DABN13 - Assignment 1

This lab comes in the form of an R Markdown document which you are supposed to fill in. All instances that require you input are marked by "??". Please replace that with the corresponding code for a given task. Additionally, you need to uncomment all commented (`#`) lines in the R code chunks below in order to make the script work. Moreover, note the following:

- Often, we have specified names for objects that you are supposed to create. Use them. Do not come up with any object names of your own.
- Tasks that require you to write a function will provide you with function name, all inputs to the function as well as the function output. Your task is to write operations within the function that work with these given parameters.
- At times, you will be asked to use a specific function to complete a task. If this is the case, please follow instructions to the point. Do not implement any alternative way of performing the task.
- Sometimes, you might have questions concerning the use of a specific R command. Please approach this situation as in a real-life programming situation: First, use the R help files to find an answer. If unsuccessful, use Google. If still unsuccessful, post your question in the discussion section of our course page on Canvas.
- Please write text answers into the corresponding string variables.

Part one: Linear regression practice

In this basic section, we are goign to use the standard canned routines for linear regression in R.

Load the data set

We will be working with `Guns.dta`, a Stata dataset containing yearly US state data between 1977 and 1999 of three different crime rates, a number of additional state characteristics, as well as an indicator for the existence of a "shall-carry" law that allows citizens to obtain a permission to wear concealed handguns. In the following, you will fit a simple predictive model for state-wide violent crime rates.

To begin with, use the read.dta command in the "foreign" package to load `Guns.dta`

```r
library("foreign")
library(haven)
setwd("/Users/viktorsjoberg/Desktop/Assignment 1")
Guns <- read_dta("Guns.dta")
```

Task 1a)

As a first task, we learn a linear regression using the `lm()`-command. This command primarily asks you to specify a model formula. The basic syntax for formulas is as follows:

outvar ~ invar1 + invar2 + invar3

Here, *outvar* and *invar1-invar3* must be replaced with the names of output variable and input variables respectively. For more detailed info on formulas, see the "Details" section in the R help file on "formula".

`lm()` also requires you to specify a "data" argument. See the "Arguments" section in the help file on "lm" for details.

Now use `lm()` to learn a regression model with log violent crime rate as output and the following input variables:

- the logarithm of state population,
- average per capita income,
- shall-carry law in effect,
- log murder rates,
- log robbery rates,
- (an intercept)

Save your learned model as object `lm_fit_1a`

```
lm_fit_1a <-  lm(formula = log(vio) ~ log(pop) + avginc + shall + log(Guns$mur) + log(rob), data = Guns
```

Task 1b)

Least squares regression coefficients can be extracted from the learned model `lm_fit_1a` by using the `coef()`-command. Save the coefficients as object `lm_coef_1b`.

```
lm_coef_1b <- coef(lm_fit_1a)
lm_coef_1b
```

```
##   (Intercept)      log(pop)        avginc         shall log(Guns$mur)
##    3.13920377   -0.02893736    0.01895990    0.03172033    0.26524857
##      log(rob)
##    0.46498794
```

Task 1c)

Model residuals can be extracted from objects created by `lm` using the `residuals()` function. Obtain the model residuals of the regression from Task 1a in this way.

Additionally, residuals are saved inside the `lm_fit_1a` object. Use the `names()`-function to report the names of all objects within `lm_fit_1a`. Save this vector of names as `lm_objnames_1c`.

Lastly, calculate the sum of squared differences between the residuals you find there and the residuals that you extracted using `residuals()`.

```
lm_res_1c <- residuals(lm_fit_1a)

lm_objnames_1c <- names(lm_fit_1a)
lm_objnames_1c
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
diff_res <-  sum((lm_res_1c-lm_fit_1a$residuals)^2)
diff_res
```

```
## [1] 0
```

Task 1d)

In order to obtain training data predictions, we can use `predict()`. Do this and save your predicted outputs as `lm_pred_1d`.

