PREDICTIVE ANALYTICS

Oindrila Chakraborty

2026-02-01

Problem Set 2: Linear Regression.

1 Problem to demonstrate that the population regression line is fixed, but least square regression line varies.

Suppose the population regression line is given by Y = 2 + 3x, while the data comes from the model y = 2 + 3x + ϵ.

Step 1: For x in the range [5,10] graph the population regression line.

Step 2: Generate xi(i = 1, 2, .., n) from Uniform(5, 10) and ϵi(i = 1, 2, .., n) from N(0, 16). Hence, compute y1, y2, .., yn.

Step 3: On the basis of the data (xi, yi)(i = 1, 2, .., n) generated in Step 2, report the least squares regression line.

Step 4: Repeat steps 2-3 five times. Graph the 5 least squares regression lines over the population regression line obtained in Step 1. Interpret the findings.

Take n = 50. Set the seed as seed=123.

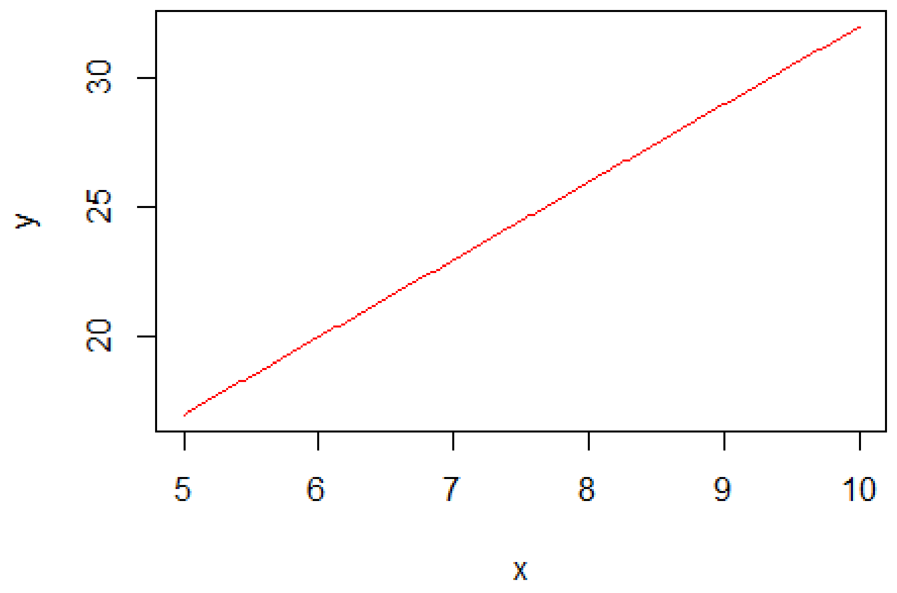
```
#Step 1.
x=seq(5,10,length.out=200)
x

##   [1]  5.000000  5.025126  5.050251  5.075377  5.100503  5.125628
5.150754
##   [8]  5.175879  5.201005  5.226131  5.251256  5.276382  5.301508
5.326633
##  [15]  5.351759  5.376884  5.402010  5.427136  5.452261  5.477387
5.502513
##  [22]  5.527638  5.552764  5.577889  5.603015  5.628141  5.653266
5.678392
##  [29]  5.703518  5.728643  5.753769  5.778894  5.804020  5.829146
5.854271
##  [36]  5.879397  5.904523  5.929648  5.954774  5.979899  6.005025
6.030151
##  [43]  6.055276  6.080402  6.105528  6.130653  6.155779  6.180905
6.206030
##  [50]  6.231156  6.256281  6.281407  6.306533  6.331658  6.356784
6.381910
##  [57]  6.407035  6.432161  6.457286  6.482412  6.507538  6.532663
6.557789
##  [64]  6.582915  6.608040  6.633166  6.658291  6.683417  6.708543
```

```
6.733668
##  [71]  6.758794  6.783920  6.809045  6.834171  6.859296  6.884422
6.909548
##  [78]  6.934673  6.959799  6.984925  7.010050  7.035176  7.060302
7.085427
##  [85]  7.110553  7.135678  7.160804  7.185930  7.211055  7.236181
7.261307
##  [92]  7.286432  7.311558  7.336683  7.361809  7.386935  7.412060
7.437186
##  [99]  7.462312  7.487437  7.512563  7.537688  7.562814  7.587940
7.613065
## [106]  7.638191  7.663317  7.688442  7.713568  7.738693  7.763819
7.788945
## [113]  7.814070  7.839196  7.864322  7.889447  7.914573  7.939698
7.964824
## [120]  7.989950  8.015075  8.040201  8.065327  8.090452  8.115578
8.140704
## [127]  8.165829  8.190955  8.216080  8.241206  8.266332  8.291457
8.316583
## [134]  8.341709  8.366834  8.391960  8.417085  8.442211  8.467337
8.492462
## [141]  8.517588  8.542714  8.567839  8.592965  8.618090  8.643216
8.668342
## [148]  8.693467  8.718593  8.743719  8.768844  8.793970  8.819095
8.844221
## [155]  8.869347  8.894472  8.919598  8.944724  8.969849  8.994975
9.020101
## [162]  9.045226  9.070352  9.095477  9.120603  9.145729  9.170854
9.195980
## [169]  9.221106  9.246231  9.271357  9.296482  9.321608  9.346734
9.371859
## [176]  9.396985  9.422111  9.447236  9.472362  9.497487  9.522613
9.547739
## [183]  9.572864  9.597990  9.623116  9.648241  9.673367  9.698492
9.723618
## [190]  9.748744  9.773869  9.798995  9.824121  9.849246  9.874372
9.899497
## [197]  9.924623  9.949749  9.974874 10.000000
```

