

# POLI803: Maximum Likelihood Estimation

Week 3: Linear Regression Review and The Linear Probability Model

Ph.D. Student, Sanghoon Park (Univ. of South Carolina)

9/11/2019

여기서는 먼저 선형회귀분석(linear regression)에 대해 간단한 리뷰를 진행한다. 그리고 선형확률모형에 대해서 살펴본다. 선형확률모형을 통해 우리는 종속변수가 연속형이 아닐 때, 즉 예측변수와 종속변수 간 선형 관계를 가정할 수 없을 때, OLS를 사용하는 것이 왜 ‘비효율적’(inefficient)이고 그 추정치가 BLUE<sup>1</sup>일 수 없는지를 살펴볼 것이다.

## 필요한 패키지를 로드하기

```
library(tidyverse) # 데이터 관리 및 전처리를 위한 주요 패키지
library(ezpickr) # 다른 확장자의 파일을 R로 불러오기 위한 패키지
library(here) # 현재 작업디렉토리를 R-스크립트가 위치한 디렉토리로 자동설정하는 패키지
library(lubridate) # 날짜시각 데이터를 원활하게 제공하는 데 특화된 패키지
library(lobstr) # str() 함수와 같지만 객체를 좀 더 구체적으로 살펴볼 수 있게 하는 패키지
library(broom) # lm, nls, t.test 등과 같은 통계분석 함수의 결과를 정연하게 정리해주는 패키지
library(ggfortify) # ggplot2 플로팅을 도와주는 패키지
library(estimatr) # 로버스트 표준오차나 신뢰구간 등과 같은 추정치들을 빠르게 계산해주는 패키지
```

선형회귀분석은 몇 가지 가정들에 바탕을 두고 있다.

1. 선형회귀분석은 다차항(polynomial terms) 등을 포함하여 비선형성(non-linearity)도 모델에 반영할 수 있지만 기본적으로는 종속변수와 예측변수 간의 관계가 선형일 것 등을 가정한다.
2. 잔차(residuals)는 정규적으로 분포되어 있어야 한다.—잔차의 정규성
3. 데이터는 iid 가정(assumed independently and identically distributed)을 충족시켜야 한다.
  1. 데이터의 각 확률변수는 상호독립적이며(independent),
  2. 모두 동일한 확률분포를 가지고(등분산성, identical)
  3. 정규분포를 따른다(normally distributed). 즉, 데이터의 각 확률변수들과 잔차는 서로 상관하지 않아야 한다.
4. 등분산성(homoscedasticity) + 모든 잔차(오차)는 같은 분산을 가진다.

## 우선 데이터를 전처리해보자.

데이터를 전처리 및 분석할 때에는 처음에 불러온 데이터의 사본을 만들어 항상 원본을 유지하는 것이 중요하다. 혹여 전처리 과정에서 실수를 하여 변수를 제거하거나 잘못 조작하였을 때, 원본에 바로 작업을 해버리면 다시 다운을 받거나 혹은 불러오는(import) 귀찮은 과정을 거쳐야 하기 때문이다.

```
here::here() %>% setwd(.)
df <- pick('example_data/diamonds_data.xlsx')
original <- df # 이렇게 원본 데이터임을 알 수 있게 라벨링을 해준다.
## 이제 데이터가 어떻게 생겨먹었는지 한 번 들여다 보자.
lobstr::obj_addr(original)
```

```
## [1] "0x1a089ed0"
```

obj\_addr() 함수는 객체의 주소(address)를 부여하는 함수이다. 이 함수를 통해서 우리는 각 객체에 특정한 주소를 부여할 수 있고, 만약 사본을 만들더라도 그 주소는 동일하게 복사된다. 따라서 두 객체의 동일 여부를 확인할 수 있다.

<sup>1</sup>불편추정량을 의미한다. Best Unbiased Linear Estimator.

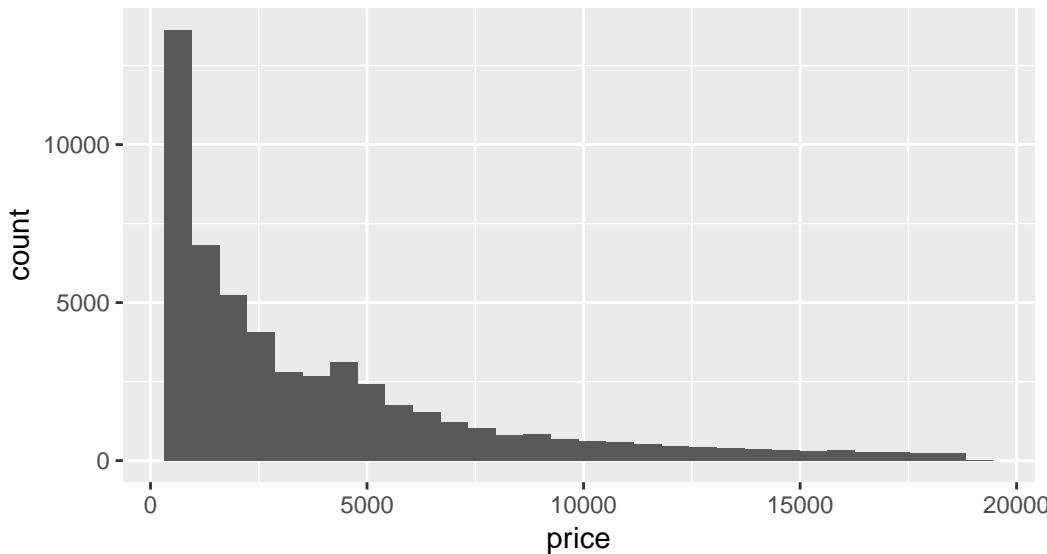
```
## original 객체의 원형인 df와의 동일 여부를 확인해보자.  
lobstr::obj_addr(df); lobstr::obj_addr(original)
```

```
## [1] "0x1a089ed0"  
## [1] "0x1a089ed0"  
## 두 객체의 주소가 동일한 것을 확인할 수 있다.
```

## 복제 및 수정(Copy and modify)

먼저 이 예제에서 사용할 종속변수는 바로 다이아몬드 데이터의 가격(price) 변수이다. 따라서 가격 변수의 분포를 살펴보자.

```
df %>%  
  ggplot(aes(price)) + geom_histogram()
```



일단 전체 다이아몬드 데이터에서 가격 변수의 분포는 좌측으로 치우친(left-skewed) 것을 확인할 수 있다. 즉, OLS의 기본 가정에서 언급되었던 종속변수가 정규적으로 분포되어 있을 것이라는 가정이 위배되고 충분히 의심해볼 수 있다.

자, 이제는 이 df 데이터의 변수들을 필요에 따라 조작해주도록 하자. 이전에 말했듯이 `mutate()` 함수는 내부에 다중의 코드를 통해서 여러 변수를 한 번에 생성 및 조작할 수 있다.

