PTT Analysis of Entrance Exam Scores in Taiwan

A Statistical Approach with R Code

Christine P. Chai
Microsoft
cpchai21@gmail.com

January 21, 2022

Ongoing work since 2019.

Executive Summary

Write something here

Disclaimer

The opinions and views expressed in this manuscript are those of the author and do not necessarily state or reflect those of Microsoft. The author has a statistics and engineering background, without any training in middle school or high school teaching.

1 Introduction

Consider making the introduction statistics-oriented, because we would like to focus on the statistical methodology. Most important is how to handle a real-time application problem.

State thie project as open-source and provide the GitHub link somewhere in the manuscript.

The author grew up in Taiwan, so we are curious about the relationship between the high school entrance exam score percentiles and the college entrance scores in Taiwan. It seems obvious that a greater percentage of students from top high schools get admitted to prestigious universities.[1] However, the high school entrance exam is much easier than the college entrance exam, so some people studied little in middle school and was able to get into a good high school. Then some of these people kept studying little and ended up with a bad score on the college entrance exam. On the other hand, we have also seen some students from mediocre high schools studied very hard, and eventually earned a stellar score in the college exam. Therefore, we decided to gather data and create our own analysis. The target audience of this document would be people with a basic understanding of statistics, equivalent to taking or having completed Statistics 101.

Since we do not have the official data from the Taiwan government that links people between their scores on the high school entrance exam and college entrance exam, it is difficult to obtain a representative sample of people's scores. Therefore, we obtained self-reported data from an existing post[2] in 2015 on PTT, the largest terminal-based bulletin board in Taiwan. Each of the 197 rows of the data contains an anonymous ID, high school entrance exam score (in the form of percentile rank), and college entrance exam score.

For the rest of this article, we begin with a short background of Taiwan's high school and college entrance exams, followed by the description of our dataset. We start the technical content with exploratory data

[1] https://bit.ly/2JSPXKc
[2] https://www.ptt.cc/bbs/SENIORHIGH/M.1432729401.A.995.html

analysis, in order to identify and visualize the distribution of exam scores. Next, we explain our decisions in statistical modeling, such as whether a linear regression is appropriate or not. We also take a closer look at the top scorers, because they typically came from prestigious high schools and/or colleges. Then we move on to introduce advanced models like logistic regression. Finally, we provide our discussion, state our conclusion, and give some personal remarks.

The goal of writing this document is to instruct the readers on how to perform data analysis using fundamental machine learning concepts. We focus on demonstrating the methodology, so the coding implementation may not be perfectly optimized.

2 Background

Unfinished below

Elaborate on the following to enhance the background context:

- When did GSAT start?
- College entrance exam GSAT and ACT: New rules in 2022 (not related to COVID)
- COVID-related rules for students who could not attend the exam
  Start writing about 2021 for both exams, because 2022 is still ongoing

Unfinished above

In Taiwan, the high school entrance exam score percentiles are between 1% and 99%, and people often refer to the percentile rank (PR value) as from 1 to 99. This scoring system existed from 2001 to 2013.[3] The actual exam score ranges were different. For example, the maximum possible score was originally set to 300, but it was increased to 312 in Year 2007. Then the maximum possible score was increased to 412 in Year 2009. Therefore, the percentile ranks (PR values) serves as a normalized tool to compare academic achievements across different years. Starting in 2014, the PR values are replaced with seven grades for each subject (C, B, B+, B++, A, A+, A++) and an independent essay score (0-6).[4]

The college entrance exams are held twice a year in Taiwan; the first exam is for early admission and the second exam is for regular admission. The first exam, typically held in late January or early February, is called the General Scholastic Ability Test (GSAT).[5] The second exam is called the Advanced Subjects Test (AST),[6] and it is almost always held on July 1st, 2nd, and 3rd. The GSAT scores are normalized to a range of 0 to 75, regardless of the difficulty level of GSAT each year. On the other hand, the scores of AST can vary widely because each subject is scored separately from 0 to 100. Since the AST scores fluctuate more due to the difficulty level of the exam questions each year, we decided to use the GSAT scores as a benchmark of the college exam scores.

Remark: The GSAT consists of five subjects (Chinese, English, mathematics, physical sciences, and social sciences), each of which are graded on a 0 to 15 point scale. Starting in 2019, students may choose four of the five subjects for the GSAT. The maximum possible score (i.e., full marks) is reduced from 75 to 60.

3 Data Description

It is a challenge to obtain individual pairs of data as a representative sample. Although it is easy to send out a spreadsheet and ask our friends to report their scores anonymously, this approach can result in a large

[3] https://bit.ly/2JNQaOI
[4] https://www.parenting.com.tw/article/5049351
[5] https://bit.ly/2W0fdUq
[6] https://bit.ly/2J7YxoW

selection bias. Many of our friends graduated from the same high school and/or college, so we are likely to have similar entrance exam scores.

Hence we retrieved data from the SENIORHIGH (high school)[7] discussion section on PTT,[8] the largest terminal-based bulletin board in Taiwan.[9] We assume the data to be more representative (than if we had collected on our own) because anyone could get a PTT account and reply to the post – at least before the restriction was announced in 2018.[10] The majority of scores were reported in May 2015, and a few scores were reported in the following month or later. The records indicate that each respondent had taken the college entrance exam in the year 2015 or earlier, so we can safely assume that neither of them had taken the new form of high school entrance exam starting in 2014.

The data `ptt_SENIORHIGH_data.csv` contain 197 rows, and the main variables are:

- **pttID**: Each person's ID on PTT, which can be anonymous. This column serves as the unique identifier of each person.
- **HighSchool_PR**: Each person's percentile rank (PR) of the high school entrance exam in Taiwan, ranging from 0 to 99.
- **College_Score**: Each person's General Scholastic Ability Test (GSAT) score, ranging from 0 to 75.

There are 6 missing values in **HighSchool_PR** and 3 missing values in **College_Score**, so we recorded each of them as "-1" (an invalid numerical value).

In some cases, the reported scores can be inaccurate based on the respondent's description, so we created two indicators for this issue:

- **HS_Inacc**: A "1" means the reported score of high school entrance exam is inaccurate.
- **College_Inacc**: A "1" means the reported score of college entrance exam is inaccurate.

Some people reported their percentile rank (PR) from the mock exam, rather than the actual high school entrance exam. In 2012 and 2013, the Ministry of Education in Taiwan allowed students to apply for high schools with their grades in middle school. During that time, if a student got admitted to a high school using this method, he/she would not need to take the high school entrance exam.[11]

Moreover, there are two college entrance exams in each school year, and some students may do much better on the second exam than the first one. Then they were admitted to a more prestigious school than the first exam score had indicated, so this is also a form of inaccuracy.

3.1 Data at a Glance

We show the first 10 rows of data here, and `NA` (not available) denotes that the value is missing. Note that only **HS_Inacc** and **College_Inacc** contain `NA`s, because we already recoded missing values to "-1" (an invalid numeric value) for **HighSchool_PR** and **College_Score**.

We also observed that **pttID** contains some information for potential inference, although we are not going to use it. For example, the 6th respondent `robinyu85` could be someone named Robin Yu, and the 8th respondent `godpatrick11` may have the English name Patrick. Nevertheless, this kind of information is simply a heuristic, so it is neither sufficient nor appropriate to include in the data analysis.

```r
data = read.csv("ptt_SENIORHIGH_data.csv")
names(data)[1] = "pttID"

data[1:10,]
```

```
##           pttID HighSchool_PR College_Score HS_Inacc College_Inacc
## 1    game275415            60            50        1            NA
```

[7] https://www.ptt.cc/bbs/SENIORHIGH/M.1432729401.A.995.html
[8] If you have a PTT account, you can log into the website using a browser. https://iamchucky.github.io/PttChrome/?site=ptt.cc
[9] https://en.wikipedia.org/wiki/PTT_Bulletin_Board_System
[10] https://www.ettoday.net/news/20200304/1659455.htm
[11] https://tsjh301.blogspot.com/2014/06/compulsory-education.html

```
## 2      a2654133            60            52       NA            NA
## 3   cookie20125            99            72       NA            NA
## 4       heejung            92            54        1            NA
## 5        shun01            87            51       NA            NA
## 6     robinyu85            -1            74       NA            NA
## 7    allengoose            69            48       NA            NA
## 8  godpatrick11            98            60       NA            NA
## 9     morgankhs            95            65       NA            NA
## 10      jazzard            88            65       NA            NA
```

Remark: Data in the real world are messy, and data scientists spend lots of time cleaning (preprocessing) the data, i.e., preparing the data for analysis.[12] But data cleaning is a necessary step for better analysis results, and there are some visualization examples that demonstrate the importance of preprocessing the data (Chai, 2020). Our dataset `ptt_SENIORHIGH_data.csv` is relatively clean, but we still had to recode and flag missing values.

4 Exploratory Data Analysis

Unfinished below

Write something more to this section, especially the foreword.

Before we perform any statistical modeling, the first step in a data project is exploratory data analysis, i.e., observing the data. The exploratory phase allows us to identify patterns, detect anomalies, and verify assumptions in the data.[13] This is an important but often overlooked step in data analysis, and failure to explore the data can lead to inefficiencies such as accurate models on the wrong data.[14]

Therefore, we start with examining the trends of the two main variables, **HighSchool_PR** and **College_Score**. For each variable, we observe the values and summarize them in a histogram as part of the univariate analysis. Then we investigate the relationship between the two variables, i.e., bivariate exploration.

4.1 High School Entrance Exam Scores (Percentile Rank)

Below shows the descriptive statistics of **HighSchool_PR**, i.e., the percentile rank of high school entrance exam scores. The missing values are removed beforehand. Approximately 75% of the respondents have a percentile rank (PR) at least 85, indicating that most of the respondents scored in the top 15% of the high school entrance exam. The histogram is also extremely left-skewed.

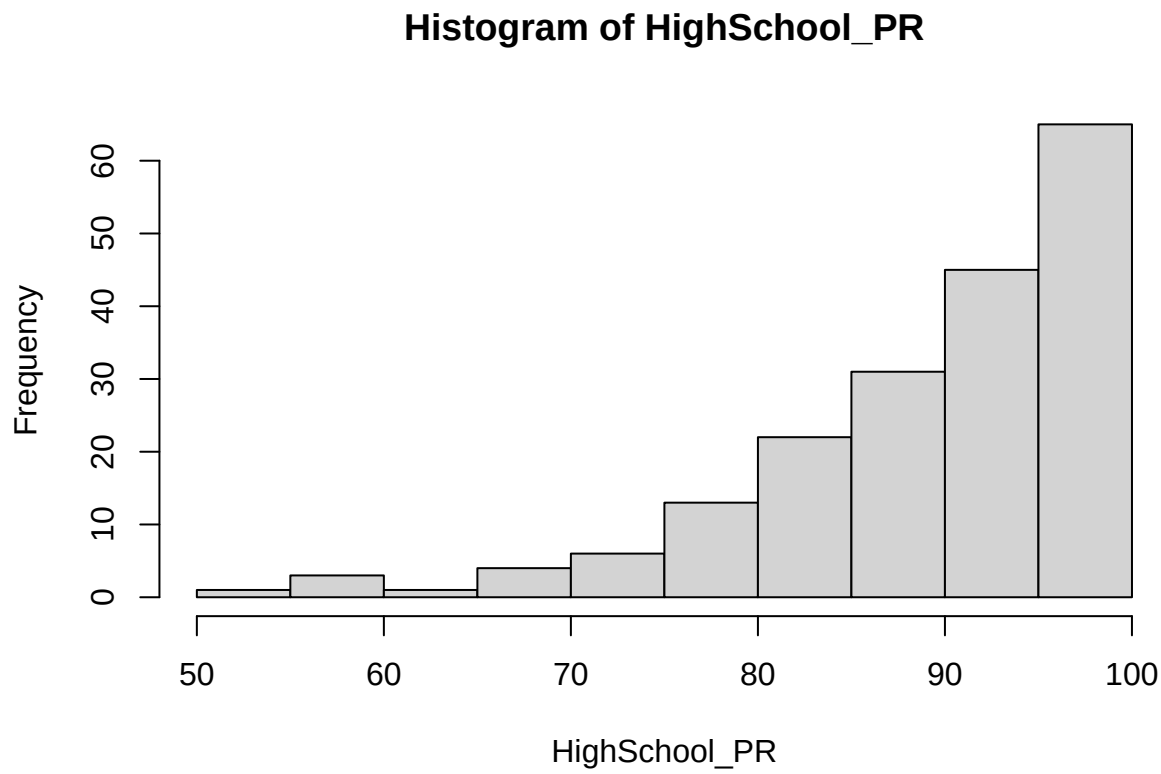
```r
# High school entrance exam scores: Remove missing values
uni_HS_score = data$HighSchool_PR[which(data$HighSchool_PR != -1)]

summary(uni_HS_score)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   51.00   85.00   92.00   89.82   97.00   99.00
```

```r
hist(uni_HS_score, main = "Histogram of HighSchool_PR",
     xlab="HighSchool_PR")
```

[12] https://bit.ly/303IWxY
[13] https://blog.eduonix.com/bigdata-and-hadoop/importance-exploratory-data-analysis-ml-modelling
[14] https://towardsdatascience.com/exploratory-data-analysis-topic-that-is-neglected-in-data-science-projects-9962ae078a56



4.2 College Entrance Exam Scores

Similarly, we also show the descriptive statistics of **College_Score**, i.e., the college entrance exam scores between 0 and 75. The histogram is also left-skewed, but less extreme than **HighSchool_PR**.

According to the reference score table[15] from Wikipedia, the 88th percentile of the college entrance score fluctuates around 60 in Years 2004-2010, and 62-65 in Years 2011-2018. Since the median of **College_Score** is 64.5, we can infer that at least 50% of the respondents scored in the top 12% of the college entrance exam.

On the other hand, the reference score table also shows that the 75th percentile of the college entrance score is between 53 and 58 in Years 2004-2018. The PTT data's 1st quantile is already at 58, so we can also infer that at least 75% of the respondents scored in the top 25% of the college entrance exam.

Since PTT contains forums for several prestigious universities in Taiwan, it is no surprise that many users attended these colleges because they scored well on the college entrance exam. Nevertheless, PTT did not limit registration to students of these colleges in the past, so the population of PTT is slightly more diverse. Note that as of 2020, PTT changed their eligibility requirements, and limited new account registrations to only people with an email address from National Taiwan University.[16]

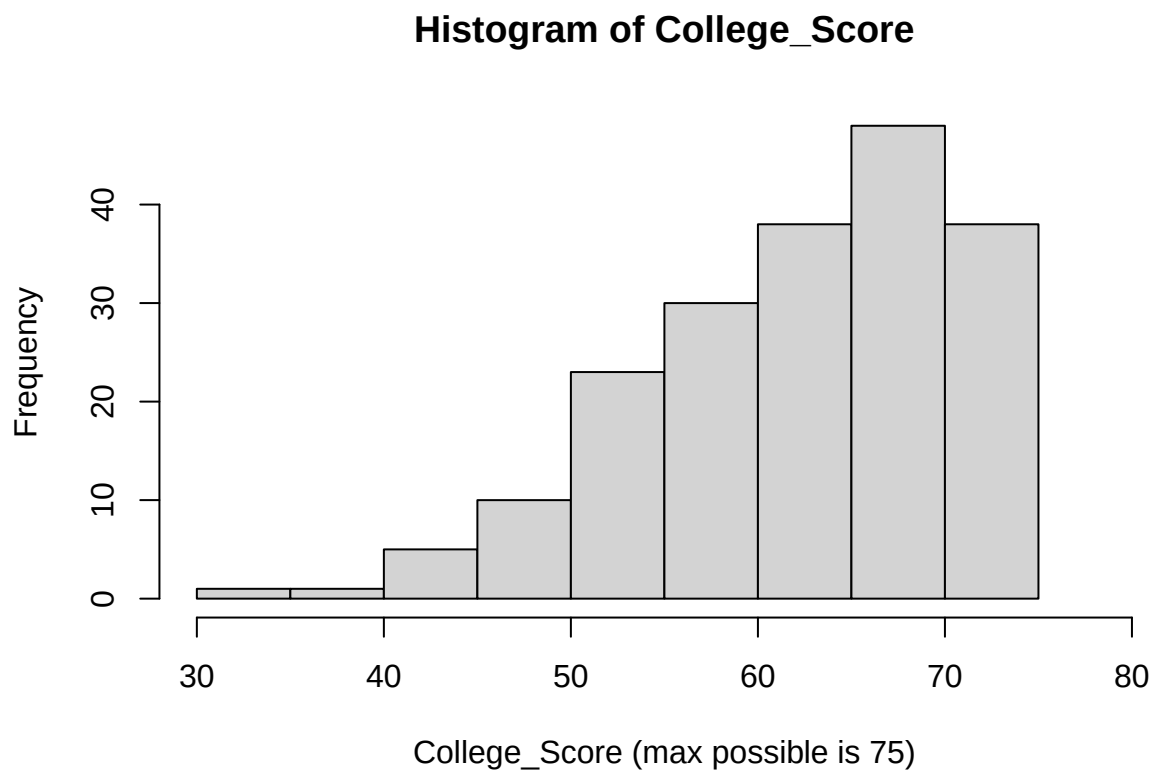
```r
# College entrance exam scores: Remove missing values
uni_college_score = data$College_Score[which(data$College_Score != -1)]

summary(uni_college_score)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    34.0    58.0    64.5    62.7    69.0    75.0
```

[15] https://bit.ly/3bAYOvO
[16] Screenshot obtained on May 26, 2020: https://imgur.com/33fwrGH

```r
hist(uni_college_score, main="Histogram of College_Score",
     xlab="College_Score (max possible is 75)",xlim=c(30,80))
```