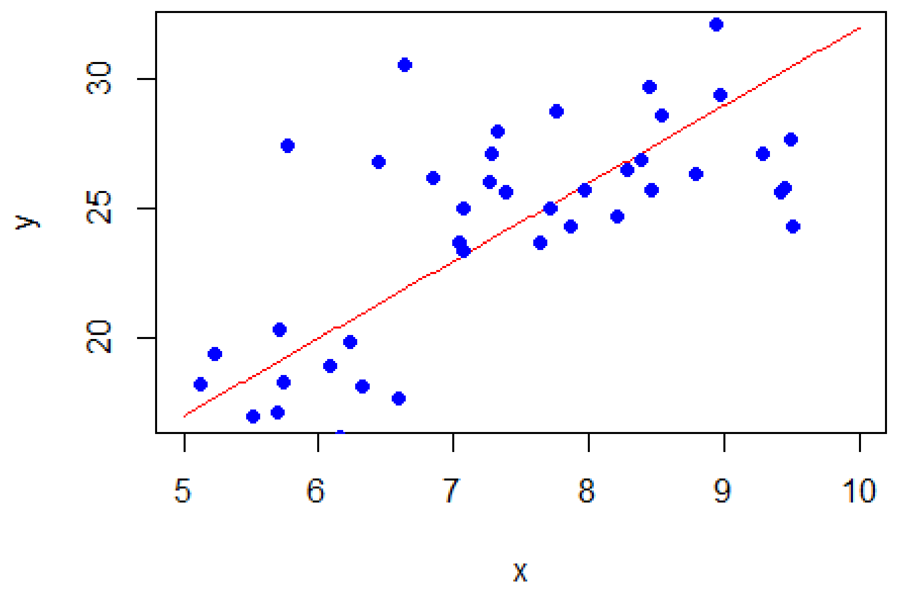
```
y=2+3*x
y
```

```
##   [1] 17.00000 17.07538 17.15075 17.22613 17.30151 17.37688 17.45226
17.52764
##   [9] 17.60302 17.67839 17.75377 17.82915 17.90452 17.97990 18.05528
18.13065
##  [17] 18.20603 18.28141 18.35678 18.43216 18.50754 18.58291 18.65829
18.73367
##  [25] 18.80905 18.88442 18.95980 19.03518 19.11055 19.18593 19.26131
19.33668
```

