Applied Econometrics I, Lab 6

Merging and Appending Data, and Heteroskedasticity (Chapter 8)

Muhammad Salman Khalid

Contents

1 Packages for This Lab **1**

2 Part 1. Combining Data Sets **1**
- 2.1 Primary Keys . . . 2
- 2.2 Loading the Data Sets . . . 2
- 2.3 The 1:1 Merge . . . 3
- 2.4 The 1:m Merge . . . 4
- 2.5 Appending Data . . . 6

3 Part 2. Heteroskedasticity (Chapter 8) **7**
- 3.1 The Data and the Model . . . 7
- 3.2 Why the LPM Is Heteroskedastic . . . 8
- 3.3 Breusch Pagan Test, Done Manually . . . 8
- 3.4 White Test, Done Manually . . . 9
- 3.5 Both Tests Using Commands . . . 10
- 3.6 Heteroskedasticity Robust Standard Errors . . . 11
- 3.7 Weighted Least Squares for the LPM . . . 12
- 3.8 Comparing the Three Approaches . . . 13

4 Exercises **15**

1 Packages for This Lab

We need `dplyr` for joining data, `wooldridge` for the MROZ data, `lmtest` and `sandwich` for heteroskedasticity tests and robust standard errors, and `stargazer` for the final regression table. Install any package you do not have with `install.packages()` before running the chunk below.

```
library(dplyr)
library(wooldridge)
library(lmtest)
library(sandwich)
library(stargazer)
```

2 Part 1. Combining Data Sets

Real research rarely comes in one clean file. Household data sit in one file, district data in another, and every survey round arrives as its own file. Before we can run a regression we have to put these pieces together. There are two ways of doing that.

A **merge** (also called a join) adds new *columns*. We line up two data sets side by side using a variable they share, and every row picks up the extra information from the other file.

An **append** adds new *rows*. We stack two data sets with the same variables on top of each other, for example the 2024 and 2025 rounds of the same survey.

2.1 Primary Keys

A merge only works if R knows which row in one file belongs with which row in the other. The variable that does this matching is the **key**. A **primary key** is a variable (or a set of variables) that uniquely identifies every row in a data set. No two rows share the same value, and no value is missing. A CNIC number is a primary key for Pakistani adults, and a student ID is a primary key in the IBA student records.

Whenever you merge, the first thing to ask is whether the key is unique in each file. The answer tells you what kind of merge you are doing.

2.2 Loading the Data Sets

This lab comes with six small CSV files, `students.csv`, `scores.csv`, `cities.csv`, `households.csv`, `survey_2024.csv` and `survey_2025.csv`. They are kept small on purpose so you can see exactly what happens to every row. Save them in the same folder as this .Rmd file, or set your working directory to the folder that holds them with `setwd()`.

The first file holds basic information on students. The primary key is `student_id`.

```
students <- read.csv("students.csv")
students
```

```
  student_id   name program
1        101 Ayesha  BSEcon
2        102  Bilal  BSEcon
3        103   Hira     BBA
4        104  Usman  BSEcon
5        105 Zainab     BBA
```

The second file holds exam scores from the registrar. Its key is also `student_id`. Notice that student 105 has no scores yet and student 106 has scores but is missing from the first file. This mismatch is very common in practice.

```
scores <- read.csv("scores.csv")
scores
```

```
  student_id midterm final
1        101      72    78
2        102      65    70
3        103      88    85
4        104      54    60
5        106      91    94
```

Before merging, we confirm that `student_id` really is a primary key in both files. If the number of unique IDs equals the number of rows, the key is unique.

```
n_distinct(students$student_id) == nrow(students)
```

```
[1] TRUE
```

```
n_distinct(scores$student_id) == nrow(scores)
```

```
[1] TRUE
```

```r
# Another way. List any ID that appears more than once
students %>% count(student_id) %>% filter(n > 1)
```

```
[1] student_id n
<0 rows> (or 0-length row.names)
```

Both checks pass, so each student appears once in each file.

2.3 The 1:1 Merge

In a **one to one (1:1) merge** the key is unique in *both* data sets. Each student in the first file matches at most one row in the second file. The merged data have one row per student and simply gain the score columns.

The only real decision is what to do with rows that do not find a partner. There are four types of join, and they differ only in which unmatched rows they keep.

Join	Rows kept
`inner_join()`	Only IDs found in both files
`left_join()`	Every row of the first (left) file
`right_join()`	Every row of the second (right) file
`full_join()`	Every row of both files

```r
inner_join(students, scores, by = "student_id")
```

```
  student_id   name program midterm final
1        101 Ayesha  BSEcon      72    78
2        102  Bilal  BSEcon      65    70
3        103   Hira     BBA      88    85
4        104  Usman  BSEcon      54    60
```

```r
left_join(students, scores, by = "student_id")
```

```
  student_id   name program midterm final
1        101 Ayesha  BSEcon      72    78
2        102  Bilal  BSEcon      65    70
3        103   Hira     BBA      88    85
4        104  Usman  BSEcon      54    60
5        105 Zainab     BBA      NA    NA
```

```r
full_join(students, scores, by = "student_id")
```

```
  student_id   name program midterm final
1        101 Ayesha  BSEcon      72    78
2        102  Bilal  BSEcon      65    70
3        103   Hira     BBA      88    85
4        104  Usman  BSEcon      54    60
5        105 Zainab     BBA      NA    NA
6        106   <NA>    <NA>      91    94
```

The inner join drops both 105 and 106. The left join keeps Zainab (105) with `NA` scores because she is in the left file. The full join keeps everyone and fills the gaps with `NA`.

You can tell `dplyr` what relationship you expect, and it will stop with an error if the data disagree. This is a good habit because it catches mistakes in the key before they reach your regression.

```
merged_11 <- left_join(students, scores, by = "student_id",
                       relationship = "one-to-one")
merged_11
```

```
  student_id   name program midterm final
1        101 Ayesha  BSEcon      72    78
2        102  Bilal  BSEcon      65    70
3        103   Hira     BBA      88    85
4        104  Usman  BSEcon      54    60
5        105 Zainab     BBA      NA    NA
```

Base R does the same thing with `merge()`. The argument `all.x = TRUE` gives a left join, `all = TRUE` gives a full join, and the default is an inner join.

```
merge(students, scores, by = "student_id", all.x = TRUE)
```

```
  student_id   name program midterm final
1        101 Ayesha  BSEcon      72    78
2        102  Bilal  BSEcon      65    70
3        103   Hira     BBA      88    85
4        104  Usman  BSEcon      54    60
5        105 Zainab     BBA      NA    NA
```

To see why checking the key matters, suppose the registrar file accidentally records student 102 twice. A 1:1 merge should now fail, and it does.

```
scores_bad <- rbind(scores, data.frame(student_id = 102, midterm = 66, final = 71))

left_join(students, scores_bad, by = "student_id",
          relationship = "one-to-one")
```

```
Error in `left_join()`:
! Each row in `x` must match at most 1 row in `y`.
i Row 2 of `x` matches multiple rows in `y`.
```

Without the `relationship` argument R would quietly give Bilal two rows, and any regression on the merged data would count him twice.

2.4 The 1:m Merge

In a **one to many (1:m) merge** the key is unique in one file but repeats in the other. The classic example is attaching area level information to individual level data. Each city appears once in a city file, but many households live in the same city.

The city file has one row per city, so `city_id` is its primary key.

```
cities <- read.csv("cities.csv")
cities
```

```
  city_id     city    province avg_rent
1       1  Karachi       Sindh    45000
2       2   Lahore      Punjab    40000
3       3 Peshawar          KP    28000
4       4   Quetta Balochistan    25000
```

The household file has one row per household. Its primary key is `hh_id`, while `city_id` repeats because several households share a city. Here `city_id` is called a **foreign key**, since it points to the primary key of another file.

```
households <- read.csv("households.csv")
households
```

```
  hh_id city_id hh_income hh_size
1     1       1    120000       5
2     2       1     85000       6
3     3       1    210000       4
4     4       2     95000       7
5     5       2    150000       5
6     6       3     70000       8
7     7       4     60000       9
8     8       4     80000       6
```

```
# hh_id is unique, city_id is not
n_distinct(households$hh_id) == nrow(households)
```

```
[1] TRUE
```

```
households %>% count(city_id)
```

```
  city_id n
1       1 3
2       2 2
3       3 1
4       4 2
```

When we merge, every household picks up the rent and province of its city. The city information is copied onto each matching household.

```
merged_1m <- left_join(cities, households, by = "city_id",
                       relationship = "one-to-many")
merged_1m
```

```
  city_id     city    province avg_rent hh_id hh_income hh_size
1       1  Karachi       Sindh    45000     1    120000       5
2       1  Karachi       Sindh    45000     2     85000       6
3       1  Karachi       Sindh    45000     3    210000       4
4       2   Lahore      Punjab    40000     4     95000       7
5       2   Lahore      Punjab    40000     5    150000       5
6       3 Peshawar          KP    28000     6     70000       8
7       4   Quetta Balochistan    25000     7     60000       9
8       4   Quetta Balochistan    25000     8     80000       6
```

The merged data now have eight rows, one per household. Karachi's average rent appears three times because three households live in Karachi.

If you put the household file on the left, the same merge is called **many to one (m:1)**. The result has the same rows, only the column order changes. In applied work this direction is more natural, because the household file is the one we analyse and we are adding context to it.

```
merged_m1 <- left_join(households, cities, by = "city_id",
                       relationship = "many-to-one")
merged_m1
```

```
  hh_id city_id hh_income hh_size     city    province avg_rent
1     1       1    120000       5  Karachi       Sindh    45000
2     2       1     85000       6  Karachi       Sindh    45000
3     3       1    210000       4  Karachi       Sindh    45000
```

```
4     4       2     95000       7   Lahore      Punjab    40000
5     5       2    150000       5   Lahore      Punjab    40000
6     6       3     70000       8 Peshawar          KP    28000
7     7       4     60000       9   Quetta Balochistan    25000
8     8       4     80000       6   Quetta Balochistan    25000
```

Now that the data are combined we can create variables that need both files, such as the share of income spent on rent.

```
merged_m1 %>%
  mutate(rent_share = avg_rent / hh_income) %>%
  select(hh_id, city, hh_income, avg_rent, rent_share)
```

```
  hh_id     city hh_income avg_rent rent_share
1     1  Karachi    120000    45000  0.3750000
2     2  Karachi     85000    45000  0.5294118
3     3  Karachi    210000    45000  0.2142857
4     4   Lahore     95000    40000  0.4210526
5     5   Lahore    150000    40000  0.2666667
6     6 Peshawar     70000    28000  0.4000000
7     7   Quetta     60000    25000  0.4166667
8     8   Quetta     80000    25000  0.3125000
```

A word of caution. If the key repeats in *both* files you have a **many to many** merge, and R pairs every matching row with every other matching row. This almost always signals a mistake in the key, which is another reason to check uniqueness before merging.

2.5 Appending Data

An **append** stacks data sets that share the same variables. Suppose the household survey was run in 2024 and again in 2025, and each round was saved as its own file.

```
survey_2024 <- read.csv("survey_2024.csv")
survey_2025 <- read.csv("survey_2025.csv")

survey_2024
```

```
  hh_id hh_income hh_size
1     1    120000       5
2     2     85000       6
3     3    210000       4
```

```
survey_2025
```

```
  hh_id hh_income hh_size
1     1    132000       5
2     2     90000       7
3     3    225000       4
4     4     99000       7
```

Before stacking we add a variable that records the survey year. Without it we could not tell the rounds apart after appending, and household 1 would appear twice with no way of knowing which row is which.

```
survey_2024$year <- 2024
survey_2025$year <- 2025
```

Base R appends with `rbind()`, which requires identical column names in both files. `dplyr` uses `bind_rows()`.

```r
panel <- bind_rows(survey_2024, survey_2025)
panel
```

```
  hh_id hh_income hh_size year
1     1    120000       5 2024
2     2     85000       6 2024
3     3    210000       4 2024
4     1    132000       5 2025
5     2     90000       7 2025
6     3    225000       4 2025
7     4     99000       7 2025
```

Notice that `hh_id` alone is no longer a primary key, since household 1 appears in both years. In the appended data the primary key is the *pair* `hh_id` and `year`. Data with this structure, the same units observed over several periods, is called **panel data**.

```r
panel %>% count(hh_id, year) %>% filter(n > 1)
```

```
[1] hh_id year  n
<0 rows> (or 0-length row.names)
```

The empty result confirms that `hh_id` and `year` together identify each row.

A useful difference between the two functions shows up when the files do not have the same columns. Suppose the 2025 round added a question on remittances.

```r
survey_2025$remit <- c(0, 15000, 0, 30000)

rbind(survey_2024, survey_2025)     # fails, columns differ
```

```
Error in rbind(deparse.level, ...): numbers of columns of arguments do not match
```

```r
bind_rows(survey_2024, survey_2025) # works, fills 2024 with NA
```

```
  hh_id hh_income hh_size year remit
1     1    120000       5 2024    NA
2     2     85000       6 2024    NA
3     3    210000       4 2024    NA
4     1    132000       5 2025     0
5     2     90000       7 2025 15000
6     3    225000       4 2025     0
7     4     99000       7 2025 30000
```

`rbind()` refuses, while `bind_rows()` keeps the new variable and fills it with `NA` for 2024, the year it was not asked.

To sum up, a merge makes the data **wider** by adding columns, and an append makes the data **longer** by adding rows.

3 Part 2. Heteroskedasticity (Chapter 8)

3.1 The Data and the Model

We use the MROZ data on 753 married women from the 1975 Panel Study of Income Dynamics. The dependent variable `inlf` equals 1 if the woman was in the labour force in 1975 and 0 otherwise. Since the outcome is binary, an OLS regression of `inlf` on the regressors is a **linear probability model (LPM)**, and each coefficient measures the change in the probability of working.

We model labour force participation as a function of other household income (`nwifeinc`, in thousands of dollars), years of education, experience and its square, age, and the number of young (`kidslt6`) and older (`kidsge6`) children.

```
data("mroz")

lpm <- lm(inlf ~ nwifeinc + educ + exper + I(exper^2) + age +
            kidslt6 + kidsge6, data = mroz)
summary(lpm)
```

```
Call:
lm(formula = inlf ~ nwifeinc + educ + exper + I(exper^2) + age +
    kidslt6 + kidsge6, data = mroz)

Residuals:
     Min       1Q   Median       3Q      Max
-0.93432 -0.37526  0.08833  0.34404  0.99417

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.5855192  0.1541780   3.798 0.000158 ***
nwifeinc    -0.0034052  0.0014485  -2.351 0.018991 *
educ         0.0379953  0.0073760   5.151 3.32e-07 ***
exper        0.0394924  0.0056727   6.962 7.38e-12 ***
I(exper^2)  -0.0005963  0.0001848  -3.227 0.001306 **
age         -0.0160908  0.0024847  -6.476 1.71e-10 ***
kidslt6     -0.2618105  0.0335058  -7.814 1.89e-14 ***
kidsge6      0.0130122  0.0131960   0.986 0.324415
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4271 on 745 degrees of freedom
Multiple R-squared:  0.2642,    Adjusted R-squared:  0.2573
F-statistic: 38.22 on 7 and 745 DF,  p-value: < 2.2e-16
```

An extra year of education raises the probability of being in the labour force by about 3.8 percentage points, holding the other factors fixed. Each child under six lowers it by about 26.2 percentage points.

3.2 Why the LPM Is Heteroskedastic

The LPM has heteroskedasticity built into it. Because `inlf` is a 0/1 variable, its conditional variance is

$$\mathrm{Var}(y \mid \mathbf{x}) = p(\mathbf{x})\,[1 - p(\mathbf{x})],$$

where $p(\mathbf{x})$ is the probability of working. Since this probability changes with education, children and the rest, the error variance changes too. Assumption MLR.5 cannot hold unless none of the regressors matter. The OLS coefficients remain unbiased, but the usual standard errors, t statistics and F statistics are no longer valid. Formal tests let us confirm what theory tells us, and the same tools work for any model.

3.3 Breusch Pagan Test, Done Manually

The Breusch Pagan (BP) test asks whether the squared residuals are related to the regressors. The null hypothesis is homoskedasticity,

$$H_0 : \mathrm{Var}(u \mid x_1, \ldots, x_k) = \sigma^2.$$

The test takes three steps.

1. Estimate the model by OLS and save the residuals $\hat{u}$.
2. Regress $\hat{u}^2$ on all the regressors and keep the R^2 of this auxiliary regression, call it $R^2_{\hat{u}^2}$.
3. Compute either the F statistic for joint significance of the regressors in step 2, or the LM statistic $LM = n \cdot R^2_{\hat{u}^2}$, which follows a χ^2_k distribution under the null.

```
mroz$uhat_sq <- resid(lpm)^2

bp_aux <- lm(uhat_sq ~ nwifeinc + educ + exper + I(exper^2) + age +
               kidslt6 + kidsge6, data = mroz)

n  <- nobs(lpm)
k  <- length(coef(bp_aux)) - 1
R2 <- summary(bp_aux)$r.squared

# F version
bp_F   <- (R2 / k) / ((1 - R2) / (n - k - 1))
bp_F_p <- pf(bp_F, k, n - k - 1, lower.tail = FALSE)

# LM version
bp_LM   <- n * R2
bp_LM_p <- pchisq(bp_LM, df = k, lower.tail = FALSE)

c(R2 = R2, F = bp_F, F_pvalue = bp_F_p, LM = bp_LM, LM_pvalue = bp_LM_p)
```

```
          R2            F     F_pvalue           LM    LM_pvalue
3.217006e-02 3.537619e+00 9.523498e-04 2.422406e+01 1.040486e-03
```

The F statistic matches the one printed at the bottom of the auxiliary regression output.

```
summary(bp_aux)$fstatistic
```

```
     value      numdf      dendf
  3.537619   7.000000 745.000000
```

The LM statistic is 24.22 with 7 degrees of freedom and a p value of 0.00104. We reject homoskedasticity at any conventional level, which is exactly what the theory of the LPM predicts.

3.4 White Test, Done Manually

The BP test only looks for a *linear* relation between $\hat{u}^2$ and the regressors. The White test also allows for squares and cross products of the regressors. With seven regressors the full version needs 35 terms, which uses up many degrees of freedom. Wooldridge (Section 8.3) recommends a special form that captures the same idea in just two terms, the fitted values $\hat{y}$ and their square $\hat{y}^2$. Since $\hat{y}$ is a linear function of all the regressors, $\hat{y}^2$ contains all their squares and cross products.

1. Estimate the model by OLS and save $\hat{u}$ and $\hat{y}$.
2. Regress $\hat{u}^2$ on $\hat{y}$ and $\hat{y}^2$ and keep the R^2.
3. Compute $LM = n \cdot R^2$, which follows a χ^2_2 distribution, or the F statistic with 2 and $n - 3$ degrees of freedom.

```
mroz$yhat <- fitted(lpm)

white_aux <- lm(uhat_sq ~ yhat + I(yhat^2), data = mroz)
```

```r
R2_w <- summary(white_aux)$r.squared

white_F   <- (R2_w / 2) / ((1 - R2_w) / (n - 3))
white_F_p <- pf(white_F, 2, n - 3, lower.tail = FALSE)

white_LM   <- n * R2_w
white_LM_p <- pchisq(white_LM, df = 2, lower.tail = FALSE)

c(R2 = R2_w, F = white_F, F_pvalue = white_F_p,
  LM = white_LM, LM_pvalue = white_LM_p)
```

```
          R2            F     F_pvalue           LM    LM_pvalue
1.248474e-01 5.349672e+01 1.911671e-22 9.401012e+01 3.854442e-21
```

The White LM statistic is 94.01 with a p value of <2e-16, so the White test also rejects homoskedasticity.

3.5 Both Tests Using Commands

The `bptest()` function in the `lmtest` package runs these tests in one line. By default it regresses $\hat{u}^2$ on the regressors of the model and reports the LM statistic, so it reproduces our manual BP test.

```r
bptest(lpm)
```

```
    studentized Breusch-Pagan test

data:  lpm
BP = 24.224, df = 7, p-value = 0.00104
```

For the special White test we give `bptest()` a second formula listing $\hat{y}$ and $\hat{y}^2$ as the variables in the auxiliary regression.

```r
bptest(lpm, ~ yhat + I(yhat^2), data = mroz)
```

```
    studentized Breusch-Pagan test

data:  lpm
BP = 94.01, df = 2, p-value < 2.2e-16
```

Let us put the manual and command results side by side.

```r
bp_cmd    <- bptest(lpm)
white_cmd <- bptest(lpm, ~ yhat + I(yhat^2), data = mroz)

data.frame(
  Test       = c("Breusch Pagan", "White (special form)"),
  LM_manual  = round(c(bp_LM, white_LM), 3),
  LM_command = round(c(bp_cmd$statistic, white_cmd$statistic), 3),
  df         = c(bp_cmd$parameter, white_cmd$parameter),
  p_value    = signif(c(bp_cmd$p.value, white_cmd$p.value), 3)
)
```

```
                 Test LM_manual LM_command df  p_value
1        Breusch Pagan    24.224     24.224  7 1.04e-03
2 White (special form)    94.010     94.010  2 3.85e-21
```

The numbers are identical, so the command is doing exactly the three steps we did by hand.

3.6 Heteroskedasticity Robust Standard Errors

Once we know heteroskedasticity is present, the simplest fix is to keep the OLS coefficients and replace the usual standard errors with **heteroskedasticity robust** standard errors (Section 8.2). These remain valid in large samples whether or not the errors are homoskedastic, so the t and F tests built on them can be trusted.

The `vcovHC()` function in `sandwich` computes the robust variance matrix. The option `type = "HC1"` applies the small sample degrees of freedom correction used by Stata and most applied work. `coeftest()` then reports the coefficients with the robust standard errors.

```
vc_robust <- vcovHC(lpm, type = "HC1")
coeftest(lpm, vcov. = vc_robust)
```

```
t test of coefficients:

               Estimate  Std. Error t value  Pr(>|t|)
(Intercept)  0.58551922  0.15225987  3.8455 0.0001306 ***
nwifeinc    -0.00340517  0.00152493 -2.2330 0.0258453 *
educ         0.03799530  0.00726604  5.2292 2.214e-07 ***
exper        0.03949239  0.00581002  6.7973 2.185e-11 ***
I(exper^2)  -0.00059631  0.00019000 -3.1384 0.0017656 **
age         -0.01609081  0.00239901 -6.7073 3.922e-11 ***
kidslt6     -0.26181047  0.03178320 -8.2374 7.892e-16 ***
kidsge6      0.01301223  0.01353293  0.9615 0.3366009
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We store the robust standard errors so we can use them in the final table.

```
se_robust <- sqrt(diag(vc_robust))

data.frame(
  usual_SE  = round(coef(summary(lpm))[, "Std. Error"], 5),
  robust_SE = round(se_robust, 5)
)
```

```
            usual_SE robust_SE
(Intercept)  0.15418   0.15226
nwifeinc     0.00145   0.00152
educ         0.00738   0.00727
exper        0.00567   0.00581
I(exper^2)   0.00018   0.00019
age          0.00248   0.00240
kidslt6      0.03351   0.03178
kidsge6      0.01320   0.01353
```

The coefficients do not change at all, only the standard errors do. In this model the two sets are close, so none of our conclusions about significance change, but you can never know that in advance without computing both.

Robust inference also applies to joint tests. To test whether the two child variables are jointly significant we pass the robust variance matrix to `waldtest()`.

```
lpm_r <- lm(inlf ~ nwifeinc + educ + exper + I(exper^2) + age, data = mroz)
waldtest(lpm, lpm_r, vcov = vc_robust, test = "F")
```

```
Wald test
```

```
Model 1: inlf ~ nwifeinc + educ + exper + I(exper^2) + age + kidslt6 +
    kidsge6
Model 2: inlf ~ nwifeinc + educ + exper + I(exper^2) + age
  Res.Df Df     F    Pr(>F)
1    745
2    747 -2 35.55 1.794e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

3.7 Weighted Least Squares for the LPM

If we know the form of the heteroskedasticity, we can do better than robust standard errors and obtain more efficient estimates using **weighted least squares (WLS)**. In the LPM we know the form exactly, because $\text{Var}(y \mid \mathbf{x}) = p(\mathbf{x})[1 - p(\mathbf{x})]$. Following Section 8.5 of Wooldridge, the steps are

1. Estimate the LPM by OLS and obtain the fitted values $\hat{y}$.
2. Compute $\hat{h} = \hat{y}(1 - \hat{y})$, the estimated variance of each observation.
3. Run OLS again with weights $1/\hat{h}$, so that observations with a noisier outcome get less weight.

There is one practical problem. The weights only make sense if every fitted value lies strictly between 0 and 1, otherwise $\hat{h}$ is zero or negative. Let us check.

```
summary(mroz$yhat)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-0.3451  0.4016  0.5880  0.5684  0.7592  1.1272
```

```
sum(mroz$yhat <= 0 | mroz$yhat >= 1)
```

```
[1] 33
```

In our sample 33 fitted values fall outside the unit interval. Wooldridge suggests a simple adjustment. We pull these fitted values back to 0.01 and 0.99 before computing the weights.

```
mroz$yhat_adj <- pmin(pmax(mroz$yhat, 0.01), 0.99)
mroz$h_hat    <- mroz$yhat_adj * (1 - mroz$yhat_adj)

lpm_wls <- lm(inlf ~ nwifeinc + educ + exper + I(exper^2) + age +
                kidslt6 + kidsge6, data = mroz, weights = 1 / h_hat)
summary(lpm_wls)
```

```

Call:
lm(formula = inlf ~ nwifeinc + educ + exper + I(exper^2) + age +
    kidslt6 + kidsge6, data = mroz, weights = 1/h_hat)

Weighted Residuals:
    Min      1Q  Median      3Q     Max
-3.8257 -0.8827  0.3150  0.6898  7.3444

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.6930956  0.1242765   5.577 3.42e-08 ***
nwifeinc     0.0010471  0.0008844   1.184  0.23682
educ         0.0235742  0.0049041   4.807 1.85e-06 ***
exper        0.0407199  0.0050655   8.039 3.56e-15 ***
I(exper^2)  -0.0006458  0.0001702  -3.794  0.00016 ***
age         -0.0158602  0.0019401  -8.175 1.27e-15 ***
```

```
kidslt6     -0.2580120  0.0190413 -13.550  < 2e-16 ***
kidsge6      0.0082076  0.0089458   0.917  0.35919
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.014 on 745 degrees of freedom
Multiple R-squared:  0.6254,    Adjusted R-squared:  0.6219
F-statistic: 177.7 on 7 and 745 DF,  p-value: < 2.2e-16
```

Most WLS estimates are close to OLS, which is what we expect since both estimate the same population parameters. The WLS standard errors are generally smaller, reflecting the efficiency gain from using what we know about the variance. Two coefficients do move, though. The return to education falls noticeably, and the effect of other income becomes small and insignificant. When OLS and WLS differ this much, it is a hint that the model may be misspecified, a point Wooldridge makes in Section 8.4.

3.8 Comparing the Three Approaches

The table below shows OLS with the usual standard errors in column 1, the same OLS coefficients with robust standard errors in column 2, and WLS in column 3. In `stargazer` we supply the same OLS model twice and override the standard errors for the second column through the `se` argument. Stars in column 2 are computed from the robust standard errors.

```r
stargazer(lpm, lpm, lpm_wls,
          se = list(NULL, se_robust, NULL),
          column.labels = c("OLS", "OLS Robust SE", "WLS"),
          dep.var.labels = "In labour force (inlf)",
          covariate.labels = c("Other income (nwifeinc)", "Education",
                               "Experience", "Experience squared", "Age",
                               "Kids under 6", "Kids 6 to 18"),
          model.numbers = TRUE,
          omit.stat = c("ser", "f"),
          title = "Linear Probability Model of Female Labour Force Participation",
          header = FALSE, table.placement = "H", type = "latex")
```

Table 2: Linear Probability Model of Female Labour Force Participation

	Dependent variable:		
	In labour force (inlf)		
	OLS	OLS Robust SE	WLS
	(1)	(2)	(3)
Other income (nwifeinc)	-0.003^{**}	-0.003^{**}	0.001
	(0.001)	(0.002)	(0.001)
Education	0.038^{***}	0.038^{***}	0.024^{***}
	(0.007)	(0.007)	(0.005)
Experience	0.039^{***}	0.039^{***}	0.041^{***}
	(0.006)	(0.006)	(0.005)
Experience squared	-0.001^{***}	-0.001^{***}	-0.001^{***}
	(0.0002)	(0.0002)	(0.0002)
Age	-0.016^{***}	-0.016^{***}	-0.016^{***}
	(0.002)	(0.002)	(0.002)
Kids under 6	-0.262^{***}	-0.262^{***}	-0.258^{***}
	(0.034)	(0.032)	(0.019)
Kids 6 to 18	0.013	0.013	0.008
	(0.013)	(0.014)	(0.009)
Constant	0.586^{***}	0.586^{***}	0.693^{***}
	(0.154)	(0.152)	(0.124)
Observations	753	753	753
R^2	0.264	0.264	0.625
Adjusted R^2	0.257	0.257	0.622
Note:			*p<0.1; **p<0.05; ***p<0.01

Reading across the table, columns 1 and 2 have identical coefficients because they come from the same OLS regression, and only the standard errors in parentheses differ. Column 3 has different coefficients because WLS reweights the observations. In every column education and experience raise participation while young children lower it sharply, so the main story survives however we deal with heteroskedasticity. Other household income is the exception, since it loses its significance under WLS. Finally, do not compare the R^2 of WLS with OLS. The WLS R^2 measures the fit of the weighted regression, so its much higher value says nothing about which model explains labour force participation better.

Which column should you report? Robust standard errors are the safe default because they do not require us to know the form of the variance. WLS is attractive in the LPM because the variance form is known, but it depends on fitted values inside the unit interval and on the adjustment we made for those outside it. Reporting both, as in this table, lets the reader see which conclusions hinge on that choice and which do not.

4 Exercises

1. Create a data set of five districts with a primary key `district_id` and a literacy rate, and a data set of ten schools that records each school's `district_id`. Merge them, state whether the merge is 1:1 or 1:m, and verify the key of each file before merging.
2. Append the 2024 and 2025 survey files after giving the 2024 file a `remit` column of `NA`. Confirm that `rbind()` now works.
3. Re-estimate the LPM with `log(nwifeinc + 1)` in place of `nwifeinc`. Run the BP and White tests manually and with `bptest()`.
4. Use the `hprice1` data from the `wooldridge` package to regress `price` on `lotsize`, `sqrft` and `bdrms`. Test for heteroskedasticity, then repeat the tests with `log(price)` as the dependent variable. What does the change in results tell you?