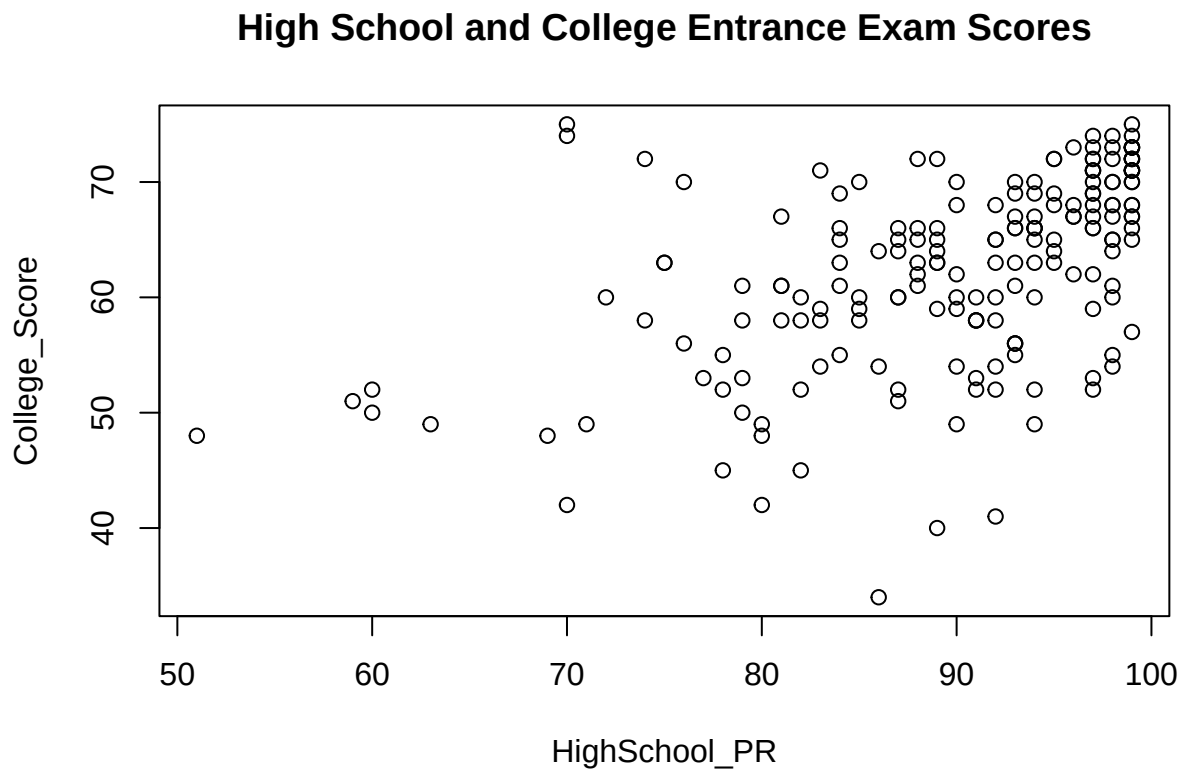
4.3 Bivariate Exploration

Next, we create a bivariate scatterplot of **HighSchool_PR** and **College_Score**, but we have to remove the records with at least one missing score. Just like what we observed in the univariate plots, both variables are largely concentrated towards the maximum possible scores.

```r
missing_rows = which(data$HighSchool_PR == "-1" | data$College_Score == "-1")
# Indices: 6  19  71  85  88  96 132 183 195 => nine in total

# Remove missing data
data_corr = data[-missing_rows,]

plot(data_corr$HighSchool_PR, data_corr$College_Score,
     main = "High School and College Entrance Exam Scores",
     xlab="HighSchool_PR",
     ylab="College_Score")
```



The correlation coefficient is approximately 0.507, showing a medium strength of positive association between **HighSchool_PR** and **College_Score**. We can interpret that a better score in the high school entrance exam is likely to lead to a better college entrance exam score, but the relationship is not as strong after **HighSchool_PR** reaches 80.

```
cor(data_corr$HighSchool_PR, data_corr$College_Score)
```

```
## [1] 0.5074473
```

To calculate the correlation coefficient between the random variables X, Y, we need to start with the covariance $\text{Cov}(X, Y)$ in the equation below. $E[X]$ denotes the expectation value of X, a.k.a. the mean of X.

$$\text{Cov}(X, Y) = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y].$$

Then we also need to compute the standard deviation σ_X:

$$\sigma_X = \sqrt{E[(X - E[X])^2]} = \sqrt{E[X^2] - (E[X])^2}.$$

Same applies to the standard deviation σ_Y.

Finally, we can calculate the correlation coefficient as:

$$\rho_{X,Y} = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y}.$$

5 Linear Regression

We are going to decide whether we should run a linear regression to predict **College_Score** from **HighSchool_PR**. If yes, we would implement the model and check the residuals. If no, we need to explore other options in analyzing the data.

A linear regression can be written in mathematical terms:

$$Y = \alpha + \beta X + \epsilon$$

Y is the response variable, i.e., what we would like to predict. X is the explanatory variable, i.e., the data used to make the predictions. α is the intercept, and it stands for the estimate $\hat{Y}$ when $X = 0$ (if applicable). β is the coefficient, and when X increases by one unit, we can expect Y to increase by β units. Last but not least, ϵ is the error term, which is normally distributed with mean zero.

OpenIntro Statistics (Diez et al., 2019) states that four requirements need to be met for a linear regression:

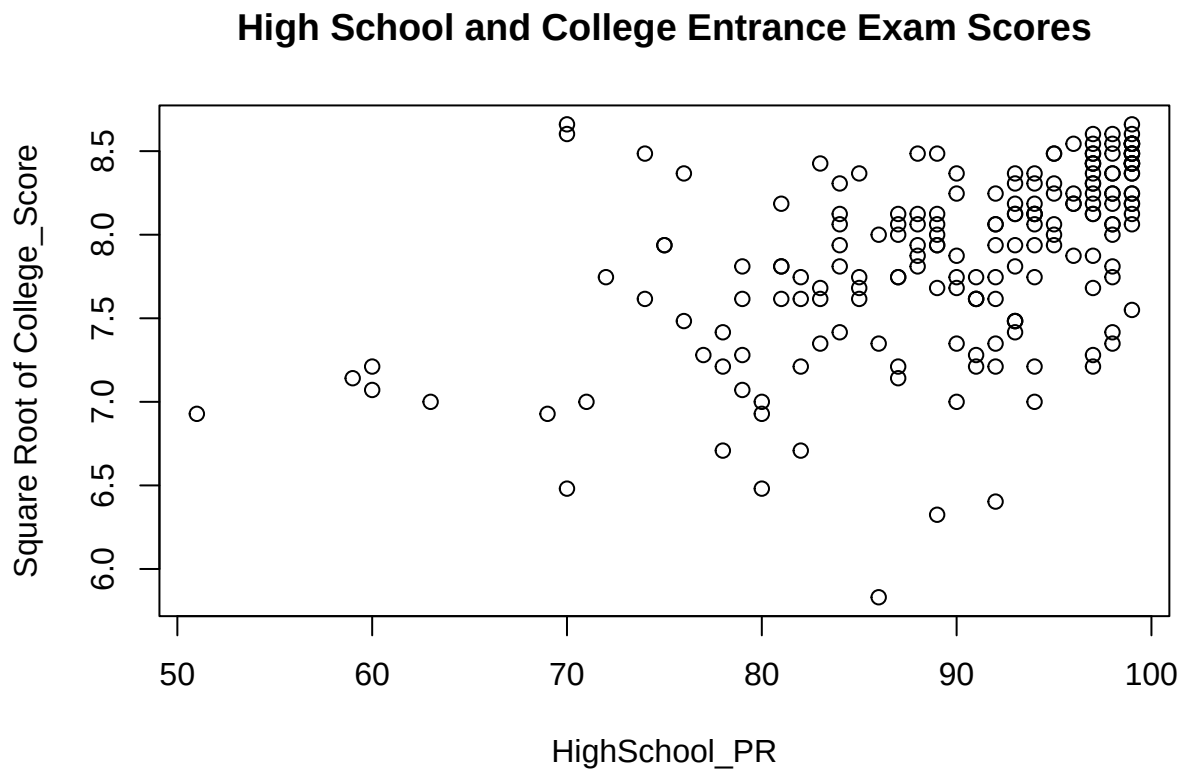
1. **Linearity**: The data has a linear trend, not a curve.
2. **Nearly normal residuals**: The residuals should be nearly normal, and we need to beware of outliers and influential points.
3. **Constant variability**: The variability of Y is constant and does not depend on the value of X.
4. **Independent observations**: Each observation (datapoint) is independent of the others.

5.1 Should we run a linear regression?

It is inappropriate to perform linear regression directly, because the data do not meet the constant variability assumption. In the bivariate exploratory plot, we can see that the variability of **College_Score** (Y) increases as **HighSchool_PR** (X) increases. One possible remedy is apply the square root transformation to **College_Score**, in order to reduce the variability. But the scatterplot below shows little to no improvement in variability, and the correlation coefficient even drops from 0.507 to 0.504. Hence we determine that it is not a good idea to run a linear regression model on the whole dataset.

```
# data version: already removed missing data
# data_corr = data[-missing_rows,]

plot(data_corr$HighSchool_PR, sqrt(data_corr$College_Score),
     main = "High School and College Entrance Exam Scores",
     xlab="HighSchool_PR",
     ylab="Square Root of College_Score")
```




```
cor(data_corr$HighSchool_PR, sqrt(data_corr$College_Score))
```

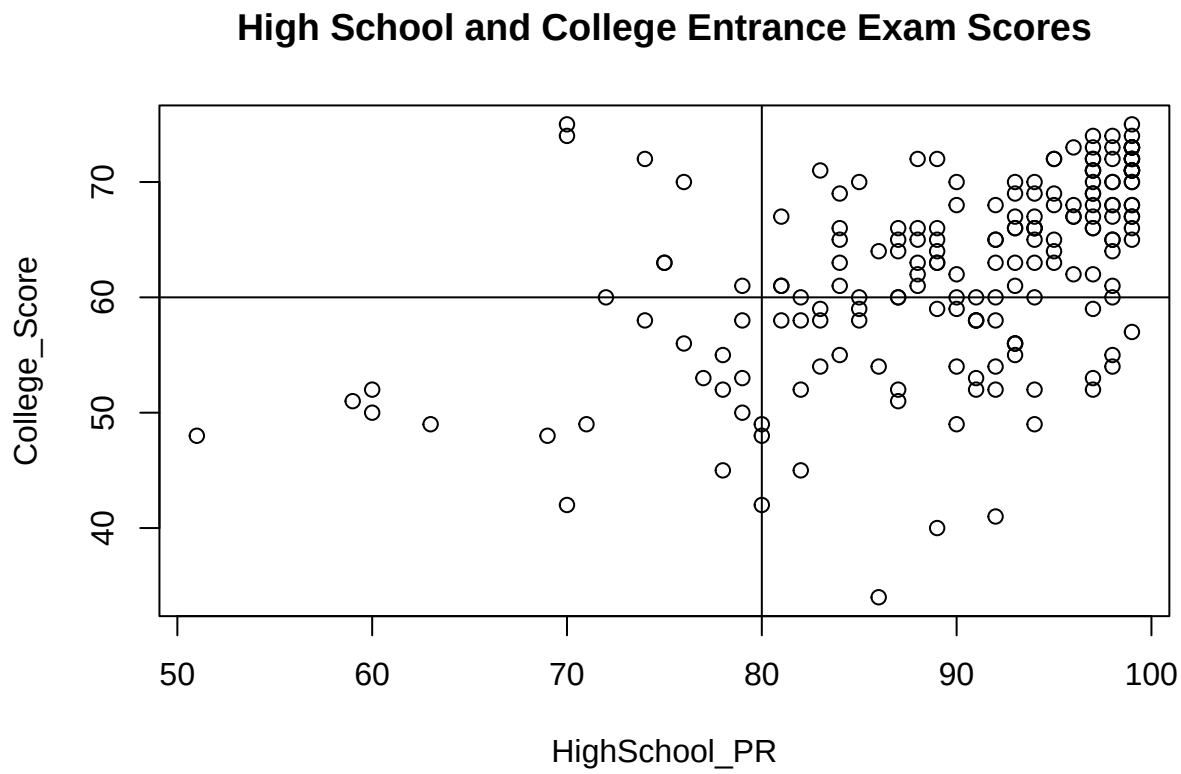
```
## [1] 0.5048132
```

5.2 Segmenting the Data

Instead, we should segment the data and further examine the top scorers in the dataset, i.e., those with **HighSchool_PR** 80 or higher. Most of these respondents have **College_Score** of 60 or higher, but the range of **College_Score** is wide. Here, we add horizontal and vertical lines to clarify the graph.

```
plot(data_corr$HighSchool_PR, data_corr$College_Score,
     main = "High School and College Entrance Exam Scores",
     xlab="HighSchool_PR",
     ylab="College_Score")

abline(h=60,v=80)
```



We can also create a contingency table (a.k.a. cross tabulation) for the two indicators **HighSchool_80up** and **College_60up**, which displays the bivariate frequency distribution in terms of counts.

- **HighSchool_80up**: Indicator of whether **HighSchool_PR** is 80 or higher
- **College_60up**: Indicator of whether **College_Score** is 60 or higher

```
data_corr$HS_80up = data_corr$HighSchool_PR >= 80
data_corr$CS_60up = data_corr$College_Score >=60

contingency = table(data_corr$HS_80up, data_corr$CS_60up,
                    dnn=c("HighSchool_80up","College_60up"))

contingency
```

```
##                College_60up
## HighSchool_80up FALSE TRUE
##           FALSE    17    8
##           TRUE     43  120
```

To make the table easier to read, we revert the order of `FALSE` and `TRUE` in the contingency table by calling the indices in reverse order.

```
contingency = contingency[2:1, 2:1]
contingency
```

```
##                College_60up
## HighSchool_80up TRUE FALSE
##           TRUE   120    43
##           FALSE    8    17
```

Below is the percentage version of the contingency table, and we can see that more than 63.5% of the respondents have both **HighSchool_PR** ≥ 80 and **College_Score** ≥ 60. This is also evidence that the PTT users tend to come from the population who scored well on the high school and college entrance exams.

```
prop.table(contingency)
```

```
##                College_60up
## HighSchool_80up       TRUE      FALSE
##           TRUE  0.63829787 0.22872340
##           FALSE 0.04255319 0.09042553
```

We can also round the percentage version to four decimal places in the ratio, so we will have two decimal places after the integer percentage. For example, 0.4528 becomes 45.28%.

```
round(prop.table(contingency),4)
```

```
##                College_60up
## HighSchool_80up   TRUE  FALSE
##           TRUE  0.6383 0.2287
##           FALSE 0.0426 0.0904
```

5.3 Conditional Probability

Using conditional probability, we can answer this question from the data: If a person scores at least 80 on the high school entrance score percentile rank (PR), how likely is he/she going to obtain a score at least 60 on the college entrance exam?

In mathematical terms, this is equivalent to finding $P(\text{College_60up is true} \mid \text{HighSchool_80up is true})$. Recall the conditional probability formula and the Bayes theorem:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

In this data, we have

- $P(\text{HighSchool_80up is true})$ = # of respondents with **HighSchool_PR** ≥ 80 / all respondents.
- $P(\text{College_60up is true})$ = # of respondents with **College_Score** ≥ 60 / all respondents.
- $P(\text{HighSchool_80up is true} \cap \text{College_60up is true})$
  = # of respondents with **HighSchool_PR** ≥ 80 and **College_Score** ≥ 60 / all respondents.

Plugging the numbers into the equation, we get

$$\begin{aligned}
&P(\text{College_60up is true} \mid \text{HighSchool_80up is true}) \\
&= \frac{P(\text{HighSchool_80up is true} \cap \text{College_60up is true})}{P(\text{HighSchool_80up is true})} \\
&= \frac{\#\text{ of respondents with HighSchool_PR} \geq 80 \text{ and College_Score} \geq 60}{\#\text{ of respondents with HighSchool_PR} \geq 80} \\
&= \frac{120}{43+120} \approx 0.7362.
\end{aligned}$$

According to this data from PTT, there is a 73.62% chance for a person to score at least 60 on the college entrance exam, given that he/she scored at least 80 on the high school entrance score percentile rank (PR). Note that we use number of respondents rather than percentage to avoid rounding errors.