- `cut` 변수는 기존에 문자형이었지만 `Fair`, `Good`, `Very Good`, `Premium`, `Ideal`을 레이블로 하는 요인형(factor)로 바꾸어줄 것이다. 그리고 결측치는 제외할 것이다.
- `df`가 티끌이기 때문에 원래 자료의 유형을 그대로 간직하고 있다. 하지만 통계분석을 수행하기 위해서는 문자형으로는 불가능하고 이를 요인형으로 바꾸어주어야 한다.
- `cut`, `color`, `clarity`는 문자형 변수이지만 모두 요인형으로 재조작하여 주었다.
- `sold`의 경우는 기존의 `sold` 변수 중 결측치는 NA, 팔린 것(sold)은 1, 안 팔린 것(unsold)은 0으로 코딩한다.
- `price_bin`은 원래 숫자형 변수인 `price`를 조건에 따라 `low`, `medium`, `high`의 레이블을 갖는 요인형 변수로 조작한다.

```
df <- df %>%  
  mutate(  
    cut = parse_factor( # 요인형으로 나누어라.  
      cut, levels = c("Fair", "Good", "Very Good", "Premium", "Ideal"),  
      include_na = FALSE # 결측체는 제외하라.  
    ),  
    color = parse_factor(
```

```

    color, levels = c("D", "E", "F", "G", "H", "I", "J"),
    include_na = F
),
clarity = parse_factor(
  clarity, levels = c(
    "I1", "SI2", "SI1", "VS2", "VS1", "VVS2", "VVS1", "IF"
  ), include_na = FALSE
),
sold = if_else( # sold 변수를 만들되,
  is.na(sold), NA_integer_, # 기존의 sold 변수 중 결측치는 NA_integer_로 값을 변경하고
  if_else(sold == 'sold', 1L, 0L) # 결측치가 아닌 것들 중 'sold'는 1로, 아닌 것은 0으로 코딩
  ## 즉, 이 코드대로라면 결측치는 NA, sold = 1, unsold = 0이 될 것이다.
),
price_bins = case_when(      # ifelse와 비슷한 조건문이라고 보면 된다.
  price <= 2500L ~ 'low', # price가 2500 이하이면 'low'라고 레이블
  price > 2500L & price <= 7500L ~ 'medium', # 2500 초과 7500 이하면 'medium'
  price > 7500L ~ 'high', # 7500 초과면 'high'
  TRUE ~ NA_character_     # 위의 세 조건을 만족시키지 못하는 관측치는 NA 처리
) %>% parse_factor(., levels = c('low', 'medium', 'high'), include_na = F)
) # 그리고 이렇게 분류된 결과를 요인형 변수로 변경, NA는 제외

```

## 변수의 순서 바꾸기

```

df$index <- 1:nrow(df) # df 데이터에 행번호를 표시하는 변수를 추가
df <- df %>% select(
  row_id = index, sold, carat, cut, color, clarity, depth, # 행을 식별하는 row_id를 만들고
  ## row_id, sold, carat, cut, color, clarity, depth, table, price, price_bins, x, y, z 순으로
  table, price, price_bins, x, y, z # 변수의 순서를 바꾼다.
)
head(df)

## # A tibble: 6 x 13
##   row_id sold carat cut   color clarity depth table price price_bins     x
##       <int> <int> <dbl> <fct> <fct> <fct> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1       1     0  0.23 Ideal    E     SI2     61.5    55    326 low     3.95
## 2       2     0  0.21 Prem~ E     SI1     59.8    61    326 low     3.89
## 3       3     0  0.23 Good   E     VS1     56.9    65    327 low     4.05
## 4       4     0  0.290 Prem~ I     VS2     62.4    58    334 low     4.2
## 5       5     1  0.31 Good   J     SI2     63.3    58    335 low     4.34
## 6       6     0  0.24 Very~ J     VVS2     62.8    57    336 low     3.94
## # ... with 2 more variables: y <dbl>, z <dbl>

```

그렇다면 이번에는 데이터 안에서 가격이 high인 다이아몬드가 몇 개나 있을지 세어보자. 이미 price\_bin이라는 변수를 새로 만들어서 숫자형인 price가 얼마를 기준으로 초과할 때 high라고 할 수 있는지를 지정해 두었다.

```

df %>% count(price_bins)

## # A tibble: 3 x 2
##   price_bins     n
##   <fct>     <int>
## 1 low         27542
## 2 medium      18016
## 3 high        8382

```

그렇다면 각각의 가격 범주에서의 평균 가격은?

```
df %>% select(price, price_bins) %>%
  group_by(price_bins) %>%
  summarise_all(mean, na.rm = T)

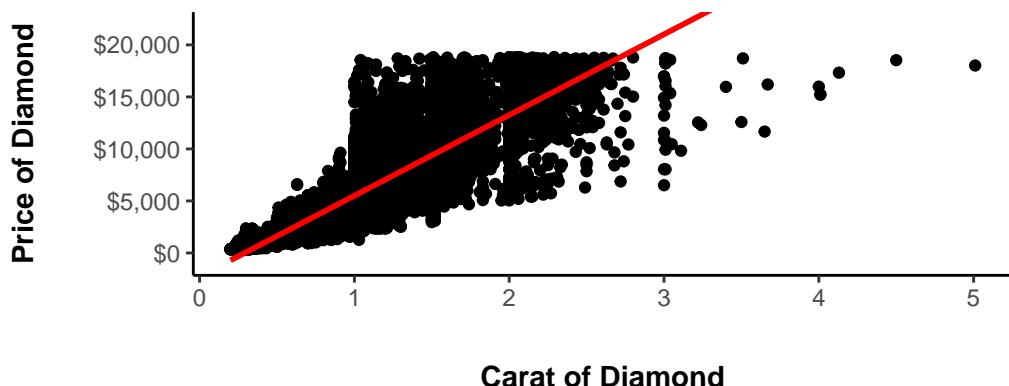
## # A tibble: 3 x 2
##   price_bins  price
##       <fct>     <dbl>
## 1 low        1153.
## 2 medium      4542.
## 3 high       11759.
```

## 선형관계 살펴보기

이번에는 다이아몬드의 무게(carat)와 가격 간의 관계를 살펴보자. 이번에는 통계치보다는 가시화된 플롯으로 살펴볼 것이다. x 축에는 무게, y축에는 가격이 설정될 것이다.

```
df %>%
  ggplot(aes(x = carat, y = price)) +
  geom_point() +
  geom_smooth(method = 'lm', se = F, color = 'red') +
  coord_cartesian(ylim = c(-1000, 22000)) +
  theme_classic() +
  labs(
    x = 'Carat of Diamond', y = 'Price of Diamond',
    title = "Effect of Carat on Diamond Price"
  ) +
  theme(
    axis.title.x = element_text(face = 'bold', margin = margin(t = 20, b = 10)),
    axis.title.y = element_text(face = 'bold', margin = margin(r = 20, l = 10)),
    plot.title = element_text(
      hjust = 0.5, margin = margin(t = 10, b = 25), face = 'bold', size = 20
    )
  ) +
  scale_y_continuous(labels = scales::dollar_format())
```

## Effect of Carat on Diamond Price

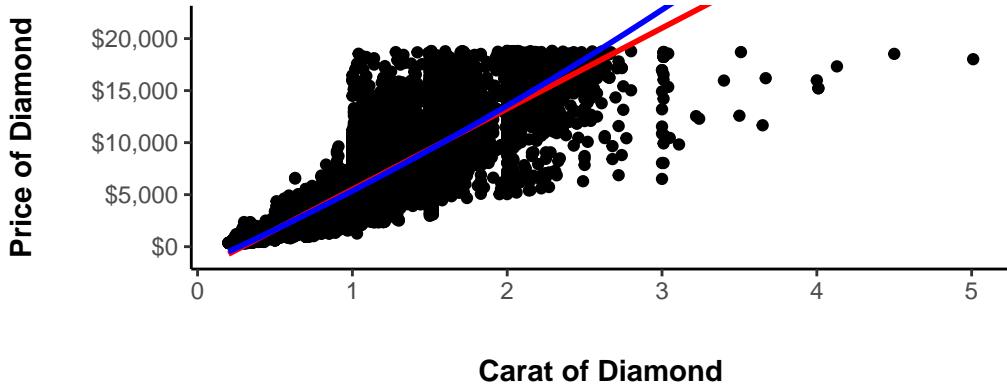


결과를 보면 선형이라기 보다는 지수함수와도 같은 형태로 우측으로 가파르게 상승하는 관계가 존재하는 것을 확인할 수 있다. 아마도 이차항(quadratic term)을 포함하는 것이 더 관계를 잘 보여줄 수 있을 것으로 보인다.

- $y = \alpha + \beta_1 x_1 + \epsilon$ 과 같은 선형 모델에서
- $y = \alpha + \beta_1 x_1^2 + \epsilon$ 과 같은 관계일 것이라고 상정해보는 것이다.

```
df %>%
  ggplot(aes(x = carat, y = price)) +
  geom_point() +
  geom_smooth(method = 'lm', se = F, color = 'red') + # 이건 아까와도 같은 선형
  geom_smooth(method = 'lm', formula = y ~ x + I(x^2), se = F, color = 'blue') + # 요건 2차항
  coord_cartesian(ylim = c(-1000, 22000)) +
  theme_classic() +
  labs(
    x = 'Carat of Diamond', y = 'Price of Diamond',
    title = "Effect of Carat on Diamond Price"
  ) +
  scale_y_continuous(labels = scales::dollar_format())
```

## Effect of Carat on Diamond Price



어떤 것이 데이터를 더 잘 보여주는 것 같은가? 파란선, 이차항으로 그 관계를 표현해준 것이 좀 더 carat과 price의 관계를 잘 보여주는 것으로 보인다.

그럼 이번에는 다이아몬드 데이터에 관측치가 너무 많기 때문에 딱 1,000의 배수가 되는 관측치들만 뽑아서 살펴보자.—천번째, 이천번째, 삼천번째…

```
temp_df <- df %>%
  dplyr::filter(row_id %% 1000 == 0) # %%는 나머지를 구하는 연산자이다.
glimpse(temp_df)
```

```
## Observations: 53
## Variables: 13
## $ row_id      <int> 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 900...
```

```

## $ sold      <int> 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, ...
## $ carat    <dbl> 1.12, 0.72, 0.90, 0.74, 1.05, 1.01, 1.00, 1.01, 0.9...
## $ cut       <fct> Premium, Ideal, Good, Ideal, Very Good, Premium, Pr...
## $ color     <fct> J, D, G, D, I, H, F, G, E, D, D, D, G, H, F, I, E, ...
## $ clarity   <fct> SI2, VS2, SI2, SI1, SI2, SI2, SI2, SI1, SI1, V...
## $ depth     <dbl> 60.6, 61.8, 63.8, 61.8, 62.3, 62.8, 62.6, 61.9, 63....
## $ table     <dbl> 59, 57, 56, 56, 59, 61, 58, 59, 57, 59, 59, 62, 60, ...
## $ price     <dbl> 2898, 3099, 3303, 3517, 3742, 3959, 4155, 4327, 451...
## $ price_bins <fct> medium, medium, medium, medium, medium, medium, med...
## $ x         <dbl> 6.68, 5.76, 6.13, 5.84, 6.42, 6.33, 6.37, 6.46, 6.1...
## $ y         <dbl> 6.61, 5.73, 6.16, 5.88, 6.46, 6.25, 6.31, 6.39, 6.1...
## $ z         <dbl> 4.03, 3.55, 3.92, 3.62, 4.01, 3.95, 3.97, 3.98, 3.8...

```

# 선형 모형을 만들어보자.

```

model <- lm(price ~ carat, data = temp_df) # 종속변수는 price, 예측변수는 carat
## 기초적인 모델 결과를 보여주는 함수는 다음과 같다.
summary(model)

```

```

##
## Call:
## lm(formula = price ~ carat, data = temp_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -6874.0  -727.7  -413.6   121.2  5785.7 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -836.3     471.6  -1.774   0.0821 .  
## carat        5719.0    440.7   12.977  <2e-16 *** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1983 on 51 degrees of freedom
## Multiple R-squared:  0.7676, Adjusted R-squared:  0.763 
## F-statistic: 168.4 on 1 and 51 DF,  p-value: < 2.2e-16

```

# 조금 세련된 방법.

```

fitted_values <- broom::augment(model) # 선형모델로부터 예측값과 그 외 기타 통계치들을
head(fitted_values)

```

```

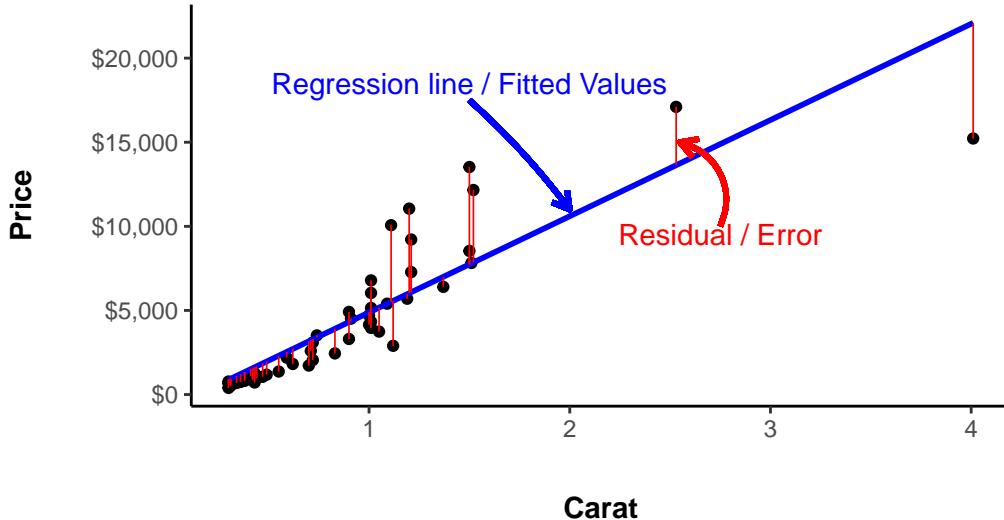
## # A tibble: 6 x 9
##   price carat .fitted .se.fit .resid   .hat .sigma   .cooksdi .std.resid
##   <dbl> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>       <dbl>
## 1 2898  1.12    5569.   293. -2671.  0.0219   1966.  0.0207    -1.36
## 2 3099  0.72    3281.   281.  -182.  0.0200   2003.  0.0000882   -0.0929
## 3 3303  0.9     4311.   273. -1008.  0.0189   1998.  0.00254    -0.513
## 4 3517  0.74    3396.   279.   121.  0.0197   2003.  0.0000384    0.0617
## 5 3742  1.05    5169.   283. -1427.  0.0204   1993.  0.00550    -0.727
## 6 3959  1.01    4940.   279.  -981.  0.0198   1998.  0.00252    -0.500

```

# *fitted\_values*라는 객체에 저장하자. 이때, *broom* 패키지를 사용하면 좀 더 깔끔하게  
## 통계치들을 추출해내는 것이 가능하다.

## 선형모형에 대한 기본적인 개념들 리뷰 및 ggplot 튜토리얼

```
fitted_values %>% # 각 1,000번째 관측치로 필터링하여 총 53개의 값을 가지는 temp_df를
  ggplot(aes(x = carat, y = price)) + # 기준으로 종속: price/예측: carat 관계를 본다.
    geom_point() + # 둘의 관계를 산포도로 보여주라는 옵션이고
    geom_smooth(method = 'lm', se = F, color = 'blue') + # 관계를 선형모델에 따라서 선으로
      # 나타내되 파란색 선으로 나타내라는 옵션이다.
    geom_segment( # 수직선을 그리는 옵션
      aes(x = carat, y = price, xend = carat, yend = .fitted), # xend는 carat, yend는 예측값
      color = "red", size = 0.3 # 즉, 관측치로부터 예측값에 수직인 선을 그으라는 코드
    ) +
    geom_curve( # 커브 화살표를 그리는 옵션
      aes(x = 2.75, y = 10000, xend = 2.55, yend = 15000), color = 'red', # 커브 명령어
      ## 보면 알겠지만 시작이 x가 2.75, y가 10000인데 끝은 2.55, 15000이다.
      ## 즉, x는 -0.2만큼, y는 +5000만큼 좌상 방향의 곡선을 그리라는 명령어
      size = 1, arrow = arrow(length = unit(.1, "inches")) # 화살표 추가(화살코)
    ) +
    annotate(
      "text", x = 2.75, y = 9500, label = "Residual / Error", color = 'red'
    ) +
    geom_curve(
      aes(x = 1.5, y = 17500, xend = 2.0, yend = 11000), color = 'blue',
      size = 1, arrow = arrow(length = unit(.1, "inches")),
      curvature = -0.05
    ) +
    annotate(
      "text", x = 1.5, y = 18500,
      label = "Regression line / Fitted Values", color = 'blue'
    ) +
    labs(x = 'Carat', y = "Price") +
    scale_y_continuous(labels = scales::dollar_format()) +
    theme_classic() +
    theme(
      axis.title.x = element_text(face = 'bold', margin = margin(t = 20, b = 10)),
      axis.title.y = element_text(face = 'bold', margin = margin(r = 20, l = 10))
    )
```



이 노트는 기본적으로 OLS 회귀분석—최소자승법을 이용한 선형회귀분석에 대한 개념을 이해하고 있다는 전제를 바탕으로 작성되었다. 이 그림은 carat과 price 간의 관계를 보여줄 때, 선형회귀선을 긋는다는 것이 어떠한 의미인지를 보여준다.

- 우리가 가진 관측치는 개별 검은 점들이다.
- 우리는 각 점들로부터 얻은 정보를 바탕으로 그 추세를 파악하기 위한 예측을 선형인 관계로 나타나게 된다.
  - 이때 중요한 것이, 어떤 기준으로 선을 그어야 하느냐는 것이다.
  - OLS는 각 점들로부터 수직거리(즉, 실제 관측치와 예측값 간의 차이), 잔차(residuals)의 제곱 합이 가장 적은 선을 긋어야 한다는 것이다.
    - \* 모집단에 대해서는 오차(errors)지만, 표본에 적용할 때에는 잔차(residuals)라고 표현한다.
  - 따라서 Ordinary least square란 결국 잔차의 제곱합이 최소가 되는 선을 긋는 방법을 말한다.
  - 제곱을 하는 이유는 가정 상 잔차의 총합은  $0(\sum_{i=1}^n \epsilon_i = 0)$ 이기 때문이다.

이번에는 모든 요인형 변수들에 대해 기준집단(reference group)을 만드는 실습을 해보자.

```
df <- df %>%
  mutate(
    cut = relevel(cut, 'Fair'),
    color = relevel(color, 'D'),
    clarity = relevel(clarity, 'I1')
  )
```

이제 cut 변수의 기준집단은 Fair가 되었고, color는 D, clarity는 I1을 기준집단으로 가지게 되었다. 그러면 전체 데이터를 가지고 다시 선형회귀모형으로 분석해보자.

```
full_model <- lm(
  price ~ carat + cut + color + clarity + depth + x + y + z, data = df
)
summary(full_model)
```

```
##
## Call:
## lm(formula = price ~ carat + cut + color + clarity + depth +
##     x + y + z, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -21340.7 -593.5 -183.7  377.9 10749.3 
## 
## Coefficients:
```

```

##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -327.268   300.749 -1.088   0.277
## carat        11241.796    48.636 231.143 <2e-16 ***
## cutGood       611.320    33.438 18.282 <2e-16 ***
## cutVery Good  785.148    31.619 24.831 <2e-16 ***
## cutPremium    809.028    31.836 25.412 <2e-16 ***
## cutIdeal      944.412    31.099 30.368 <2e-16 ***
## colorE        -210.190   17.906 -11.738 <2e-16 ***
## colorF        -271.843   18.106 -15.014 <2e-16 ***
## colorG        -480.375   17.729 -27.096 <2e-16 ***
## colorH        -979.791   18.850 -51.978 <2e-16 ***
## colorI       -1467.034   21.178 -69.271 <2e-16 ***
## colorJ       -2370.696   26.150 -90.657 <2e-16 ***
## claritySI2    2706.550   43.849  61.724 <2e-16 ***
## claritySI1    3668.956   43.665  84.024 <2e-16 ***
## clarityVS2    4272.056   43.883  97.350 <2e-16 ***
## clarityVS1    4584.500   44.575 102.850 <2e-16 ***
## clarityVVS2   4957.509   45.884 108.045 <2e-16 ***
## clarityVVS1   5014.778   47.189 106.270 <2e-16 ***
## clarityIF     5356.537   51.047 104.933 <2e-16 ***
## depth         -49.037    4.237 -11.574 <2e-16 ***
## x            -1010.763   32.922 -30.702 <2e-16 ***
## y              12.900    19.344   0.667   0.505
## z             -47.722    33.511  -1.424   0.154
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1131 on 53917 degrees of freedom
## Multiple R-squared:  0.9197, Adjusted R-squared:  0.9196
## F-statistic: 2.806e+04 on 22 and 53917 DF,  p-value: < 2.2e-16
tidy_model <- broom::tidy(full_model) %>% print(n = Inf) # 분석 결과를 티블로 바꿔주자.

```

```

## # A tibble: 23 x 5
##   term      estimate std.error statistic  p.value
##   <chr>     <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept) -327.     301.     -1.09    2.77e- 1
## 2 carat        11242.    48.6     231.     0.
## 3 cutGood       611.     33.4     18.3     1.92e- 74
## 4 cutVery Good  785.     31.6     24.8     2.37e-135
## 5 cutPremium    809.     31.8     25.4     1.27e-141
## 6 cutIdeal      944.     31.1     30.4     7.23e-201
## 7 colorE        -210.    17.9     -11.7    8.86e- 32
## 8 colorF        -272.    18.1     -15.0    7.54e- 51
## 9 colorG        -480.    17.7     -27.1    1.31e-160
## 10 colorH       -980.    18.9     -52.0     0.
## 11 colorI      -1467.    21.2     -69.3     0.
## 12 colorJ      -2371.    26.2     -90.7     0.
## 13 claritySI2   2707.    43.8     61.7     0.
## 14 claritySI1   3669.    43.7     84.0     0.
## 15 clarityVS2   4272.    43.9     97.4     0.
## 16 clarityVS1   4585.    44.6     103.     0.
## 17 clarityVVS2  4958.    45.9     108.     0.
## 18 clarityVVS1  5015.    47.2     106.     0.
## 19 clarityIF    5357.    51.0     105.     0.

```

```

## 20 depth      -49.0    4.24   -11.6   6.07e- 31
## 21 x          -1011.   32.9    -30.7   3.15e-205
## 22 y          12.9     19.3     0.667  5.05e- 1
## 23 z          -47.7    33.5    -1.42   1.54e- 1

tidy_conf <- broom::confint_tidy(full_model) # 신뢰구간을 추정해보자.
tidy_conf <- tidy_conf %>%
  rename(ll = conf.low, ul = conf.high) # 신뢰구간을 나타내기 위한 변수들의 이름을 바꿔준다.
tidy_model <- bind_cols(tidy_model, tidy_conf) # 한 번에 티블로 보여줄 수 있도록 결합한다.
print(tidy_model, n = Inf) # 자, 결과는?

```

```

## # A tibble: 23 x 7
##   term      estimate std.error statistic p.value    ll     ul
##   <chr>     <dbl>    <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
## 1 (Intercept) -327.    301.    -1.09    2.77e- 1   -917.   262.
## 2 carat       11242.   48.6    231.     0.        11146.  11337.
## 3 cutGood      611.    33.4     18.3    1.92e- 74   546.    677.
## 4 cutVery Good 785.    31.6     24.8    2.37e-135  723.    847.
## 5 cutPremium   809.    31.8     25.4    1.27e-141  747.    871.
## 6 cutIdeal      944.    31.1     30.4    7.23e-201  883.    1005.
## 7 colorE       -210.    17.9    -11.7   8.86e- 32  -245.    -175.
## 8 colorF       -272.    18.1    -15.0   7.54e- 51  -307.    -236.
## 9 colorG       -480.    17.7    -27.1   1.31e-160  -515.    -446.
## 10 colorH      -980.    18.9    -52.0   0.        -1017.   -943.
## 11 colorI      -1467.   21.2    -69.3   0.        -1509.   -1426.
## 12 colorJ      -2371.   26.2    -90.7   0.        -2422.   -2319.
## 13 claritySI2   2707.   43.8     61.7   0.        2621.   2792.
## 14 claritySI1   3669.   43.7     84.0   0.        3583.   3755.
## 15 clarityVS2   4272.   43.9     97.4   0.        4186.   4358.
## 16 clarityVS1   4585.   44.6     103.   0.        4497.   4672.
## 17 clarityVVS2  4958.   45.9     108.   0.        4868.   5047.
## 18 clarityVVS1  5015.   47.2     106.   0.        4922.   5107.
## 19 clarityIF    5357.   51.0     105.   0.        5256.   5457.
## 20 depth        -49.0    4.24   -11.6   6.07e- 31  -57.3   -40.7
## 21 x           -1011.   32.9    -30.7   3.15e-205 -1075.   -946.
## 22 y           12.9     19.3     0.667  5.05e- 1   -25.0    50.8
## 23 z           -47.7    33.5    -1.42   1.54e- 1   -113.    18.0

```

보면 만들어진 티블은 선형회귀분석 결과를 잘 정리하고 있다. 먼저 상수항(절편)을 포함한 변수들의 계수값(estimate)이 나타나 있고, 이어서 표준오차(std.error)를 보여주고 있다. t값(statistic)을 바탕으로 유의수준(p.value)도 보여주고 있고, 신뢰구간을 위해 최저값(ll)과 최고값(ul)도 나타난다.

