Introduction to R for data analysis

In this session we are going to learn how to open a data file in R and how to obtain some of the summary statistics described during the presentation. We will use the data set `lcfs_2013.dta`. The data contains information from a random sample of N=5144 households in the UK Living Costs and Food Survey, 2013 (LCF), available from the UK Data Archive. LCF constitutes real data which are used by government, business and other organisations. You should download the file `lcfs_2013.dta` and save it in a convenient location.

Setting up the working directory

Let's start by changing the *working directory* to the location in our computer where our data set is stored. We do this with the command `setwd`, as in the following example.

```
setwd("D:/Dropbox/Teaching/Manchester/methods@manchestre/labs_1")
```

You can perform this operation also by pressing `ctrl+shift+h` in RStudio or selecting `Session/Set Working Directory/Choose Directory.../` in the menus. Either of these actions will open a navigation window where you can select the working directory.Once this is done we can load the data.

Loading Data.

Data can come in many different formats and structures. The type of data we are going to handle most of the time will be stored as a table, with columns representing variables and rows representing observations. The easiest files to read into R are human-readable text files with variables separated by a character such as a comma `,`, a semicolon `;`, quotations `"`, tabs, spaces and so on. These files normally include a "header" containing the names of the different variables. For this simple type of file we can use the command `read.table`, as follows:

```
frame <- read.table("lcfs_2013.csv", sep =",", header=T)
```

In the preceding expression `<-` is the assignment operator. Therefore the preceding command is telling R to load the specified data set and assign to it a name, which in this case I have chosen to be *frame* (you could choose your own name, but the general rule is that you choose a short and simple to remember name). More precisely you are allocating your data to an *object*. In other software and computer languages, the assignment operator is `=` or `==`. You could use these in R too, but the convention is to use `<-`.

The command `read.table` takes a number of arguments. The first is the name of the file you want to load, `lcfs_2013.csv` in our case. The second argument defines the separator, that is the character separating columns of data. In a comma-separated-value file, `sep` is `,`. The final argument tells R if the data file has a header containing the names of the variables.

Libraries and other data formats.

Often you will be importing data specifically stored for alternative software, such as Excel or Stata. In such cases, R has specific commands to load the data. However, these commands are not loaded by default when you run R at the beginning. To be able to use these specialised commands you have to load the corresponding **library**. The same will be the case when using specialised estimators and functions.

To load Stata files into R, we need the library `haven`. To load this (or any other) library in R, you use the command `library()` and place the name of the desired library within the brackets.

```
library(haven)
```

The library `haven` is designed to allow you to import files in Stata, SPSS or SAS format. Once the library is loaded you use `read_dta()` to load Stata files, whereas `read_sas()` and `read_spss()` do the same for SAS and SPSS files.

```
frame <- read_dta("lcfs_2013.dta")
```

Decriptive analyses.

Once you have loaded your data, you can proceed to undertake a descriptive analysis. This achieved simply by typing,

```
summary(frame)
```

You can also calculate sample means, variances, and so on individually. This is relatively straightforward. For example, lets obtain these statistics for the variable `P550tpr` which collects household's weekly expenditure:

```
summary(frame$P550tpr)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   30.52  254.13  419.90  479.76  644.97 1175.00
```

Note how we have referred to the variable: we first wrote the name of the object where the data is stored (`frame` in our example). Then we use the character `$` and then we write the actual name of the variable. If we had not specified any variable, R would have produced descriptive statistics for all the variables in the data set.

We can calculate the variance and standard deviation of a variable using the commands `var()` and `sd()` respectively. and

```
sd(frame$P550tpr)
```

```
## [1] 292.3652
```

```
var(frame$P550tpr)
```

```
## [1] 85477.43
```

The correlation coefficient between two variables can be calculated using `cor()`. For instance, the variable `P344pr` in our data collects household income

```
cor(frame$P550tpr, frame$P344pr)
```

```
## [1] 0.7064488
```

We can also obtain the skewness and kurtosis for each variable. For this we need to load an additional library called "moments". This library has two functions, `skewness` and `kurtosis` which produce the corresponding statistics. Now, this library is not installed in R by default, so we need to import it from R's online repository. To do that, we use the command `install.packages()`, by writing

```
install.packages("moments")
```

Once installed, we can load the package and obtain the relevant statistics

```
library(moments)
skewness(frame$P550tpr)
```

```
## [1] 0.8262061
```

```
kurtosis(frame$P550tpr)
```

```
## [1] 2.937132
```

Quite often, you will want to produce descriptive statistics by group. For instance, you might want to obtain sample means by gender, ethnic group, treatment group, country, etc. For this, you can use the command `tapply()` -a very general function which can apply a function to a group of variables by categories; you can find out more about it by searching its help file:

```
help(tapply)
```

For the purpose of obtaining descriptive statistics by group, the syntax is `tapply(variable, factor, summary)` where `variable` is the variable you want to summarize, factor is the variable defining the categories and `summary` is the function `summary()` that we introduced earlier in this note. For example in the `lcfs_2013` data set we can analyse expenditure by gender (gender is defined by the variable `SexHRP`)

```
tapply(frame$P550tpr, frame$SexHRP, summary)
```

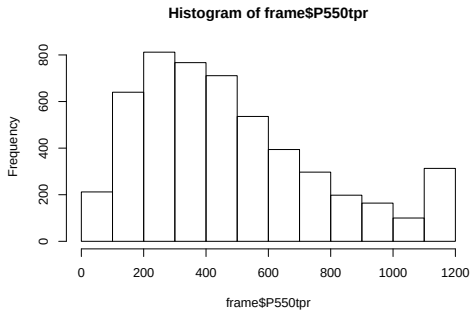
```
## $`1`
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   33.86  301.16  472.05  529.09  704.74 1175.00
##
## $`2`
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   30.52  202.60  333.54  401.44  539.40 1175.00
```

In the above the category 1 corresponds to male and 2 to female.

Plotting data

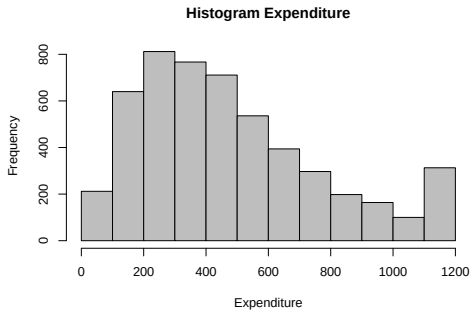
R has a reputation for its ability to produce graphs. There are indeed several powerful libraries which you can use to create visually appealing graphs (`ggplot2` in particular). However, you can achieve a lot with R's default commands.

```
hist(frame$P550tpr)
```



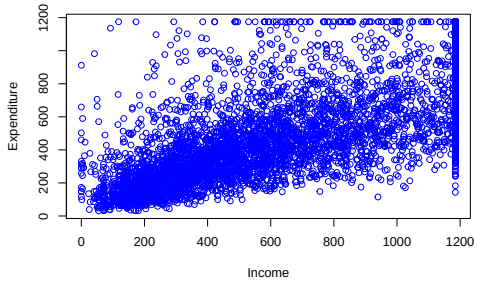
You can change the attributes of the graph by defining further options in the `hist()` command. To define a title, add the option `main="title of the graph"`. To change the X and Y labels, use `xlab="X title"` and `ylab="y title"` respectively. You might even want to change the color filling the bars. This can be done by adding the option `col`:

```
hist(frame$P550tpr, main="Histogram Expenditure",
     xlab="Expenditure", ylab="Frequency", col="grey")
```



Scatter plots can be produced equally easy using `plot(x-var, y-var, options...)`

```
plot( frame$P344pr,frame$P550tpr, xlab="Income", ylab="Expenditure", col="blue")
```



This all does the deed (as MacBeth told to Lady MacBeth), but... clunky... In recent years the `ggplot2' library (part of the`tidyverse`library), as become a popular choice to produce more eye-catching results. Install the`tidyverse`package and then load the library (which also included`plyr`', a handy package for data manipulation)

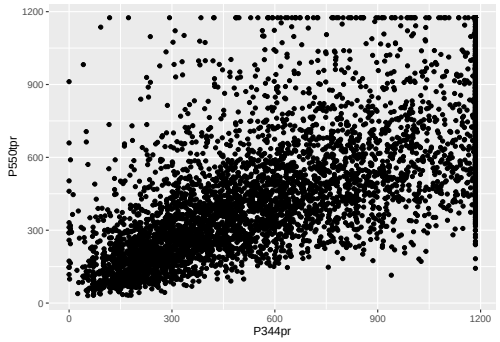
```
library(tidyverse)
```

The idea underlying ggplot2 is simple: you specify the data frame and variables to use with 'ggplot(dataframe, aes(x=..., y=...))'. Then, you specify a *geometry*, that is, the type of graph. There are various options,

1. The command `geom_histogram(fill = colorOrTheme)` produces a histogram, with `fill` determining the color of the bars
2. `geom_point(color= colorOrTheme)` produces a scatter plot, with `color` determining the color of the dots
3. `geom_smooth()` produces a *smooth* estimator of the conditional mean of two variables
4. `geom_density()` produces an estimate of the density function of a continuous variable

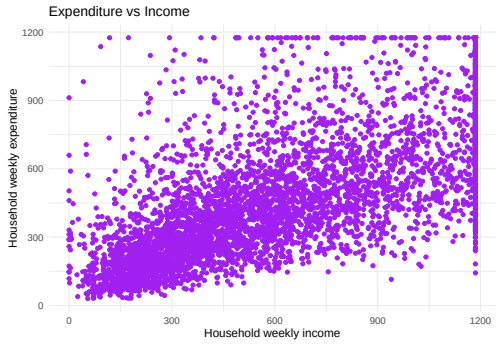
there are many more *geometries*. To produce the previous scatterplot on its own, just write,

```
ggplot(frame, aes(y=P550tpr,x=P344pr)) + geom_point()
```



The final component of any `ggplot` are all those features that make graphs more readable. These include, titles, subtitles, color themes, background colors, labels and so on. For starters, the grey background is not particularly appealing. We can over-ride this using the `theme_minimal()' option. We can then also replace the black dots for something more cheerful, say`purple. We can add a title `with`ggtitle("your title"), `labels to the axis with`xlab("your label")`and`ylab("your label")', as in this example,

```
ggplot(frame, aes(y=P550tpr, x=P344pr))+
   geom_point(color="purple")+
   theme_minimal()+
   xlab("Household weekly income")+
   ylab("Household weekly expenditure")+
   ggtitle("Expenditure vs Income")
```



We might also want to add a density estimator of household expenditure, after all the latter is a continuous variable. But, rather than producing separate graphs, we might want to align the density estimator and the scatter plot with each other. No problem, we can use `plot_grid(graph1, graph2)`, from the library `cowplot`

. For this we need to store the separate graphs in objects,

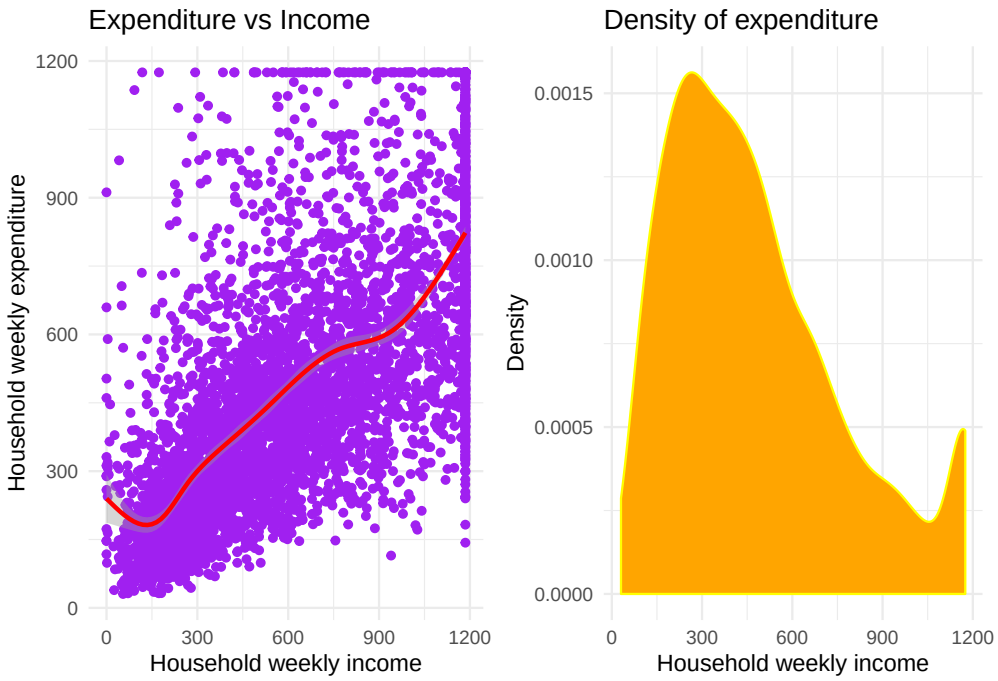
```
library("cowplot")

graph1<- ggplot(frame, aes(y=P550tpr, x=P344pr))+
   geom_point(color="purple")+
   theme_minimal()+
   xlab("Household weekly income")+
   ylab("Household weekly expenditure")+
   ggtitle("Expenditure vs Income") +
   geom_smooth(color = "red")

graph2<- ggplot(frame, aes(x=P550tpr))+
   geom_density(color="yellow", fill="orange")+
```

```
   theme_minimal()+
   xlab("Household weekly income")+
   ylab("Density")+
   ggtitle("Density of expenditure")

plot_grid(graph1, graph2)
```



Note how we added a `smooth` *geometry* to the scatter plot.

Data handling with `dplyr`

The `dplyr` library helps do a number of data transformations quickly; it also allows us to do analysis by categories easily. We we learn about these capabilities as we go. Quite often higher efficiency and code transparency is obtained by using the *pipe* operator, `%>%`, which can be read as "then apply". This operator is part of the `tidyverse` package -though originally developed for handling regular expressions with the package `maggrittr`. For example, to summarise the data set, rather than writing `summary(frame)`, you could do,

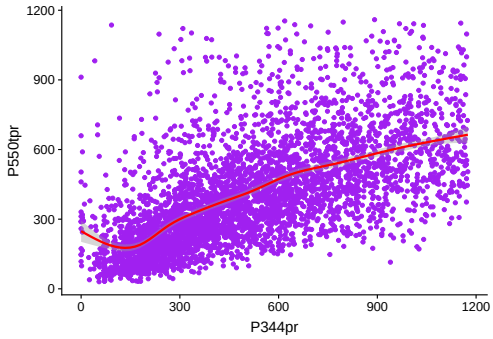
```
frame %>% summary()
```

which reads *first pick up the data in frame, then apply* ***`summary()`*** *to the data.* The first thing we can use `dplyr` for is to summarise data by categories. For instance, the variable `A049r` is the household size (number of individuals). We coud compute the average income by household size, using `dplyr`'s `group_by(var)`, where `var` must be a factor (categorical) variable,

```
frame %>% group_by(A049r) %>% summarise(meanIncome = mean(P344pr))
```

Looking at the scatter plot, we see that many observations have been censored at 1175; these are units with income/expenditure above that level. We could repeat the graph without these observations using `dplyr`'s `filter` command,

```
 frame %>% filter(P344pr<1175 & P550tpr <1175 ) %>% ggplot( aes(y=P550tpr, x=P344pr))+
  geom_point(color="purple")+
  geom_smooth(color="red")
```



Note that `filter` does not remove the filtered observations from the dataset. The ampersand, `&` inside `filter` tells `R` that it should filter (keep) observations *if P344pr<1175 AND P550tpr*.

#Regression

Running a regression in R is done using the command `lm(dependent~independent)` or `lm(dependent~independent, data=nameFile)`. Here `lm` standards for "linear model". The second version simplifies the task in so far, by identifying the object containing the data (`nameFile`) you can write the names of the variables directly (rather than using the `nameFile$variableName` format). To illustrate, consider the estimation of the simple regression model of expenditure on income,

```
lm(frame$P550tpr~frame$P344pr)
```

```
##
## Call:
## lm(formula = frame$P550tpr ~ frame$P344pr)
##
## Coefficients:
##  (Intercept)  frame$P344pr
##     122.9632        0.5751
```

```
lm(P550tpr~P344pr, data=frame)
```

```
##
## Call:
## lm(formula = P550tpr ~ P344pr, data = frame)
##
## Coefficients:
## (Intercept)       P344pr
##    122.9632       0.5751
```

`lm` prints only the estimated coefficients, but it estimates many other quantities that will be of interest to us. So to preserve all the results that `lm` calculates, we are going to assign all these results to another object, say `expend.lm`,

```
expend.lm <- lm(P550tpr~P344pr, data=frame)
```

Note that R does not print any output when the `lm` is assigned to an object. To extract the parameter values, we use `coefficients`

```
coeffs<-coefficients(expend.lm)
coeffs
```

```
## (Intercept)      P344pr
## 122.9632419   0.5750739
```

In the preceding box, the first line stores the OLS coefficients in an object, `coeffs` (you could give it any other name). Then the second line tells R to print the values stored in `coeffs`.

Prediction

You could now obtain some predicted values. For example suppose you want to predict expected expenditure when income is 100 a week. To do so is simple. If the estimated regression is $\hat{y} = 122.96 + 0.5751X$ then you only need to replace X by 100, which in R is done as follows:

```
predictedExpenditure = coeffs[1]+coeffs[2]*100
predictedExpenditure
```

```
## (Intercept)
##    180.4706
```

When we stored the coefficients of the OLS regression into the object `coeffs`, R stored each of these in a list. In that list, the first value is the coefficient of the intercept, and you can refer to it using `coeffs[1]` where the number in brackets refers to the *first element in the list* `coeffs`. The second value is the coefficient of expenditure and you can refer to it using `coeffs[2]`. If there were additional regressors, $X_3, X_4, \ldots, X_K$, these would be stored in the order you introduced them in the `lm` command and then, to use the k coefficient in later calculations you would use `coeffs[k]`. That explains the syntax in the first line of the preceding box. It is simply telling R to take the coefficient of the intercept, `coeffs[1]`, and add 100 times the coefficient of `coeffs[2]`. The second line simply tells R to print the result of that operation.

Running regressions for subsets of data

You will often want to run regressions for subsets of the sample, perhaps defined by a range of values of some variable or a category. In this case, we add the option `subset=(condition)` to `lm()`. The `condition` will be specific to each application. For example, to obtain a regression of expenditure on income for households with income between 400 and 700, you would type

```
lm(P550tpr~P344pr, data = subset(frame, P344pr >400 & P344pr <700))
```

```
##
## Call:
## lm(formula = P550tpr ~ P344pr, data = subset(frame, P344pr >
##     400 & P344pr < 700))
##
## Coefficients:
## (Intercept)       P344pr
##    113.8824       0.6169
```

But, of course, we could have used pipes instead,

```
 frame %>% filter(P344pr>400 & P344pr <700 ) %>%lm(P550tpr~P344pr, data =.)
```

```
##
## Call:
## lm(formula = P550tpr ~ P344pr, data = .)
##
## Coefficients:
## (Intercept)       P344pr
##    113.8824       0.6169
```

Notice what is going on here: the last argument in `lm` sets `data = .`, not `data = frame`; here `.` refers to the filetered data.

Residuals and Predicted Values

`lm` automatically calculates the residuals $e_i = Y_i - \hat{\beta}_0 - \hat{\beta}_1 X$ and predicted values $\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X$ for each observation. You can retrieve these values using the commands `residuals()` and `fitted.values()`, as follows

```
ehat <- residuals(expend.lm)
yhat <- fitted.values(expend.lm)
```

Then you can verify a number of properties that OLS has. For example, the mean of the predicted values equals the mean of the dependent variable, whereas the mean of the residuals is exactly 0,

```
summary(yhat)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   123.0   303.6   446.8   479.8   656.7   804.4
```

```
summary(frame$P550tpr)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   30.52  254.13  419.90  479.76  644.97 1175.00
```

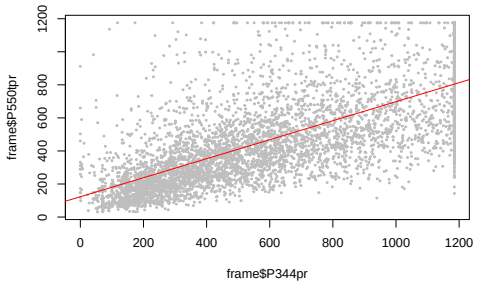
```
summary(ehat)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -661.37 -126.61  -36.88    0.00   93.20  984.11
```

Note, however, that the overall distribution of the predicted values is not equal to the distribution of the dependent variable (you can see that from the quantiles of the distribution).

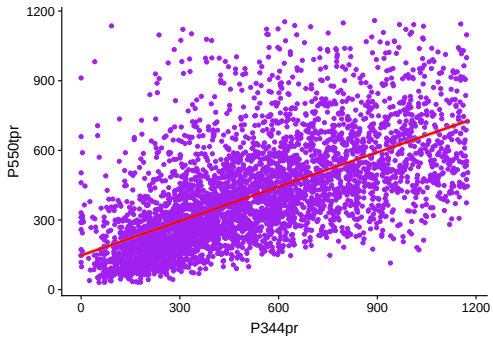
One of the things you can do after estimating a simple regression is to plot your estimated model over a scatter of the original data in order to see how the model fits the overall data set. To do so, we need to store the coefficients from the regression in an object, then draw a scatter plot and finally add the regression line using the command `abline(intercept, slope)`. The latter command adds straight lines to a plot by specifying the values of the intercept and the slope parameters.

```
plot( frame$P344pr,frame$P550tpr,  col="grey", pch = 20, cex=.5)
abline(coeffs[1], coeffs[2], col="red")
```



But with `ggplot` things are far easier...

```
frame %>% filter(P344pr<1175 & P550tpr <1175 ) %>% ggplot( aes(y=P550tpr, x=P344pr))+
 geom_point(color="purple")+
 geom_smooth(color="red", method="lm")
```



Multiple regression in R is an straightforward application of `lm`. Suppose you want to estimate a regression of expenditure on income as well as gender. Then you add the new regressor as follows

```
lm(P550tpr~P344pr+SexHRP, data=frame)
```

```
##
## Call:
## lm(formula = P550tpr ~ P344pr + SexHRP, data = frame)
##
## Coefficients:
## (Intercept)       P344pr       SexHRP
##    157.9892       0.5676     -21.9048
```

More generally, if you have k variables, you would write something along these lines

```
lm(dependent~var1+var2+var3+...+vark, data=name)
```

#Testing hypotheses

Testing hypotheses with R is relatively straightforward, at least at an elementary level. The tests one is often most concern with are those about $H_0 : \beta_k = 0$, for some β_k in the assumed population model $Y = \beta_0 + \beta_1 X_1 + \ldots \beta_K X_K + u$. These tests are automatically provided by R. For example, let's regress expenditure on income, a dummy variable taking value 1 if an observation comes from London (Gorx=7), and another dummy variable taking value 1 if the respondent is a man. In our data, the respondent's gender is recorded in the variable `SexHRP`. Men have been coded with value 1 and women with value 2. So, we need to create a new dummy variable for "man" as well.

```
frame$London <- with(frame, Gorx==7)
summary(frame$London)
```

```
##    Mode   FALSE    TRUE
## logical    4664     480
```

```
frame$man <- with(frame, SexHRP==1)
summary(frame$man)
```

```
##    Mode   FALSE    TRUE
## logical    1988    3156
```

```
frame.lm <- lm(frame$P550tpr ~ frame$P344pr + frame$London+frame$man)
summary(frame.lm)
```

```
##
## Call:
## lm(formula = frame$P550tpr ~ frame$P344pr + frame$London + frame$man)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -642.40 -126.32  -36.49   93.78  994.57
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)      1.134e+02  6.296e+00  18.018  < 2e-16 ***
## frame$P344pr     5.671e-01  8.308e-03  68.261  < 2e-16 ***
## frame$LondonTRUE 1.001e+01  9.925e+00   1.009 0.313108
## frame$manTRUE    2.205e+01  6.119e+00   3.604 0.000316 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 206.7 on 5140 degrees of freedom
## Multiple R-squared:  0.5004, Adjusted R-squared:  0.5001
## F-statistic:  1716 on 3 and 5140 DF,  p-value: < 2.2e-16
```

You will recall from that the formal test for $H_0 : \beta_k = 0$ is given by $t = \hat{\beta}_k / s.e.(\hat{\beta}_k)$. You can compute each of the tests for each regressor in your model by taking the value under "Estimate" and dividing it by the corresponding value under "Std. Error". For example, the test for $H_0 : \beta_{P344pr} = 0$ is $\hat{t} = 5.671e-01/8.308e-03$. The value of this test is provided in the third column of output, under "t value". For the example at hand the value of the test is 68.261. To see if this value is or not significant, we focus on the "p value" of the test, given under the column `Pr(>|t|)`. We check if the p-value is below the significance level we have decided to work with. If that is the case, then we reject the null hypothesis and conclude that the coefficient of the variable is statistically significant. Therefore the variable is important to explain the variation in Y. In the example at hand the p-value associated with the coefficient of income (P344pr) is 2e-16 (which is computer language for 2 divided by 10 to the power 16 -basically 0). A p-value of 0 means that we reject the null hypothesis (and therefore income is a statistically significant regressor) at any significance level.

R further provides a visual aid to quickly assess if these tests are significant or not: the number of `*`. As described in the legend `***` means that the p-value is essentially 0; `**` means that the p-value is below 0.001 and so the coefficient would be significant at 0.1% significance level;`*` means that the p-value is below 0.01 but above 0.001 and so the coefficient would be significant at 1% significance level... and so on. No stars means that the p-value exceeds 0.1 and therefore the coefficient is not significant at 10% level. IN that case we would conclude that the associated variable cannot explain the variation in Y.