In comparison, if we do not know anything about the person's high school entrance score percentile rank (PR), we have a probability of 63.82% in observing the person scoring at least 60 on the college entrance

exam. There is an increase of 9.80% in probability after we learn information about his/her high school entrance exam score.

$$\begin{aligned} P(\text{College_60up is true}) &= \text{\# of respondents with College_Score} \geq 60 \text{ / all respondents} \\ &= \frac{120}{188} \approx 0.6382. \end{aligned}$$

```
nrow(data_corr) # number of all respondents without missing data
```

```
## [1] 188
```

Remark: Conditional probability is the foundation of Bayesian statistics, which updates the existing probabilities given the new data. For the interested readers, we recommend the book *An Introduction to Bayesian Thinking: A Companion to the Statistics with* R *Course* (Clyde et al., 2020) as a starting point to learn Bayesian statistics. It is the supplementary materials for the Bayesian statistics course on Coursera from Duke University.[17]

6 Top Scorers: A Closer Look

We are going to take a closer look at the top scorers, given the observation in Section 5.2. In Taiwan's education system, the top tier of high schools and colleges can be further segmented. The top of the top tier can be very different than the bottom of top tier. Therefore, we consider the following subcategories:

- **HighSchool_PR** ranges: 80-89, 90-94, 95-99
- **College_Score** ranges: 60-64, 65-69, 70-75

6.1 Data Consistency

Before we start the analysis, we need to ensure consistency in the data. For instance, there are 191 records of valid high school entrance exam scores in the data. But if we consider only the ones with a valid college entrance exam score, the number of available records drops to 188. Although which version we use does not matter much when we look at the univariate distribution, this is going to be problematic when we combine the univariate analysis with the bivariate analysis. Thus, we should use only the 188 records whose college entrance exam scores are also valid.

```
# High school entrance exam scores: Remove missing values
# uni_HS_score = data$HighSchool_PR[which(data$HighSchool_PR != -1)]
length(uni_HS_score)
```

```
## [1] 191
```

```
# Consider only the records with both valid HighSchool_PR and College_Score
length(data_corr$HighSchool_PR)
```

```
## [1] 188
```

The same data consistency requirement also applies to the college entrance scores. There are 194 records of valid high school entrance exam scores in the data, but only 188 of them also have corresponding valid high school entrance exam scores. Accordingly, we should use only the 188 records whose high school entrance exam scores are also valid.

```
# College entrance exam scores: Remove missing values
# uni_college_score = data$College_Score[which(data$College_Score != -1)]
length(uni_college_score)
```

```
## [1] 194
```

[17] https://www.coursera.org/learn/bayesian

```r
# Consider only the records with both valid HighSchool_PR and College_Score
length(data_corr$College_Score)
```

```
## [1] 188
```

6.2 High School Entrance Exam Scores (Percentile Rank) at least 80

We use the `R` function `table` to show the frequency of each **HighSchool_PR** value that is at least 80, and we have 163 values in total. In the table below, the first row is the PR (percentile rank), and the second row is the counts. Although we truncated the **HighSchool_PR** to 80 and above, the distribution is still left-skewed. The **HighSchool_PR** 99 has the highest counts, followed by **HighSchool_PR** 97 and **HighSchool_PR** 98.

```r
HS_PR_seg = data_corr$HighSchool_PR[which(data_corr$HS_80up == TRUE)]
length(HS_PR_seg)
```

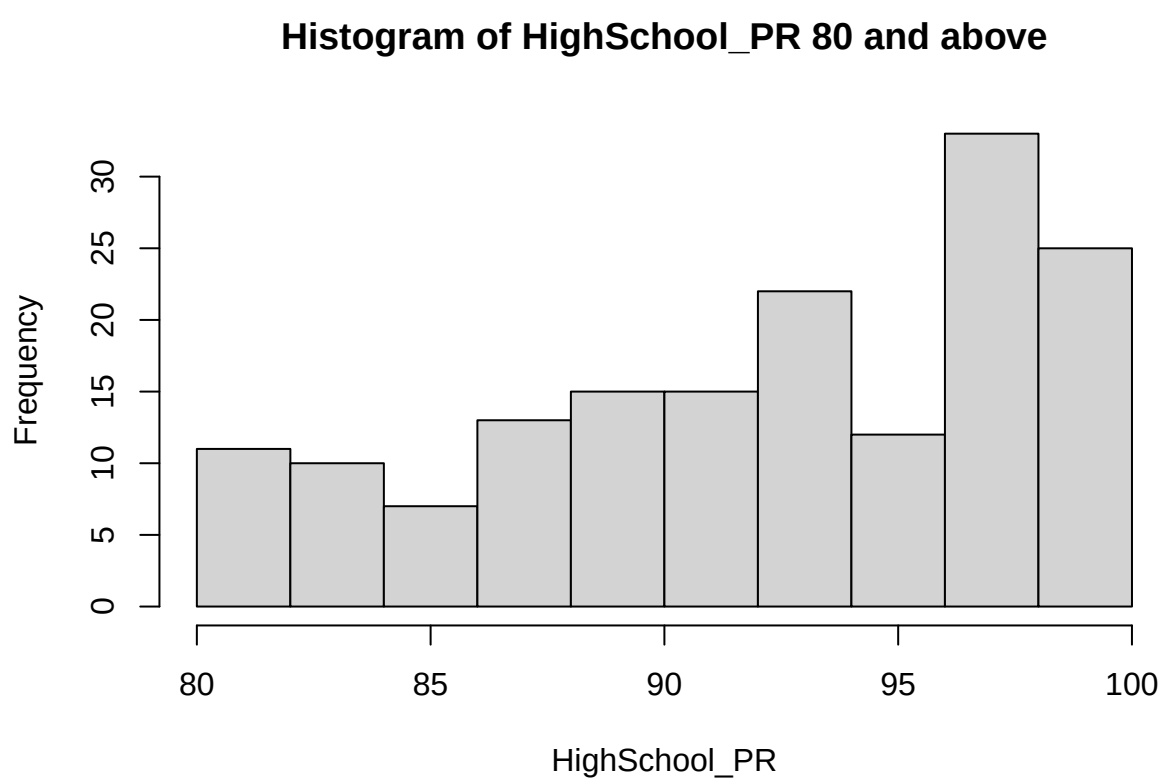
```
## [1] 163
```

```r
table(HS_PR_seg)
```

```
## HS_PR_seg
## 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
##  3  4  4  4  6  4  3  7  6  8  7  6  9 11 11  7  5 18 15 25
```

Or if you prefer a histogram, we can also create one.

```r
hist(HS_PR_seg, xlab="HighSchool_PR",
     main="Histogram of HighSchool_PR 80 and above")
```



Therefore, we create the breakdown of the **HighSchool_PR** ranges: 80-89, 90-94, 95-99. There are 49 records in the 80-89 range, 44 records in 90-94, and 70 records in 95-99. We divided the 90-99 range into 90-94 and 95-99, but the number of **HighSchool_PR** records in the 95-99 range is still higher than any of the other two categories.

However, it is not a good idea to further divide the 95-99 range into 95-97 and 98-99, due to the lack of geographic information. In Taipei, the high school enrollment is extremely competitive. Students with **HighSchool_PR** 95 and those with **HighSchool_PR** 99 would get admitted to high schools of different rankings.[18] But in other parts of Taiwan, most students with **HighSchool_PR** at least 95 would already qualify for the top local high school, and some rural parts even require a lower **HighSchool_PR** to get into the county's top high school.

```
HS80to89 = length(which(HS_PR_seg >= 80 & HS_PR_seg <= 89))
HS90to94 = length(which(HS_PR_seg >= 90 & HS_PR_seg <= 94))
HS95to99 = length(which(HS_PR_seg >= 95 & HS_PR_seg <= 99))
```

```
print(paste("HighSchool_PR 80-89:",HS80to89))
```

```
## [1] "HighSchool_PR 80-89: 49"
```

```
print(paste("HighSchool_PR 90-94:",HS90to94))
```

```
## [1] "HighSchool_PR 90-94: 44"
```

```
print(paste("HighSchool_PR 95-99:",HS95to99))
```

```
## [1] "HighSchool_PR 95-99: 70"
```

6.3 College Entrance Exam Scores at least 60

Similar to the previous section, we also show the frequency of each **College_Score** value that is at least 60. The total counts is 128, fewer than the 163 counts with **HighSchool_PR** 80 or above. The distribution is relatively uniform for **College_Score** values between 60 and 73, with a steep decline in the counts of **College_Score** 74 and 75 (max possible score). On the college entrance exam, only four respondents scored 74 and two scored 75. According to the historical statistics of the college entrance exam in Taiwan, **College_Score** 74 and 75 account for approximately 0.2% of all test takers each year, which is quite a small percentage.

```
CS_Score_seg = data_corr$College_Score[which(data_corr$CS_60up == TRUE)]
length(CS_Score_seg)
```

```
## [1] 128
```

```
table(CS_Score_seg)
```

```
## CS_Score_seg
## 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
## 10  7  4 10  5 11 12  9  9  6 10  9 12  8  4  2
```

Before we display the histogram, let's create a table to (approximately) convert **College_Score** into PR (Percentile Rank) using 2001-2014 data.[19] This gives readers an understanding of what percentage of test takers (high school students in grade 12) get what scores. For example, a **College_Score** of 70 would be at the 98.5th percentile, i.e., PR 98.5.

```
college_score = c(60,65,70,74,75)
college_pr = c(88, 95, 98.5, 99.9, 100)
```

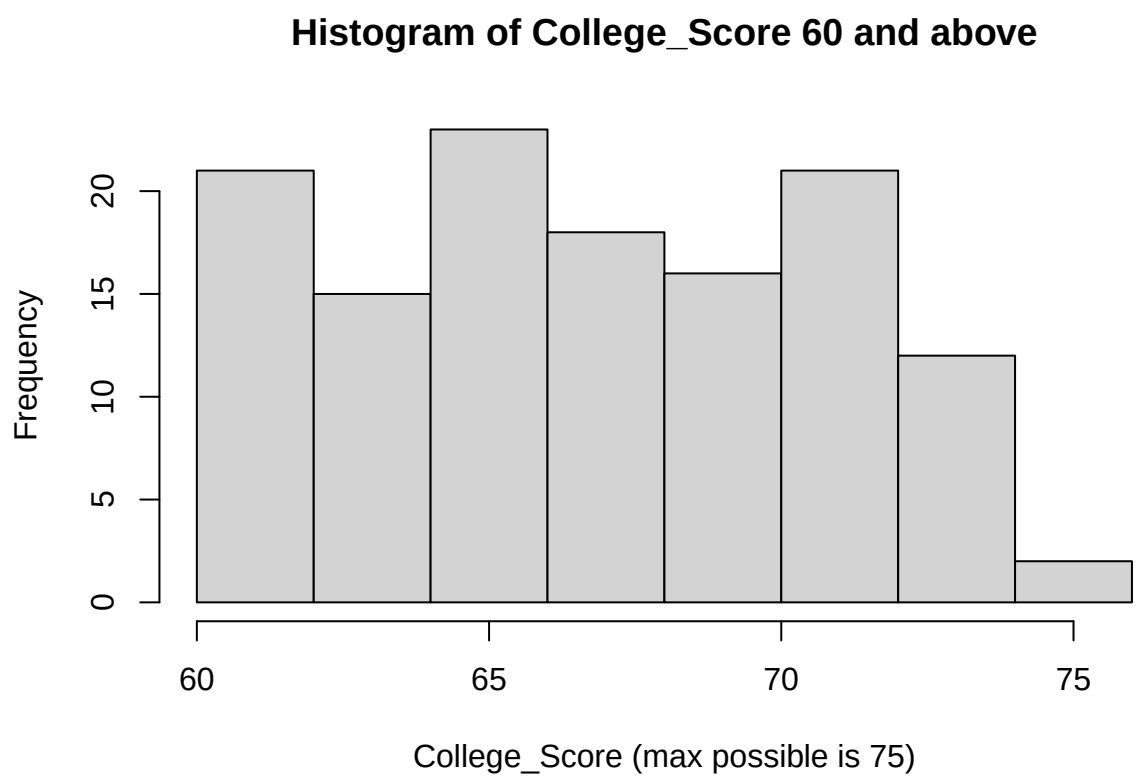
[18] https://w199584.pixnet.net/blog/post/28321887
[19] https://web.archive.org/web/20150207042900/http://www.ceec.edu.tw/AbilityExam/AbilityExamStat.htm

```r
data.frame(college_score, college_pr)
```

```
##   college_score college_pr
## 1            60       88.0
## 2            65       95.0
## 3            70       98.5
## 4            74       99.9
## 5            75      100.0
```

Here is the histogram of the **College_Score** values 60 and above.

```r
hist(CS_Score_seg, xlab="College_Score (max possible is 75)",
     main="Histogram of College_Score 60 and above")
```



We also create the breakdown of the **College_Score** ranges: 60-64, 65-69, 70-75. There are 36 records in the 60-64 range, 47 records in 65-69, and 45 records in 70-75. This is also relatively more uniform than the **HighSchool_PR** breakdown (49, 44, and 70 records each).

```r
CS60to64 = length(which(CS_Score_seg >= 60 & CS_Score_seg <= 64))
CS65to69 = length(which(CS_Score_seg >= 65 & CS_Score_seg <= 69))
CS70to75 = length(which(CS_Score_seg >= 70 & CS_Score_seg <= 75))
```

```r
print(paste("College_Score 60-64:",CS60to64))
```

```
## [1] "College_Score 60-64: 36"
```

```r
print(paste("College_Score 65-69:",CS65to69))
```

```
## [1] "College_Score 65-69: 47"
```

```r
print(paste("College_Score 70-75:",CS70to75))
```

```
## [1] "College_Score 70-75: 45"
```

6.4 Bivariate Exploration of High Scorers

Section 4.3 explored the bivariate relationship between **HighSchool_PR** and **College_Score**, and this time we would like to focus on the high scorers: respondents with **HighSchool_PR** $\geq$ 80 and/or **College_Score** $\geq$ 60. There are 163 respondents with **HighSchool_PR** $\geq$ 80, 128 respondents with **College_Score** $\geq$ 60, and 120 respondents with both. Since the number of respondents with **HighSchool_PR** $\geq$ 80 does not equal to the number of respondents with **College_Score** $\geq$ 60, we should consider the full 188 records in the data. Hence we add the range 0-79 to **HighSchool_PR**, and the range 0-59 for **College_Score**. We would like to create a 4x4 table for the following ranges:

- **HighSchool_PR** ranges: 0-79, 80-89, 90-94, 95-99
- **College_Score** ranges: 0-59, 60-64, 65-69, 70-75

Here, we use the `for` loop and `if-else` logic to map **HighSchool_PR** and **College_Score** into their corresponding ranges. The `else if` statement is executed when and only when the `if` statement is not true, so we can assign the score to the appropriate category using sequential `if-else` statements for range boundaries.

```r
data_corr$HS_range = "set"
data_corr$CS_range = "set"

for(ii in 1:nrow(data_corr)) {
        # High School PR categories
        if (data_corr$HighSchool_PR[ii] <= 79) {
                data_corr$HS_range[ii] = "0-79"
        } else if (data_corr$HighSchool_PR[ii] <= 89) {
                data_corr$HS_range[ii] = "80-89"
        } else if (data_corr$HighSchool_PR[ii] <= 94) {
                data_corr$HS_range[ii] = "90-94"
        } else {
                data_corr$HS_range[ii] = "95-99"
        }

        # College Score Categories
        if (data_corr$College_Score[ii] <= 59) {
                data_corr$CS_range[ii] = "0-59"
        } else if (data_corr$College_Score[ii] <= 64) {
                data_corr$CS_range[ii] = "60-64"
        } else if (data_corr$College_Score[ii] <= 69) {
                data_corr$CS_range[ii] = "65-69"
        } else {
                data_corr$CS_range[ii] = "70-75"
        }
}
```

We continue to use the `R` function `table` to create the 4x4 contingency table between the ranges of **HighSchool_PR** and **College_Score**. As we can see in the horizontal rows, the majority of respondents with **HighSchool_PR** less than 80 have a **College_Score** less than 60. For the respondents with **HighSchool_PR** between 80 and 94, the **College_Score** varies widely. The respondents with **HighSchool_PR** 95 or above performed the best in terms of **College_Score** – most of them scored 65 or higher.

In the vertical columns, the respondents with **College_Score** less than 60 mostly had a **HighSchool_PR** 94 or below; few came from the group of **HighSchool_PR** 95-99. For the respondents with **College_Score** between 60 and 64, their **HighSchool_PR** varied widely. Approximately half of the respondents with **College_Score** had **HighSchool_PR** 95 or above. For the top group of **College_Score** 70 or above, more than three quarters (34 out of 45) came from the respondents with **HighSchool_PR** 95 or higher.

```
table_4x4 = table(data_corr$HS_range, data_corr$CS_range,
                  dnn=c("HighSchool_PR","College_Score"))
table_4x4
```

```
##              College_Score
## HighSchool_PR 0-59 60-64 65-69 70-75
##         0-79    17     4     0     4
##         80-89   19    16    10     4
##         90-94   18     9    14     3
##         95-99    6     7    23    34
```

The contingency table shows a positive association between **HighSchool_PR** and **College_Score**: Respondents with a good **HighSchool_PR** score are more likely to achieve a good **College_Score**, but this is not guaranteed. Respondents who scored poorly in **HighSchool_PR** still had a small chance to do exceptionally well in **College_Score** later. Our findings align with the conventional wisdom that **HighSchool_PR** and **College_Score** are somewhat related, but a high score on **HighSchool_PR** does not guarantee a high score on **College_Score**.

7 Logistic Regression

We decided to perform a different model to quantify the relationship between **HighSchool_PR** and **College_Score**, because we concluded in Chapter 5 that it is inappropriate to perform an ordinary linear regression. (We also tried the square root transformation, but it did not work out, either.) Let's try another statistical model to evaluate the relationship between the two variables. Typically when the ordinary linear regression model is ruled out, the next candidate is a generalized linear model, such as logistic regression and Poisson regression. Choosing a different model may involve modifying the details of the problem statement. The original problem statement is to investigate the relationship between **College_Score** and **HighSchool_PR**, but it is flexible and does not require the relationship to be linear.

We would like to redefine the problem statement to "estimate the probability of **College_Score** at least 65, given the student's **HighSchool_PR**." Since the variance of **College_Score** depends on **HighSchool_PR**, the assumptions of linear regression are violated, making linear regression an inappropriate model. We decided to focus on whether **College_Score** is at least a particular value instead, so the response variable is binary. We selected 65 as the cutoff point for **College_Score** because this is close to the median, making the number of values about the same in the two categories. We would like to balance the number of datapoints in each category of the response variable.

Logistic regression is a generalized linear model that uses a binary response variable, and the equation models the probability of an event occurring or not. That's why we set up the new problem statement this way, and Section 7.1 gives a brief introduction to the logistic regression model. Although logistic regression is not typically covered at the Statistics 101 level, we would like to give the readers a head start of generalized linear models. We explained the requirements of linear regression in Section 5, and now we would like to expand the regression model to additional forms. We decided to introduce generalized linear models early on, because we would like to show that there exist methods to analyze the data `ptt_SENIORHIGH_data.csv` other than linear regression.

We try to explain the logistic regression in basic terminology. But if the readers feel like they cannot understand the mathematics, it is totally fine to skip this chapter and move on to Chapter 8 for model validation. The model evaluation concepts are not related to the logistic regression itself, so understanding the model details is not always a prerequisite. We can regard the model as a blackbox, which simply produces

the binary classification output.

7.1 Brief Introduction of Logistic Regression

Logistic regression is used to model categorical outcomes, especially a binary response variable. For example, if the response variable is whether an event Y occurs or not, then it can only have two values – not occurred (0) and occurred (1). We model the probability that the event Y occurred, denoted as $p = P(Y = 1)$, and it is between 0 and 1. But in a linear regression $y = \alpha + \beta x$, the response variable y can be any real number. Therefore, we transform the probability p to the log odds $\log(\frac{p}{1-p})$, so that its range spans the whole real line like y. Note that the odds $\frac{p}{1-p}$ is always positive, as long as p is not exactly 0 or 1.

The equation for logistic regression is written as below:

$$\text{logit}(p) = \log(\frac{p}{1-p}) = \alpha + \beta X$$

The notation is similar to a linear regression, but the interpretation is slightly different. α is the intercept, and β is the coefficient. When β increases by one unit, the log odds $\log(\frac{p}{1-p})$ increases by β units. In other words, when β increases by one unit, the odds $\frac{p}{1-p}$ are multiplied by $e = \exp(1) \approx 2.71828$.

The probability p can be estimated from the logistic regression model with the equation below.

$$p = \frac{\exp(\alpha + \beta X)}{1 + \exp(\alpha + \beta X)}$$

The intercept α serves as a baseline when $X = 0$, and the probability p can be estimated by $p = \frac{\exp(\alpha)}{1 + \exp(\alpha)}$. Just like in an ordinary linear regression, the intercept may or may not have practical meaning. For example, if we examine the relationship between people's height and weight, nobody is going to have height of 0 inches.

For the advanced readers, we recommend reading the textbook *Categorical Data Analysis* (Agresti, 2003) to learn more about logistic regression and other generalized linear models for categorical data.

7.2 Estimate the Probability of Scoring Well on the College Entrance Exam

We would like to redefine the problem statement to "estimate the probability of **College_Score** at least 65, given the student's **HighSchool_PR**." Since the variance of **College_Score** depends on **HighSchool_PR**, the assumptions of linear regression are violated, making linear regression an inappropriate model. That's why we decided to focus on whether **College_Score** is at least a particular value instead, so the response variable is binary. We selected 65 as the cutoff point for **College_Score** because this is close to the median, making the number of values about the same in the two categories. We would like to balance the number of datapoints in each category of the response variable.

```
print(paste("College_Score at least 65:", sum(data_corr$College_Score >= 65)))
```

```
## [1] "College_Score at least 65: 92"
```

```
print(paste("College_Score below 65:", sum(data_corr$College_Score < 65)))
```

```
## [1] "College_Score below 65: 96"
```

The event Y we would like to observe is getting a **College_Score** at least 65. We define

$$p = P(Y = 1) = P(\textbf{College_Score} \geq 65),$$

i.e., the probability of getting a **College_Score** at least 65. And the independent variable X is the **HighSchool_PR**, whose values are between 0 and 99.

Then we implement the logistic regression with the equation

$$\text{logit}(p) = \log(\frac{p}{1-p}) = \alpha + \beta X.$$

The model is written as `glm(y ~ x, data=data, family="binomial")` in `R` code, where `glm` stands for generalized linear regression. The `family` option is set to binomial, because the response variable is binary and can only have values 0 or 1. Hence our logistic model can be written as

$$\text{logit}(P(\textbf{College_Score} \geq 65)) \sim \alpha + \beta * \textbf{HighSchool_PR}.$$

Let's create the logistic regression model in `R` as below.

```r
# 1. Create the binary variable first.
# 2. model = glm( y ~ x, data=data, family="binomial")
# 3. summary(model)
# https://stats.idre.ucla.edu/r/dae/logit-regression/

data_corr$CS_65up = data_corr$College_Score >= 65
model = glm(CS_65up ~ HighSchool_PR, data=data_corr, family="binomial")
summary(model)
```

```
##
## Call:
## glm(formula = CS_65up ~ HighSchool_PR, family = "binomial", data = data_corr)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7122  -0.9948  -0.1334   0.8248   2.5246
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -13.63939    2.45755  -5.550 2.86e-08 ***
## HighSchool_PR   0.14993    0.02674   5.607 2.06e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 260.54  on 187  degrees of freedom
## Residual deviance: 211.98  on 186  degrees of freedom
## AIC: 215.98
##
## Number of Fisher Scoring iterations: 5
```

7.3 Model Interpretation: Point Estimates

The `Pr(>|z|)` is the p-value of each independent variable, and we can see that **HighSchool_PR** is statistically significant because p-value < 0.05. When **HighSchool_PR** increases by one, the log odds $\log(\frac{p}{1-p})$ of getting **College_Score** at least 65 increases by approximately 0.15. After transforming log odds $\log(\frac{p}{1-p})$ back to odds $\frac{p}{1-p}$, we get that the odds are multiplied by $e = \exp(0.15) \approx 1.161$. In other

words, when **HighSchool_PR** increases by one, the odds of getting **College_Score** at least 65 increases by approximately 16.1%.

For better reproducibility of coefficients, these can be retrieved using the code below.

```
model$coefficients
```

```
##   (Intercept) HighSchool_PR
##   -13.6393909     0.1499257
```

We can also round the coefficients to the third digit after the decimal point. But for better precision, we do not recommend rounding numbers until we reach the final calculation results.

```
round(model$coefficients, digits=3)
```

```
##   (Intercept) HighSchool_PR
##       -13.639         0.150
```

Here is the exponential of the coefficients, because we need to transform log odds into odds.

```
exp(model$coefficients)
```

```
##   (Intercept) HighSchool_PR
##  1.192581e-06  1.161748e+00
```

The intercept serves as a baseline for the logistic regression model when **HighSchool_PR** is 0. We can predict p under this condition, and find out how likely this (fictitious) person is going to get **College_Score** at least 65. The estimated probability is extremely low, less than 0.01%. Nevertheless, the value of **HighSchool_PR** starts at 1, so the intercept does not have an intrinsic meaning. (And typically most students who score less than 10 in **HighSchool_PR** would not be interested in attending college, either.)

```
alpha = as.numeric(model$coefficients[1])
beta = as.numeric(model$coefficients[2])

p_intercept = exp(alpha)/(1+exp(alpha))
p_intercept
```

```
## [1] 1.192579e-06
```

In comparison, when **HighSchool_PR** is 99, the model estimates that the student has a 76.9% chance to achieve a **College_Score** of 65 or higher.

```
p_pr99 = exp(alpha + beta*99)/(1+exp(alpha + beta*99))
p_pr99
```

```
## [1] 0.7691025
```

If we look at the data, there are 25 respondents with **HighSchool_PR** 99, and only one of them scored below 65 in the **College_Score**. The data shows the conditional probability to be 96%.

$$P(\textbf{College_Score} \geq 65 | \textbf{HighSchool_PR} = 99) = \frac{24}{25} = 96\%.$$

In this **HighSchool_PR** 99 group, more than half of the respondents (14, to be exact) has a **College_Score** between 71 and 73. Nevertheless, **HighSchool_PR** 99 is not an (almost) necessary condition to achieve **College_Score** 65 or higher. In the group of **College_Score** 65 or higher, only 24 out of the 92 respondents had **HighSchool_PR** 99, which is less than a quarter.

$$P(\textbf{HighSchool_PR} = 99 | \textbf{College_Score} \geq 65) = \frac{24}{92} \approx 26\%.$$

```r
num_pr99 = sum(data_corr$HighSchool_PR == 99)
num_cs65 = sum(data_corr$College_Score >= 65)
num_pr99_and_cs65 = sum(data_corr$HighSchool_PR == 99 & data_corr$College_Score >= 65)
```

```r
print(paste("Number of respondents with HighSchool_PR 99:",num_pr99))
```

```
## [1] "Number of respondents with HighSchool_PR 99: 25"
```

```r
print(paste("Number of respondents with College_Score 65 or better:",num_cs65))
```

```
## [1] "Number of respondents with College_Score 65 or better: 92"
```

```r
print(paste("Number of respondents with HighSchool_PR 99 and College_Score 65 or better:",
            num_pr99_and_cs65))
```

```
## [1] "Number of respondents with HighSchool_PR 99 and College_Score 65 or better: 24"
```

```r
sort(data_corr$College_Score[which(data_corr$HighSchool_PR==99)])
```

```
##  [1] 57 65 66 67 67 68 68 70 70 71 71 71 71 71 72 72 72 72 73 73 73 73 73 74 75
```

7.4 Model Interpretation: 95% Confidence Intervals

In addition to the point estimates, we also need to provide the corresponding 95% confidence intervals to account for uncertainty. The lower bound is the 2.5th percentile, and the upper bound is the 97.5th percentile. Statistical significance at 5% means that the 95% confidence interval does not include 0. Let's start with the intercept and the coefficient for **HighSchool_PR** using the `R` function `confint`. Neither the intercept nor the coeffient's confidence interval includes 0, so both are statistically significant.

Although the intercept α's confidence interval seems wide, the exponential version $\exp(\alpha)$ is extremely small for both ends. Especially when the intercept (baseline for **HighSchool_PR** $= 0$) does not have practical meaning in this data, we do not need to be overly concerned about the intercept.

On the other hand, the coefficient β has a point estimate of approximately 0.15, with a 95% confidence interval [0.101, 0.206]. When **HighSchool_PR** increases by one, we can expect an increase between 0.101 and 0.206 in the log odds $\log(\frac{p}{1-p})$ of getting **College_Score** at least 65. We can also transform log odds $\log(\frac{p}{1-p})$ back to odds $\frac{p}{1-p}$, and get an expected increase factor between 1.106 and 1.229. We are 95% confident that the odds of getting **College_Score** at least 65 would increase by between 10.6% and 22.9%, given that **HighSchool_PR** increases by one.

```r
confint(model)
```

```
## Waiting for profiling to be done...
```

```
##                     2.5 %     97.5 %
## (Intercept)   -18.7973934 -9.1378744
## HighSchool_PR   0.1008108  0.2059154
```

```r
exp(confint(model))
```

```
## Waiting for profiling to be done...
```

```
##                      2.5 %       97.5 %
## (Intercept)   6.861132e-09 0.0001075156
## HighSchool_PR 1.106067e+00 1.2286493088
```

We can also calculate the 95% confidence interval for the predicted probability of getting **College_Score** at least 65, given that the respondent scored a 99 on **HighSchool_PR**. However, the confidence interval is [0.00012, 0.99999], which does not make practical sense because a probability has natural boundaries of [0,1].

Why is this interval so wide? The linear component is $\alpha + \beta X$, so the range depends on the value of **HighSchool_PR**. When **HighSchool_PR** is large (say, 99), the 95% confidence interval of $\alpha + \beta X$ becomes extremely wide. The interval would be narrow for a small value of **HighSchool_PR**, such as less than 10. But people with extremely low **HighSchool_PR** are unlikely to be interested in taking the college entrance exam at all. Hence we are not going to calculate the 95% confidence interval for small **HighSchool_PR** values.

```
ci_matrix = confint(model)
alpha_ci = ci_matrix[1,]
beta_ci = ci_matrix[2,]
p_pr99_ci = exp(alpha_ci + beta_ci*99)/(1+exp(alpha_ci + beta_ci*99))
p_pr99_ci
```

```
##        2.5 %       97.5 %
## 0.0001481513 0.9999869636
```

Remark: The readers may wonder how the author(s) remember all of the `R` commands for the model. In fact, we don't need to! We can use the function `objects` to find all available outputs from the model, and the object names are descriptive.

```
objects(model)
```

```
##  [1] "aic"               "boundary"          "call"
##  [4] "coefficients"      "contrasts"         "control"
##  [7] "converged"         "data"              "deviance"
## [10] "df.null"           "df.residual"       "effects"
## [13] "family"            "fitted.values"     "formula"
## [16] "iter"              "linear.predictors" "method"
## [19] "model"             "null.deviance"     "offset"
## [22] "prior.weights"     "qr"                "R"
## [25] "rank"              "residuals"         "terms"
## [28] "weights"           "xlevels"           "y"
```

7.5 Overall Model Results

In addition to the coefficient estimates, we also need to perform model validation to examine how well the model fits the data. Let's start with visualizing the predicted probability of getting **College_Score** at least 65 and the **HighSchool_PR** values in the data. The former can be obtained from the `fitted.values` object of the model. The highest predicted probability occurs at **HighSchool_PR** 99, but the predicted probability of getting **College_Score** at least 65 is still less than 80%. (So high school students should still study hard for the college entrance exam, despite getting an excellent **HighSchool_PR** score.)

```
print(paste("The highest predicted probability is",max(model$fitted.values)))
```

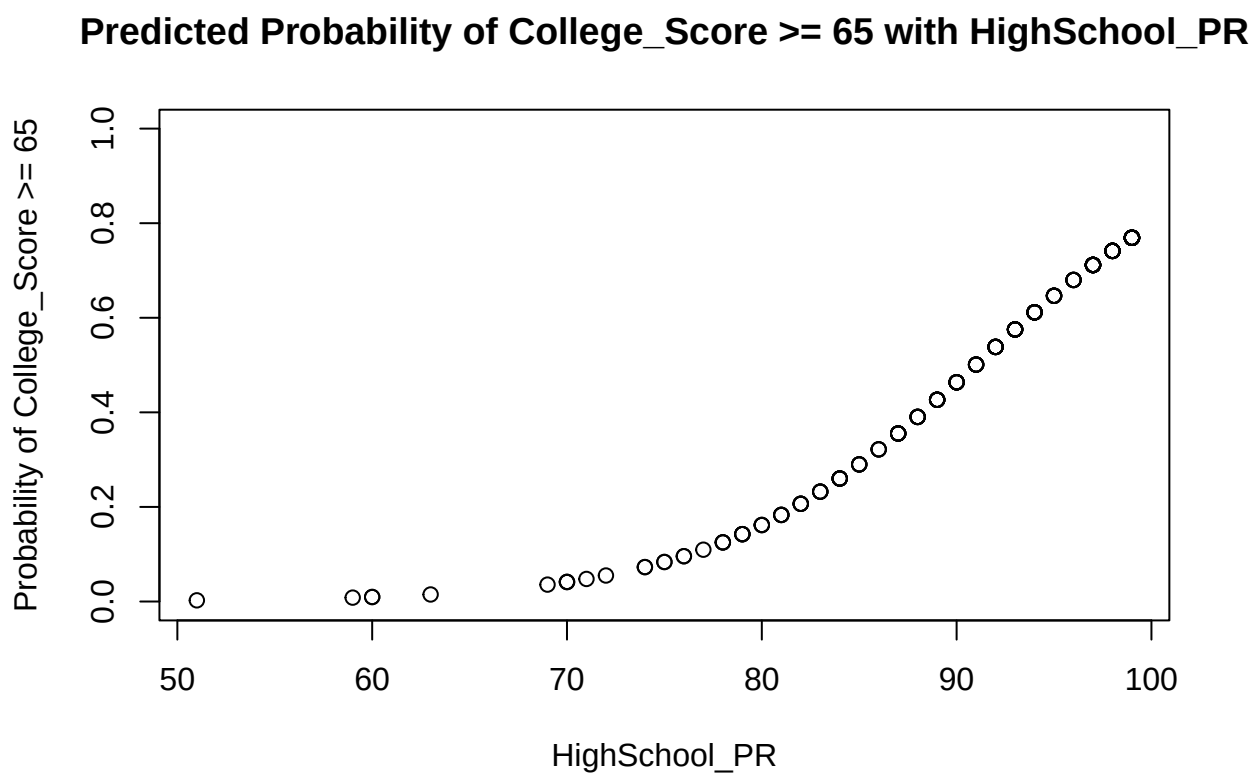
```
## [1] "The highest predicted probability is 0.769102549200389"
```

```
print(paste("This occurs at HighSchool_PR",
      data_corr$HighSchool_PR[which.max(model$fitted.values)]))
```

```
## [1] "This occurs at HighSchool_PR 99"
```

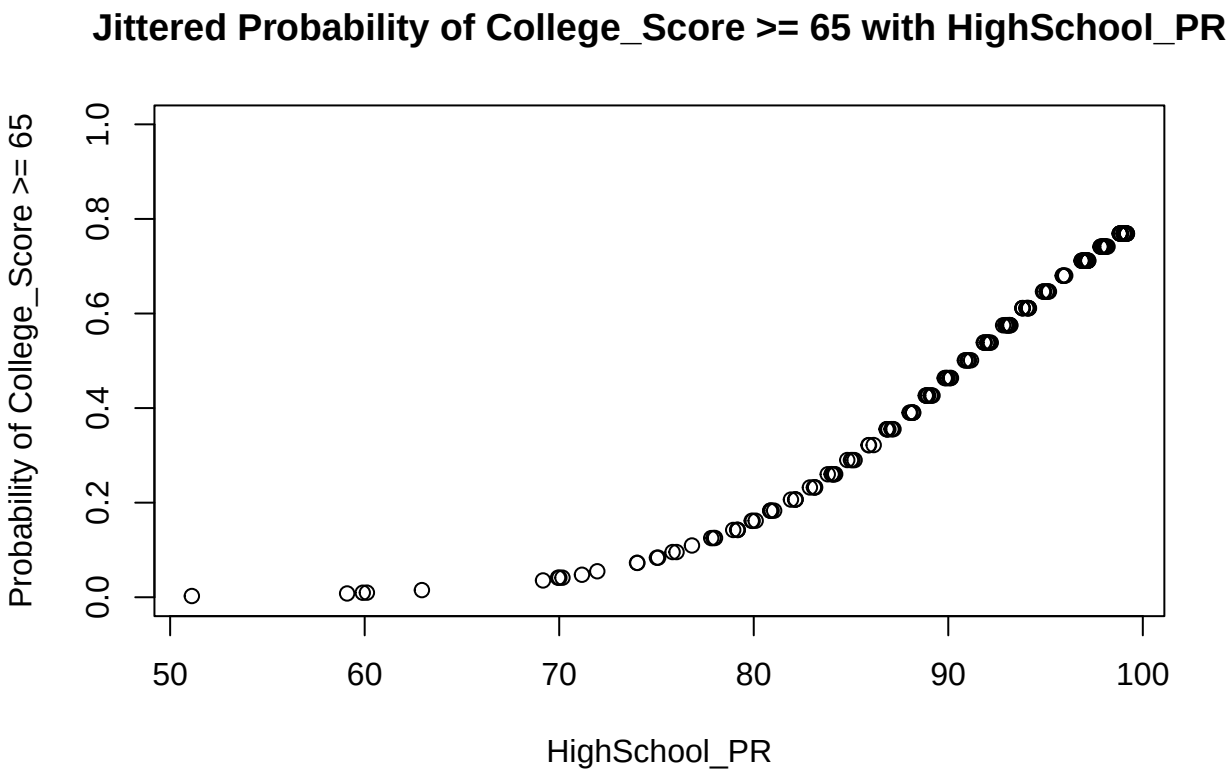
The graph shows different trends for different segments of the **HighSchool_PR**. When **HighSchool_PR** is less than 70, the predicted probability of getting **College_Score** at least 65 is close to zero. But we should take this observation with caution, because we have few data points with **HighSchool_PR** less than 70. When **HighSchool_PR** is between 70 and 79, the predicted probability increases with an almost linear trend. Starting at **HighSchool_PR** 80, the predicted probability increases with a steeper slope. Finally, the growth of the predicted probability slows down when **HighSchool_PR** reaches 98 (the maximum possible **HighSchool_PR** is 99).

```r
# First version of graph
yy = model$fitted.values
xx = data_corr$HighSchool_PR
plot(xx, yy, ylim=c(0,1),
     main="Predicted Probability of College_Score >= 65 with HighSchool_PR",
     xlab="HighSchool_PR",
     ylab="Probability of College_Score >= 65")
```



Since there are many repetitive values in **HighSchool_PR**, let's apply the `jitter` function to add random noise to the data for display. (Remember to set a random seed for reproducibility.) We can see that the deeper black circles indicate more records in the data, which are concentrated in the higher **HighSchool_PR** values.

```r
# Add jittering because of many repetitive values in HighSchool_PR
set.seed(21)
new_xx = jitter(xx)
new_yy = jitter(yy)
plot(new_xx, new_yy, ylim=c(0,1),
     main="Jittered Probability of College_Score >= 65 with HighSchool_PR",
     xlab="HighSchool_PR",
     ylab="Probability of College_Score >= 65")
```



8 Model Validation: In-Sample Prediction

We built the logistic regression model in Chapter 7, and now it is time for model validation, i.e., evaluate the model performance. We will focus on machine learning concepts in this chapter, and the readers simply need to know that **the model does binary classification**. For more about machine learning, we recommend the book *Introduction to Machine Learning* (Alpaydin, 2020). It explains a wide range of machine learning algorithms and applications, including the recent advances in deep learning and neural networks.

We start with in-sample prediction to obtain the binary classification results, then we explain how to interpret the 2x2 confusion matrix output. There are two actual outcomes for **College_Score** – at least 65 or not. There are also two predicted outcomes for **College_Score**, and we compare the predicted outcomes with the actual ones. Next, we would like to examine the model performance for different **HighSchool_PR** scores, in order to identify whether the model does better in higher or lower **HighSchool_PR**, or performs about the same. In Chapter 9, we will perform out-of-sample prediction on the model; that is, train the model on some data and test it on different data.

8.1 Implementation of In-Sample Prediction

Let's start the model validation with in-sample prediction; that is, using the data to predict the outcome of whether **College_Score** is at least 65 for each value of **HighSchool_PR** for values already in the data. If predicted probability of **College_Score** at least 65 (i.e., `fitted.values`) is greater than or equal to 0.5, we assign the predicted value as `TRUE`. We can safely use 0.5 as the probability threshold because the data are quite balanced. In other words, the proportion of **College_Score** at least 65 is close to 0.5 in the data (0.489, to be exact). If the data are imbalanced, say 80% of the records belong to one category, we should adjust the probability threshold in our classification prediction model.

```r
# nrow(data_corr) # 188
# sum(data_corr$CS_65up) # 92
# sum(data_corr$CS_65up)/nrow(data_corr) # 0.489

print(paste("There are",nrow(data_corr),"records in the data, with",
            sum(data_corr$CS_65up),
            "of them have College_Score at least 65."))
```

```
## [1] "There are 188 records in the data, with 92 of them have College_Score at least 65."
```

```r
print(paste("This is a proportion of",
            round(sum(data_corr$CS_65up)/nrow(data_corr),digits=3),
            ", which is close to 0.5."))
```

```
## [1] "This is a proportion of 0.489 , which is close to 0.5."
```

Let's create a confusion matrix to show the comparison between the actual outcomes and predicted outcomes, i.e., whether each respondent obtained **College_Score** at least 65 or not. Note that a confusion matrix is slightly different than the contingency table in Section 5.2, although both record counts in a matrix. A confusion matrix involves the predicted results, while a contingency table simply observes the categories in the data.

```r
# Data
actual_65up = data_corr$CS_65up

# Predicted results
predicted_65up = model$fitted.values >= 0.5

# Confusion matrix
confusion = table(actual_65up, predicted_65up)
# revert the order of FALSE and TRUE
confusion = confusion[2:1, 2:1]
confusion
```

```
##            predicted_65up
## actual_65up TRUE FALSE
##       TRUE    72    20
##       FALSE   35    61
```

Below is the percentage version of the confusion matrix.

```r
prop.table(confusion)
```

```
##            predicted_65up
## actual_65up      TRUE     FALSE
##       TRUE  0.3829787 0.1063830
##       FALSE 0.1861702 0.3244681
```

The table below shows the meaning of each cell of the confusion matrix. Assume that "positive" means getting **College_Score** at least 65, which is equivalent to the `TRUE` label in `actual_65up` and `predicted_65up`. The term "negative" means not getting **College_Score** at least 65, so it is equivalent to the `FALSE` label. For each cell, we have:[20]

- **True Positive (TP)**: The datapoint is actually positive and is predicted as positive, so the model correctly predicts the positive outcome.
- **True Negative (TN)**: The datapoint is actually negative and is predicted as negative, so the model correctly predicts the negative outcome.

[20] https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative

- **False Positive (FP)**: The datapoint is actually negative but is predicted as positive, so the model **in**correctly predicts the positive outcome, i.e., false alarm.
- **False Negative (FN)**: The datapoint is actually positive but is predicted as negative, so the model **in**correctly predicts the negative outcome, i.e., error.

```
##                 | Predicted Positive | Predicted Negative
## --------------  | ------------------ | ------------------
## Actual Positive |      True Positive |     False Negative
## --------------- | ------------------ | ------------------
## Actual Negative |     False Positive |      True Negative
```

We can retrieve each element in the confusion matrix by specifying the labels for each row and each column. The row indicates how many respondents actually obtained **College_Score** at least 65, and the column indicates how many respondents were predicted to have the positive outcome. Instead of writing the syntax like `confusion[1,2]`, we write it in a clearer way `confusion["TRUE","FALSE"]`. (Don't make the confusion matrix more confusing!) The readers can easily see that this is the number of respondents who we did not predict to have a **College_Score** at least 65, but they actually did.

```r
# row = actual_65up, column = predicted_65up
tp = confusion["TRUE","TRUE"]
fn = confusion["TRUE","FALSE"]
fp = confusion["FALSE","TRUE"]
tn = confusion["FALSE","FALSE"]

print(paste(tp,fn,fp,tn))
```

```
## [1] "72 20 35 61"
```

We can also verify that the retrieved numbers are exactly the same as in the original confusion matrix.

```r
confusion
```

```
##            predicted_65up
## actual_65up TRUE FALSE
##       TRUE    72    20
##       FALSE   35    61
```

8.2 Interpretation of Confusion Matrix

After creating the confusion matrix to record the model performance, we need to interpret the numbers and define metrics to measure the performance. We start with the overall **accuracy** to calculate how many datapoints the model predicted correctly. A datapoint is correctly predicted if one of the two scenarios occurs:

- True Positive: The datapoint is actually positive and it is predicted as positive.
- True Negative: The datapoint is actually negative and it is predicted as negative.

In the context of our dataset, "positive" means getting a **College_Score** 65 or higher, and "negative" means getting a **College_Score** of 64 or lower. We run the model to predict the respondents' college entrance exam outcome given their **HighSchool_PR**. In mathematical terms, accuracy can be calculated as below:

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Actual Positive} + \text{Actual Negative}} = \frac{\text{True Positive} + \text{True Negative}}{\text{All Datapoints}}$$

Plugging in the numbers from our model, we get

$$\text{Accuracy} = \frac{72 + 61}{72 + 20 + 35 + 61} \approx 70.74\%.$$

Our model correctly predicts whether the **College_Score** would be at least 65 or not around 70% of the time, which is better than a 50-50 coin flip. This means our model is informative, and the results align with the prior knowledge that a higher **HighSchool_PR** is more likely to lead to **College_Score** at least 65.

Note that when the data are imbalanced (say, 98% of the records belong to one category), accuracy is not a good measure of model performance.[21] The model can simply predict all datapoints to be in the larger category, and achieve 98% accuracy. The good news is that we get a relatively balanced dataset by setting the classification threshold of **College_Score** to be 65, as we explained at the beginning of Section 7.2. In fact, 48.9% of the respondents have a **College_Score** of 65 or higher.

```
print(paste("The proportion of respondents with College_Score 65 or higher is",
            round(sum(data_corr$CS_65up)/nrow(data_corr), digits = 3)))
```

```
## [1] "The proportion of respondents with College_Score 65 or higher is 0.489"
```

8.2.1 Precision and Recall

Precision and **recall** are also two widely-used metrics to measure the performance of the prediction model, and most importantly, they do not depend much on the proportions of data categories. Precision is defined as the percentage of true positives among all datapoints the model predicted to be positive. In our example, precision is the percentage of the respondents who actually got **College_Score** at least 65, among the respondents we predicted to make this achievement given their **HighSchool_PR**.

$$\text{Precision} = \frac{\text{True Positive}}{\text{Predicted Positive}} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

Plugging in the numbers from our model, we get

$$\text{Precision} = \frac{72}{72 + 35} \approx 67.29\%.$$

The precision would be useful if we use the predictive information to decide to invest in which high school students based on their **HighSchool_PR**. If we predict a student to get **College_Score** at least 65, he/she has about a 67.29% chance to make it, which is more than two-thirds. Since there are different tiers of high schools in Taiwan based on **HighSchool_PR**, many resources are given to the top tier high schools, because these students have the highest chance to do well on the college entrance exam. Nevertheless, other social factors also play a role in the overall policy decision-making process. For example, the government may decide to put more resources into remote rural high schools to empower disadvantaged students to succeed, which is beneficial for upward social mobility.

On the other hand, recall is defined as the percentage of model-predicted positives among all datapoints that are actually positive. In our example, recall is the percentage of the respondents we predicted to have **College_Score** at least 65 given their **HighSchool_PR**, among the respondents who actually made this achievement.

$$\text{Recall} = \frac{\text{True Positive}}{\text{Actual Positive}} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

Plugging in the numbers from our model, we get

$$\text{Recall} = \frac{72}{72 + 20} \approx 78.26\%.$$

The recall would also be useful if we use the predictive information to decide to invest in which high school students based on their **HighSchool_PR**. We need to remember that among the high school graduates

[21] https://www.analyticsvidhya.com/blog/2017/03/imbalanced-data-classification/

who got a **College_Score** at least 65, only 78.26% of them were predicted to have such potential. In other words, the remaining 21.74% of high school students did better than the predictive model had expected. It is possible that they were smart but accidentally did poorly on the high school entrance exam, and they would have achieved **College_Score** at least 65 regardless. Another possibility is that they did not study much in middle school and got a low **HighSchool_PR**, but they received extra help and/or studied much harder for the college entrance exam to get a **College_Score** at least 65. The lesson is that by pre-selecting students based on **HighSchool_PR**, we would still get some "dark horses", i.e., the students who performed much better on the **College_Score** than we had expected.

8.2.2 False Positive Rate and False Negative Rate

False positive rate and **false negative rate** are typically used to measure the accuracy of a medical screening test for a disease (NCSS, nd), where "positive" means having the disease and "negative" means not having the disease. In our dataset, "positive" means something much better – getting a **College_Score** 65 or higher. "Negative" means not achieving this.

False positive rate is the probability of an actual negative being classified as a positive, and it is also called a "false alarm". In medical terms, this means someone is tested positive for a disease, but actually does not have it. In our dataset, this means a student was predicted to get **College_Score** at least 65, but actually did not.

$$\text{False Positive Rate} = \frac{\text{False Positive}}{\text{Actual Negative}} = \frac{\text{False Positive}}{\text{True Negative} + \text{False Positive}}$$

Plugging in the numbers from our model, we get

$$\text{False Positive Rate} = \frac{35}{35 + 61} \approx 36.46\%.$$

Given that a student did not get **College_Score** at least 65, there is a 36.46% chance that we predicted him/her to achieve this. We would like to give the benefit of the doubt, saying that the student simply did not do well on the first college entrance exam for early admission. It is possible that he/she was able to get into a better school through taking the second college entrance exam for regular admission.

On the other hand, false negative rate is the probability of an actual positive being classified as a negative. In medical terms, this means someone is tested negative for a disease, but actually has the disease. In our dataset, this means a student actually got **College_Score** at least 65, but we predicted him/her as not achieving this.

$$\text{False Negative Rate} = \frac{\text{False Negative}}{\text{Actual Positive}} = \frac{\text{False Negative}}{\text{True Positive} + \text{False Negative}}$$

False negative rate means how likely the model missed actual positive datapoints, so this rate is the opposite of recall.

$$\text{False Negative Rate} = 1 - \text{Recall}$$

Plugging in the numbers from our model, we get

$$\text{False Negative Rate} = \frac{20}{72 + 20} \approx 21.74\%.$$

Given that a student actually got **College_Score** at least 65, there is a 21.74% chance that we did not predict him/her to achieve this. We need to remember that the model is imperfect, so there always exist students who did better in **College_Score** than the model had expected. To put it differently, **HighSchool_PR**

is not a full indicator of achieving **College_Score** at least 65 or not. This is an encouraging message to students who did not do well in **HighSchool_PR**, because they still have a chance in **College_Score** to get admitted to a good school.

In our data analysis, false positive rate and false negative rate have about equal importance. But in certain situations, one can be much more important than the other. For instance, false positive rate is an essential measure in the effectiveness of prompting users to re-enter login information to verify identity for social media. The goal of re-entering credentials is to prevent unauthorized logins, but when people get prompted too many times while using their own account, they would get frustrated and leave the website. This leads to significant revenue loss in business.[22] On the contrary, we are more concerned about the false negative rate in medical testing for a rare disease. The goal of medical testing is to identify as many people with the disease as possible, so that these people can receive timely medical treatment. Hence we are more concerned when the test fails to detect a person who actually has the disease.[23]

8.2.3 Sensitivity and Specificity

We are going off on a tangent here, because this section is not directly related to the data of high school and college entrance exam scores. But we think it is importance to introduce the concepts of **sensitivity** and **specificity** to the readers, since they are also used to describe the overall testing results, especially in a clinical setting.[24]

Sensitivity means that when a patient actually has the disease (actual positive), the medical test is able to sense it and produce a positive result. That is, the medical test is sensitive enough to identify patients who have the disease.

$$\text{Sensitivity} = \frac{\text{True Positive}}{\text{Actual Positive}} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}.$$

Sensitivity is equivalent to the true positive rate (a.k.a. recall), or the opposite of the false negative rate.

$$\text{Sensitivity} = \text{Recall} = 1 - \text{False Negative Rate}.$$

On the other hand, specificity means when a patient does not have the disease (actual negative), the medical test is able to produce a negative result. That is, the medical test is specific enough that it filters out patients who do not have the disease.

$$\text{Specificity} = \frac{\text{True Negative}}{\text{Actual Negative}} = \frac{\text{True Negative}}{\text{True Negative} + \text{False Positive}}.$$

Specificity is equivalent to the true negative rate, or the opposite of the false positive rate.

$$\text{Specificity} = 1 - \text{False Positive Rate}.$$

Let's see an example in medical testing.[25] Assume 0.1% of the population have a specific disease. In a population of 500,000 people, 500 people would have the disease. Now we have a medical test that claims to be 99% accurate, which means the test has 99% sensitivity and 99% specificity. Hence the false positive rate and the false negative rate are both 1%.

- For the 500 people who actually have the disease, 495 tested positive and 5 tested negative.
- For the 499,500 people who do not have the disease, 4,995 people tested positive and the remaining 494,505 people tested negative.

[22] https://fcase.io/a-major-challenge-false-positives/
[23] https://brownmath.com/stat/falsepos.htm
[24] https://www.healthnewsreview.org/toolkit/tips-for-understanding-studies/understanding-medical-tests-sensitivity-specificity-and-positive-predictive-value/
[25] https://math.hmc.edu/funfacts/medical-tests-and-bayes-theorem/

Given that a patient tested positive, how likely does he/she actually have the disease?

Using the Bayes theorem, we get

$$
\begin{aligned}
P(\text{Actual Positive}|\text{Test Positive}) &= \frac{P(\text{Actual Positive} \cap \text{Test Positive})}{P(\text{Test Positive})} \\
&= \frac{P(\text{Actual Positive} \cap \text{Test Positive})}{P(\text{Actual Positive} \cap \text{Test Positive}) + P(\text{Actual Negative} \cap \text{Test Positive})} \\
&= \frac{495}{495 + 4995} \approx 9.16\%.
\end{aligned}
$$

The patient has a 9.16% of chance of actually having the disease, despite the positive test outcome. The patient does not have the disease for more than 90% of the time. Since the disease infects only 0.1% of the population, the medical test creates many false positives, i.e., false alarms. Nevertheless, the test is still informative because given a positive test result, the probability of the patient having the disease increases by 91.6 times.

$$
\frac{P(\text{Actual Positive}|\text{Test Positive})}{P(\text{Actual Positive})} = \frac{9.16\%}{0.1\%} = 91.6.
$$

The statistical calculation tells us that we do not have to be overly concerned about a positive medical test outcome, because the chances are still low for the person to have the disease. However, upon learning the 99% sensitivity and 99% specificity of the test, many doctors seem to associate a positive test with a high probability of having the disease.[26] If any of the readers become a medical doctor in the future, please remember the lesson here and make better treatment decisions for the patients.

8.3 Breakdown by High School Entrance Exam Scores

Let's examine the confusion matrices for each group of **HighSchool_PR**: 0-79, 80-89, 90-94, 95-99. We would like to see if the model performance varies across these groups. Readers can refer to Section 6.2 for the details of this categorization. Before going into the analysis, we need to ensure that each group has a sufficiently large number of respondents within the 188 total records.

The group with **HighSchool_PR** 0-79 has the smallest number of respondents, and 25 is a sufficient sample size. The groups with **HighSchool_PR** 80-89 and 90-94 contain 49 and 44 respondents, respectively. The group with **HighSchool_PR** 95-99 includes 70 respondents, which is the largest of the four categories.

```
HS0to79_ind = which(data_corr$HighSchool_PR >=0 & data_corr$HighSchool_PR <= 79)
HS80to89_ind = which(data_corr$HighSchool_PR >= 80 & data_corr$HighSchool_PR <= 89)
HS90to94_ind = which(data_corr$HighSchool_PR >= 90 & data_corr$HighSchool_PR <= 94)
HS95to99_ind = which(data_corr$HighSchool_PR >= 95 & data_corr$HighSchool_PR <= 99)

print(paste("HighSchool_PR 0-79:",length(HS0to79_ind), "respondents"))
```

```
## [1] "HighSchool_PR 0-79: 25 respondents"
```

```
print(paste("HighSchool_PR 80-89:",length(HS80to89_ind), "respondents"))
```

```
## [1] "HighSchool_PR 80-89: 49 respondents"
```

```
print(paste("HighSchool_PR 90-94:",length(HS90to94_ind), "respondents"))
```

```
## [1] "HighSchool_PR 90-94: 44 respondents"
```

[26] https://www.washingtonpost.com/news/posteverything/wp/2018/10/05/feature/doctors-are-surprisingly-bad-at-reading-lab-results-its-putting-us-all-at-risk/

```r
print(paste("HighSchool_PR 95-99:",length(HS95to99_ind), "respondents"))
```

```
## [1] "HighSchool_PR 95-99: 70 respondents"
```

Now we can produce a confusion matrix for each of the four groups. Except for **HighSchool_PR** 90-94, all the other three groups contain only one value in the predicted outcomes. The higher **HighSchool_PR** the student had, the higher probability the model would predict him/her to achieve **College_Score** at least 65. This is not completely unexpected, but we would like to emphasize that the model is imperfect. There are still some students with a low **HighSchool_PR** but an impressively good **College_Score**.

- **HighSchool_PR** 0-79 has only FALSE predicted outcomes in **College_Score** at least 65.
- **HighSchool_PR** 80-89 has only FALSE predicted outcomes in **College_Score** at least 65.
- **HighSchool_PR** 90-94 has TRUE and FALSE predicted outcomes in **College_Score** at least 65.
- **HighSchool_PR** 95-99 has only TRUE predicted outcomes in **College_Score** at least 65.

When the predicted outcomes do not include both TRUE and FALSE, the confusion matrix produced in `R` would be 2x1 instead of the full 2x2. Section 8.3.1 shows the incorrect 2x1 confusion matrices, and we will add the missing second column back in Section 8.3.2. Finally, we will interpret these results in Section 8.3.3.

8.3.1 Confusion Matrices (Incorrect)

Let's write a function to summarize the actual outcomes and the predicted outcomes into a confusion matrix, using the default settings in `table`. As the readers can see, `table` outputs only the nonzero columns, and that's why we have several 2x1 confusion matrices here.

```r
# Data
actual_65up = data_corr$CS_65up
# Predicted results
predicted_65up = model$fitted.values >= 0.5

confusion_original <- function(HS_inds, actual, predicted) {
  actual = actual[HS_inds]
  predicted = predicted[HS_inds]
  confusion = table(actual, predicted)
  return(confusion)
}
```

Confusion matrix for **HighSchool_PR** 0-79

```r
confusion_0to79 = confusion_original(HS0to79_ind, actual_65up, predicted_65up)
confusion_0to79
```

```
##        predicted
## actual  FALSE
##   FALSE    21
##   TRUE      4
```

Confusion matrix for **HighSchool_PR** 80-89

```r
confusion_80to89 = confusion_original(HS80to89_ind, actual_65up, predicted_65up)
confusion_80to89
```

```
##        predicted
## actual  FALSE
##   FALSE    35
##   TRUE     14
```

Confusion matrix for **HighSchool_PR** 90-94

```r
confusion_90to94 = confusion_original(HS90to94_ind, actual_65up, predicted_65up)
confusion_90to94
```

```
##        predicted
## actual  FALSE TRUE
##   FALSE     5   22
##   TRUE      2   15
```

Confusion matrix for **HighSchool_PR** 95-99

```r
confusion_95to99 = confusion_original(HS95to99_ind, actual_65up, predicted_65up)
confusion_95to99
```

```
##        predicted
## actual  TRUE
##   FALSE   13
##   TRUE    57
```

8.3.2 Confusion Matrices (Correct)

When the confusion matrix has nonzero values in all four cells, we can output the 2x2 confusion matrix. But before doing so, we need to revert the order of FALSE and TRUE in the rows and columns, since `R` sorts names in alphabetical order by default. This can be done by producing the second index (TRUE) before the first index (FALSE).

When all predicted values are FALSE, the confusion matrix is missing a TRUE column and we need to add it back. Similarly, When all predicted values are TRUE, the confusion matrix is missing a FALSE column and we also need to add it back. Finally, we revert the order of FALSE and TRUE in the rows and columns, as in the case of an 2x2 confusion matrix.

By checking the dimensions of the confusing matrices and modifying if necessary, we obtain all four 2x2 confusion matrices for the four categories of **HighSchool_PR**.

```r
confusion_subset <- function(HS_inds, actual, predicted) {
  actual = actual[HS_inds]
  predicted = predicted[HS_inds]
  confusion = table(actual, predicted)

  # When the confusion matrix has nonzero values in all four cells
  if ((dim(confusion)[1] == 2) && (dim(confusion)[2] == 2)) {
    # Revert the order of FALSE and TRUE
    confusion = confusion[2:1, 2:1]
    # Exit the function because the operation is complete
    return(confusion)
  }

  # When all predicted values are FALSE
  else if (colnames(confusion) == c("FALSE")) {
    confusion = as.table(cbind(confusion,c(0,0)))
    colnames(confusion) = c("FALSE","TRUE")
    names(dimnames(confusion)) = c("actual","predicted")
  }

  # When all predicted values are TRUE
  else if (colnames(confusion) == c("TRUE")) {
    confusion = as.table(cbind(c(0,0), confusion))
    colnames(confusion) = c("FALSE","TRUE")
```

```
    names(dimnames(confusion)) = c("actual","predicted")
  }

  # Revert the order of FALSE and TRUE
  confusion = confusion[2:1, 2:1]
  return(confusion)
}
```

Confusion matrix for **HighSchool_PR** 0-79

```
confusion_0to79 = confusion_subset(HS0to79_ind, actual_65up, predicted_65up)
confusion_0to79
```

```
##        predicted
## actual  TRUE FALSE
##   TRUE     0     4
##   FALSE    0    21
```

Confusion matrix for **HighSchool_PR** 80-89

```
confusion_80to89 = confusion_subset(HS80to89_ind, actual_65up, predicted_65up)
confusion_80to89
```

```
##        predicted
## actual  TRUE FALSE
##   TRUE     0    14
##   FALSE    0    35
```

Confusion matrix for **HighSchool_PR** 90-94

```
confusion_90to94 = confusion_subset(HS90to94_ind, actual_65up, predicted_65up)
confusion_90to94
```

```
##        predicted
## actual  TRUE FALSE
##   TRUE    15     2
##   FALSE   22     5
```

Confusion matrix for **HighSchool_PR** 95-99

```
confusion_95to99 = confusion_subset(HS95to99_ind, actual_65up, predicted_65up)
confusion_95to99
```

```
##        predicted
## actual  TRUE FALSE
##   TRUE    57     0
##   FALSE   13     0
```

8.3.3 Interpretations

Table 1 shows the model results for each **HighSchool_PR** category (0-79, 80-89, 90-94, 95-99) – in terms of accuracy, precision, recall, FPR, and FNR. Any metric that cannot be calculated is marked as N/A. The results are quite extreme because the model predicted negative for all **HighSchool_PR** 0-79 and 80-89, and it predicted positive for all **HighSchool_PR** 95-99. Here, positive means achieving **College_Score** 65 or higher, and negative means not achieving this.

The overall accuracy is slightly above 70%, while the accuracy for **HighSchool_PR** 90-94 is below 50%. Nevertheless, the group of **HighSchool_PR** 90-94 is the only category with nonzero values in all four cells

of the confusion matrix. All the other three **HighSchool_PR** categories have good accuracy, but their FPR and FNR are either 0% or 100% because all predictions within each group have the same value.

HighSchool_PR	Accuracy	Precision	Recall	FPR	FNR
0-79	84.00%	N/A	0%	0%	100%
80-89	71.43%	N/A	0%	0%	100%
90-94	45.45%	40.54%	88.23%	81.48%	11.76%
95-99	81.43%	81.43%	100%	100%	0%
Overall	70.74%	67.29%	78.26%	36.46%	21.74%

Table 1: Model results for each **HighSchool_PR** category

For **HighSchool_PR** 0-79 and 80-89, the precision is not available (N/A) due to division-by-zero. The model made zero positive predictions, i.e., it predicted all students in the two categories not to achieve **College_Score** 65 or higher. Note that these students had a non-zero probability to obtain this achievement, but their predicted probability is less than the threshold of 50%, and that's why the datapoints are predicted as negative. The remaining metrics of **HighSchool_PR** 0-79 and 80-89 are straightforward, because they are either 0% or 100%. The recall is 0% because none of the students in these categories received a positive prediction. The FPR is also 0% because all students who did not achieve **College_Score** 65 or higher were not predicted to do so anyway. The FNR is 100% because all students who actually obtained **College_Score** 65 or higher were not predicted to make it, so these datapoints are regarded as unexpected successes. Finally, the accuracy here equals to the percentage of students who received **College_Score** below 65, and it is reasonable to see a lower accuracy (non-success rate) for **HighSchool_PR** 80-89 than 70-79.

HighSchool_PR 90-94 is the most interesting group, because it is the only category where the model predicted both TRUE and FALSE values. Although the accuracy is less than 50%, the recall is impressively high (over 80%). The model predicted 37 out of 44 students in this category to achieve **College_Score** 65 or higher, but 22 of the 37 students did not make it. This resulted in a high FPR and a low FNR. Section 6.3 shows that a **College_Score** of 65 is around the 95th percentile of all college entrance exam takers in Taiwan each year. Therefore, it is reasonable to predict that most students with **HighSchool_PR** 90-94 are going to achieve **College_Score** at least 65. Recall that not all high school students are willing and able to attending college. However, students with **HighSchool_PR** 90-94 do not have the same level of access to academic resources. **HighSchool_PR** of this range can mean attending a top high school in the rural areas.[27] But in Taipei, this **HighSchool_PR** is nowhere sufficient to get admitted into any prestigious high schools (a.k.a. "the top three") within the city.[28] Since we do not have the geographic information of the respondents, we cannot make any inferences on which ones with **HighSchool_PR** 90-94 are more likely to achieve **College_Score** at least 65.

The **HighSchool_PR** 95-99 group contains respondents with top 5% scores in the high school entrance exam, and that's why the model predicted all of them to achieve **College_Score** at least 65. The recall is 100% due to all datapoints being predicted as TRUE. The FPR is also 100% because all respondents in this category who did not achieve **College_Score** at least 65 were predicted to do so. The FNR is 0% because none of the respondents were predicted not to achieve this. The accuracy is over 80% because most respondents with **HighSchool_PR** 95-99 actually obtained **College_Score** at least 65. The 4x4 table in Section 6.4 shows that 23 of them (32.86%) got **College_Score** between 65 and 69, while 34 of them (48.57%) got **College_Score** between 70 and 75. One extension would be increasing the **College_Score** cutoff point to 70 for this group, but we do not think this is practical without external data about these respondents.

In our opinion, **HighSchool_PR** 95-99 does little in distinguishing students' academic capabilities within the group, because it is possible to drop one percentile by getting just one critical question wrong on the high school entrance exam. Prior to 2009, there was a huge score difference between full marks and missing by just one question.[29] This could result in a significant difference of **HighSchool_PR**, because missing five

[27] https://bit.ly/3qxuqMw
[28] https://cclccl-life.blogspot.com/2013/06/blog-post_9.html
[29] https://news.ltn.com.tw/news/life/paper/217325

questions in a single subject (full marks on the other four) would have a better score than missing one question in each of the five subjects. Hence it is not meaningful to evaluate students' academic performance solely from **HighSchool_PR**. Similar to the case in **HighSchool_PR** 90-94, we cannot make inferences about the within-group differences of **HighSchool_PR** 95-99. In rural areas, students with **HighSchool_PR** 95-99 already meet the admission requirements for the top high school in their region. Almost all of them would attend that school unless they move to a different region. As a comparison, Taipei's first choice high school requires **HighSchool_PR** at least 99, second choice requires at least 98, and third choice requires at least 97.[30] As a result, students in Taipei with **HighSchool_PR** 95 or 96 often end up attending local high schools outside the top tier. These schools are still high-quality and provide ample resources, but they are not as prestigious as the top three choices.

9 Model Validation: Out-of-Sample Prediction

The goal of building this model is to optimize the performance for future input, i.e., incoming students who just obtained their **HighSchool_PR** scores. We need to use the model to predict new data, and this validation method is called out-of-sample prediction. That is, the model has to be able to predict data outside the training sample.

In-sample prediction (Chapter 8) is insufficient because we need to test on unseen data to **avoid overfitting**.[31] But why is overfitting bad? Because the model would do well on the existing data but perform poorly on the new data, which is undesirable. This is similar to a student who memorizes the answers to score 100% on quizzes without understanding the actual content. Then this student may not do well on the final exam because he/she has not seen the questions before. In order to measure the student's grasp of the knowledge, the instructor usually gives exam questions similar to the practice questions, but not exactly the same.

Some readers may be wondering how to get "new" data to perform out-of-sample prediction, and the good news is that we already have them. New data means **previously unseen** data by the model; in other words, the data was not involved in training the model. Although training the model requires data, we do not have to feed in all 188 records at once. We can use a part of the records to train the model, and leverage the remaining data to test the model for performance evaluation. In this way, the latter part of the data are considered "new" because they are not seen by the model beforehand. The data involved in the training phase is called the training dataset, and the data used for testing is called the testing dataset.

In this chapter, we demonstrate two methods to implement out-of-sample prediction. The first method is using **separate training and testing datasets** to validate the model, where the two datasets are mutually exclusive. We train the model on the training set, and test the model on the testing set. The second method is **cross validation**, which involves partitioning the data into a number of subsets, then we reserve one subset for testing and train the model on all the remaining subsets. Each subset take turns to be used for testing, and finally we combine the results to estimate the overall prediction performance.

9.1 Separate Training and Testing Datasets

In this section, **randomly divide the data into a training set and a testing set**. Then we use the training set to train the model for the parameters, and use the testing set to evaluate the model performance. In other words, the model is trained on some data independent of the testing set, because it would not see the testing data beforehand. Using unseen data is helpful to get a better measure of the prediction power of the model. An extension is to divide the data into **training, validation, and testing** sets. We still use the training set to train the model, but we use the validation set to fine-tune the model parameters.[32] Finally we test the model using the testing set. By incorporating a validation set as a mid-step, we do not look at the model results for the testing set until the end. This further reduces the risk of overfitting the testing data. But since our logistic regression model does not involve fine-tuning, we use only the training and testing sets for simplicity.