The data for which we predict here is the same data used for model training. Accordingly, only need to specify one argument (i.e. input) for `predict()`.

```
lm_pred_1d <- predict(lm_fit_1a)
```

Task 1e)

A good prediction model for violent crime rates should capture all systematic patterns in the variation of this variable. A simple, but very effective way of finding out whether this is the case is to look at residual plots. If model residuals look like more than completely random noise, then there must be patterns left that we can exploit. Conduct the following steps:

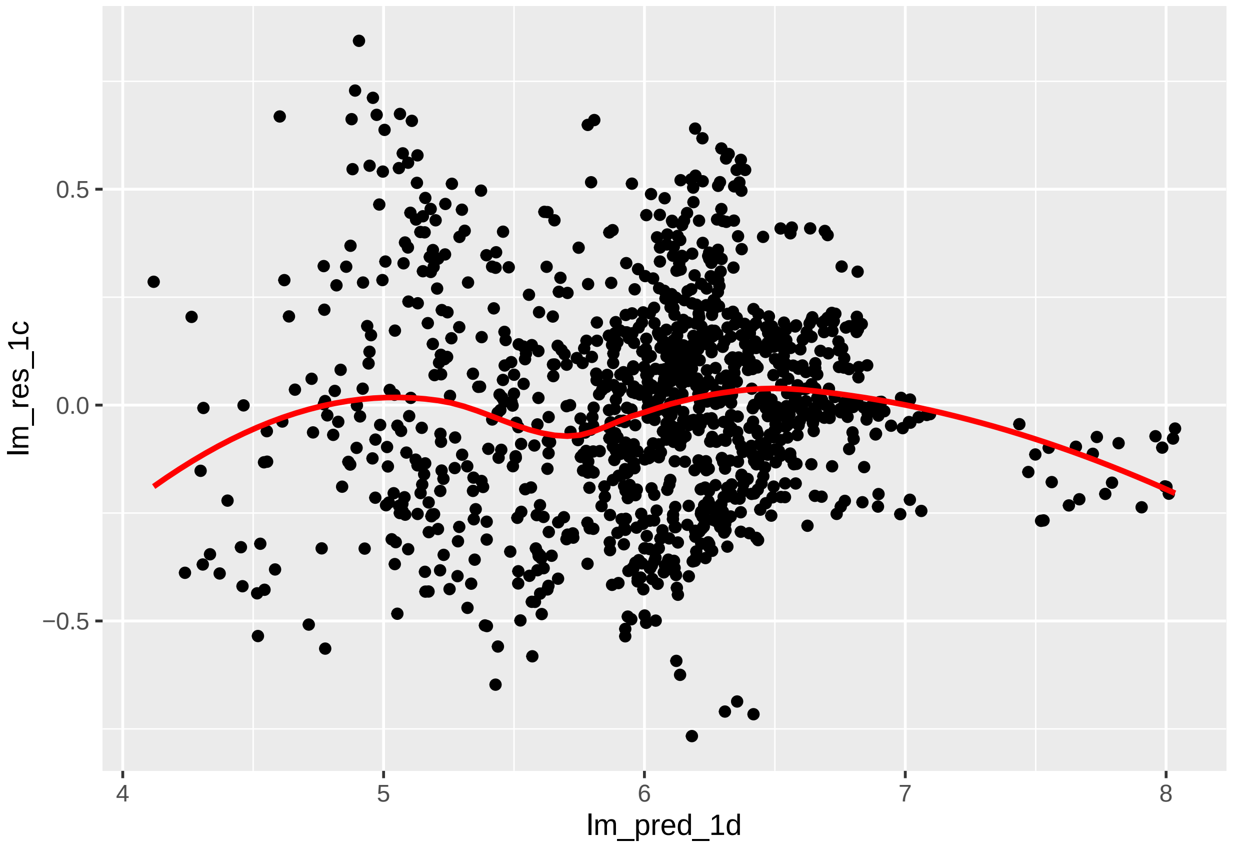
1. Create a data frame `plotdata_1e` that contains training data predictions and residuals from `lm_fit_1a`.
2. Complete the code chunk that creates the object `figure_1e` by making appropriate replacements for `??`. We want a plot that has model residuals from on the y-axis and outcome predictions on the x-axis.
3. Do you see any remaining systematic patterns in the data? Write your answers into the string variable `rem_patterns_1e`.

```
library(ggplot2)

## 1.
plotdata_1e <- data.frame(Predictions = lm_pred_1d, Residuals = lm_res_1c)

## 2.
figure_1e <- ggplot(data=plotdata_1e, mapping=aes(x=lm_pred_1d, y=lm_res_1c)) +         # opens plot su
              geom_point() + # adds scatter plot
              geom_smooth(se=FALSE,method='loess', col='red') # adds "a fitted smooth curve"
print(figure_1e)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