R further provides the F-test for the joint significance of the model, that is $H_0 : \beta_1 = \beta_2 = \ldots = \beta_K = 0$. The value and p-value of this test as given at the bottom of output. In the example above the F-test for this hypothesis equals 1716 with p-value smaller than 2.2e-16 (or 2.2 divided by 10^{16}, which again is essentially 0). So we would reject the null hypothesis at any significance level and would conclude that at least one of the variables in our model is relevant to explain Y.

Running an F-test to evaluate the joint significance of a subset of variables in a model requires a bit of extra work in R. This involves comparing the RSS (Residual Sum of Squares) of the full model (including all the variables) to the RSS of the restricted model (excluding the variables whose coefficients we think are equal to zero). To illustrate, consider again the regression model of expenditure on income, and the dummy variables for London and "man". Suppose that you want to test that the dummy variables are jointly irrelevant to explain Y. In other words, you want to test the null hypothesis that the coefficients of these variables are each zero, $H_0 : \beta_{London} = 0, \beta_{man} = 0$. To do this, we need to first estimate the full model (as we did above) and, second, estimate the "restricted model", that is the model without the variables whose joint significant we are testing (the dummies for London and man in this example).

```
frame.lm_unrestricted <- lm( P550tpr ~  P344pr +  London+ man, data=frame)
summary(frame.lm_unrestricted)
```

```
##
## Call:
```

```
## lm(formula = P550tpr ~ P344pr + London + man, data = frame)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -642.40 -126.32  -36.49   93.78  994.57
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.134e+02  6.296e+00  18.018  < 2e-16 ***
## P344pr      5.671e-01  8.308e-03  68.261  < 2e-16 ***
## LondonTRUE  1.001e+01  9.925e+00   1.009 0.313108
## manTRUE     2.205e+01  6.119e+00   3.604 0.000316 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 206.7 on 5140 degrees of freedom
## Multiple R-squared:  0.5004, Adjusted R-squared:  0.5001
## F-statistic:  1716 on 3 and 5140 DF,  p-value: < 2.2e-16
```

```
frame.lm_restricted <- lm( P550tpr ~  P344pr, data=frame )
summary(frame.lm_restricted)
```

```
##
## Call:
## lm(formula = P550tpr ~ P344pr, data = frame)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -661.37 -126.61  -36.88   93.20  984.11
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.230e+02  5.760e+00   21.35   <2e-16 ***
## P344pr      5.751e-01  8.035e-03   71.57   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 206.9 on 5142 degrees of freedom
## Multiple R-squared:  0.4991, Adjusted R-squared:  0.499
## F-statistic:  5123 on 1 and 5142 DF,  p-value: < 2.2e-16
```

R has stored the RSS for each model in `lcfs.lm_unrestricted` and `lcfs.lm_restricted`. The formal test can now be run using the command `anova()`,

```
anova(frame.lm_unrestricted, frame.lm_restricted)
```

```
## Analysis of Variance Table
##
## Model 1: P550tpr ~ P344pr + London + man
## Model 2: P550tpr ~ P344pr
##   Res.Df       RSS Df Sum of Sq      F   Pr(>F)
## 1   5140 219622691
## 2   5142 220214057 -2   -591366 6.9201 0.000997 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The critical information in the above table are the values under F (the value of the F test itself) and Pr(>F) (the corresponding p-value). In particular the p-value in this example is 0.000997, which means that we can confidently reject the null hypothesis at virtually any significance level. We conclude that at least one of the dummy variables (London, man or both) is significant for the analysis.

Robust Estimation

Let's run a regression of expenditure on income and dummies for London and man -just as above,

```
options(scipen=999)
library(haven)
lcfs <- read_dta("lcfs_2013.dta")
lcfs$London <- with(lcfs, Gorx==7)
summary(lcfs$London)
```

```
##    Mode   FALSE    TRUE
## logical    4664     480
```

```
lcfs$man <- with(lcfs, SexHRP==1)
summary(lcfs$man)
```

```
##    Mode   FALSE    TRUE
## logical    1988    3156
```

```
lcfs.lm <- lm(lcfs$P550tpr ~ lcfs$P344pr + lcfs$London+lcfs$man)
summary(lcfs.lm)
```