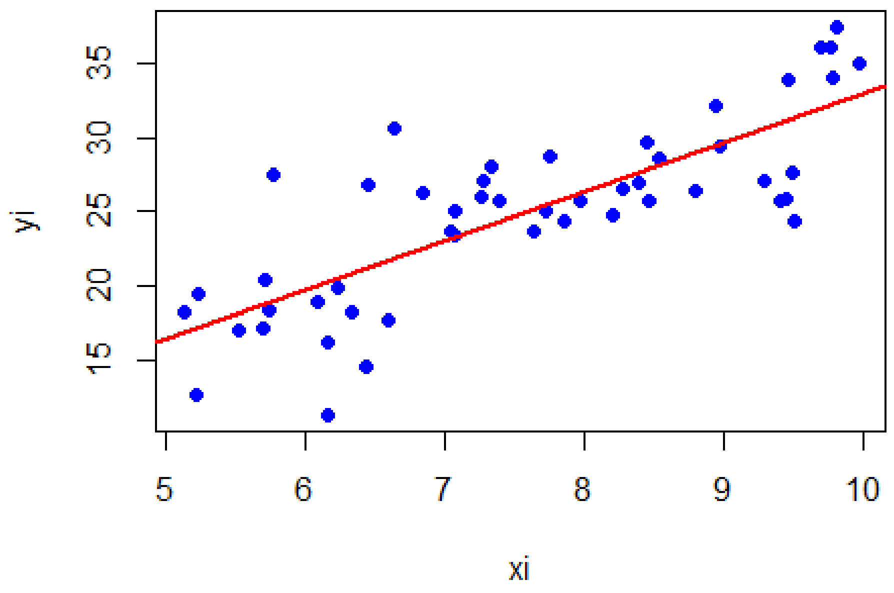
```
##  [33] 19.41206 19.48744 19.56281 19.63819 19.71357 19.78894 19.86432
19.93970
##  [41] 20.01508 20.09045 20.16583 20.24121 20.31658 20.39196 20.46734
20.54271
##  [49] 20.61809 20.69347 20.76884 20.84422 20.91960 20.99497 21.07035
21.14573
##  [57] 21.22111 21.29648 21.37186 21.44724 21.52261 21.59799 21.67337
21.74874
##  [65] 21.82412 21.89950 21.97487 22.05025 22.12563 22.20101 22.27638
22.35176
##  [73] 22.42714 22.50251 22.57789 22.65327 22.72864 22.80402 22.87940
22.95477
##  [81] 23.03015 23.10553 23.18090 23.25628 23.33166 23.40704 23.48241
23.55779
##  [89] 23.63317 23.70854 23.78392 23.85930 23.93467 24.01005 24.08543
24.16080
##  [97] 24.23618 24.31156 24.38693 24.46231 24.53769 24.61307 24.68844
24.76382
## [105] 24.83920 24.91457 24.98995 25.06533 25.14070 25.21608 25.29146
25.36683
## [113] 25.44221 25.51759 25.59296 25.66834 25.74372 25.81910 25.89447
25.96985
## [121] 26.04523 26.12060 26.19598 26.27136 26.34673 26.42211 26.49749
26.57286
## [129] 26.64824 26.72362 26.79899 26.87437 26.94975 27.02513 27.10050
27.17588
## [137] 27.25126 27.32663 27.40201 27.47739 27.55276 27.62814 27.70352
27.77889
## [145] 27.85427 27.92965 28.00503 28.08040 28.15578 28.23116 28.30653
28.38191
## [153] 28.45729 28.53266 28.60804 28.68342 28.75879 28.83417 28.90955
28.98492
## [161] 29.06030 29.13568 29.21106 29.28643 29.36181 29.43719 29.51256
29.58794
## [169] 29.66332 29.73869 29.81407 29.88945 29.96482 30.04020 30.11558
30.19095
## [177] 30.26633 30.34171 30.41709 30.49246 30.56784 30.64322 30.71859
30.79397
## [185] 30.86935 30.94472 31.02010 31.09548 31.17085 31.24623 31.32161
31.39698
## [193] 31.47236 31.54774 31.62312 31.69849 31.77387 31.84925 31.92462
32.00000

plot(x,y,type='l',col="red")
```