하지만 변수 이름에 .이 들어가 있는 것이 보기 좋지 않으니 이름을 바꾸어 보자.

```

tidy_model <- tidy_model %>%
  rename_at(vars(contains('.')), ~ str_replace(., '\\.', '_')) # .을 가진 변수들의 이름을 변경
## 변수명의 .들이 모두 _로 바뀐 것을 볼 수 있다.

```

```

fitted_model <- augment(full_model) # 선형회귀모형의 결과를 fitted_model이라는 객체로
fitted_model                         # broom 패키지를 이용해 정리

```

```

## # A tibble: 53,940 x 16
##   price carat cut   color clarity depth     x     y     z .fitted .se.fit
##   <dbl> <dbl> <fct> <fct> <fct>   <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>
## 1 326  0.23 Ideal E     SI2      61.5  3.95  3.98  2.43 -1374.   21.7
## 2 326  0.21 Prem~ E     SI1      59.8  3.89  3.84  2.31 -624.    22.5

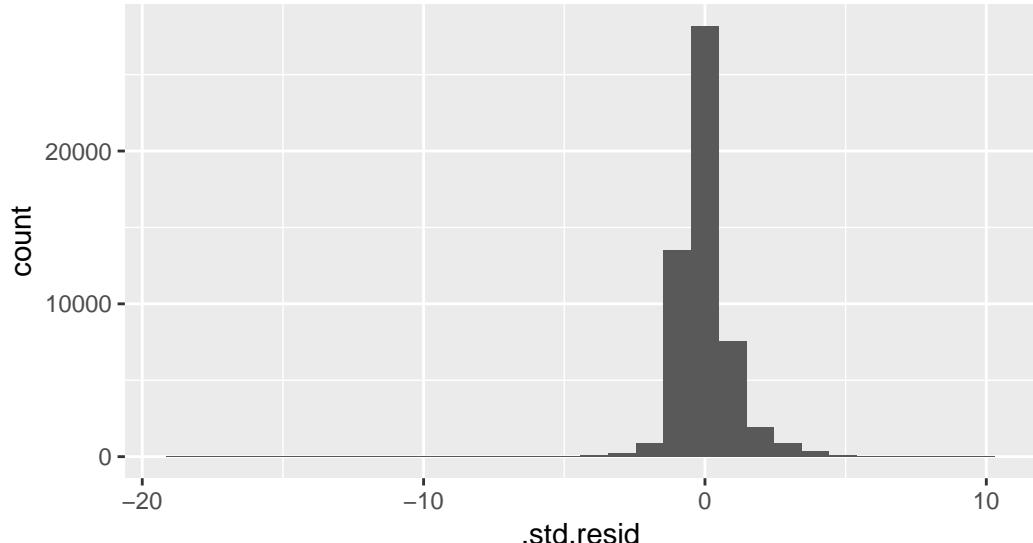
```

```

## 3 327 0.23 Good E VS1 56.9 4.05 4.07 2.31 302. 32.6
## 4 334 0.290 Prem~ I VS2 62.4 4.2 4.23 2.63 -829. 22.8
## 5 335 0.31 Good J SI2 63.3 4.34 4.35 2.75 -3461. 31.4
## 6 336 0.24 Very~ J VVS2 62.8 3.94 3.96 2.48 -1387. 30.4
## 7 336 0.24 Very~ I VVS1 62.3 3.95 3.98 2.47 -410. 27.2
## 8 337 0.26 Very~ H SI1 61.9 4.07 4.11 2.53 -1147. 21.3
## 9 337 0.22 Fair E VS2 65.1 3.87 3.78 2.49 -966. 34.3
## 10 338 0.23 Very~ H VS1 59.4 4 4.05 2.39 -369. 24.9
## # ... with 53,930 more rows, and 5 more variables: .resid <dbl>,
## # .hat <dbl>, .sigma <dbl>, .cooksdi <dbl>, .std.resid <dbl>

# 표준화된 잔차를 히스토그램으로 살펴보자
fitted_model %>%
  ggplot(aes(x = .std.resid)) + geom_histogram()

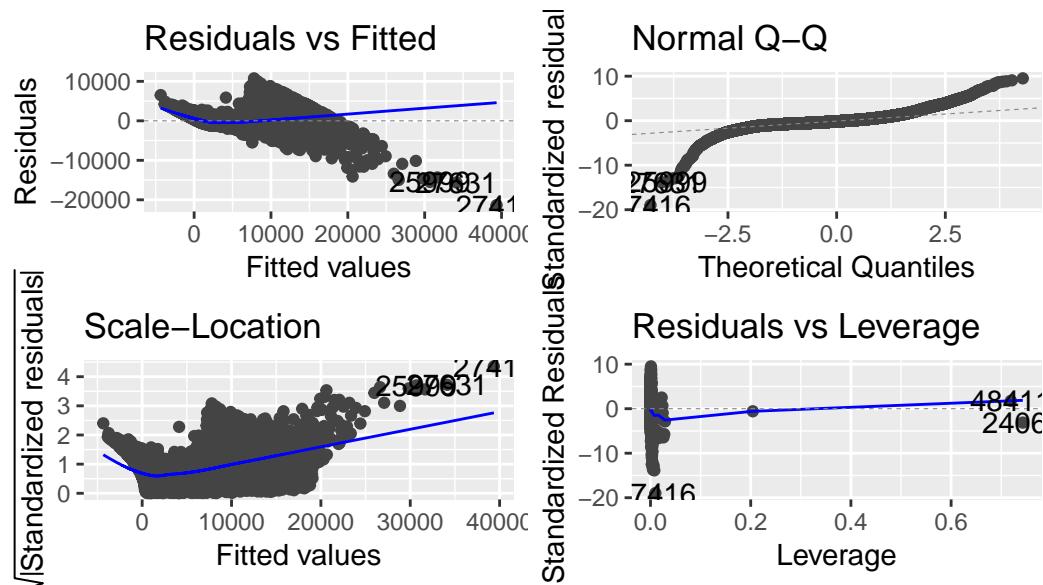
```



```

# 모형의 가정을 점검해보자.
autoplot(full_model)

```



마지막 `autoplot()`은 `ggplot2` 패키지에서 주로 사용되는 함수들을 데이터에 맞게 적용시켜주는 `ggfortify` 패키지에 속한 함수이다. 순서대로 보자면,

#### 1. 잔차와 예측값 간의 관계 (Residuals vs Fitted)

- 이 플롯은 잔차와 예측값의 관계가 선형인지 아닌지를 보여주는 플롯이다.
- 선형회귀분석의 가정이 만족되기 위해서는 이 플롯은 0의 값을 갖는 수평선을 보여주어야 한다.
- 여기서는 심하지는 않지만 약간의 선형성을 보여주고는 있다.
- 따라서 이 선형 모형에 제곱항 (squared terms)을 포함해주는 등의 조치를 취해줄 필요가 있다는 것을 보여준다.

#### 2. QQ 플롯

- QQ 플롯은 잔차가 정규분포를 따르고 있는지를 보여준다.
- 점선을 따라 분포되어 있어야 정규분포인데, 이 데이터의 결과는 역시 잔차의 정규성 또한 위반의 소지가 있음을 보여준다.

#### 3. 잔차의 동분산성 확인(Scale-Location or Spread-Location)

- 잔차의 분산의 동질성을 확인하기 위해 그리는 플롯이다(동분산성).
- 선들이 평행하게 확산되는 수평선이 동분산성을 보여주는 좋은 지표이다.
- 이 예제 데이터는 일정한 경향성을 보여주기 때문에 이분산성의 문제 (heteroscedasticity)가 존재할 수 있음을 보여준다.
- 이분산성은 로버스트 표준오차를 통해서 쉽게 해결할 수 있다.

#### 4. 잔차 대 레버리지 (Residuals vs Leverage)

- 극단적인 값들이 분석에 포함/제거될 경우에는 회귀분석 결과에 영향을 미칠 수 있다.
- 따라서 이 플롯은 그러한 극단적인 이상치들이 있는지를 확인하는 데 사용된다.
- 우측 극단에 값들이 나타나는 것을 확인할 수 있다.
- 확실히 몇몇 극단적인 이탈치들이 제거될 필요가 있음을 보여준다.<sup>2</sup>

그렇다면 종속변수가 이항변수(binary)일때는 선형회귀모형(OLS)을 사용할 수 없는가? 아니다. 사용할 '수는' 있다. 우리는 그것을 선형확률모형(linear probability model, LPM)이라고 부른다.

그러나 LPM을 사용하는 데 있어서는 여러 가지 단점들이 존재한다.

#### 1. LPM은 이분산성이 거의 확정적이다.

- 이 문제는 표본의 크기가 크면 로버스트 표준오차를 취함으로써 쉽게 해결해줄 수 있다.
- 2. LPM을 통해 우리는 0···1의 확률을 넘어서는 예측을 할 수 있다.

- 만약 단위 구간(0···1)을 넘어서는 예측확률이 없거나, 거의 없다면 LPM은 대개 불편(unbiased)하고 일관될 것으로 기대된다.

그렇다면 LPM을 사용하는 장점은?

#### 1. 빅데이터 분석을 할 때, 더 빠르게 계산이 가능하다.

- 물론 속도 문제는 이제 컴퓨터 기술이 발전해서 큰 문제는 아닐 수 있다.
- 그러나 아무리 컴퓨터가 좋아도 확률론이 아니라 최대가능도를 반복계산하게 되면 시간이 더 걸리는 것은 사실이다.

#### 2. 해석하기가 정말 쉽다.

그럼 한 번 LPM 모형을 만들어보자. 종속변수를 다이아몬드의 판매 여부(sold)로 볼 것이다. 이분산성은 거의 확실하기 때문에 표준오차를 로버스트 표준오차로 설정해줄 것이다(`se_type = "HC1"`).

```
lmp <- lm_robust(  
  sold ~ price + carat + cut + color + clarity + depth + x + y + z,  
  data = df, se_type = "HC1"  
) %>% tidy()
```

<sup>2</sup>다만 이것은 통계적 진단일 뿐이다. 극단적인 값을 지니는 이른바 이탈치(outliers)에 대한 처치를 어떻게 할 것인가는 연구자의 관점에 좌우된다. 자세한 논의는 김웅진. 2016. “Deviant Case in the Social Scientific Generalization: A Methodological Discourse on the Analytic Conventions for Detecting Deviance.” 21세기정치학회보. 26권 4호, pp. 49-66. 또는 김웅진. 2016. “사회과학적 개념의 방법론적 경직성 : 국소성과 맥락성의 의도적 훼손.” 국제지역연구. 19권 4호, pp. 3-22.를 읽어보라.

```

# 만약에 위의 se_type을 지정해주지 않으면? 계수값은 변하지 않지만 표준오차가 달라진다.
non_lmp <- lm(
  sold ~ price + carat + cut + color + clarity + depth + x + y + z,
  data = df
) %>% tidy()
# 결과를 한 번 보자.
lmp <- as_tibble(lmp[, 1:5])

lmp.table <- lmp %>% print(n = Inf)

knitr::kable(lmp.table)

```

term	estimate	std.error	statistic	p.value
(Intercept)	0.3657903	0.1285860	2.8447123	0.0044468
price	-0.0000025	0.0000019	-1.2877022	0.1978552
carat	0.0214591	0.0304262	0.7052835	0.4806368
cutGood	0.0159846	0.0148174	1.0787681	0.2806959
cutVery Good	0.0115692	0.0140438	0.8237900	0.4100625
cutPremium	0.0154833	0.0141476	1.0944138	0.2737784
cutIdeal	0.0121528	0.0138536	0.8772339	0.3803635
colorE	-0.0109737	0.0079250	-1.3846932	0.1661520
colorF	-0.0160942	0.0080186	-2.0070998	0.0447440
colorG	-0.0020303	0.0078893	-0.2573431	0.7969149
colorH	-0.0065396	0.0085398	-0.7657743	0.4438139
colorI	-0.0154642	0.0097701	-1.5828074	0.1134712
colorJ	0.0058370	0.0124041	0.4705699	0.6379498
claritySI2	-0.0182485	0.0200410	-0.9105561	0.3625334
claritySI1	-0.0040441	0.0205093	-0.1971824	0.8436856
clarityVS2	-0.0053402	0.0210149	-0.2541143	0.7994082
clarityVS1	0.0093340	0.0215302	0.4335289	0.6646323
clarityVVS2	-0.0072611	0.0223522	-0.3248506	0.7452954
clarityVVS1	0.0108147	0.0229192	0.4718602	0.6370285
clarityIF	-0.0266747	0.0247418	-1.0781201	0.2809850
depth	0.0021288	0.0017931	1.1872084	0.2351507
x	0.0003846	0.0131787	0.0291827	0.9767190
y	0.0031623	0.0072927	0.4336255	0.6645621
z	-0.0062441	0.0115441	-0.5408904	0.5885854

```

non_lmp.table <- non_lmp %>% print(n = Inf)

knitr::kable(non_lmp.table)

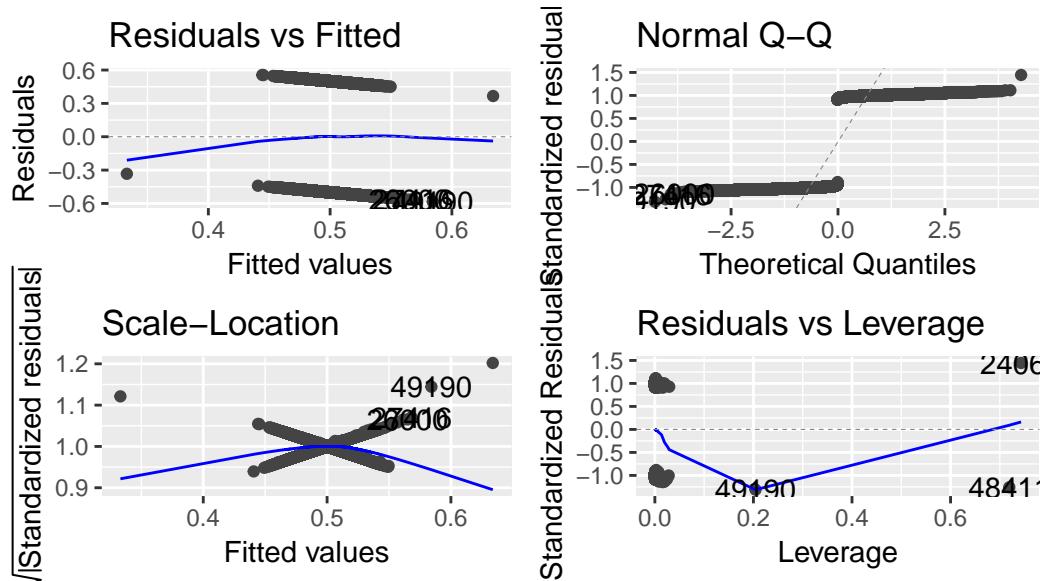
```