```r
## 3.
rem_patterns_1e <- "I dont see any remaining patterns"
```

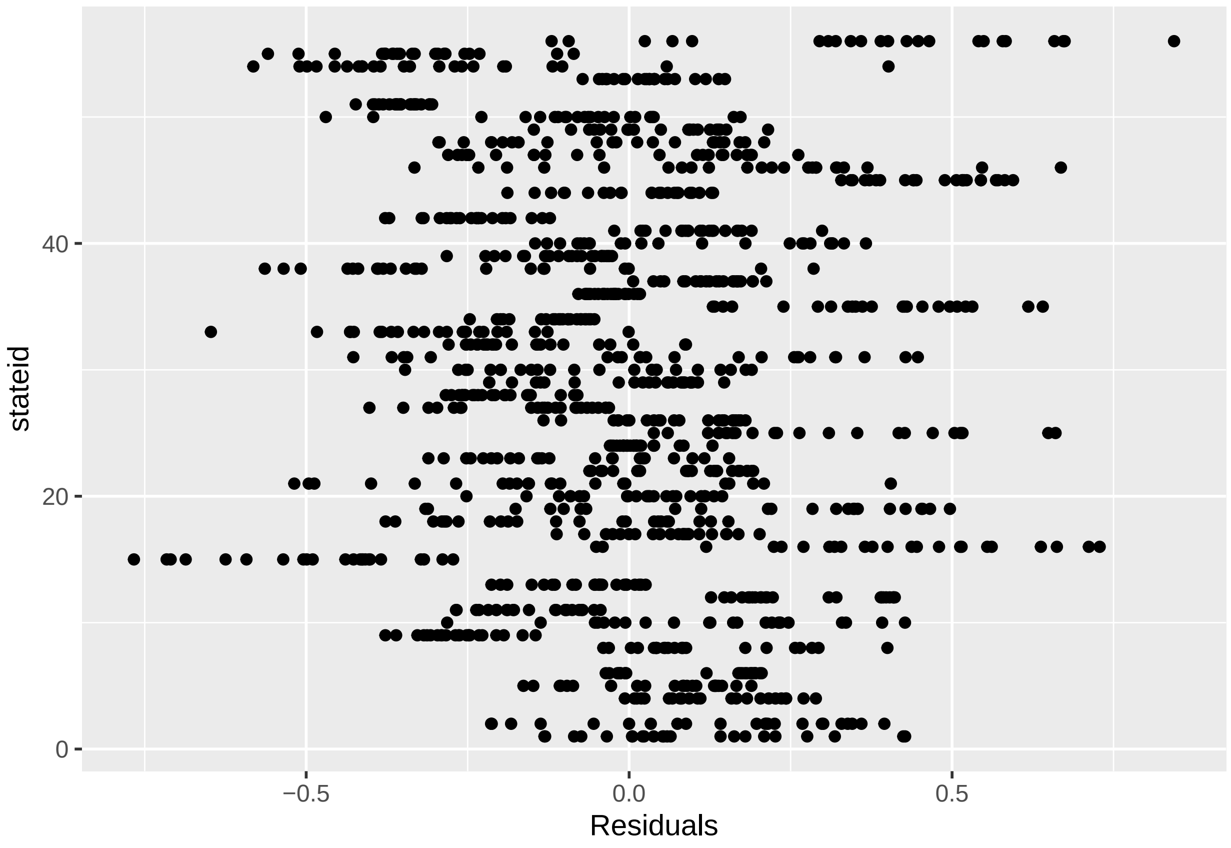
Task 1f)

Let us proceed with another plot that should highlight an obvious source of unaccounted patterns in the data. Conduct the following steps:

1. Create a data frame `plotdata_1f` which adds the variable `stateid` from `Guns` to `plotdata_1e`.
2. Plot the model residuals (x-axis) against `stateid` (y-axis).
3. Describe any patterns that you see or conclude that you don't see any. Irrespective of your choice, write your answer into the string variable `whatIsee_1f`