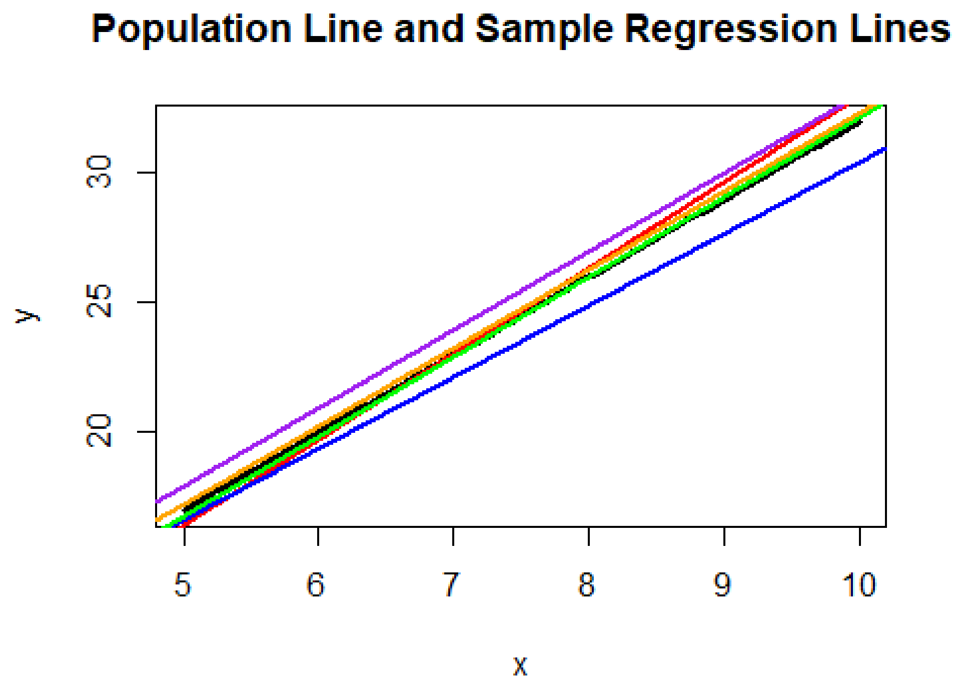
```
#Step 2.
set.seed(123)
xi=runif(50,5,10)
xi

##  [1] 6.437888 8.941526 7.044885 9.415087 9.702336 5.227782 7.640527
9.462095
##  [9] 7.757175 7.283074 9.784167 7.266671 8.387853 7.863167 5.514623
9.499125
## [17] 6.230439 5.210298 6.639604 9.772518 9.447697 8.464017 8.202534
9.971349
## [25] 8.278529 8.542652 7.720330 7.970710 6.445799 5.735568 9.815121
9.511495
## [33] 8.453526 8.977337 5.123068 7.388980 8.792298 6.082040 6.590905
6.158129
## [41] 5.714000 7.072732 7.068622 6.844227 5.762224 5.694030 6.165170
7.329812
## [49] 6.329863 9.289139

ei=rnorm(50,0,4)
ei

##  [1] -6.7467732  3.3511482  0.6134925 -4.5525477  5.0152597  1.7058569
##  [7] -1.1802859  3.5805026  3.5125340  3.2863243  2.7545610  2.2156706
## [13] -0.2476468 -1.2238507 -1.5218840 -2.7788279 -0.8316691 -5.0615854
## [19]  8.6758239  4.8318480 -4.4924343 -1.6115393 -1.8666214  3.1198605
## [25] -0.3334763  1.0132741 -0.1141870 -0.1714818  5.4744091 -0.9030839
## [31]  6.0658824 -6.1950112  2.3384550  0.4954170  0.8637663  1.5185579
## [37] -2.0092938 -1.3328295 -4.0743015 -4.2871649  1.2141146  1.7928391
## [43]  0.2120169  3.6890699  8.2003387 -1.9641247 -9.2366755  4.0229541
## [49] -2.8368031 -2.7520345

yi=2+3*xi+ei
yi

##  [1] 14.56689 32.17573 23.74815 25.69271 36.12227 19.38920 23.74130
33.96679
##  [9] 28.78406 27.13555 34.10706 26.01568 26.91591 24.36565 17.02199
27.71855
## [17] 19.85965 12.56931 30.59463 36.14940 25.85066 25.78051 24.74098
35.03391
## [25] 26.50211 28.64123 25.04680 25.74065 26.81181 18.30362 37.51125
24.33947
## [33] 29.69903 29.42743 18.23297 25.68550 26.36760 18.91329 17.69841
16.18722
## [41] 20.35611 25.01103 23.41788 26.22175 27.48701 17.11797 11.25884
28.01239
## [49] 18.15279 27.11538

plot(x,y,type='l',col="red")
points(xi, yi, col = "blue", pch = 16)
```



```
#Step 3.
model=lm(yi~xi)
model

##
## Call:
## lm(formula = yi ~ xi)
##
## Coefficients:
## (Intercept)           xi
##    -0.09639      3.30540

summary(model)

##
## Call:
## lm(formula = yi ~ xi)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.0231 -2.2314 -0.2627  2.1970  8.7445
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.09639    2.82610  -0.034    0.973
## xi           3.30540    0.36519   9.051 5.96e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.761 on 48 degrees of freedom
## Multiple R-squared:  0.6306, Adjusted R-squared:  0.6229
## F-statistic: 81.93 on 1 and 48 DF,  p-value: 5.962e-12

plot(xi, yi, col = "blue", pch = 16)
abline(model,col='red',lwd = 2)
```



```
coef(model)

## (Intercept)          xi
## -0.09638929  3.30539569

# Step 4.
set.seed(123)
n=50

# Population regression line
x_pop=seq(5, 10, length.out = 100)
y_pop=2 + 3 * x_pop

plot(x_pop, y_pop, type = "l", lwd = 3, col = "black",
     xlab = "x", ylab = "y",
     main = "Population Line and Sample Regression Lines")

beta_table=data.frame(
  Simulation = 1:5,
  Beta_0 = numeric(5),
  Beta_1 = numeric(5)
)

colors=c("red", "blue", "green", "purple", "orange")

for (i in 1:5) {
  x=runif(n, 5, 10)
  eps=rnorm(n, 0, 4)
  y=2 + 3 * x + eps

  fit=lm(y ~ x)

  abline(fit, col = colors[i], lwd = 2)

  beta_table$Beta_0[i] <- coef(fit)[1]
  beta_table$Beta_1[i] <- coef(fit)[2]
}
```



```
beta_table

##   Simulation       Beta_0   Beta_1
## 1          1 -0.09638929 3.305396
## 2          2  2.79218839 2.761042
## 3          3  1.39299737 3.073267
## 4          4  2.82308856 3.023608
## 5          5  2.03250638 3.028097
```