term	estimate	std.error	statistic	p.value
(Intercept)	0.3657903	0.1329503	2.7513312	0.0059373
price	-0.0000025	0.0000019	-1.2900308	0.1970455
carat	0.0214591	0.0303362	0.7073751	0.4793365
cutGood	0.0159846	0.0148271	1.0780638	0.2810101
cutVery Good	0.0115692	0.0140572	0.8230067	0.4105078
cutPremium	0.0154833	0.0141576	1.0936345	0.2741202
cutIdeal	0.0121528	0.0138646	0.8765344	0.3807435
colorE	-0.0109737	0.0079257	-1.3845661	0.1661909
colorF	-0.0160942	0.0080207	-2.0065961	0.0447977

term	estimate	std.error	statistic	p.value
colorG	-0.0020303	0.0078903	-0.2573118	0.7969391
colorH	-0.0065396	0.0085391	-0.7658427	0.4437733
colorI	-0.0154642	0.0097697	-1.5828680	0.1134574
colorJ	0.0058370	0.0124098	0.4703553	0.6381031
claritySI2	-0.0182485	0.0200572	-0.9098198	0.3629216
claritySI1	-0.0040441	0.0205276	-0.1970065	0.8438232
clarityVS2	-0.0053402	0.0210350	-0.2538711	0.7995961
clarityVS1	0.0093340	0.0215511	0.4331086	0.6649376
clarityVVS2	-0.0072611	0.0223716	-0.3245696	0.7455081
clarityVVS1	0.0108147	0.0229412	0.4714067	0.6373522
clarityIF	-0.0266747	0.0247631	-1.0771948	0.2813981
depth	0.0021288	0.0018753	1.1351741	0.2563074
x	0.0003846	0.0146799	0.0261984	0.9790992
y	0.0031623	0.0085513	0.3698041	0.7115299
z	-0.0062441	0.0148140	-0.4214995	0.6733921

표준오차가 미묘하게 다른 것을 확인할 수 있다. 좀 더 가시적으로 보기 위해서는 이 결과를 `autoplot()`으로 보자.

```
## 먼저 로버스트 표준오차를 취하지 않은 경우.
lm(
  sold ~ price + carat + cut + color + clarity + depth + x + y + z, data = df
) %>% autoplot()
```



```
## emmeans(non_lmp, pairwise~color)
## contrasts <- emmeans(non_lmp, pairwise~"color")
## conf_int <- confint(contrasts, side = ">", level = .95, type = "response")
```

```
## 계수값을 비교해보자.
round(lmp$estimate, 4) %>% unname()
```

```
## [1] 0.3658 0.0000 0.0215 0.0160 0.0116 0.0155 0.0122 -0.0110
## [9] -0.0161 -0.0020 -0.0065 -0.0155 0.0058 -0.0182 -0.0040 -0.0053
## [17] 0.0093 -0.0073 0.0108 -0.0267 0.0021 0.0004 0.0032 -0.0062
```

```

round(non_lmp$estimate, 4)

## [1] 0.3658 0.0000 0.0215 0.0160 0.0116 0.0155 0.0122 -0.0110
## [9] -0.0161 -0.0020 -0.0065 -0.0155 0.0058 -0.0182 -0.0040 -0.0053
## [17] 0.0093 -0.0073 0.0108 -0.0267 0.0021 0.0004 0.0032 -0.0062

## 표준오차를 비교해보자.
round(lmp$std.error, 7) %>% unname()

## [1] 0.1285860 0.0000019 0.0304262 0.0148174 0.0140438 0.0141476 0.0138536
## [8] 0.0079250 0.0080186 0.0078893 0.0085398 0.0097701 0.0124041 0.0200410
## [15] 0.0205093 0.0210149 0.0215302 0.0223522 0.0229192 0.0247418 0.0017931
## [22] 0.0131787 0.0072927 0.0115441

round(non_lmp$std.error, 7)

## [1] 0.1329503 0.0000019 0.0303362 0.0148271 0.0140572 0.0141576 0.0138646
## [8] 0.0079257 0.0080207 0.0078903 0.0085391 0.0097697 0.0124098 0.0200572
## [15] 0.0205276 0.0210350 0.0215511 0.0223716 0.0229412 0.0247631 0.0018753
## [22] 0.0146799 0.0085513 0.0148140

## 계수는 한계효과를 보여주기 때문에 이 경우에는 해석이 꽤 쉬운 편이다.
## 이번에는 가격에 대한 선형회귀모형을 분석하고
## 그 결과를 좀 더 보기 좋은 플롯으로 바꾸어보자.
## 가격을 종속변수로 쓰는 이유는 다이아몬드 데이터는 실제 데이터기 때문에
## 무작위 값을 가지지 않기 때문이다. (관계가 나타나면 잘 보일 것)

# 먼저 변수들을 좀 조작해주어야 한다.
library(wesanderson)

new_df <- df %>% sample_n(1000)
full_model <- lm(
  price ~ carat + cut + color + clarity + depth + x + y + z, data = new_df
)

summary(full_model)

##
## Call:
## lm(formula = price ~ carat + cut + color + clarity + depth +
##     x + y + z, data = new_df)
##
## Residuals:
##       Min     1Q   Median     3Q    Max 
## -6695.9 -483.0 -118.5  349.2 7421.5 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -22460.3     7814.7  -2.874 0.004140 ** 
## carat        12681.0      355.7  35.649 < 2e-16 *** 
## cutGood      378.6       227.4   1.664 0.096365 .    
## cutVery Good 479.1       217.5   2.203 0.027846 *  
## cutPremium   526.8       215.4   2.446 0.014637 *  
## cutIdeal     575.6       212.1   2.714 0.006769 ** 
## colorE       -197.8      107.2  -1.844 0.065413 .    
## colorF       -331.6      111.1  -2.985 0.002903 ** 

```

```

## colorG      -579.2    106.4   -5.445 6.57e-08 ***
## colorH      -850.9    111.9   -7.601 6.88e-14 ***
## colorI     -1322.6    130.4  -10.145 < 2e-16 ***
## colorJ     -2443.3    156.4  -15.623 < 2e-16 ***
## claritySI2  1475.5    295.1    5.000 6.80e-07 ***
## claritySI1  2521.3    294.1    8.572 < 2e-16 ***
## clarityVS2  3032.5    295.3   10.268 < 2e-16 ***
## clarityVS1  3494.5    298.9   11.692 < 2e-16 ***
## clarityVVS2 3693.4    306.9   12.036 < 2e-16 ***
## clarityVVS1 3611.1    314.7   11.476 < 2e-16 ***
## clarityIF   4386.2    347.3   12.630 < 2e-16 ***
## depth       373.7     126.4   2.956 0.003191 **
## x           593.4     924.4   0.642 0.521068
## y           2579.7    980.9   2.630 0.008672 **
## z          -7830.4   2107.4  -3.716 0.000214 ***
##
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 939.2 on 977 degrees of freedom
## Multiple R-squared:  0.938, Adjusted R-squared:  0.9366
## F-statistic: 671.5 on 22 and 977 DF, p-value: < 2.2e-16

```

먼저 new\_df는 기존의 df 데이터에서 1,000개의 관측치만을 표본으로 뺀 것이다. full\_model은 가격을 종속변수로 하고, carat, cut, color, clarity, depth, x, y, z를 예측변수로 하고 있다.