```r
## 1.
plotdata_1f <- cbind(plotdata_1e, stateid = Guns$stateid)

## 2.
figure_1f <- ggplot(data = plotdata_1f, mappin=aes(x=Residuals, y=stateid)) +
             geom_point()
print(figure_1f)
```



```r
## 3.
whatIsee_1f <- "I dont see any patterns"
```

Task 1g)

`stateid` is a variable that want to add to our model specification in some form. Before doing this, use the `summary()` command to get some descriptive statistics this variable in `Guns` and save them as object summary_1g.

You will see that a mean and a median are reported. Hence, as what type of variable is stateid apparently seen by R? Write your answer into the string variable typeofvarb_1g.

Would it make sense to add stateid variable into our model from Task 1a) as it currently is? Why or why not? Write your answer into the string variable in_regmodel_1g.

```
summary_1g <- summary(Guns)
print(summary_1g)
```

```
##       year          vio              mur              rob
##  Min.   :77   Min.   :  47.0   Min.   : 0.200   Min.   :   6.4
##  1st Qu.:82   1st Qu.: 283.1   1st Qu.: 3.700   1st Qu.:  71.1
##  Median :88   Median : 443.0   Median : 6.400   Median : 124.1
##  Mean   :88   Mean   : 503.1   Mean   : 7.665   Mean   : 161.8
##  3rd Qu.:94   3rd Qu.: 650.9   3rd Qu.: 9.800   3rd Qu.: 192.7
##  Max.   :99   Max.   :2921.8   Max.   :80.600   Max.   :1635.1
##   incarc_rate        pb1064           pw1064          pm1029
##  Min.   :  19.0   Min.   : 0.2482   Min.   :21.78   Min.   :12.21
##  1st Qu.: 114.0   1st Qu.: 2.2022   1st Qu.:59.94   1st Qu.:14.65
##  Median : 187.0   Median : 4.0262   Median :65.06   Median :15.90
##  Mean   : 226.6   Mean   : 5.3362   Mean   :62.95   Mean   :16.08
##  3rd Qu.: 291.0   3rd Qu.: 6.8507   3rd Qu.:69.20   3rd Qu.:17.53
##  Max.   :1913.0   Max.   :26.9796   Max.   :76.53   Max.   :22.35
##       pop              avginc          density              stateid
##  Min.   : 0.4027   Min.   : 8.555   Min.   : 0.000707   Min.   : 1.00
##  1st Qu.: 1.1877   1st Qu.:11.935   1st Qu.: 0.031911   1st Qu.:16.00
##  Median : 3.2713   Median :13.402   Median : 0.081569   Median :29.00
##  Mean   : 4.8163   Mean   :13.725   Mean   : 0.352038   Mean   :28.96
##  3rd Qu.: 5.6856   3rd Qu.:15.271   3rd Qu.: 0.177718   3rd Qu.:42.00
##  Max.   :33.1451   Max.   :23.647   Max.   :11.102116   Max.   :56.00
##      shall
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :0.000
##  Mean   :0.243
##  3rd Qu.:0.000
##  Max.   :1.000
```

```
typeofvarb_1g <- "Categorical"
in_regmodel_1g <- "No, it does not make sense to add stateid as it currently is.
Stateid is a categorical variable with many unique values, and including it
directly in the regression model may lead to overfitting and multicollinearity
issues. It would be more appropriate to encode stateid as a factor variable and
include it in the model."
```

Task 1h)

The way in which R treats a specific variable can change considerably if we encode it as a factor variable. Hence, replace the variable stateid in Guns with a version if itself that is encoded as factor variable. Use the factor() command for that.

Next, get the summary statistics of this modified variable and save them as object summary_1h. What has changed? Write your answer into the string variable whatchanged_1h

```r
Guns$stateid <- factor(Guns$stateid)
summary_1h <- summary(Guns)
print(summary_1h)
```

```
##       year          vio              mur               rob        
##  Min.   :77   Min.   :  47.0   Min.   : 0.200   Min.   :   6.4  
##  1st Qu.:82   1st Qu.: 283.1   1st Qu.: 3.700   1st Qu.:  71.1  
##  Median :88   Median : 443.0   Median : 6.400   Median : 124.1  
##  Mean   :88   Mean   : 503.1   Mean   : 7.665   Mean   : 161.8  
##  3rd Qu.:94   3rd Qu.: 650.9   3rd Qu.: 9.800   3rd Qu.: 192.7  
##  Max.   :99   Max.   :2921.8   Max.   :80.600   Max.   :1635.1  
##                                                                 
##   incarc_rate         pb1064            pw1064          pm1029     
##  Min.   :  19.0   Min.   : 0.2482   Min.   :21.78   Min.   :12.21  
##  1st Qu.: 114.0   1st Qu.: 2.2022   1st Qu.:59.94   1st Qu.:14.65  
##  Median : 187.0   Median : 4.0262   Median :65.06   Median :15.90  
##  Mean   : 226.6   Mean   : 5.3362   Mean   :62.95   Mean   :16.08  
##  3rd Qu.: 291.0   3rd Qu.: 6.8507   3rd Qu.:69.20   3rd Qu.:17.53  
##  Max.   :1913.0   Max.   :26.9796   Max.   :76.53   Max.   :22.35  
##                                                                    
##       pop              avginc          density              stateid    
##  Min.   : 0.4027   Min.   : 8.555   Min.   : 0.000707   1      :  23  
##  1st Qu.: 1.1877   1st Qu.:11.935   1st Qu.: 0.031911   2      :  23  
##  Median : 3.2713   Median :13.402   Median : 0.081569   4      :  23  
##  Mean   : 4.8163   Mean   :13.725   Mean   : 0.352038   5      :  23  
##  3rd Qu.: 5.6856   3rd Qu.:15.271   3rd Qu.: 0.177718   6      :  23  
##  Max.   :33.1451   Max.   :23.647   Max.   :11.102116   8      :  23  
##                                                         (Other):1035  
##      shall      
##  Min.   :0.000  
##  1st Qu.:0.000  
##  Median :0.000  
##  Mean   :0.243  
##  3rd Qu.:0.000  
##  Max.   :1.000  
## 
```

```r
whatchanged_1h <- "The stateid variable has been encoded as a factor variable,
and its summary statistics now include the counts of each unique stateid value
along with the number of missing values. This change reflects the transformation
of stateid from a numeric variable to a categorical factor variable."
```

Task 1i)

Learn the regression model from Task 1a with factor variable `state_id` as an additional regressor. Use the `summary()` command to report a summary of the regression results. How has `lm()` included `stateid` into the model?

```r
lm_fit_1i  <- lm(formula = log(vio) ~ log(pop) + avginc + shall + log(Guns$mur) + log(rob) + stateid, da
summary_1i <- summary(lm_fit_1i)
print(summary_1i$coefficients[1:15,])
```

```
##                   Estimate  Std. Error     t value      Pr(>|t|)
## (Intercept)     3.12755578 0.111474729  28.0561864 1.562219e-131
## log(pop)        0.05307272 0.053785764   0.9867428  3.239824e-01
## avginc         0.04293095 0.002909938  14.7532177  3.875286e-45
## shall          0.02504379 0.013495284   1.8557442  6.375327e-02
## log(Guns$mur)  0.03141796 0.017985416   1.7468576  8.093688e-02
## log(rob)       0.52152651 0.018630560  27.9930671 4.419585e-131
## stateid2       0.04525298 0.125772357   0.3598007  7.190642e-01
## stateid4      -0.04897295 0.039323406  -1.2453893  2.132499e-01
## stateid5      -0.02172743 0.047076675  -0.4615328  6.445063e-01
## stateid6      -0.30500591 0.104495948  -2.9188300  3.583924e-03
## stateid8      -0.19440480 0.042910659  -4.5304548  6.518311e-06
## stateid9      -0.73174451 0.052041679 -14.0607398  1.816559e-41
## stateid10     -0.11565959 0.115130616  -1.0045946  3.153097e-01
## stateid11     -0.05061507 0.124730027  -0.4057970  6.849695e-01
## stateid12     -0.04150074 0.065002277  -0.6384505  5.233113e-01
```

```
# Explain how lm() included stateid into the model
howincluded1i <- "lm() automatically included stateid as a set of dummy
variables representing each unique state. Each dummy variable serves as an
additional predictor in the model, allowing it to account for state-level
variations in the response variable."
```

Task 1j)

The regression results in Task 1i) look the way they do because the `lm()` command conveniently transforms the factor variable `stateid` into numerical variables before fitting the model. In particular, the `model.matrix()` command is automatically used to arrive at a set of input variables that one can directly feed into a least squares estimation routine.

Some important R-commands are less convenient and require you to transform the predictors yourselves. In order to prepare for this situation, use the `model.matrix()` command manually with the same model specification as in Task 1i to get the set of predictors internally generated by `lm()`. Inspect the resulting matrix (e.g. using the `View()` command) and describe in how far it differs from the variables that you specified.

```
inputs_1j <- model.matrix(lm_fit_1i)
howxmatdiffers_1j <- "The manually created model matrix includes the intercept
term and dummy variables for stateid, just like the model generated by lm().
The primary difference is that the manually created model matrix does not
include the original categorical stateid variable, as it has been replaced by
its dummy variable representation."
```

Part two: Least squares regression mechanics

In this more advanced section, you are supposed to create your own algorithms for some standard statistics of a fitted linear regression model from scratch. You are only allowed to use matrix and scalar operations like

```
## solve(), inv(), t(), %*%, /, sum(), mean(), rep()
```

to manually produce results otherwise produced by the `lm()`-command.

Task 2a)

Build a function that takes the two arguments`x` and `y` and that returns the least squares estimate $\hat{\beta}$ of the slope coefficients on `x`.

```
get_beta <- function(x, y){
  betahat <- solve(t(x)%*%x) %*% t(x) %*% y
  return(betahat)
}
```

Task 2b)

Build a function that computes the model residuals as an object called `res`. Refer to the function written in Task 2a to get an estimate of the slope coefficients.

```
get_res <- function(x, y){
  betahat <-get_beta(x, y)
  res <- y - x %*% betahat
  return(res)
    }
```

Task 2c)

Build a function that computes R^2, i.e. the estimated proportion of variance of y that is explained by the model inputs. Refer to the function written in Task 2b to get model residuals.

```
r2 <- function(x, y){
  res <- get_res(x, y)
  SST <- sum((y - mean(y))^2)
  SSE <- sum(res^2)
  R2 <- 1- SSE/SST
  return(R2)
}
```

Task 2d)

Next, prepare the inputs that we can feed into our functions. Construct an input matrix `x_2d` containing the following variables from `Guns` as columns:

- a vector of ones,
- the logarithm of state population,
- average per capita income,
- shall-carry law in effect,
- log murder rates,
- log robbery rates.

Additionally, create an output vector `y_2d` containing the log violent crime rate (for the state that year).

```
x_2d <- cbind(1, log(Guns$pop), Guns$avginc, Guns$shall, log(Guns$mur), log(Guns$rob))
y_2d <- log(Guns$vio)
```

Task 2e)

To what degree did the degree to which we picked up variation in the training data increase when we included state indicators? The functions that you wrote in Tasks 2a-2c allow us to answer this question.

Use the function from Task 2c to obtain the training R^2 for a regression model with inputs `x_2d` and for a model with inputs `inputs_1j`. Save them as `R2_nostate_2e` and `R2_withstate_2e`, respectively.

Then, use the string variable `effect_of_stateid_2e` to explain how much R^2 changed due to the use of state indicators as additional inputs.

```
R2_nostate_2e <- r2(x_2d, y_2d)
R2_withstate_2e <- r2(inputs_1j, y_2d)
```

```r
diff_in_r2 <- R2_withstate_2e-R2_nostate_2e
change_in_r2 <- print(c(R2_withstate_2e,R2_nostate_2e ))
```

```
## [1] 0.9648717 0.8548569
```

```r
diff_in_r2
```

```
## [1] 0.1100148
```

```r
effect_of_stateid_2e <- "The inclusion of state indicators as additional inputs (inputs_1j)
significantly improved the model's ability to explain the variation in the training data.
The R-squared value increased from R2_nostate_2e to R2_withstate_2e, indicating that
the model with state indicators better captures the underlying patterns in the data.
This suggests that state-level variations play a significant role in explaining
violent crime rates."
```