Interpretation:-

The population regression line (black) is fixed: Y=2+3X Each sample regression line differs slightly due to random error.This shows sampling variability in least squares estimation.

2.Problem to demonstrate that β ̂0 and β ̂ minimises RSS

Step 1: Generate xi from Uniform(5, 10) and mean centre the values. Generate εi from N(0, 1). Calculate yi = 2 + 3xi + εi, i = 1,2,.., n. Take n=50 and seed=123.

Step 2: Now imagine that you only have the data on (xi, yi), i = 1, 2, .., n, without knowing the mechanism that was used to generate the data in step 1. Assuming a linear regression of the type yi = β0 + βxi + εi,and based on these data (xi, yi), i = 1, 2,.., n, obtain the least squares estimates of β0 and β.

Step 3: Take a large number of grid values of (β0, β) that also include the least squares estimates obtained from step 2. Compute the RSS for each parametric choice of (β0, β), where RSS = (y1 − β0 − βx1)2 + (y2 − β0 − βx2)2 + ...(yn −β0 − βxn)2. Find out for which combination of (β0, β), RSS is minimum.

```
#step 1.
set.seed(123)
n=50
x=runif(n,5,10)
x=x-mean(x)
e=rnorm(n,0,1)
y=2+3*x+e
y
```

```
##  [1] -3.17439510  6.86099949  0.48666236  6.30575953  9.55945960 -
4.69155288
##  [7]  1.82514625  8.48004674  3.34829411  1.86943752  9.23977584
1.55256541
## [13]  4.30028323  2.48217378 -4.63796535  7.00130299 -2.31796585 -
6.43586794
## [19]  1.28640216  9.72415215  6.41861657  4.18780167  3.33958226
9.89264718
## [25]  3.95085333  5.07991095  2.33107903  3.06789526 -0.09536625 -
3.82043087
## [31] 10.16046950  6.18436828  5.14382834  6.25450092 -5.21621775
1.74521446
## [37]  5.07320502 -2.88845294 -2.04722486 -3.39876904 -3.35583561
0.86504023
## [43]  0.45750453  0.65358464 -1.46460869 -4.21030480 -4.61502197
2.19381069
## [49] -2.52097575  6.37804252
```

```
#Step 2:-
model1=lm(y~x)
coef(model1)
```

```
## (Intercept)           x
##    2.056189    3.076349
```

```
#Step 3:-
beta0_grid=seq(1,3,length = 100)
beta1_grid=seq(2,4,length = 100)

RSS=matrix(NA, 100, 100)

for (i in 1:100) {
  for (j in 1:100) {
    RSS[i, j]=sum((y-beta0_grid[i]-beta1_grid[j]*x)^2)
  }
}

min_RSS=which(RSS == min(RSS), arr.ind = TRUE)

beta0_grid[min_RSS[1]]

## [1] 2.050505

beta1_grid[min_RSS[2]]

## [1] 3.070707
```

Interpretation:- The minimum RSS occurs at the least squares estimates.Confirms that Ordinary least squares(OLS) estimators minimize RSS.

3.Problem to demonstrate that least square estimators are unbiased

Step 1: Generate xi(i = 1, 2, .., n) from Uniform(0, 1), εi(i = 1, 2, .., n) from N(0, 1) and hence generate y using yi = β0 + βxi + εi.(Take β0 = 2, β = 3).

Step 2: On the basis of the data (xi,yi)(i = 1, 2, .., n) generated in Step 1,btain the least square estimates of β0 and β.Repeat Steps 1-2, R = 1000 times. In each simulation obtain βˆ0 and βˆ.Finally,the least-square estimates will be given by the average of these estimated values.Compare these with the true β0 and β and comment.

Take n = 50 and seed=123.

```
#step 1:-
set.seed(123)
x_3=runif(50,0,1)
x_3

##  [1] 0.28757752 0.78830514 0.40897692 0.88301740 0.94046728 0.04555650
##  [7] 0.52810549 0.89241904 0.55143501 0.45661474 0.95683335 0.45333416
## [13] 0.67757064 0.57263340 0.10292468 0.89982497 0.24608773 0.04205953
## [19] 0.32792072 0.95450365 0.88953932 0.69280341 0.64050681 0.99426978
## [25] 0.65570580 0.70853047 0.54406602 0.59414202 0.28915974 0.14711365
## [31] 0.96302423 0.90229905 0.69070528 0.79546742 0.02461368 0.47779597
## [37] 0.75845954 0.21640794 0.31818101 0.23162579 0.14280002 0.41454634
## [43] 0.41372433 0.36884545 0.15244475 0.13880606 0.23303410 0.46596245
## [49] 0.26597264 0.85782772
```

```
e_3=rnorm(50,0,1)
e_3
```

```
##  [1] -1.68669331  0.83778704  0.15337312 -1.13813694  1.25381492
0.42646422
##  [7] -0.29507148  0.89512566  0.87813349  0.82158108  0.68864025
0.55391765
## [13] -0.06191171 -0.30596266 -0.38047100 -0.69470698 -0.20791728 -
1.26539635
## [19]  2.16895597  1.20796200 -1.12310858 -0.40288484 -0.46665535
0.77996512
## [25] -0.08336907  0.25331851 -0.02854676 -0.04287046  1.36860228 -
0.22577099
## [31]  1.51647060 -1.54875280  0.58461375  0.12385424  0.21594157
0.37963948
## [37] -0.50232345 -0.33320738 -1.01857538 -1.07179123  0.30352864
0.44820978
## [43]  0.05300423  0.92226747  2.05008469 -0.49103117 -2.30916888
1.00573852
## [49] -0.70920076 -0.68800862
```

```
y_3=2+3*x_3+e_3
y_3
```

```
##  [1] 1.1760392 5.2027025 3.3803039 3.5109153 6.0752168 2.5631337 3.2892450
##  [8] 5.5723828 4.5324385 4.1914253 5.5591403 3.9139201 3.9708002 3.4119375
## [15] 1.9283030 4.0047679 2.5303459 0.8607822 5.1527181 6.0714729 3.5455094
## [22] 3.6755254 3.4548651 5.7627744 3.8837483 4.3789099 3.6036513 3.7395556
## [29] 4.2360815 2.2155700 6.4055433 3.1581443 4.6567296 4.5102565 2.2897826
## [36] 3.8130274 3.7730552 2.3160164 1.9359676 1.6230861 2.7319287 3.6918488
## [43] 3.2941772 4.0288038 4.5074189 1.9253870 0.3899334 4.4036259 2.0887172
## [50] 3.8854745
```

```
model=lm(y_3~x_3)
model
```

```
## 
## Call:
## lm(formula = y_3 ~ x_3)
## 
## Coefficients:
## (Intercept)          x_3  
##       1.858        3.382
```

```
summary(model)
```

```
## 
## Call:
## lm(formula = y_3 ~ x_3)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.25578 -0.55786 -0.06567  0.54926  2.18613
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.8576     0.2721   6.827 1.36e-08 ***
## x_3           3.3817     0.4565   7.408 1.75e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9404 on 48 degrees of freedom
## Multiple R-squared:  0.5334, Adjusted R-squared:  0.5237
## F-statistic: 54.88 on 1 and 48 DF,  p-value: 1.745e-09
```

```
#Step 2:-
set.seed(123)
n=50
R=1000

beta0_hat=numeric(R)
beta1_hat=numeric(R)

for (i in 1:R) {
  x=runif(n,0,1)
  e=rnorm(n,0,1)
  y=2+3*x+e

  model2=lm(y~x)
  beta0_hat[i]=coef(model2)
  beta1_hat[i]=coef(model2)
}
```

```
## Warning in beta0_hat[i] <- coef(model2): number of items to replace is not
a
## multiple of replacement length

## Warning in beta1_hat[i] <- coef(model2): number of items to replace is not
a
## multiple of replacement length
```

```
mean(beta0_hat)
```

```
## [1] 2.013053
```

```
mean(beta1_hat)
```

```
## [1] 2.013053
```

Interpretation: -

Average of estimates are close to the true value.Hence, OLS estimators are unbiased.The simulation results show that the average of the estimated intercept ($\hat{\beta_0}$) and slope ($\hat{\beta_1}$) over repeated samples is very close to the true parameter values $\beta_0 = 2$ and $\beta_1 = 3$. This confirms that the least squares estimators are unbiased. Any deviation in individual samples is due to random error. As the number of simulations increases, the estimates converge to the true parameters.

4.Comparing several simple linear regressions

Attach “Boston” data from MASS library in R. Select median value of owner-occupied homes, as the response and per capita crime rate, nitrogen oxides concentration, proportion of blacks and percentage of lower status of the population as predictors.

(a) Selecting the predictors one by one, run four separate linear regressions to the data. Present the output in a single table.

(b) Which model gives the best fit?

(c) Compare the coefficients of the predictors from each model and comment on the usefulness of the predictors.

```
library(MASS)
data(Boston)
fit1=lm(medv~crim,data=Boston)
fit2=lm(medv~nox,data=Boston)
fit3=lm(medv~black, data=Boston)
fit4=lm(medv~lstat,data=Boston)

summary_table=data.frame(
  Predictor=c("crim", "nox", "black", "lstat"),
  Coefficient=c(coef(fit1), coef(fit2),coef(fit3),coef(fit4)),
  R_squared=c(summary(fit1)$r.squared,
              summary(fit2)$r.squared,
              summary(fit3)$r.squared,
              summary(fit4)$r.squared)
)

summary_table

##   Predictor  Coefficient R_squared
## 1      crim  24.03310617 0.1507805
## 2       nox  -0.41519028 0.1826030
## 3     black  41.34587447 0.1111961
## 4     lstat -33.91605501 0.5441463
## 5      crim  10.55103414 0.1507805
## 6       nox   0.03359306 0.1826030
## 7     black  34.55384088 0.1111961
## 8     lstat  -0.95004935 0.5441463
```

(b) Best fit:- The model with lstat has the highest R^2.Best predictor of median house value.

(c) Interpretation:-

crim: Negative impact, weak explanatory power **nox:** Strong negative relationship **black:** Positive but weak predictor **lstat:** Strongest negative effect and best fit

Conclusion:- Among the four predictors considered, the percentage of lower status population (lstat) provides the best fit, as it has the highest R^2 value. Crime rate (crim) and nitrogen oxides concentration (nox) show negative relationships with median house value, but with weaker explanatory power. The proportion of blacks (black) has a relatively weak effect. Overall, lstat is the most useful predictor of median house prices.