[30] https://chendaneyl.pixnet.net/blog/post/31436728
[31] https://elitedatascience.com/overfitting-in-machine-learning
[32] https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

9.1.1 Implementation

Below is the code to divide the data into the training and testing partitions. We randomly selected 50% of the data (94 out of the 188 records) to be in the training part, and the remaining 50% are in the testing part. This can be done by a random permutation of the indices 1-194. Then the first half of the indices correspond to the training records, and the second half of the indices correspond to the testing records. We set a random seed to ensure reproducibility.

```
set.seed(10)

nn = nrow(data_corr) # total 188 rows of data

row_inds = c(1:nn)

ind_permute = sample(row_inds)

train_inds = ind_permute[1:94]
test_inds = ind_permute[95:188]
```

The `train_inds` are the indices for the training part of the data:

```
print(train_inds)
```

```
##  [1] 137  74 112 183  72 182 167  88  15 143 170 187 162  24  13  95 136 110   7
## [20] 155  86  82  29 166 121  92  50 109 154 101 122  33 135 160  68  93 114 181
## [39]  51  32  11  79 163  91  42  78 174 105 117  26  89  48 180 171 175  61 132
## [58]  14  35  10 177  58  39  16 172  31 129 159 150  53  63 164 161  47 126 120
## [77] 138  18   3 168 144 158  64  59 147  77  90 179 146  34 106   4 118  20
```

The `test_inds` are the indices for the testing part of the data:

```
print(test_inds)
```

```
##  [1]  96 153  87 188 173  54  57  27   9  80 145 116 104 108  73 184  25 130 141
## [20]  46 113  22 142   6  40  71 119 149 123 134 169 176  45  23  17  43 140   1
## [39]  97  30 165 131 115  65  62  38 102 152  55   2 186  28  76  12  52  81  75
## [58] 100 156  37  98  49 103 107  67   8  69   5  83 111 124 125  60  99 157  70
## [77] 185  36  41  21  85 133  44 127  19  94  84 148 151 178 139  56  66 128
```

We can sort each set of the indices in ascending order, so it will be easier to refer to them later, i.e., better readability. When we obtain the testing results, the records would be in the same order as in the original dataset.

```
train_inds = sort(train_inds)
test_inds = sort(test_inds)
```

After sorting, the 94 training indices are in ascending order.

```
print(train_inds)
```

```
##  [1]   3   4   7  10  11  13  14  15  16  18  20  24  26  29  31  32  33  34  35
## [20]  39  42  47  48  50  51  53  58  59  61  63  64  68  72  74  77  78  79  82
## [39]  86  88  89  90  91  92  93  95 101 105 106 109 110 112 114 117 118 120 121
## [58] 122 126 129 132 135 136 137 138 143 144 146 147 150 154 155 158 159 160 161
## [77] 162 163 164 166 167 168 170 171 172 174 175 177 179 180 181 182 183 187
```

The 94 testing indices are also sorted in ascending order.

```
print(test_inds)
```

```
##  [1]   1   2   5   6   8   9  12  17  19  21  22  23  25  27  28  30  36  37  38
## [20]  40  41  43  44  45  46  49  52  54  55  56  57  60  62  65  66  67  69  70
## [39]  71  73  75  76  80  81  83  84  85  87  94  96  97  98  99 100 102 103 104
## [58] 107 108 111 113 115 116 119 123 124 125 127 128 130 131 133 134 139 140 141
## [77] 142 145 148 149 151 152 153 156 157 165 169 173 176 178 184 185 186 188
```

Now we slice the data into the training and testing parts using the two sets of indices.

```
train_data = data_corr[train_inds,]
test_data = data_corr[test_inds,]
```

Then we train the logistic regression model using the 188 records in the training part, and the model summary shows the coefficient point estimates along with the standard error.

```
train_model = glm(CS_65up ~ HighSchool_PR, data=train_data, family="binomial")
summary(train_model)
```

```
##
## Call:
## glm(formula = CS_65up ~ HighSchool_PR, family = "binomial", data = train_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6541  -0.9403  -0.1133   0.8232   2.6852
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -15.2067     3.7687  -4.035 5.46e-05 ***
## HighSchool_PR   0.1661     0.0408   4.072 4.66e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 130.27  on 93  degrees of freedom
## Residual deviance: 104.80  on 92  degrees of freedom
## AIC: 108.8
##
## Number of Fisher Scoring iterations: 5
```

Next, we use the trained model to predict the testing part of the data. The function `predict.glm` allows us to fit the generalized linear model (GLM) on new data. The type 'response' gives the predicted probabilities. The output is a numeric vector with the predicted probabilities, and the header is the record index from the original data. For example, the 1st record in the original data is included in the testing part, and the model predicts the respondent to have a 0.5% probability of obtaining **College_Score** 65 or higher.

```
test_prob = predict.glm(train_model, test_data, type="response")
round(test_prob, digits=3)
```

```
##     1     2     5     7     9    10    13    18    21    23    24    25    27
## 0.005 0.005 0.320 0.023 0.640 0.357 0.640 0.745 0.052 0.222 0.561 0.745 0.677
##    29    30    32    38    39    40    42    43    45    46    47    48    51
## 0.222 0.519 0.640 0.396 0.776 0.640 0.222 0.776 0.561 0.601 0.713 0.776 0.677
##    54    56    57    58    59    62    64    67    68    69    72    73    74
## 0.437 0.396 0.519 0.170 0.677 0.070 0.032 0.561 0.148 0.776 0.111 0.060 0.095
##    76    78    79    83    84    87    89    90    92   100   102   103   104
## 0.745 0.745 0.148 0.713 0.601 0.519 0.111 0.478 0.745 0.128 0.478 0.561 0.601
```

```
##   105   106   108   109   110   113   114   117   119   121   122   125   129
## 0.776 0.320 0.396 0.776 0.601 0.111 0.561 0.437 0.252 0.776 0.320 0.713 0.320
##   130   131   134   135   137   138   140   141   146   147   148   149   152
## 0.601 0.252 0.776 0.357 0.252 0.776 0.561 0.745 0.745 0.128 0.478 0.437 0.776
##   155   156   158   159   160   163   164   172   176   180   184   186   192
## 0.713 0.776 0.745 0.437 0.713 0.195 0.285 0.037 0.357 0.745 0.001 0.396 0.601
##   193   194   197
## 0.745 0.052 0.027
```

Then we follow the procedures in Section 8.1 to convert the test probabilities into binary classification results, i.e., the confusion matrix.

```r
# Convert the test probabilities into binary classification results
test_actual_65up = test_data$CS_65up
test_pred_65up = test_prob > 0.5

# Confusion matrix
test_confusion = table(test_actual_65up, test_pred_65up)
# revert the order of FALSE and TRUE
test_confusion = test_confusion[2:1, 2:1]

test_confusion
```

```
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    34    12
##            FALSE   14    34
```

We can also produce the percentage version of the confusion matrix.

```r
prop.table(test_confusion)
```

```
##                 test_pred_65up
## test_actual_65up      TRUE     FALSE
##            TRUE  0.3617021 0.1276596
##            FALSE 0.1489362 0.3617021
```

Now we show the number of true positives, false negatives, false positives, and false negatives.

```r
# row = actual_65up, column = predicted_65up
tp = test_confusion["TRUE","TRUE"]
fn = test_confusion["TRUE","FALSE"]
fp = test_confusion["FALSE","TRUE"]
tn = test_confusion["FALSE","FALSE"]

print(paste(tp,fn,fp,tn))
```

```
## [1] "34 12 14 34"
```

We can also calculate the evaluation metrics for the predictive model. The process is similar to Section 8.2.

$$
\begin{aligned}
\text{Accuracy} &= \frac{TP+TN}{TP+FN+FP+TN} = \frac{34+34}{34+12+14+34} = \frac{68}{94} \approx 72.34\% \\
\text{Precision} &= \frac{TP}{TP+FP} = \frac{34}{34+14} = \frac{34}{48} \approx 70.83\% \\
\text{Recall} &= \frac{TP}{TP+FN} = \frac{34}{34+12} = \frac{34}{46} \approx 73.91\% \\
\text{False Positive Rate (FPR)} &= \frac{FP}{TN+FP} = \frac{14}{34+14} = \frac{14}{48} \approx 29.17\% \\
\text{False Negative Rate (FNR)} &= \frac{FN}{TP+FN} = \frac{12}{34+12} = \frac{12}{46} \approx 26.09\%
\end{aligned}
$$

We also compare the results with the in-sample prediction, and they show similar trends. The accuracy, precision, and recall all hover around 70%.

	Accuracy	Precision	Recall	FPR	FNR
In-Sample Prediction	70.74%	67.29%	78.26%	36.46%	21.74%
Separate Training and Testing	72.34%	70.83%	73.91%	29.17%	26.09%

Table 2: Comparison of results with in-sample prediction

9.1.2 Organizing the Code for Reusability

We have demonstrated how to train the logistic regression model on a part of the data, and test the model on the remaining data. But there is one problem – the code is messy and hence not reusable. If we are going to build another training-testing framework, we may have to copy-paste lots of code from Section 9.1.1, which is undesirable. We should avoid excessive copy-pasting because this is prone to mistakes, making the code more difficult to debug.

A better solution is to incorporate repetitive code into a function, so that we can keep the same work in one place. In software development, "don't repeat yourself" (DRY) is a principle to reduce code repetitions (Foote, 2014). When we change this part of the program, we only need to edit the code within the function. The modifications would automatically be performed anytime the function is called. In this way, the code can be easily reused and maintained.

As a first example, we need to convert the predictive probabilities into the binary classification results, and show them in a confusion matrix. This compound task is performed in almost every model validation involving binary classifications, so we should encapsulate the task into a function. This function compares the test probabilities with their ground truth (0/1), and outputs the number of true positives, false negatives, false positives, and true negatives. We predict a datapoint to be positive if the estimated probability is at or above a given threshold, which is set to 0.5 by default. Otherwise, we predict the datapoint to be negative.

```
prob_to_matrix <- function(test_data, test_prob, threshold=0.5) {
  # Convert the test probabilities into binary classification results.
  # Threshold should be between 0 and 1, set to 0.5 by default.

  test_actual_65up = test_data$CS_65up
  test_pred_65up = test_prob >= threshold

  # Confusion matrix
  test_confusion = table(test_actual_65up, test_pred_65up)
  # revert the order of FALSE and TRUE
  test_confusion = test_confusion[2:1, 2:1]
```

```
  return(test_confusion)
}
```

We can call the `prob_to_matrix` function to obtain the confusion matrix, and the output is the same.

```
another_test = prob_to_matrix(test_data, test_prob)
another_test
```

```
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    34    12
##            FALSE   14    34
```

The results in Table 2 are for separate training and testing sets from a single random seed. We would like to try more versions of such out-of-sample prediction, so we created the function `train_and_test` to automate the procedure. Note that this procedure calls `prob_to_matrix` at the end. We wrote the latter as a single function because we may also use it in other types of model validation. Eventually, we can run this function multiple times and take the average of the accuracy/precision/recall/etc.

```
train_and_test <- function(data, seed) {
  # Automate the procedure of using training and testing datasets
  # for out-of-sample model validation.

  # Input: data_corr, random_seed
  # Output: confusion_matrix

  set.seed(seed)
  nn = nrow(data)
  row_inds = c(1:nn)
  ind_permute = sample(row_inds)
  mid_pt = floor(nn/2) # round down

  # Randomly split the data into 50% training and 50% testing
  train_inds = ind_permute[1:mid_pt]
  test_inds = ind_permute[(mid_pt+1):nn]
  train_inds = sort(train_inds)
  test_inds = sort(test_inds)

  train_data = data[train_inds,]
  test_data = data[test_inds,]

  train_model = glm(CS_65up ~ HighSchool_PR, data=train_data, family="binomial")
  # summary(train_model)

  test_prob = predict.glm(train_model, test_data, type="response")
  # round(test_prob, digits=3)

  test_confusion = prob_to_matrix(test_data, test_prob)

  return(test_confusion)
}
```

With this function, we can reproduce the predictive outcomes using the same random seed.

```
train_and_test(data_corr, seed=10)
```

```
##                 test_pred_65up
```

```
## test_actual_65up TRUE FALSE
##            TRUE    34    12
##            FALSE   14    34
```

We can try a different random seed, and obtain results with a different split of training/testing data.

```r
train_and_test(data_corr, seed=123)
```

```
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    38     9
##            FALSE    8    39
```

Let's try five iterations with different random seeds and output the results. We will get five confusion matrices, and we can summarize the numbers across them. The main reason to try multiple iterations is to avoid getting an unlucky draw, i.e., a single partition that leads to extreme outcomes.

In the code, we generate the sequence of random seeds from a sequence of random numbers between 1 and 1000 without replacement. In this way, we only need to set a single random seed to get all five runs (which we can increase in the future).

```r
set.seed(37)
runs = 5

# Discrete uniform distribution:
# Generate a sequence of random numbers between 1 and 1000
# (sample without replacement)
seed_each = sample(1:1000, runs, replace=F)

# Initialize the list with size = number of runs.
# Don't start with an empty list and append elements later,
# because the append function may not work for matrix elements.

out_matrices = rep(list("results"), runs)

for (iter in 1:runs) {
  output = train_and_test(data_corr, seed=seed_each[iter])
  out_matrices[[iter]] = output
}

out_matrices
```

```
## [[1]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    36    11
##            FALSE   15    32
##
## [[2]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    31    14
##            FALSE   12    37
##
## [[3]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
```

```
##            TRUE    36    11
##            FALSE   23    24
##
## [[4]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    38     9
##            FALSE   14    33
##
## [[5]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    40    10
##            FALSE   16    28
```

For each confusion matrix, we need to calculate the accuracy, precision, recall, FPR, and FNR. We calculated these by hand earlier, and now it is time to do them in `R` code for reproducibility. We write a function to do the calculations, because these would be reused in other model validation schemes. We also use the first confusion matrix to demonstrate how this function works.

```r
confusion_to_measures <- function(output) {
  tp = output[1,1]
  fn = output[1,2]
  fp = output[2,1]
  tn = output[2,2]

  accuracy = (tp+tn)/(tp+fn+fp+tn)
  precision = tp/(tp+fp)
  recall = tp/(tp+fn)
  fpr = fp/(tn+fp)
  fnr = fn/(tp+fn)

  measures = c(accuracy, precision, recall, fpr, fnr)
  names(measures) = c("Accuracy","Precision","Recall","FPR","FNR")

  return(measures)
}

sample_output = confusion_to_measures(out_matrices[[1]])
round(sample_output, digits = 4)
```

```
##  Accuracy Precision    Recall       FPR       FNR
##    0.7234    0.7059    0.7660    0.3191    0.2340
```

Then we output the metrics of each iteration to a table, using a for loop to run through the iterations. We converted this process into the function `combine_results`, because we need to use the same script for k-fold cross validation as well.

```r
combine_results <- function(out_matrices) {
  # Combine the output results
  # Input: out_matrices (list of matrices)
  # Output: out_measures (matrix array)

  runs = length(out_matrices)

  out_measures = c(0,0,0,0,0,0)
```

```r
  names(out_measures) = c("Iteration","Accuracy","Precision","Recall","FPR","FNR")

  for (iter in 1:runs) {
    output = confusion_to_measures(out_matrices[[iter]])
    measures = c(iter, output)
    out_measures = rbind(out_measures, measures)
  }

  row.names(out_measures) = rep(c(""), runs+1) # remove row names
  out_measures = out_measures[-1,] # remove the first placeholder row

  return(out_measures)
}

out_measures = combine_results(out_matrices)

# out_measures
round(out_measures, digits=4)
```

```
##  Iteration Accuracy Precision Recall    FPR    FNR
##          1   0.7234    0.7059 0.7660 0.3191 0.2340
##          2   0.7234    0.7209 0.6889 0.2449 0.3111
##          3   0.6383    0.6102 0.7660 0.4894 0.2340
##          4   0.7553    0.7308 0.8085 0.2979 0.1915
##          5   0.7234    0.7143 0.8000 0.3636 0.2000
```

We also calculate the average of the five iterations for each metric. The accuracy, precision, and recall all hover around 70% as expected, just like in Table 2 in Section 9.1.1.

```r
calc_average <- function(out_measures) {
  avg_results = c(mean(out_measures[,"Accuracy"]),
            mean(out_measures[,"Precision"]),
            mean(out_measures[,"Recall"]),
            mean(out_measures[,"FPR"]), mean(out_measures[,"FNR"]))

  names(avg_results) = c("Accuracy","Precision","Recall","FPR","FNR")

  return(avg_results)
}

average = calc_average(out_measures)

# average
round(average, digits=4)
```

```
##  Accuracy Precision    Recall       FPR       FNR
##    0.7128    0.6964    0.7659    0.3430    0.2341
```

We are going to have fewer descriptions in the result evaluation of later sections, because we assume that at this point, the readers would already be familiar with the relevant concepts.

9.2 Cross Validation

Next, we are going to talk about **cross validation**. The "cross" means that each record in the data has the opportunity to serve as the training set AND the testing set (obviously, not at the same time). Cross validation involves partitioning data into a number of subsets, then we reserve one subset for testing and train

the model on all the remaining subsets. Each subset take turns to be used for testing, and finally we combine the results to estimate the overall prediction performance. Two common cross validation methods are **k-fold cross validation** and **leave-one-out cross validation**. We will demonstrate both ways of cross validation in this section.

9.2.1 K-fold Cross Validation

In **k-fold cross validation**, we randomly divide the data into k subsets to cross-validate each other. (Typically $k = 10$.) For each round of validation, we train the model on the $k - 1$ subsets and test the model on the one subset which was excluded in the training. Finally, we combine all k rounds of validation, and each subset gets its predicted result for performance evaluation.

To implement 10-fold cross validation, we divide the data into 10 partitions of near-equal sizes. We have 188 records, so there should be eight partitions of 19 records and two partitions of 18 records. We store the indices of each partition in the `partition_list`.

```
# 10-fold cross validation:
# Divide 188 records into 10 partitions of near-equal size

# Number of records in each partition:
# 19, 19, 19, 19, 19, 19, 19, 19, 18, 18
k_fold = c(19, 19, 19, 19, 19, 19, 19, 19, 18, 18)
k_accumulate = c(19, 38, 57, 76, 95, 114, 133, 152, 170, 188)

k_fold_partition <- function(data, k_fold, seed) {
  # Generate the 10 partitions for the data

  set.seed(seed)
  nn = nrow(data) # total 188 rows of data
  row_inds = c(1:nn)
  ind_permute = sample(row_inds)
  # random permutation of row indices
  # => prepare for the training/testing partitions

  partition_list = list(0,0,0,0,0,0,0,0,0,0)

  # Need to sort the indices within each partition
  partition_list[[1]] = sort(ind_permute[1:k_fold[1]])
  for (ii in 2:length(k_fold)) {
    start = k_accumulate[ii-1]+1
    end = start + k_fold[ii] - 1
    partition_list[[ii]] = sort(ind_permute[start:end])
  }

  return(partition_list)
}

partition_list = k_fold_partition(data_corr, k_fold, seed=21)
partition_list
```

```
## [[1]]
##  [1]   3  16  21  47  57  62  63  67  94 106 107 115 123 138 142 157 161 166 170
##
## [[2]]
##  [1]   1   5  33  49  71  72  74  91  92  96  98 105 112 124 131 137 151 159 184
##
```

```
## [[3]]
##  [1]   8  11  15  20  40  45  50  56  65  66  86 117 125 126 135 145 160 162 165
##
## [[4]]
##  [1]   6  22  34  37  53  58  69  89 101 103 104 116 127 132 144 146 172 174 175
##
## [[5]]
##  [1]  28  29  31  35  55  80  81  88 111 113 118 140 152 154 163 173 176 186 187
##
## [[6]]
##  [1]  25  36  44  51  52  59  61  73 110 121 128 129 130 139 150 153 167 177 185
##
## [[7]]
##  [1]   4  10  12  18  19  27  38  43  46  68  75  84  85  87 136 148 156 164 168
##
## [[8]]
##  [1]   7  23  42  64  77  90  93 100 108 109 119 134 143 147 158 178 179 181 183
##
## [[9]]
##  [1]   9  14  26  30  39  41  54  76  78  79  82  95  99 114 120 141 155 180
##
## [[10]]
##  [1]   2  13  17  24  32  48  60  70  83  97 102 122 133 149 169 171 182 188
```

After obtaining the 10 partitions, we reserve one partition as the testing set and feed the other 9 partitions into the training. Each partition gets the chance to be the testing set, and we obtain all 10 confusion matrices.

Other `R` packages may have functions to handle k-fold cross validation, but we decided to write our own code to show the readers how the method is implemented from scratch. Moreover, this allows maximum flexibility for us to modify the code for future needs.

```r
k_fold_train_test <- function(data, partition_list, k_fold) {
  # Training and testing process for k-fold cross validation

  # Use the partitions for training and testing
  partition_probs = list(0,0,0,0,0,0,0,0,0,0)
  partition_matrices = list(0,0,0,0,0,0,0,0,0,0)

  for (exclude in 1:length(k_fold)) {
    # Testing parts
    testing_with_k = partition_list[[exclude]]
    test_kfold_data = data_corr[testing_with_k,]

    # Training parts
    # partition_list[-exclude] shows all elements except the exclude.
    training_without_k = unlist(partition_list[-exclude])
    # integer vector of training indices
    train_kfold_data = data_corr[training_without_k,]

    train_kfold_model = glm(CS_65up ~ HighSchool_PR,
                            data=train_kfold_data, family="binomial")
    # summary(train_kfold_model)
    test_kfold_prob = predict.glm(train_kfold_model,
                                  test_kfold_data, type="response")
    # type="response" gives the predicted probabilities
```

```r
    # Store the predicted probabilities of each partition in a list
    partition_probs[[exclude]] = test_kfold_prob

    # Store the confusion matrix of each partition in another list
    partition_matrices[[exclude]] = prob_to_matrix(test_kfold_data, test_kfold_prob)
  }

  # partition_probs
  return(partition_matrices)
}

partition_matrices = k_fold_train_test(data_corr, partition_list, k_fold)
partition_matrices
```

```
## [[1]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE     7     3
##            FALSE    1     8
##
## [[2]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE     7     1
##            FALSE    4     7
##
## [[3]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE     7     1
##            FALSE    1    10
##
## [[4]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    10     1
##            FALSE    3     5
##
## [[5]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE     4     2
##            FALSE    5     8
##
## [[6]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE     6     5
##            FALSE    4     4
##
## [[7]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
```

```
##            TRUE     8     2
##            FALSE    3     6
##
## [[8]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE     9     2
##            FALSE    5     3
##
## [[9]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE     9     2
##            FALSE    2     5
##
## [[10]]
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE     5     1
##            FALSE    3     9
```

Now we summarize the results in k-fold cross-validation, i.e., combine all 10 confusion matrices into one. We write a function to encapsulate the sum-up process of the confusion matrices, in order to reuse the code later.

```r
sum_up_confusion <- function(k_fold, partition_matrices) {
  # Sum up all k confusion matrices into a single one.
  tp = 0
  fp = 0
  fn = 0
  tn = 0

  for (part in 1:length(k_fold)) {
    tp = tp + partition_matrices[[part]][1]
    fp = fp + partition_matrices[[part]][2]
    fn = fn + partition_matrices[[part]][3]
    tn = tn + partition_matrices[[part]][4]
  }

  # Use an existing confusion matrix as a template.
  k_fold_table = partition_matrices[[1]]

  k_fold_table[1,1] = tp
  k_fold_table[1,2] = fn
  k_fold_table[2,1] = fp
  k_fold_table[2,2] = tn

  return(k_fold_table)
}

k_fold_table = sum_up_confusion(k_fold, partition_matrices)
k_fold_table
```

```
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    72    20
```

```
##            FALSE   31    65
```

After obtaining the 10-fold cross validation results in a single confusion matrix, we can calculate the accuracy, precision, recall, FPR, FNR using the `confusion_to_measures` function from the previous section.

```
k_fold_results = confusion_to_measures(k_fold_table)
round(k_fold_results, digits=4)
```

```
##  Accuracy Precision    Recall       FPR       FNR
##    0.7287    0.6990    0.7826    0.3229    0.2174
```

Since 10-fold cross validation involves randomly partitioning the data into 10 parts of nearly equal size, we can try different random seeds to see how the results change. For each of the five random seeds, the confusion matrices are quite similar.

```
set.seed(37)
runs = 5
# Discrete uniform distribution:
# Generate a sequence of random numbers between 1 and 1000
# (sample without replacement)
seed_each = sample(1:1000, runs, replace=F)

for (iter in 1:runs){
  partition_list = k_fold_partition(data_corr, k_fold,seed=seed_each[iter])
  partition_matrices = k_fold_train_test(data_corr, partition_list, k_fold)
  out_matrices[[iter]] = sum_up_confusion(k_fold, partition_matrices)
  print(out_matrices[[iter]])
}
```

```
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    72    20
##            FALSE   35    61
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    72    20
##            FALSE   34    62
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    72    20
##            FALSE   33    63
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    72    20
##            FALSE   33    63
##                 test_pred_65up
## test_actual_65up TRUE FALSE
##            TRUE    72    20
##            FALSE   32    64
```

We also output the results of the five iterations.

```
out_measures = combine_results(out_matrices)

# out_measures
round(out_measures, digits=4)
```

```
##  Iteration Accuracy Precision Recall    FPR    FNR
```

```
##          1   0.7074    0.6729 0.7826 0.3646 0.2174
##          2   0.7128    0.6792 0.7826 0.3542 0.2174
##          3   0.7181    0.6857 0.7826 0.3438 0.2174
##          4   0.7181    0.6857 0.7826 0.3438 0.2174
##          5   0.7234    0.6923 0.7826 0.3333 0.2174
```

Then we calculate the average of the five iterations, and the mean accuracy is slightly over 70%. Note that the actual numbers can vary depending on the random seed.

```
average = calc_average(out_measures)

# average
round(average, digits=4)
```

```
##  Accuracy Precision    Recall       FPR       FNR
##    0.7160    0.6832    0.7826    0.3479    0.2174
```

9.2.2 Leave-one-out Cross Validation

In **leave-one-out cross validation**, each record is considered an independent subset. This is essentially setting K to be the number of total records in the data, say N. We train the model on the $N-1$ records and test the model on the one left-out record. This allows each record to get its own prediction. The key advantage is that the results are a relatively accurate estimate of the model performance.[33] Note that when N is extremely large, the computational cost would be high because we need to perform N rounds of validation with $N-1$ records each. The complexity is $O(N^2)$.

Here is the code for leave-one-out cross validation. Since each record is predicted by all the other records in the data, no randomness is involved in creating the data split. Hence we do not have to set a random seed in the process.

```
nn = nrow(data_corr) # total 188 rows of data

prob_leave1out = rep(c(-1), nn)

for (ii in 1:nn) {
  data_test = data_corr[ii,] # reserve one record for testing
  data_exclude = data_corr[-ii,]

  train_leave1out = glm(CS_65up ~ HighSchool_PR, data=data_exclude, family="binomial")
  # summary(train_leave1out)

  test_leave1out = predict.glm(train_leave1out, data_test, type="response")
  # type="response" gives the predicted probabilities

  # Store the predicted probability to the general list
  prob_leave1out[ii] = test_leave1out
}
```

Now we summarize the predictive probabilities in a single confusion matrix. The results are exactly the same as the in-sample prediction in Section 8, which may be a coincidence.

```
matrix_leave1out = prob_to_matrix(data_corr, prob_leave1out)
matrix_leave1out
```

```
##                 test_pred_65up
## test_actual_65up TRUE FALSE
```

[33] https://machinelearningmastery.com/loocv-for-evaluating-machine-learning-algorithms/

```
##            TRUE    72    20
##            FALSE   35    61
```

We also calculate the five metrics: accuracy, precision, recall, FPR, and FNR.

```
leave1out_results = confusion_to_measures(matrix_leave1out)
round(leave1out_results, digits=4)
```

```
##  Accuracy Precision    Recall       FPR       FNR
##    0.7074    0.6729    0.7826    0.3646    0.2174
```

9.3 Comparison of Results

Table 3 summarizes the results of in-sample prediction and out-of-sample prediction. Separate training & testing and k-fold cross validation are the average results of five iterations each. The accuracy is slightly above 70%, the precision is around 68%, and the recall is approximately 78%. Since the outcomes are similar across each method, we are not concerned about overfitting in the logistic regression model. Note that the logistic regression is straightforward to run and does not require parameter tuning. In contrast, some machine learning models have a large number of parameters and they are at more risk of overfitting.

	Accuracy	Precision	Recall	FPR	FNR
In-Sample Prediction	70.74%	67.29%	78.26%	36.46%	21.74%
Separate Training & Testing (Average)	71.28%	69.64%	76.59%	34.30%	23.41%
K-Fold Cross Validation (Average)	71.60%	68.32%	78.26%	34.79%	21.74%
Leave-one-out Cross Validation	70.74%	67.29%	78.26%	36.46%	21.74%

Table 3: Comparison of results with in-sample and out-of-sample prediction

This model uses **HighSchool_PR** scores to predict whether a student would get **College_Score** at least 65 or not. But the model is imperfect in prediction, just as we explained in Section 8.2.

- The **precision** is the number of true positives divided by the predicted positives. This means among the students with good **HighSchool_PR** scores, around 68% of them achieved **College_Score** at least 65 three years later.[34]
- The **recall** is the number of true positives divided by the actual positives. This means among the students with **College_Score** at least 65, approximately 78% them had good **HighSchool_PR** scores three years ago.
- The **FPR (false positive rate)** is the number of false positives divided by the actual negatives. This means among the students with **College_Score** 64 or below, about 35% of them were originally predicted to have **College_Score** at least 65.
- The **FNR (false negative rate)** is the number of false negatives divided by the actual positives. This means among the students with **College_Score** at least 65, slightly over 20% of them were "pleasant surprises" because we did not predict them to achieve such scores given their **HighSchool_PR**.

In summary, given the student's **HighSchool_PR**, the model is only about 70% accurate to predict **College_Score** at least 65 or not. In practical terms, if a student obtains a great **HighSchool_PR** score, he/she should keep up with the good work in order to perform well in **College_Score** three years later. On the other hand, if a student does not obtain a great **HighSchool_PR** score for any reason, he/she still has a second chance to do well in the **College_Score**.

Unfinished below

Continue to separate this enormous `.Rmd` into smaller files.

[34] The high school is three years in Taiwan (grades 10-12).

10 Discussion and Conclusion

Unfinished below

Add a summary of our thought process in performing the data analysis

From data collection to meaningful insights

Ideas to elaborate on:

- In-sample prediction vs out-of-sample prediction
- Accuracy, precision, recall, FPR, FNR + extreme data situations
- Good coding practices: user-defined functions, descriptive variable names, and helpful comments in code

Incorporated already:

- Data preprocessing (data cleaning)
- The importance of exploratory data analysis (link to Section)
- What we can do when the data do not satisfy the assumptions for linear regression
- Problem formulation (into a binary prediction task)

In addition to statistical concepts, we also demonstrated:

machine learning, good coding practices, etc. Add something more!

Don't just list statistics resources. Need to have a context-specific discussion!

Unfinished above

The Statistics 101 course provides a starting point for students to perform data analysis, and this manuscript is a detailed example of a project from data collection to meaningful insights. We obtained small-scale data that are publicly available, and articulated our thought process in the step-to-step execution. We started with data preprocessing (i.e., cleaning the data), explored the data, and applied statistical models to quantify the relationship between the two variables of interest. The first model we try does not always work out, and the model validation can reveal more data issues previously undiscovered in the exploratory phase. Given the new findings, we can try different ways to continue the analysis.

Our original problem is to find the relationship between high school entrance exam scores and college entrance exam scores. The exploratory analysis in Chapter 4 shows a moderate positive association between the two sets of scores, with the correlation coefficient 0.50. However, this does not necessarily mean that the linear regression is the most appropriate model. Linear regression is widely used, but it is not a panacea for data analysis. Chapter 5 states the four assumptions to be met in linear regression: linearity, nearly normal residuals, constant variability, and independent observations. When these assumptions are violated, we should not use the linear regression model.

The good news is that we can continue the data analysis through several ways:

- Transform the data to meet the assumption requirements
- Choose another model to investigate the relationship between the two variables
- Reformulate the problem and answer a different question from the data

In Chapter 7, we reformulated the problem to: Given a student's high school entrance exam score, estimate the probability of getting a college entrance exam score of at least 65. This statement binarizes the outcome of college entrance exam scores, so we discovered the relationship of the two variables in a different way. Sections 7.3 and 7.4 show that when the high school entrance exam score increases by one percentile, the

odds of "success" increases by about 16.1% on average, i.e., getting a college entrance exam score at least 65. We are 95% confident that the odds increase is between 10.6% and 22.9%.

Unfinished below

After implementing the logistic regression model, we also validated the results using in-sample prediction (Chapter 8) and out-of-sample prediction (Chapter 9). The accuracy is approximately 70% and consistent across both methods of prediction. About half of the respondents obtained a college entrance exam score at least 65, so the baseline is around 50% like a coin-flip. Compared with the baseline, the logistic regression model is doing much better. The detailed metrics are listed and explained in Section 9.3.

What about the breakdown by high school entrance exam score category?

Why are we interested in this part? Add link to the relevant sections.

More comments here

What else did we demonstrate in this manuscript?

Elaborate on this statement above

Write something more

Consider moving the above to an earlier section as transition

For the next steps in learning statistics, we suggest reading *The Statistical Sleuth: A Course in Methods of Data Analysis* (Ramsey and Schafer, 2013), which is the textbook for undergraduate-level regression analysis at Duke Statistical Science.[35] The book covers intermediate topics such as ANOVA (Analysis of Variance) and multiple linear regression. It also provides data files for case studies and exercises.[36]

For the advanced readers, we recommend the following graduate level statistics textbooks:

- *A First Course in Bayesian Statistical Methods* (Hoff, 2009)
- *Statistical Inference* (Casella and Berger, 2021)
- *Categorical Data Analysis* (Agresti, 2003)

There are many other high-quality statistics textbooks, and we selected these as a starting point.

Write something more

11 Final: Personal Remarks

Write something here

Taipei First Girls' High School[37] typically requires **HighSchool_PR** 99 for admission. There are some exceptions, such as recruited athletes, students with disabilities,[38] and students under other extraordinary situations.[39]

The Department of Electrical Engineering at National Taiwan University (NTUEE)[40] typically requires full marks (15 out of 15) in English, mathematics, and science in **College_Score** for the early admission.[41] Most students at NTUEE had a **College_Score** of 70 or higher, at the time when 75 was the max possible score. But still a significant number of students got admitted through the regular college entrance exam process in July.

[35] https://www2.stat.duke.edu/courses/Fall18/sta210.001/
[36] http://www.statisticalsleuth.com/
[37] http://web.fg.tp.edu.tw/~tfghweb/EnglishPage/index.php
[38] http://www.rootlaw.com.tw/LawArticle.aspx?LawID=A040080080001900-1020822
[39] https://bit.ly/2WtRY63
[40] https://web.ee.ntu.edu.tw/eng/index.php
[41] https://university.1111.com.tw/univ_depinfo9.aspx?sno=100102&mno=520101

Finally, include my own scores as a datapoint for prediction.

Don't show the numbers until I am ready to work on this section!

Acknowledgments

The author would like to thank her Microsoft colleagues Smit Patel and Dylan Stout for troubleshooting GitHub issues.

More to add

Appendix

RStudio, code reproducibility, etc.

Need to specify the R libraries.

The code is mainly for demonstration purposes. To learn about better coding styles, we recommend the book *Code Complete* (McConnell, 2004) instead.

References

Agresti, A. (2003). *Categorical Data Analysis*, volume 482. John Wiley & Sons.

Alpaydin, E. (2020). *Introduction to Machine Learning*. MIT press, 4th edition.

Casella, G. and Berger, R. L. (2021). *Statistical Inference*. Cengage Learning.

Chai, C. P. (2020). The importance of data cleaning: Three visualization examples. *CHANCE*, 33(1):4–9. Available from: https://chance.amstat.org/2020/02/data-cleaning/.

Clyde, M., Cetinkaya-Rundel, M., Rundel, C., Banks, D., Chai, C., and Huang, L. (2020). *An Introduction to Bayesian Thinking: A Companion to the Statistics with* R *Course*. GitHub. Available from: https://statswithr.github.io/book/.

Diez, D. M., Cetinkaya-Rundel, M., and Barr, C. D. (2019). *OpenIntro Statistics*. OpenIntro, 4th edition. Available from: https://www.openintro.org/book/os/.

Foote, S. (2014). *Learning to Program*. Addison-Wesley Professional.

Hoff, P. D. (2009). *A First Course in Bayesian Statistical Methods*, volume 580. Springer.

McConnell, S. (2004). *Code Complete*. Pearson Education, 2nd edition. Available from: https://tinyurl.com/code-complete-book.

NCSS (n.d.). Binary diagnostic tests – single sample. In *NCSS Statistical Software*, chapter 535. NCSS (Number Cruncher Statistical System). Available from: https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Binary_Diagnostic_Tests-Single_Sample.pdf.

Ramsey, F. and Schafer, D. (2013). *The Statistical Sleuth: A Course in Methods of Data Analysis*. Cengage Learning. Data files are available on: http://www.statisticalsleuth.com/.