이제 full\_model의 결과를 broom 패키지의 함수들을 이용해 깔끔하게 정리해보자.

```
new_tidy_model <- broom::tidy(full_model) %>% print(n = Inf) # 결과를 티블로 바꾼다.
```

```

## # A tibble: 23 x 5
##   term        estimate std.error statistic  p.value
##   <chr>      <dbl>     <dbl>     <dbl>      <dbl>
## 1 (Intercept) -22460.    7815.    -2.87  4.14e- 3
## 2 carat       12681.    356.     35.6   5.68e-179
## 3 cutGood      379.     227.     1.66  9.64e- 2
## 4 cutVery Good 479.     218.     2.20  2.78e- 2
## 5 cutPremium    527.     215.     2.45  1.46e- 2
## 6 cutIdeal      576.     212.     2.71  6.77e- 3
## 7 colorE      -198.     107.    -1.84  6.54e- 2
## 8 colorF      -332.     111.    -2.99  2.90e- 3
## 9 colorG      -579.     106.    -5.44  6.57e- 8
## 10 colorH     -851.     112.    -7.60  6.88e-14
## 11 colorI     -1323.    130.   -10.1   4.64e-23
## 12 colorJ     -2443.    156.   -15.6   2.79e-49
## 13 claritySI2  1475.    295.     5.00  6.80e- 7
## 14 claritySI1  2521.    294.     8.57  3.94e-17
## 15 clarityVS2  3033.    295.    10.3   1.47e-23
## 16 clarityVS1  3495.    299.    11.7   1.19e-29
## 17 clarityVVS2 3693.    307.    12.0   3.27e-31
## 18 clarityVVS1 3611.    315.    11.5   1.09e-28
## 19 clarityIF   4386.    347.    12.6   5.54e-34
## 20 depth       374.     126.     2.96  3.19e- 3
## 21 x           593.     924.     0.642 5.21e- 1
## 22 y           2580.    981.     2.63  8.67e- 3
## 23 z          -7830.   2107.    -3.72  2.14e- 4

```

```

new_tidy_conf <- broom::confint_tidy(full_model) # 신뢰구간을 구한다.
new_tidy_conf <- new_tidy_conf %>%
  rename(ll = conf.low, ul = conf.high) # 변수 이름을 변경한다.
new_tidy_model <- bind_cols(new_tidy_model, new_tidy_conf)
## 회귀분석 결과와 추가로 구한 신뢰구간 변수를 하나의 티블로 구성한다.
print(new_tidy_model, n = Inf) # 결과를 살펴보자.

## # A tibble: 23 x 7
##   term      estimate std.error statistic p.value     ll     ul
##   <chr>     <dbl>    <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
## 1 (Intercept) -22460.    7815.    -2.87  4.14e- 3 -37796.   -7125.
## 2 carat       12681.    356.     35.6   5.68e-179 11983.   13379.
## 3 cutGood      379.     227.     1.66   9.64e- 2   -67.8    825.
## 4 cutVery Good 479.     218.     2.20   2.78e- 2    52.3    906.
## 5 cutPremium    527.     215.     2.45   1.46e- 2    104.    949.
## 6 cutIdeal      576.     212.     2.71   6.77e- 3    159.    992.
## 7 colorE        -198.    107.    -1.84   6.54e- 2   -408.    12.6
## 8 colorF       -332.    111.    -2.99   2.90e- 3   -550.   -114.
## 9 colorG       -579.    106.    -5.44   6.57e- 8   -788.   -370.
## 10 colorH       -851.    112.    -7.60   6.88e- 14  -1071.   -631.
## 11 colorI      -1323.    130.   -10.1   4.64e- 23  -1578.  -1067.
## 12 colorJ      -2443.    156.   -15.6   2.79e- 49  -2750.  -2136.
## 13 claritySI2    1475.    295.     5.00   6.80e- 7    896.   2055.
## 14 claritySI1    2521.    294.     8.57   3.94e- 17   1944.   3099.
## 15 clarityVS2    3033.    295.     10.3   1.47e- 23   2453.   3612.
## 16 clarityVS1    3495.    299.     11.7   1.19e- 29   2908.   4081.
## 17 clarityVVS2   3693.    307.     12.0   3.27e- 31   3091.   4296.
## 18 clarityVVS1   3611.    315.     11.5   1.09e- 28   2994.   4229.
## 19 clarityIF      4386.    347.     12.6   5.54e- 34   3705.   5068.
## 20 depth         374.     126.     2.96   3.19e- 3     126.    622.
## 21 x             593.     924.     0.642  5.21e- 1   -1221.   2407.
## 22 y            2580.    981.     2.63   8.67e- 3     655.   4505.
## 23 z           -7830.   2107.    -3.72   2.14e- 4   -11966.  -3695.

```

마지막으로 이렇게 '표'로 보여주는 것이 아니라 가독성이 좋게 계수플롯(coef.plot)의 형태로 나타내 보자.

```

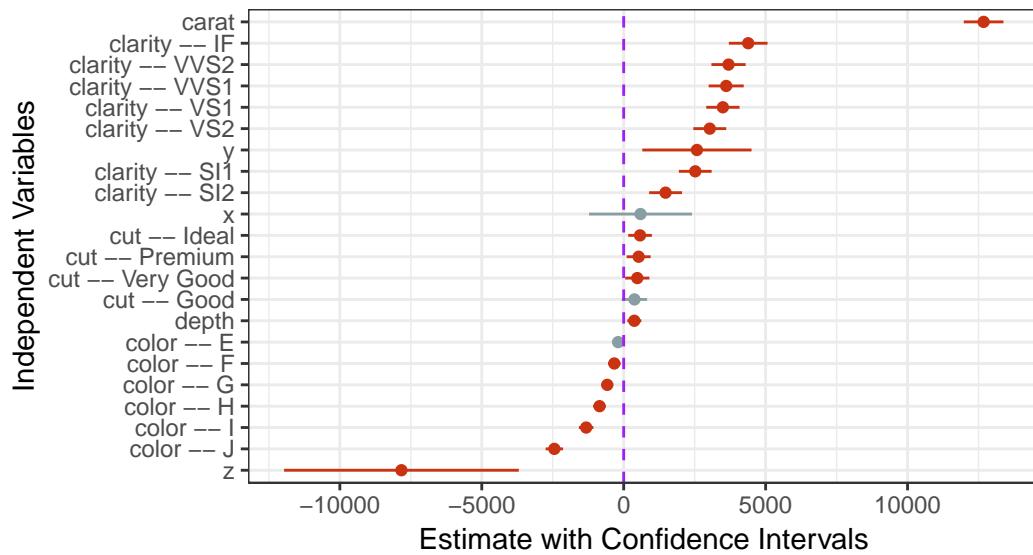
new_tidy_model %>% # 정리된 결과 티블을 활용한다.
  dplyr::filter(term != "(Intercept)") %>% # 절편, 상수항을 제외한 나머지를 대상으로 필터링
  mutate(                                # 변수를 새롭게 만들어라
    ## term 안의 문자열을 대체하라 (예: 'cut'에서 'cut -- '으로)
    term = str_replace(term, 'cut', 'cut -- '),
    term = str_replace(term, 'color', 'color -- '),
    term = str_replace(term, 'clarity', 'clarity -- '))
  ) %>%
  rename(variable = term) %>% # 그렇게 적용된 term이라는 변수의 이름을 variable로 바꾸어라
  ggplot(
    aes(
      x = reorder(variable, estimate), y = estimate, # x축에 변수, y축에 추정치를 놓는다.
      ## 나타나는 순서는 변수명, 추정치 순이다.
      color = p.value <= 0.05 # 유의수준이 0.05 이하인 경우와 아닌 경우의 색을 구분
    )
  ) +
  geom_point(show.legend = F) + # 점으로 나타내되 점의 범례는 보여주지 마라
  geom_linerange(aes(ymin = ll, ymax = ul), show.legend = F) + # 신뢰구간 그리기
  geom_hline(yintercept = 0, color = 'purple', linetype = 'dashed') + # 0의 수직선을 그리기

```

```

coord_flip() + # x축과 y축 그림에서 서로 바꾸기
labs(x = 'Independent Variables', y = 'Estimate with Confidence Intervals') +
scale_color_manual(values = wes_palette("Royal1")) +
theme_bw()

```



이 결과는 예측변수들이 가격과 어떠한 관계를 가지는지 다변량(multivariate) 관계를 바탕으로 보여주고 있다.