```
##
## Call:
## lm(formula = lcfs$P550tpr ~ lcfs$P344pr + lcfs$London + lcfs$man)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -642.40 -126.32  -36.49   93.78  994.57
##
## Coefficients:
##                   Estimate Std. Error t value             Pr(>|t|)
## (Intercept)     113.447623   6.296226  18.018 < 0.0000000000000002 ***
## lcfs$P344pr       0.567096   0.008308  68.261 < 0.0000000000000002 ***
## lcfs$LondonTRUE  10.012471   9.924923   1.009              0.313108
## lcfs$manTRUE     22.054692   6.119012   3.604              0.000316 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 206.7 on 5140 degrees of freedom
## Multiple R-squared:  0.5004, Adjusted R-squared:  0.5001
## F-statistic:  1716 on 3 and 5140 DF,  p-value: < 0.00000000000000022
```

Estimating a regression model in R with heteroskedasticity-robust standard errors can be done using the command `coeftest()`, after loading two libraries, `sandwich` and `lmtest`. The basic version of this command takes two arguments. The first is the name of the object where you have saved the output from a regression (in the preceding box, `lcfs.lm`). The second argument is a command, `vcov=vcovHC()`. This latter command takes also two arguments. The first one is, again, the name of the object where you have saved the output from a regression (in the preceding box, `lcfs.lm`). The second argument defines how the heteroskedasticity

robust standard errors are going to be calculated. We will use the option `HC1` by default here. All together, this is how you produce heteroskedasticity-robust standard errors.

```
library(sandwich)
library(lmtest)
coeftest(lcfs.lm, vcov=vcovHC(lcfs.lm, "HC1"))
```

```
##
## t test of coefficients:
##
##                     Estimate  Std. Error t value              Pr(>|t|)
## (Intercept)     113.4476226   5.4204898 20.9294 < 0.00000000000000022 ***
## lcfs$P344pr       0.5670958   0.0088875 63.8083 < 0.00000000000000022 ***
## lcfs$LondonTRUE  10.0124706  11.1064700  0.9015              0.367365
## lcfs$manTRUE     22.0546920   5.9684002  3.6952              0.000222 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

You can see how the standard errors in this second regression are larger than in the original model. This need not be always the case (the relationship between normal, heteroskedasticity-robust and clustered standard errors depends on a number of technical factors which are beyond the scope of this course).

The second type of standard-error correction we saw in the lecture was clustered standard errors. This is appropriate when individuals are sampled in groups or clusters (households, regions, classrooms, etc) such that, whereas the clusters can be assumed to be independent from each other, the behavior of individuals are likely to be correlated within a cluster. If this is the case, the normal standard errors reported by `lm` will likely be misleading (and so will be the tests of hypotheses built on those standard errors). To take into account the clustering in data and obtain more reliable standard errors we can use the command `coeftest` again, together with the option `cluster.vcov()`. The latter option takes two arguments. The first is the he name of the object where you have saved the output from a regression (`lcfs.lm` in the current example). The second argument is a variable (or variables) which define the clusters. To be able to implement the `cluster.vcov()` option, we need to install and load the library `multiwayvcov`.
If the library is not installed in R, run the command

```
install.packages("multiwayvcov")
```

In our example, we might be concerned about regional correlations in expenditure among observations, in which case we would want to obtain cluster-robust standard errors. The variable `Gorx` indicates the region from which each observation is coming from, therefore we could run the following command.

```
library(multiwayvcov)

coeftest(lcfs.lm, cluster.vcov(lcfs.lm, lcfs$Gorx))
```

```
##
## t test of coefficients:
##
##                    Estimate Std. Error t value              Pr(>|t|)
## (Intercept)     113.447623   4.423507 25.6465 < 0.00000000000000022 ***
## lcfs$P344pr       0.567096   0.010801 52.5028 < 0.00000000000000022 ***
## lcfs$LondonTRUE  10.012471   6.628481  1.5105                0.131
## lcfs$manTRUE     22.054692   4.868726  4.5299          0.000006036 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

On this particular occasion, the clustered-standard errors happen to be smaller than the heteroskedasticity-robust and normal standard errors. This is not always the case (and indeed, if the observations within a cluster are, as is generally the case, positive correlated, then the cluster-robust standard errors will normally

be larger than the normal and heteroskedasticity robust standard errors.) **WARNING**: Remember that cluster-robust standard errors will be reliable *only* if the number of clusters is sufficiently large. In the example above `Gorx` defines only 12 clusters. That is too a small number and therefore the results in the preceding table are likely to be misleading.