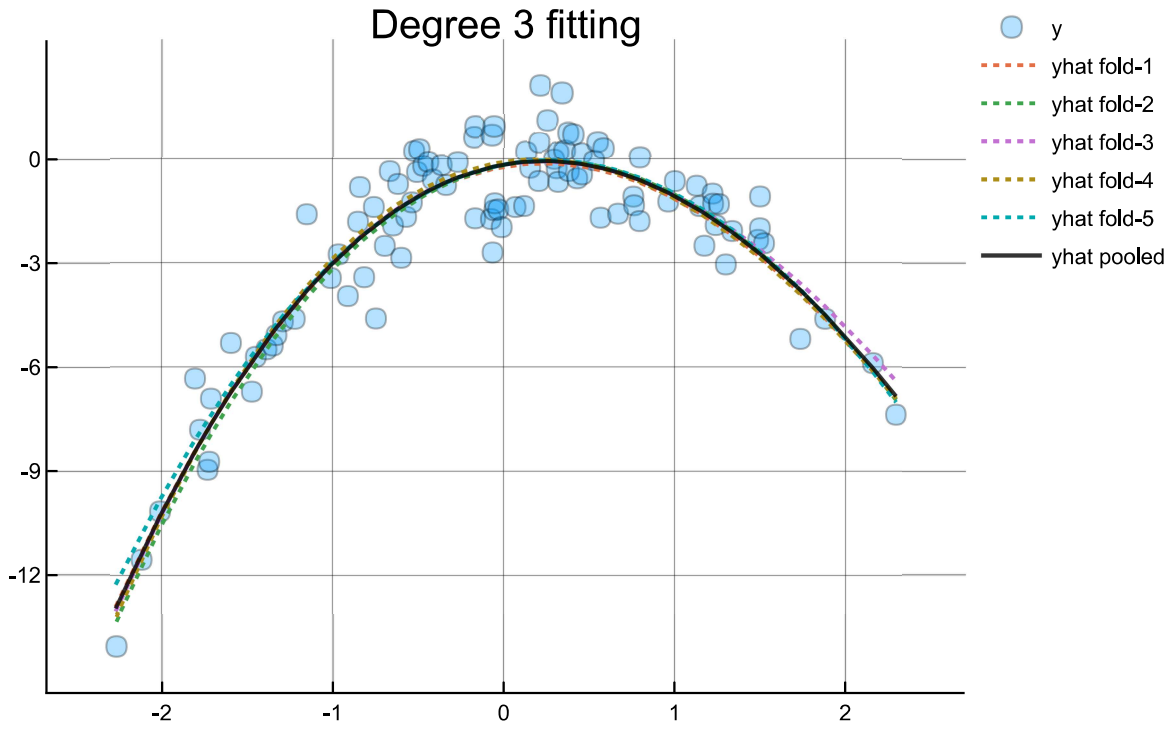
Out[11]:



In [12]:

```
deg = 3
Xdeg = Xo[:,1:(deg + 1)]
yhat_folds = hcat([Xdeg*βhat_all[i][deg] for i in 1:k]...)
yhat_pooled = Xdeg*βpooled[deg]
plot(xo, yo, st = :scatter, alpha = 0.3, ms = 5,
    label = "y", title = "Degree 3 fitting")
plot!(xo, yhat_folds, st = :line, lw = 2,
    label = hcat(["yhat fold-$i" for i in 1:5]...), ls = :dot)
plot!(xo, yhat_pooled, st = :line, lw = 2,
    alpha = 0.8, label = "yhat pooled", color = :black)
```

Out[12]:



In [13]:

```
deg = 4
Xdeg = Xo[:,1:(deg + 1)]
yhat_folds = hcat([Xdeg*βhat_all[i][deg] for i in 1:k]...)
yhat_pooled = Xdeg*βpooled[deg]
plot(xo, yo, st = :scatter, alpha = 0.3, ms = 5,
    label = "y", title = "Degree 4 fitting")
plot!(xo, yhat_folds, st = :line, lw = 2,
    label = hcat(["yhat fold-$i" for i in 1:5]...), ls = :dot)
plot!(xo, yhat_pooled, st = :line, lw = 2, alpha = 0.8,
    label = "yhat pooled", color = :black)
```

Out[13]: