Exercise 3

We are almost done with the R primer. You learnt the neccessary basics to start crunching the data and build fancy models. The final exercise will cover the usage of scripts, introduce to the working horse of statistics - regression - and learn to visualize and communicate the results of our work.

1. Loading scripts and automating data cleaning

You have not been introduced to the data preparation routine (that comes in Exercise #6), however, you do know what a function is. When one expects certain set of actions to be repeated many times, writing those functions down into a function will save a lot of time. In this case we have prepared a "cleaner" function for you that will prepare the Loan data for modeling with a couple of clicks only. We will learn to apply this sort of external functions that are not included into any packages. This custom function takes a form of a special script, stored ina file 'helperfunctions.R'. Don't worry if you do not understand everything that happens in a function just yet , it will get obvious in couple of weeks. In the future you will be able to create this sort of 'little helpers' in no time.

1. Open a new R script and make sure your current working directory contains the file **helperfunctions.R** as well as the **loan_data.csv**.
2. Use function **source()** to load and execute the script *helperfunctions.R* containing our custom function. The function should now be visible in your R environment. You can check by calling **GetLoanDataset** without the brackets to see the function code. Don't worry if it takes some time when you call it for the first time - it is installing all the packages for you.
3. Use your custom function **GetLoanDataset()** to load and clean the data and save the resulting data frame to an object **loans**.

Well done, this will allow us to load the dataset in just a couple of lines from now on. In case you are curious, do go through the helperfunction and try to see what exactly it takes care of.

```r
# Source (=run) the R script storing your function(s)
source("Rintro_HelperFunctions.R")
loans <- GetLoanDataset()

# If the previous line did not work on your machine, the most likely
# cause is that something with the file directories went wrong.
# Check whether the data is available in your working directory.
# Recall that you can query your current working directory using the
# commend getwd(). Also have a look into the function's source code
# to understand where it tries to find the data file on your disk.
# If everything went well, you can now use the data set, e.g., to
# build models. Let's produce the usual summary for start
str(loans)
```

```
## 'data.frame':    1225 obs. of  16 variables:
##  $ YOB        : num  19 41 66 51 65 42 59 43 52 65 ...
##  $ nKIDS      : num  4 2 0 2 0 2 0 1 0 0 ...
##  $ nDEP       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ PHON       : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ dINC_SP    : num  0 0 0 0 0 10500 6500 13500 0 0 ...
##  $ EMPS_A     : Factor w/ 11 levels "B","E","M","N",..: 6 5 4 5 5 2 1 2 2 5 ...
##  $ dINC_A     : num  0 36000 30000 464 15000 48000 30000 9000 22500 19500 ...
##  $ RES        : Factor w/ 5 levels "F","N","O","P",..: 3 3 2 3 4 3 3 3 4 3 ...
##  $ dHVAL      : num  14464 0 0 24928 0 ...
##  $ dMBO       : num  4 0 0 8464 0 ...
##  $ dOUTM      : num  0 280 0 584 0 1120 520 0 0 540 ...
##  $ dOUTL      : num  0 664 0 320 0 0 0 200 200 0 ...
##  $ dOUTHP     : num  0 0 0 0 0 0 96 0 0 0 ...
##  $ dOUTCC     : num  0 80 0 60 0 0 0 0 80 0 ...
##  $ BAD        : Factor w/ 2 levels "good","bad": 1 1 1 1 1 1 1 1 1 1 ...
##  $ YOB_missing: num  0 0 0 0 0 0 0 0 0 0 ...
```

2. Visualization

Data visualization can be a great tool to analyze vast amounts of data. R provides some very nice tools for visualization. For example, the base version of R has several functions to plot data such as **plot**. Corresponding charts can look look pretty good, if options of **plot** are set in a suitable way. However, for professional charts, the package **ggplot2** is recommended and we will use that package for this exercise.

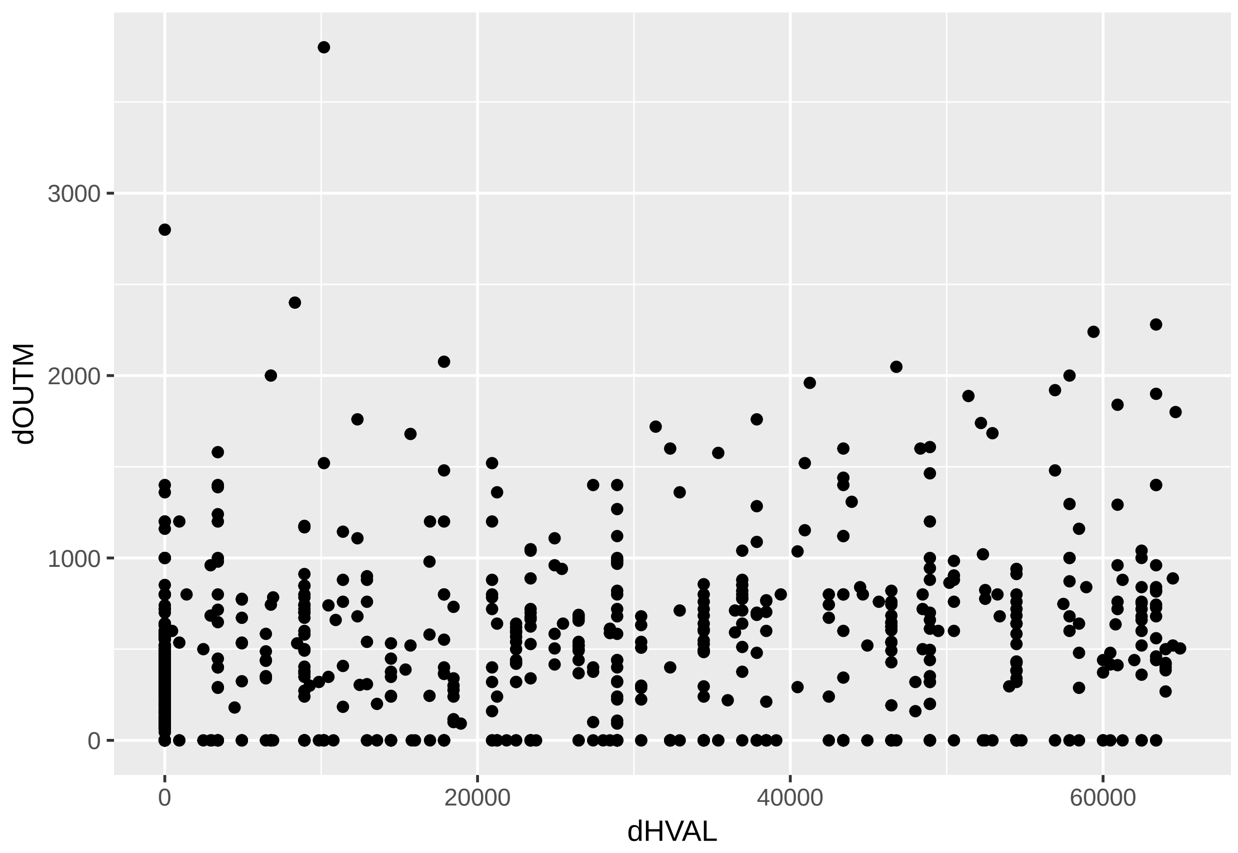
ggplot follows a so-called *grammer of graphics*. This means that you build a graph structure-by-structure with '+' in between layers. For your first steps, it is sufficient to know that 1) **ggplot()** in combination with **aes()** specifices the general data and aesthetics of a plot, e.g. the axis labels. It has the form **ggplot(data = *your.data*, aes(*options*))**. A range of functions starting with **geom__** (e.g. **geom__point()**) are used to add data to the plot layer by layer in the form of, for example, **+ geom__point(*optional options*)**.

Let's start with an example to see how **ggplot2** works. Make sure you have installed and loaded package **ggplot2**.

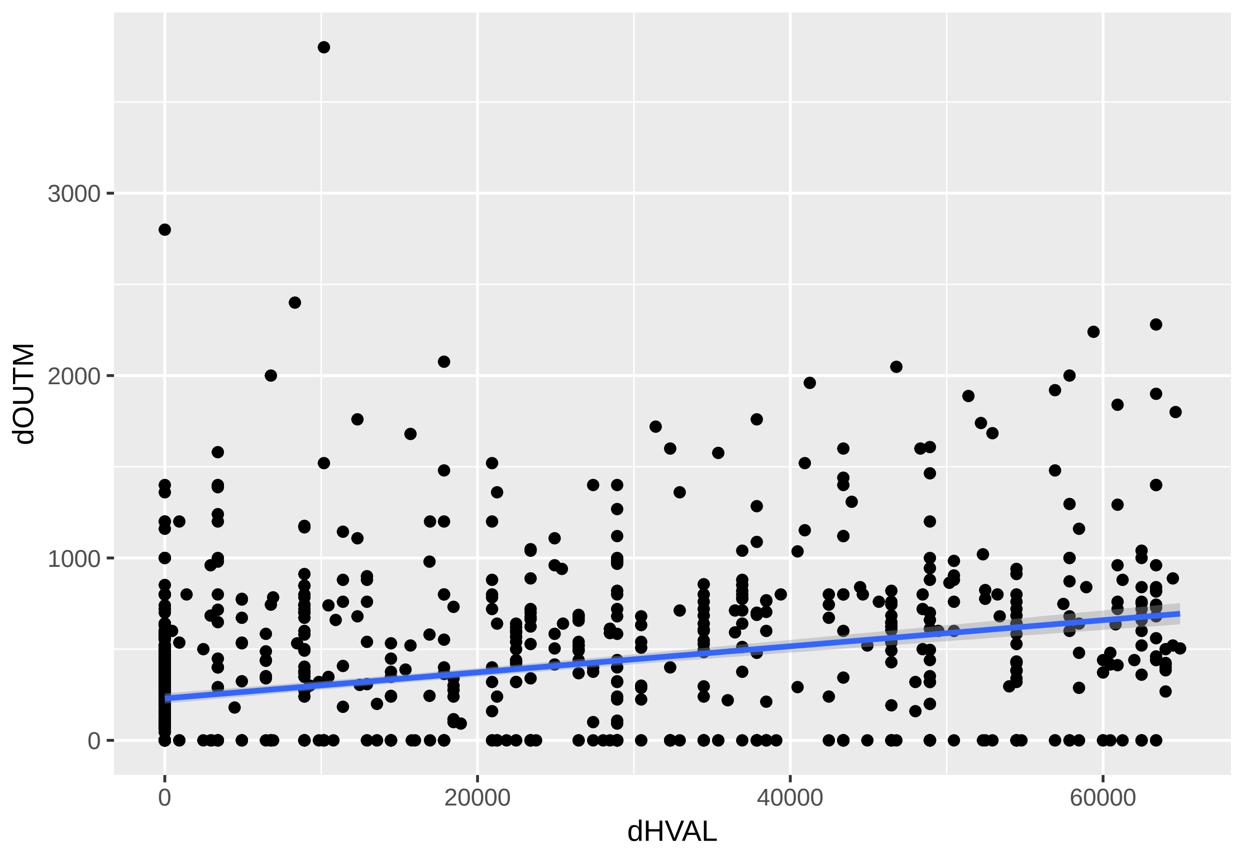
1. Run **ggplot(data = loans, aes(x = dINC__A, y = dINC__SP))** and look at the resulting plot. The structure is there, but the information is missing!
2. Try the following command **ggplot(data = loans, aes(x = dINC__A, y = dINC__SP)) + geom__point()** and look at the resulting plot. **Point**s are now drawn onto the plot frame!
3. Make sure you understand the input in **aes()** and the roles of the two parts of the function separated by the plus. After specifying a data frame as data, you can use variable names from that data frame directly and without quotation marks in the function call (here: **dINC__A** instead of **loans$dINC__A**).
4. Add **+ geom__smooth(method=lm)** after the command and run it again to add another layer to the original plot. What have you added?

```r
#Make sure you got your cleaned Loan dataset.

### Plot
# install and load package ggplot2
# install.packages("ggplot2")
library("ggplot2")
# Specify the data to be used and the relevant variables in function ggplot
# Then add the layers of data visualization on the base of this data
ggplot(data = loans, aes(x = dHVAL, y = dOUTM)) + geom_point()
```



```
# You can add more than one layer of visualization, e.g. an additional regression line (lm = linear mod
ggplot(data = loans, aes(x = dHVAL, y = dOUTM)) + geom_point() + geom_smooth(method=lm)
```

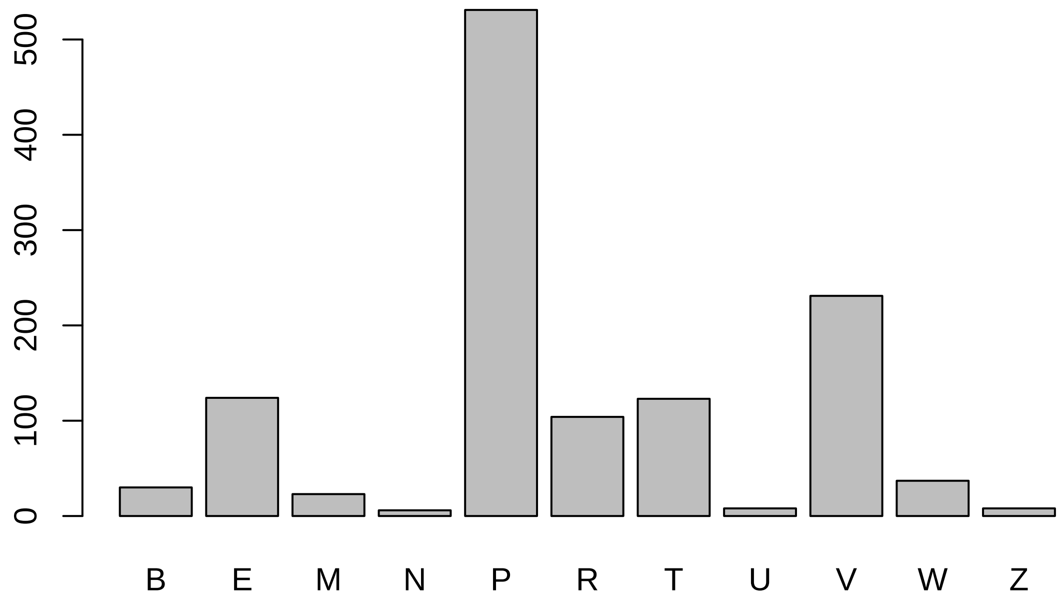


Visualisations are great to make sense of your data quickly. Let's explore the data on our loan applicants and try to derive some conclusions:

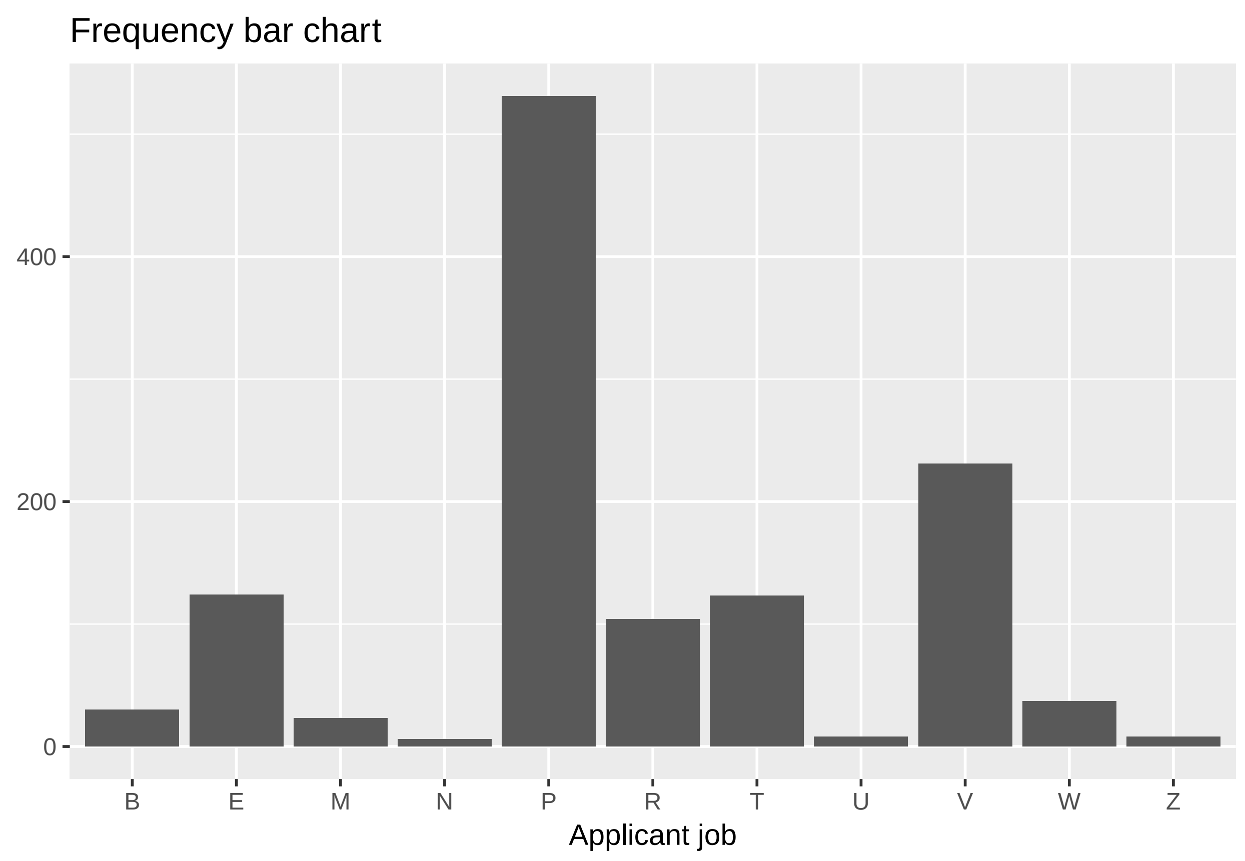
- Who are our applicants?
- Who is likely to default?

When doing exploratory analysis, we should always keep our goals in mind while simultaneously keeping an eye open for unexpected findings. When doing visualizations, keep in mind that people (including your bosses) can have a whole range of visual impairements, so pick your colors wisely.

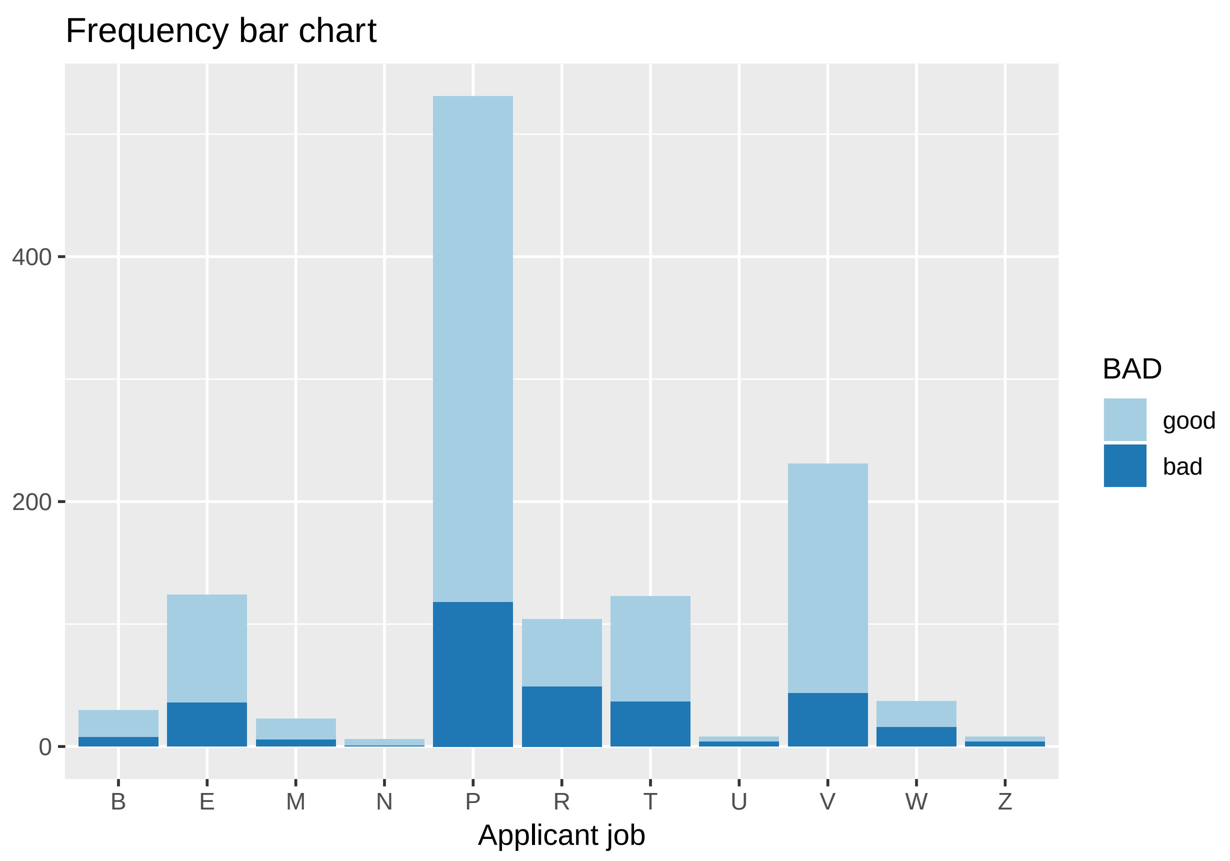
```
## Employment
# Histogram
plot(loans$EMPS_A) # This works because EMPS_A is a factor variable
```



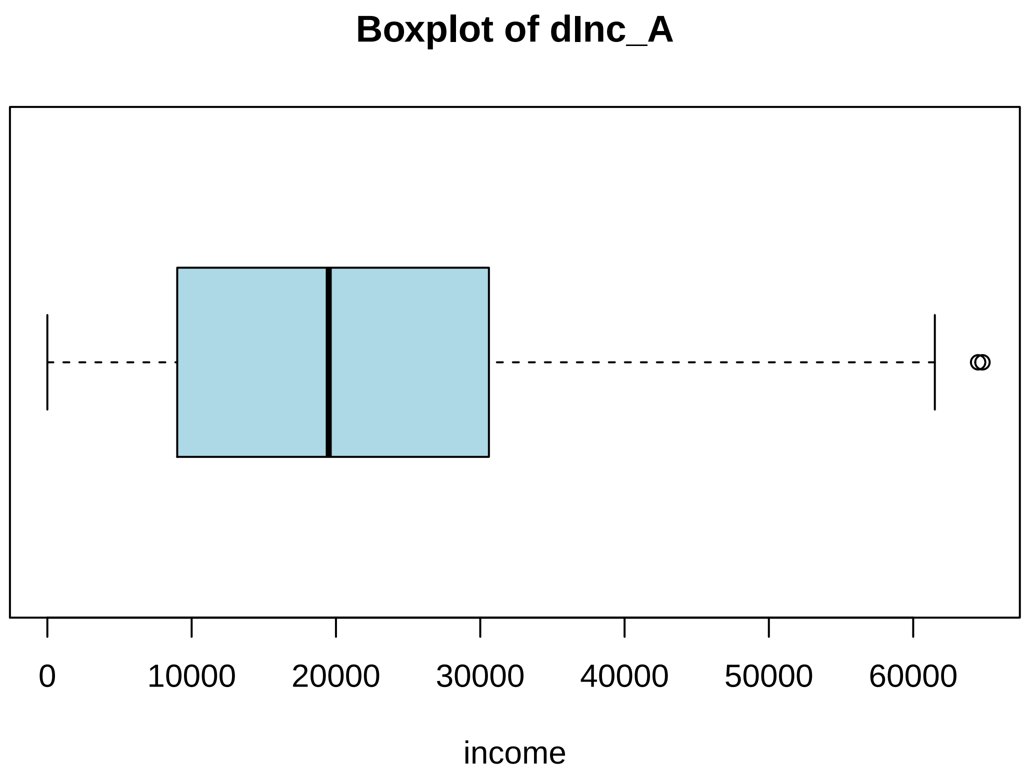
```
ggplot(loans, aes(x=EMPS_A)) + geom_bar() + labs(title="Frequency bar chart", x="Applicant job", y=NULL
```



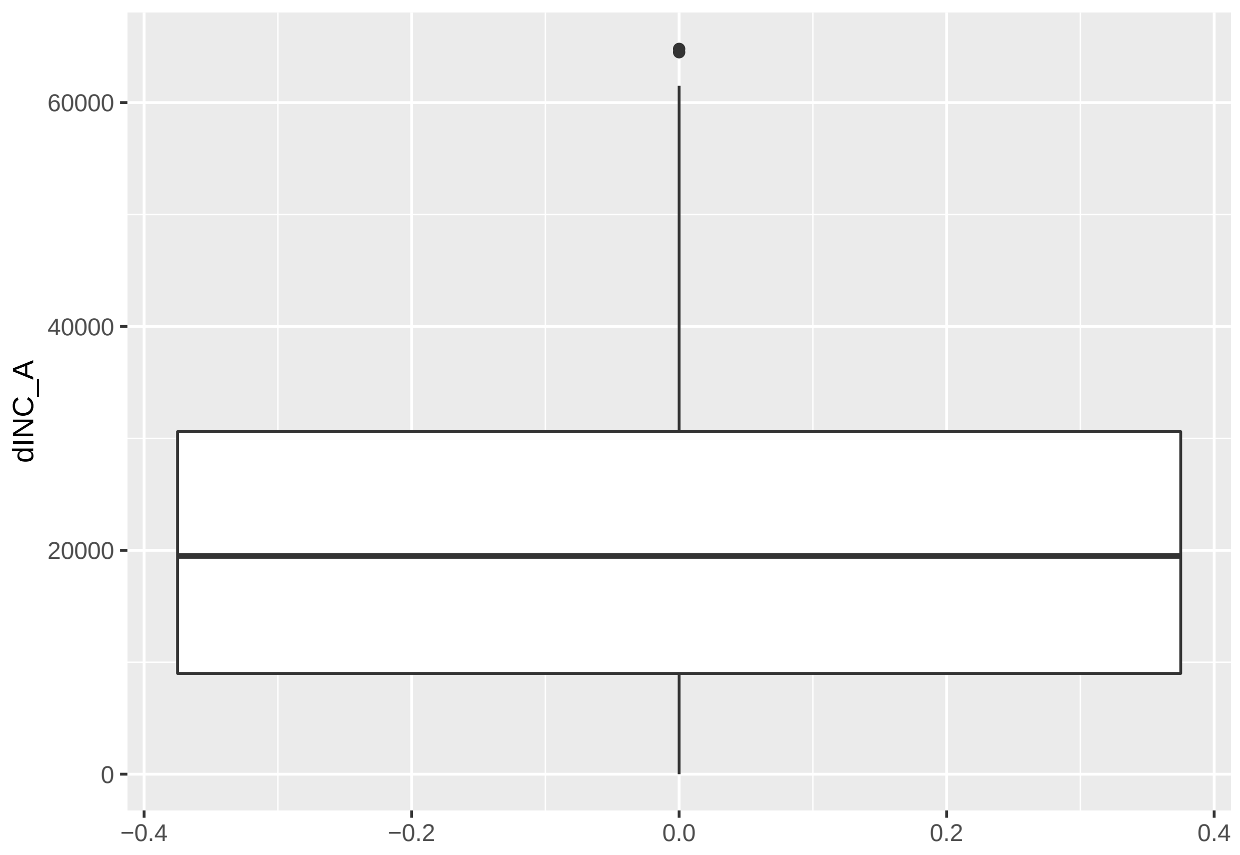
```
# Stacked histogram by default status
ggplot(loans, aes(x=EMPS_A, fill=BAD)) + geom_bar() + scale_fill_brewer(palette="Paired") + labs(title=
```



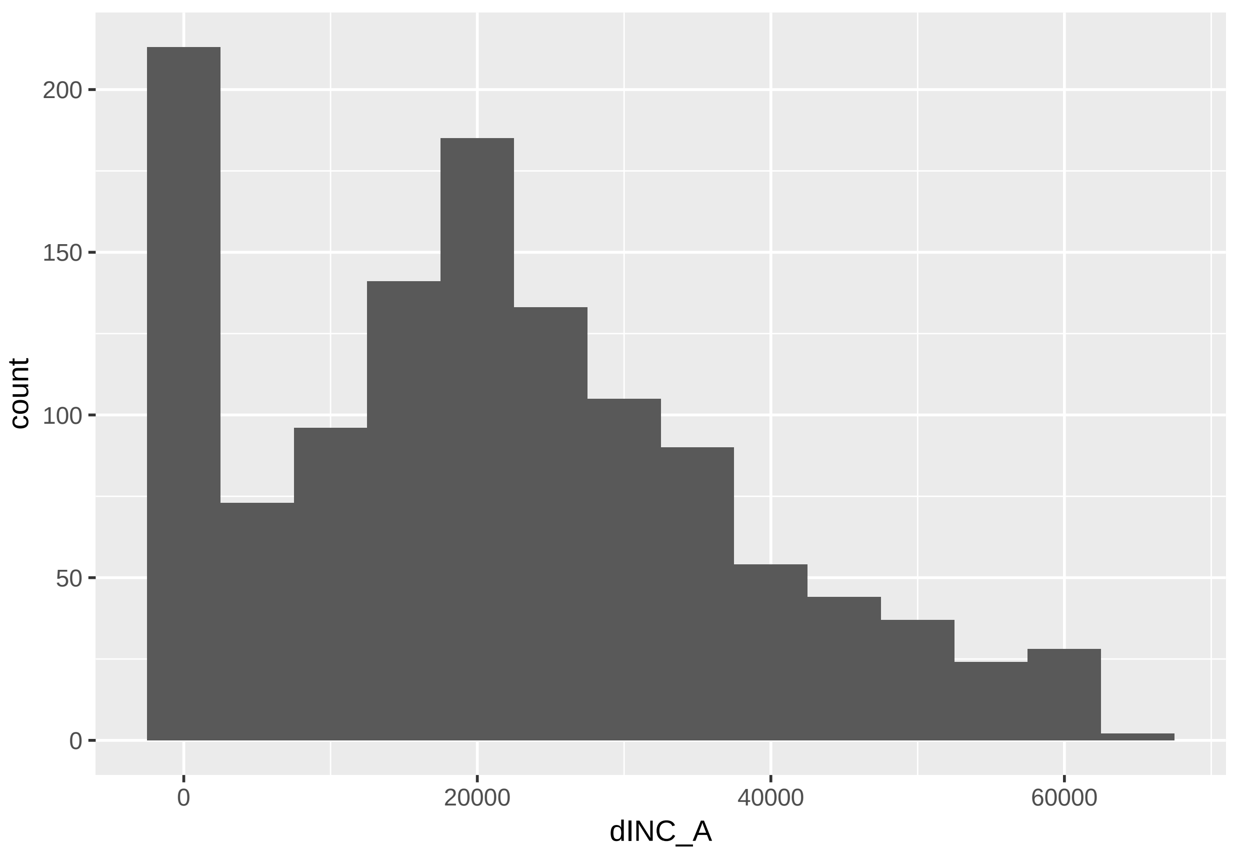
```
## Income
boxplot(loans$dINC_A, main="Boxplot of dInc_A", horizontal = TRUE, xlab="income", col = "lightblue") #
```



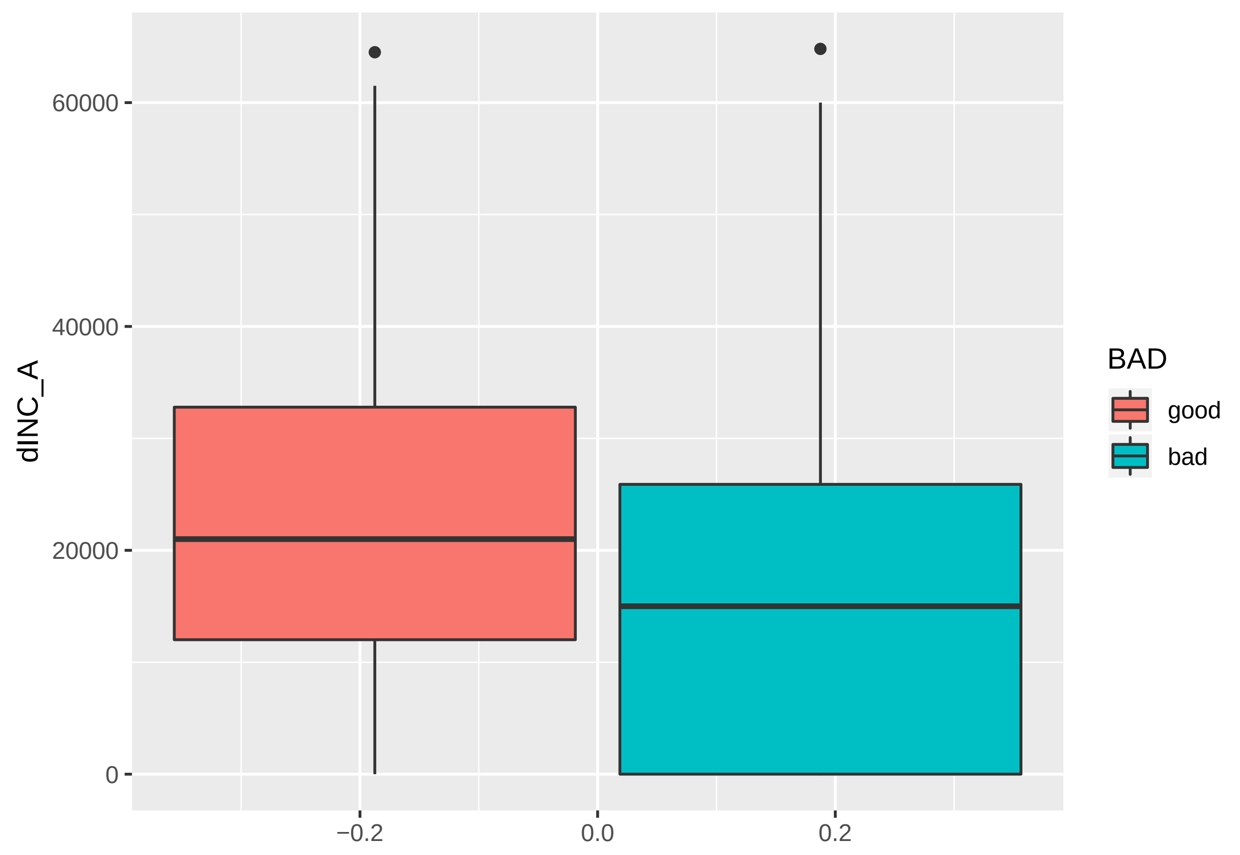
```r
ggplot(loans) + geom_boxplot(aes(y=dINC_A))
```



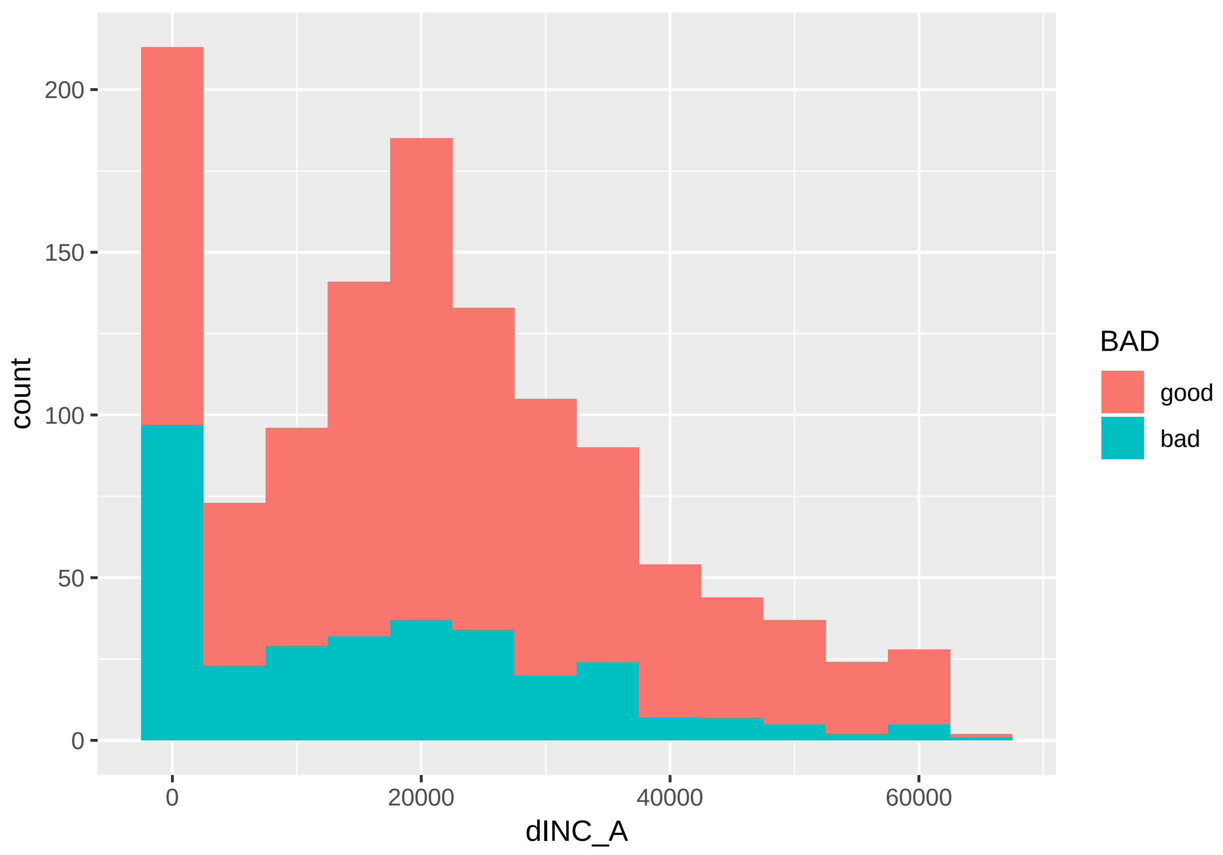
```r
ggplot(loans, aes(x=dINC_A)) + geom_histogram(binwidth = 5000)
```



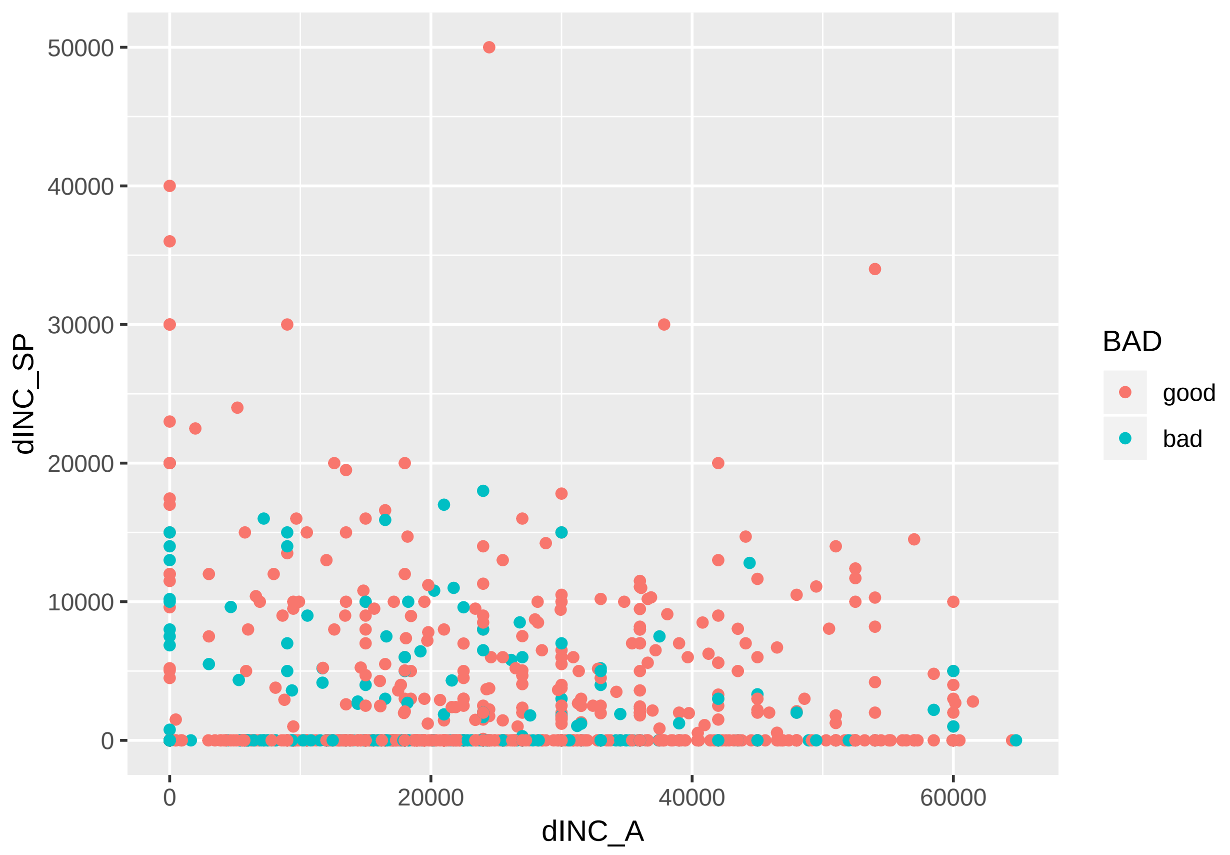
```
## Default by income
ggplot(loans) + geom_boxplot(aes(y=dINC_A, fill=BAD))
```



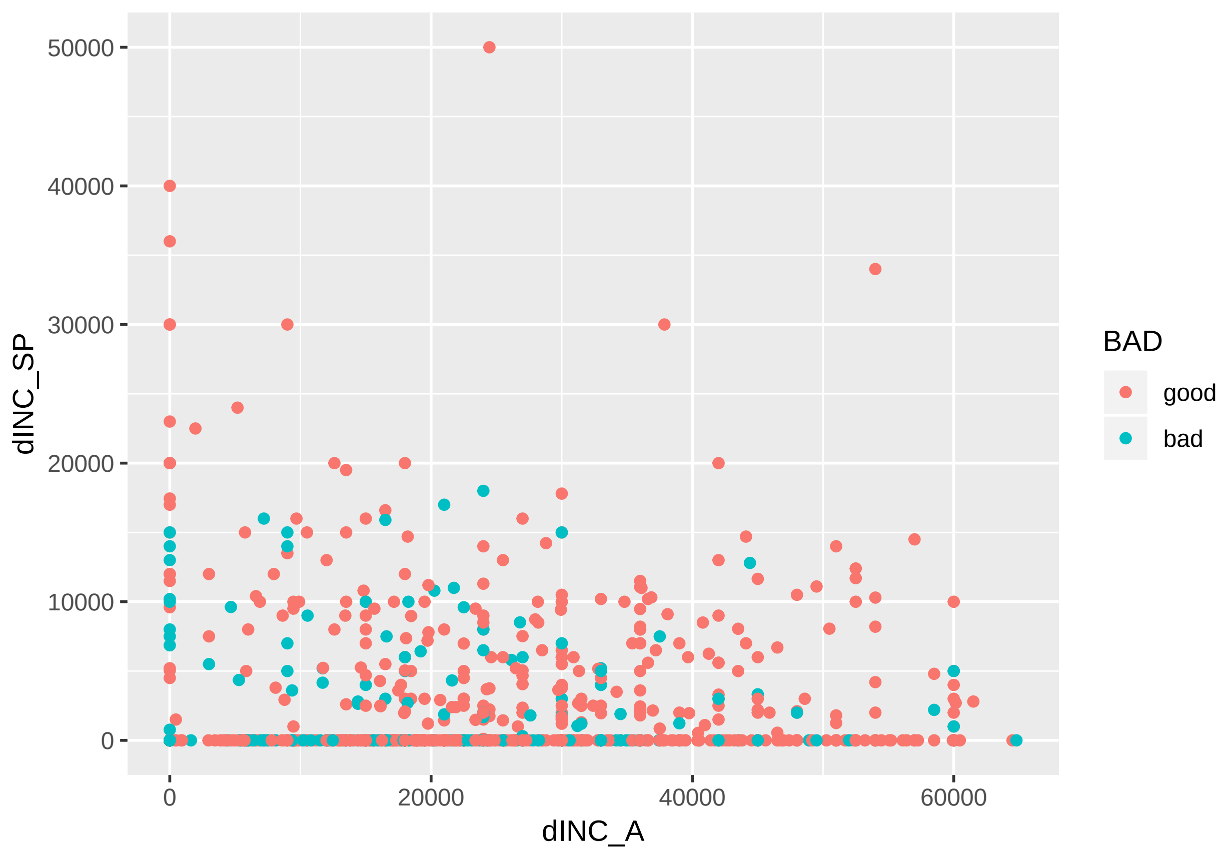
```r
ggplot(loans, aes(x=dINC_A, fill=BAD)) + geom_histogram(binwidth = 5000)
```



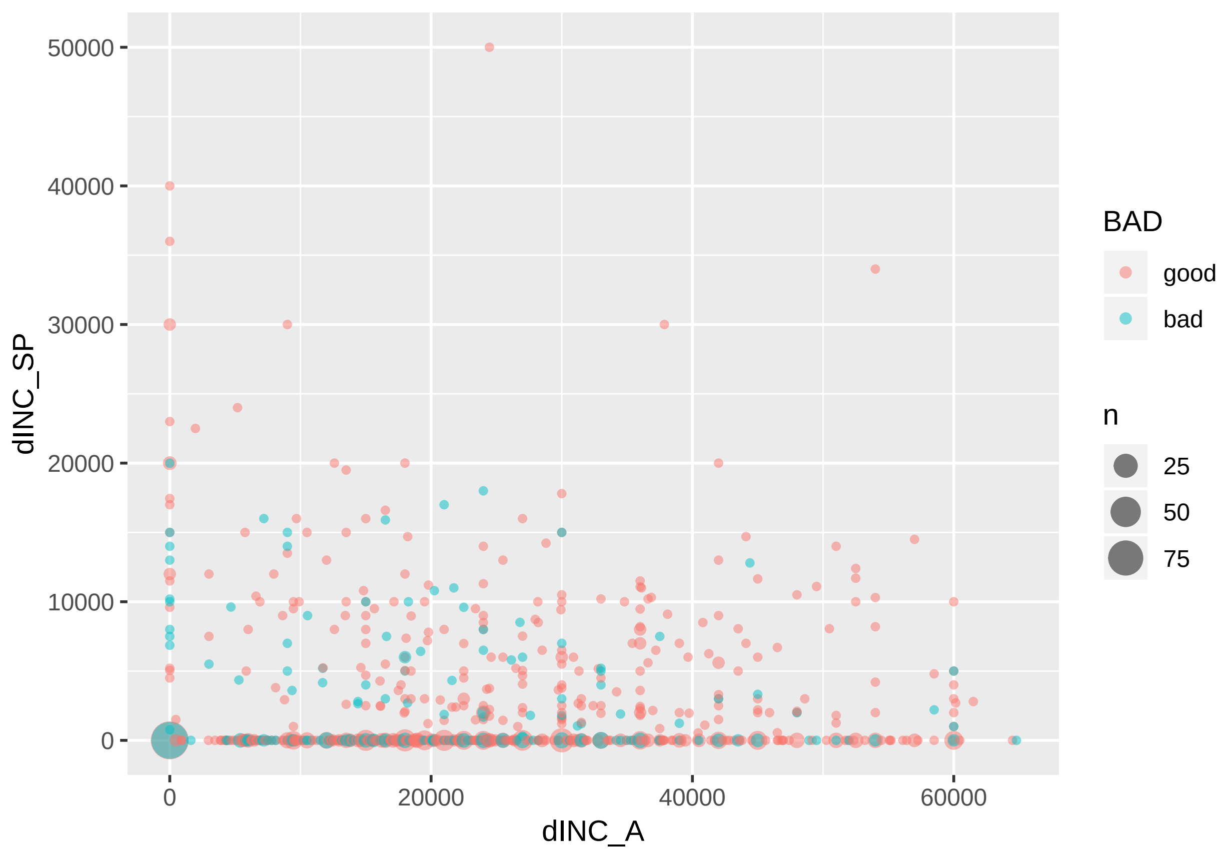
```
## What's the 3-way relation between applicant income, spouse income and default risk?
# Scatter plot by default status
ggplot(data=loans, aes(x=dINC_A, y=dINC_SP, color=BAD)) + geom_point()
```



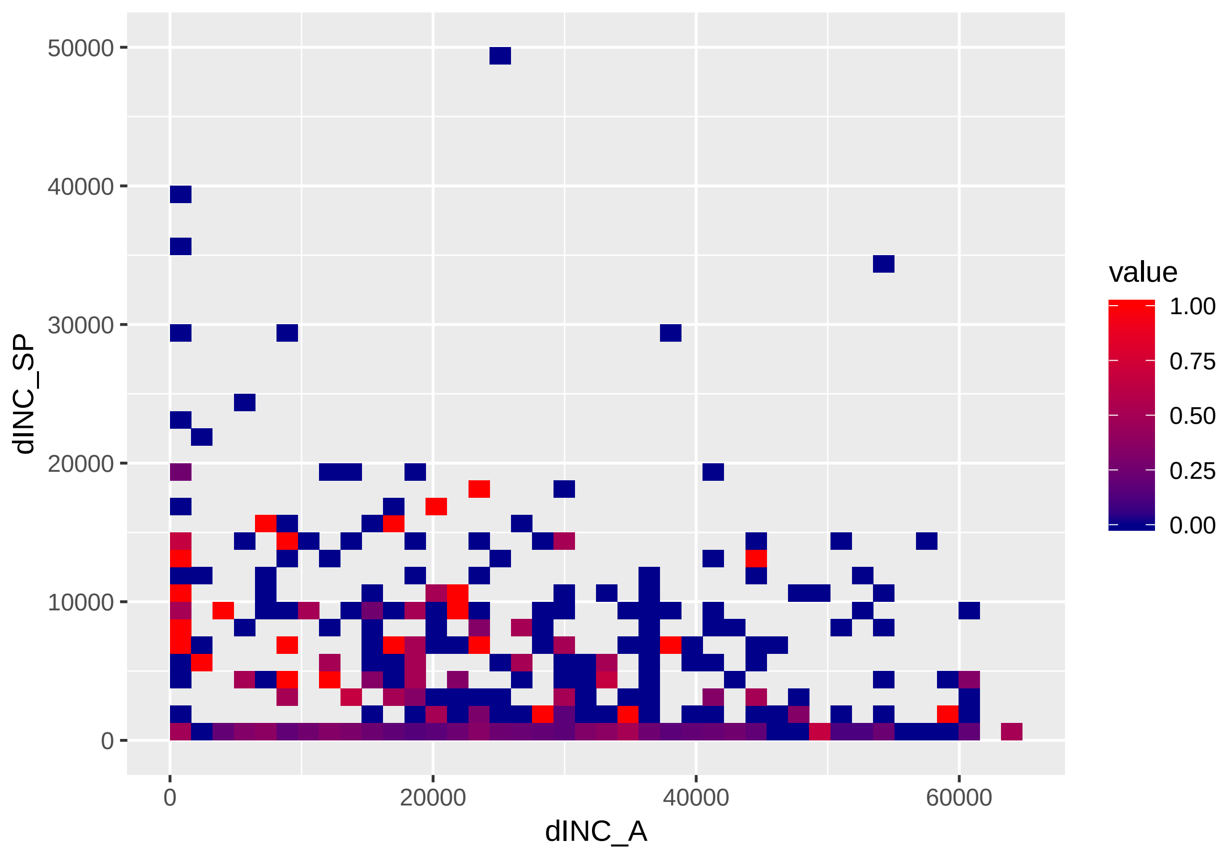
```
# ...or as a quick plot
qplot(data=loans, x=dINC_A, y=dINC_SP, color=BAD)
```



```
# We can't really see what's going on due to overplotting (read: too many damn points)
# Let's try plotting observation in the same area as bigger dots and make them transparent (alpha = 0.5
ggplot(data=loans, aes(x=dINC_A, y=dINC_SP, color = BAD)) + geom_count(alpha=0.5)
```

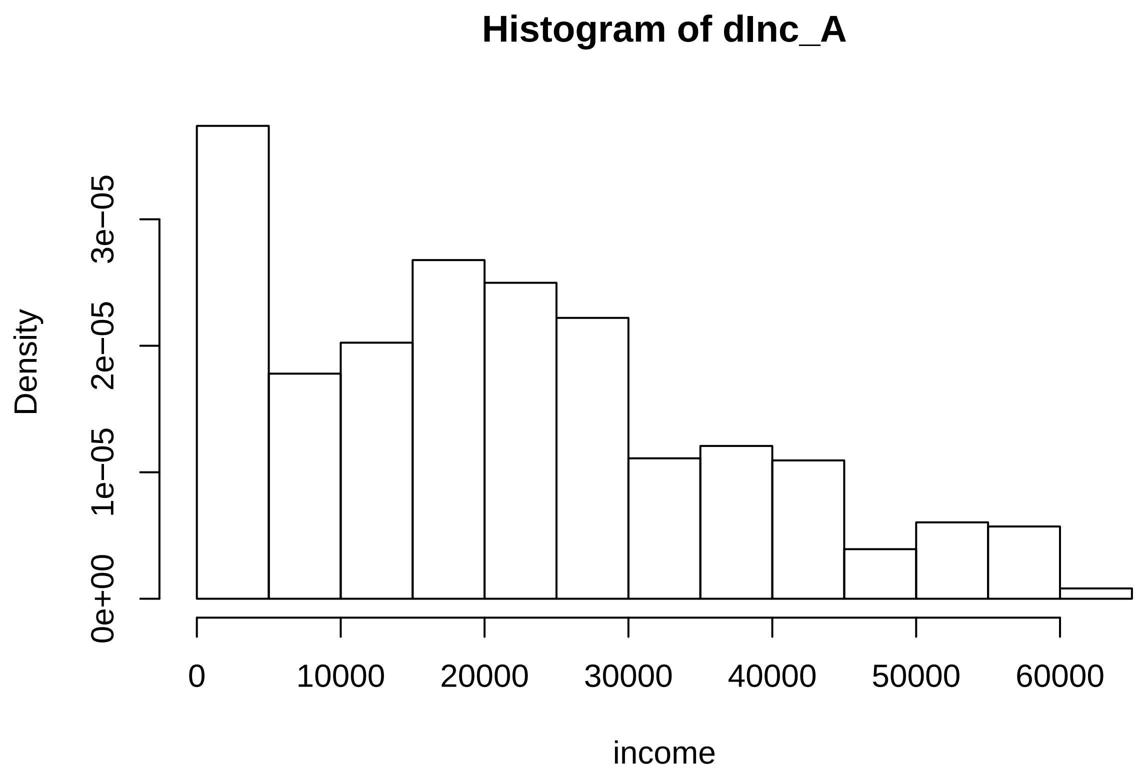


```
# Still not good, let's try to plot the average default rate
loans$BAD_binary <- as.numeric(loans$BAD == 'bad')
ggplot(data=loans, aes(x=dINC_A, y=dINC_SP, z=BAD_binary)) + stat_summary_2d(bins=40, fun="mean") + scal
```

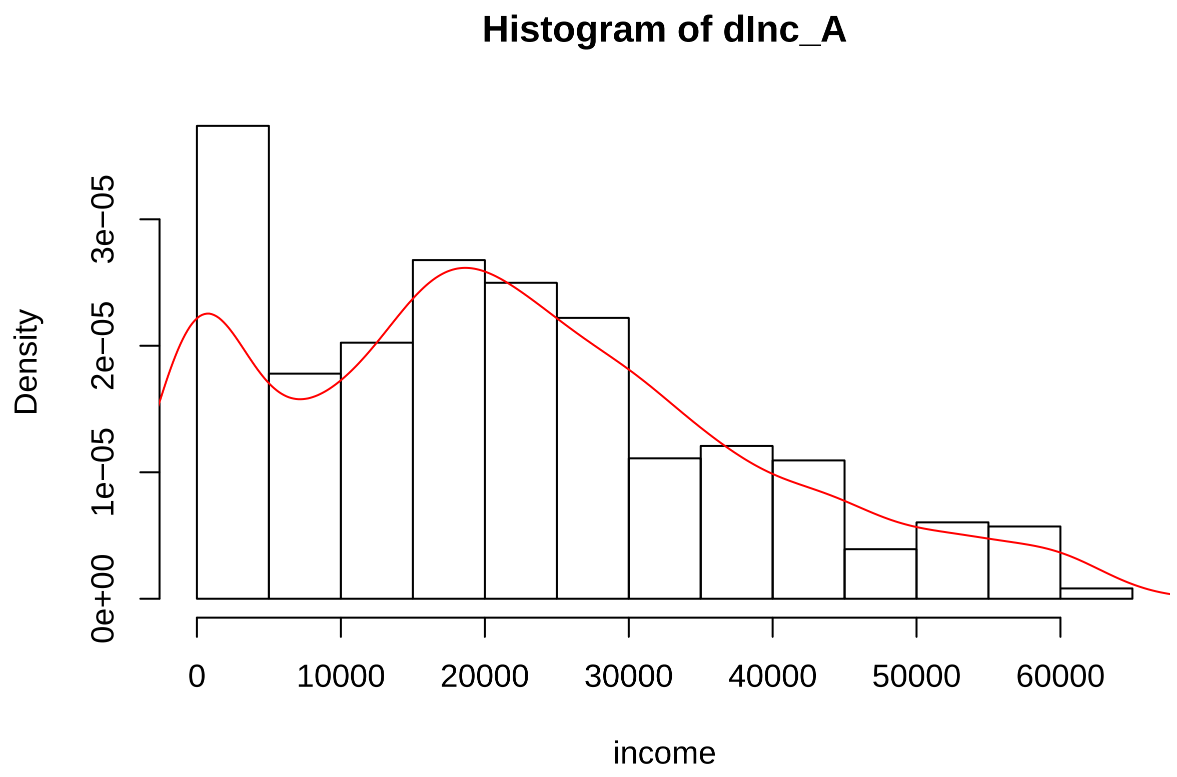


```r
# We didn't pick green to red because ~4% of the population are red-green color blind

#According to the density curve, the variable does not look normally distributed.
hist(loans$dINC_A, probability = TRUE, main = "Histogram of dInc_A", xlab = "income")
```

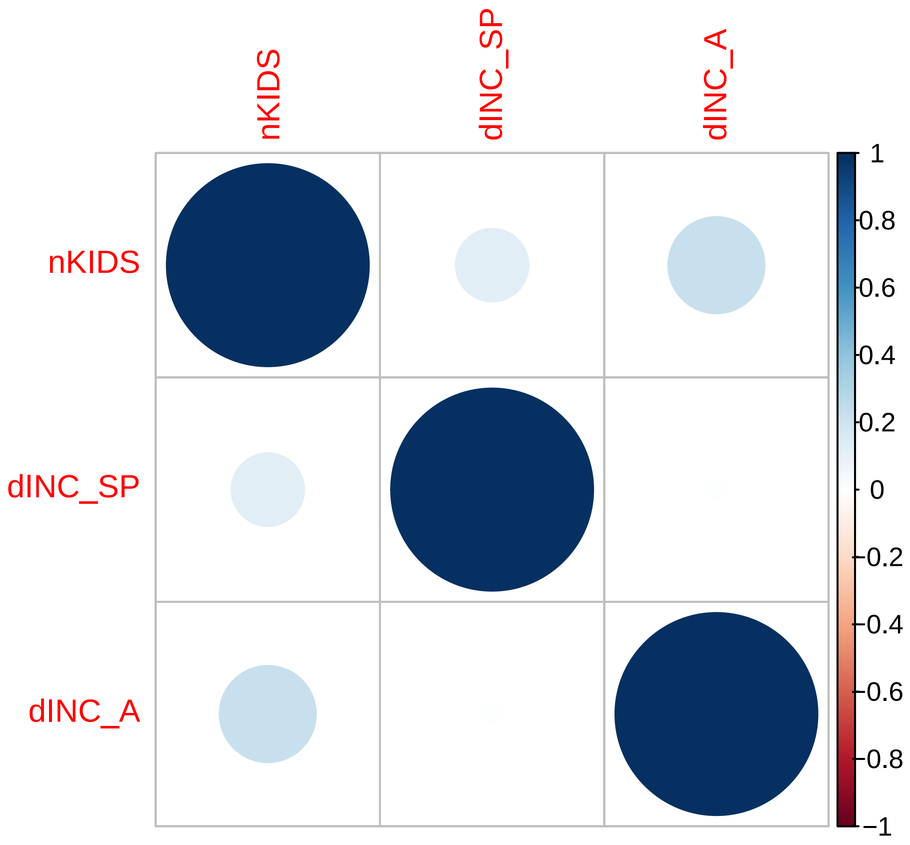


```r
lines(density(loans$dINC_A), col="red")
```



Another important piece of information lies in correlations. Fortunately, there's a package for that and it's called *corrplot*. Let's calculate the correlation between **dINC_A**, **dINC_SP** and **nKIDS** and plot it. Be careful to interpret correlation only as a linear relation between the variables.

```
library(corrplot)
cor <- cor(loans[,c(2,5,7)])
corr_graph <- corrplot(cor)
```



3. Regression

Regression is one of the standard models in predictive analytics, modeling the outcome of a variable by a linear combination of independent variables (the regression part). Most model functions specify the predictors by argument **x**, the target variable by argument **y**. Since we usually have one data frame that contains both our target variable and the predictive variables, many functions also accept a *formula* argument. A formula argument is entered with a ~ with the target on the left side and the predictor variables on the right, like this $y \sim x1 + x2 + x3$.
There are several convenient abbreviations that you can find with **?formula**. The most common are the dot **.** as a placeholder for "all other variables" and **-** to specify which variables to exclude, e.g. $y \sim . -x1 -1$ for a model without $x1$ and the intercept.

1. Run a regression to predict the income of a person **dINC_A** based on all other variables except variable **BAD** in the data set. Search for the appropriate function to create the model and save the trained model as **lr** (for linear regression).
2. Use the trained model **lr** to predict the income of each applicant in the **loans** data frame. Use function **predict()** and store the predictions in a variable **pred.lr**
3. Which variables have a significant impact on the predicted income of an applicant? Be careful when making any causal interpretations here. Its the income that casually determines the value of the house rather than the other way around!

```r
# Create your model
lr <- lm(dINC_A ~.-BAD, data = loans)
# Look into what happens to the data inside the function
# I found this by looking into the lm function
prepData <- model.matrix(dINC_A ~ .-BAD, data = loans)
```

```r
head(prepData)
```

```
##   (Intercept) YOB nKIDS nDEP PHON dINC_SP EMPS_AE EMPS_AM EMPS_AN EMPS_AP
## 1           1  19     4    0    1       0       0       0       0       0
## 2           1  41     2    0    1       0       0       0       0       1
## 3           1  66     0    0    1       0       0       0       1       0
## 4           1  51     2    0    1       0       0       0       0       1
## 5           1  65     0    0    1       0       0       0       0       1
## 6           1  42     2    0    1   10500       1       0       0       0
##   EMPS_AR EMPS_AT EMPS_AU EMPS_AV EMPS_AW EMPS_AZ RESN RESO RESP RESU
## 1       1       0       0       0       0       0    0    1    0    0
## 2       0       0       0       0       0       0    0    1    0    0
## 3       0       0       0       0       0       0    1    0    0    0
## 4       0       0       0       0       0       0    0    1    0    0
## 5       0       0       0       0       0       0    0    0    1    0
## 6       0       0       0       0       0       0    0    1    0    0
##   dHVAL  dMBO dOUTM dOUTL dOUTHP dOUTCC YOB_missing BAD_binary
## 1 14464     4     0     0      0      0           0          0
## 2     0     0   280   664      0     80           0          0
## 3     0     0     0     0      0      0           0          0
## 4 24928  8464   584   320      0     60           0          0
## 5     0     0     0     0      0      0           0          0
## 6 43392 46464  1120     0      0      0           0          0
```

```r
# The factors have been split up into a set of binary dummy variables automatically!

# Check the model results by printing the summary
summary(lr)
```

```
## 
## Call:
## lm(formula = dINC_A ~ . - BAD, data = loans)
## 
## Residuals:
##    Min     1Q Median     3Q    Max 
## -39895  -7079   -685   6328  51957 
## 
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  2.383e+04  3.484e+03   6.839 1.27e-11 ***
## YOB         -7.326e+01  3.894e+01  -1.881 0.060154 .  
## nKIDS        6.676e+02  3.796e+02   1.759 0.078874 .  
## nDEP         1.977e+03  1.596e+03   1.239 0.215697    
## PHON         3.164e+03  1.225e+03   2.582 0.009931 ** 
## dINC_SP     -3.415e-01  8.164e-02  -4.183 3.08e-05 ***
## EMPS_AE     -7.619e+03  2.464e+03  -3.092 0.002034 ** 
## EMPS_AM     -2.845e+03  3.370e+03  -0.844 0.398789    
## EMPS_AN     -4.595e+02  5.415e+03  -0.085 0.932381    
## EMPS_AP     -3.847e+03  2.265e+03  -1.698 0.089700 .  
## EMPS_AR     -1.725e+04  2.814e+03  -6.132 1.17e-09 ***
## EMPS_AT     -1.930e+04  2.532e+03  -7.624 4.99e-14 ***
## EMPS_AU     -1.925e+04  4.833e+03  -3.983 7.20e-05 ***
## EMPS_AV     -1.789e+03  2.343e+03  -0.764 0.445285    
## EMPS_AW     -2.424e+04  3.080e+03  -7.871 7.85e-15 ***
## EMPS_AZ     -2.033e+04  5.338e+03  -3.808 0.000147 ***
## RESN         2.496e+03  1.920e+03   1.300 0.193725
## RESO         5.169e+03  1.568e+03   3.296 0.001008 **
## RESP         6.214e+02  1.366e+03   0.455 0.649246
## RESU        -1.127e+02  1.532e+03  -0.074 0.941386
## dHVAL        4.174e-02  2.409e-02   1.733 0.083430 .
## dMBO         3.181e-02  2.478e-02   1.284 0.199546
## dOUTM        6.959e+00  9.879e-01   7.045 3.13e-12 ***
## dOUTL        1.177e-02  4.154e-01   0.028 0.977392
## dOUTHP       1.049e+01  2.928e+00   3.581 0.000356 ***
## dOUTCC       2.872e+00  2.162e+00   1.328 0.184406
## YOB_missing  1.392e+02  5.225e+03   0.027 0.978742
## BAD_binary  -3.782e+03  8.037e+02  -4.706 2.82e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11990 on 1197 degrees of freedom
## Multiple R-squared:  0.4438, Adjusted R-squared:  0.4313
## F-statistic: 35.38 on 27 and 1197 DF,  p-value: < 2.2e-16
```

```
# Coefficients
coef(summary(lr))
```

```
##                  Estimate   Std. Error     t value     Pr(>|t|)
## (Intercept)  2.382913e+04 3.484216e+03  6.83916615 1.267317e-11
## YOB         -7.326301e+01 3.893971e+01 -1.88144751 6.015358e-02
## nKIDS        6.676449e+02 3.796113e+02  1.75875946 7.887387e-02
## nDEP         1.977467e+03 1.596395e+03  1.23870768 2.156966e-01
## PHON         3.164397e+03 1.225398e+03  2.58234142 9.931150e-03
## dINC_SP     -3.415335e-01 8.163841e-02 -4.18349083 3.080269e-05
## EMPS_AE     -7.619417e+03 2.464195e+03 -3.09205056 2.033751e-03
## EMPS_AM     -2.844669e+03 3.370111e+03 -0.84408759 3.987891e-01
## EMPS_AN     -4.595197e+02 5.414558e+03 -0.08486744 9.323810e-01
## EMPS_AP     -3.847494e+03 2.265422e+03 -1.69835664 8.970036e-02
## EMPS_AR     -1.725498e+04 2.813732e+03 -6.13241522 1.173784e-09
## EMPS_AT     -1.930109e+04 2.531622e+03 -7.62400130 4.987796e-14
## EMPS_AU     -1.925097e+04 4.832712e+03 -3.98347141 7.201291e-05
## EMPS_AV     -1.789212e+03 2.343273e+03 -0.76355235 4.452845e-01
## EMPS_AW     -2.424285e+04 3.080171e+03 -7.87061891 7.852810e-15
## EMPS_AZ     -2.033006e+04 5.338112e+03 -3.80847325 1.469197e-04
## RESN         2.496152e+03 1.919571e+03  1.30036990 1.937245e-01
## RESO         5.168968e+03 1.568112e+03  3.29629954 1.008412e-03
## RESP         6.213863e+02 1.365911e+03  0.45492444 6.492461e-01
## RESU        -1.126837e+02 1.532218e+03 -0.07354288 9.413864e-01
## dHVAL        4.174010e-02 2.409144e-02  1.73256993 8.342971e-02
## dMBO         3.180892e-02 2.478190e-02  1.28355443 1.995463e-01
## dOUTM        6.959029e+00 9.878490e-01  7.04462848 3.130087e-12
## dOUTL        1.177337e-02 4.153715e-01  0.02834419 9.773924e-01
## dOUTHP       1.048657e+01 2.928442e+00  3.58093623 3.560884e-04
## dOUTCC       2.871927e+00 2.162470e+00  1.32807698 1.844059e-01
## YOB_missing  1.392453e+02 5.224710e+03  0.02665129 9.787423e-01
## BAD_binary  -3.782367e+03 8.037150e+02 -4.70610464 2.819146e-06
```

```
## Model prediction
# Remember that it is bad practice to train and test a model on the same data! Think about why.
# We will discuss this later in class.
pred.lr <- predict(lr, newdata = loans)
summary(pred.lr)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   -7953   15661   22304   21244   28953   49118
```

```
# Income statistics for comparison
summary(loans$dINC_A)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0    9000   19500   21244   30600   64800
```

```
## Mean absolute error
MAE <- mean(abs(loans$dINC_A - pred.lr))
MAE
```

```
## [1] 8840.065
```