이대영 (2022311154)

2022-03-10

HW1 Report

1. $\bar{x} = \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)$

Using the dot product, the sample mean vector can be obtained by the code below:

```
In [5]: #sample mean (x_bar)
        x_bar=ones.transpose().dot(auto1)/len(auto1)
        x_bar
```

Out[5]:

	price	mpg	hroom	rseat	weight	length	gratio
ones	6192.283784	21.297297	2.986486	26.817568	3010.810811	188.067568	3.018108

$$S = \frac{1}{n-1}\left[\sum_{i=1}^{n}\left(x_i - \bar{x}\right)\left(x_i - \bar{x}\right)'\right]$$

By using 'x_bar' we create a new matrix 'auto2' which corresponds to $(x_i - \bar{x})$ in the formula. We use the dot product again to compute the sample covariance matrix by the code below:

```
In [10]: # sample covariance matrix (S)
         S=auto2.transpose().dot(auto2)/(len(auto1)-1)
         S
```

Out[10]:

	price	mpg	hroom	rseat	weight	length	gratio
price	8.632196e+06	-8086.619770	315.401148	3950.367549	1.270005e+06	28977.720289	-434.503565
mpg	-8.086620e+03	33.472047	-1.989078	-8.945020	-3.732025e+03	-103.239541	1.609200
hroom	3.154011e+02	-1.989078	0.705294	1.353665	3.280933e+02	9.672158	-0.140437
rseat	3.950368e+03	-8.945020	1.353665	9.777906	1.438712e+03	43.944002	-0.564117
weight	1.270005e+06	-3732.025176	328.093299	1438.711588	6.144925e+05	16777.615698	-268.835431
length	2.897772e+04	-103.239541	9.672158	43.944002	1.677762e+04	502.091262	-7.057953
gratio	-4.345036e+02	1.609200	-0.140437	-0.564117	-2.688354e+02	-7.057953	0.205394

$$R = [r_{ij}] \quad \text{where} \quad r_{ij} = \frac{\hat{\sigma}_{ij}}{\sqrt{\hat{\sigma}_{ii}\,\hat{\sigma}_{jj}}}$$

Since $\hat{\sigma}_{ij}$ corresponds to the elements of S, we can compute R by dividing each element of S by the square root of the diagonal elements. 'Si' and 'Sj' which contain elements corresponding to $\hat{\sigma}_{ii}$ and $\hat{\sigma}_{jj}$ in the formula respectively are used to compute the sample correlation matrix as in the code below:

In [14]:
```
#sample correlation matrix (R)
R=S/np.sqrt(Si)/np.sqrt(Sj)
R
```

Out[14]:

	price	mpg	hroom	rseat	weight	length	gratio
price	1.000000	-0.475735	0.127825	0.429986	0.551425	0.440162	-0.326317
mpg	-0.475735	1.000000	-0.409379	-0.494444	-0.822896	-0.796368	0.613727
hroom	0.127825	-0.409379	1.000000	0.515470	0.498372	0.513981	-0.368980
rseat	0.429986	-0.494444	0.515470	1.000000	0.586938	0.627170	-0.398064
weight	0.551425	-0.822896	0.498372	0.586938	1.000000	0.955170	-0.756719
length	0.440162	-0.796368	0.513981	0.627170	0.955170	1.000000	-0.695015
gratio	-0.326317	0.613727	-0.368980	-0.398064	-0.756719	-0.695015	1.000000

Each result yields values similar to those of the Python built-in functions.

2. We repeat the same steps in problem 1 to obtain the sample correlation matrix of Applicant data for variables 'LC', 'SC', 'SMS' (which we name 'applicant1_R').

Out[19]:

	LC	SC	SMS
LC	1.000000	0.807545	0.818021
SC	0.807545	1.000000	0.799631
SMS	0.818021	0.799631	1.000000

For our hypothesis test, we set our null and alternative hypothesis as

$H_0: \rho = 0$ vs $H_1: \rho \neq 0$ (ρ: population correlation)

If $r = Corr(x, y)$ and assuming x, y have a bivariate normal distribution, our test statistic can be computed by the following equation:

$$T = \frac{r\sqrt{n-2}}{\sqrt{1-r^2}} \sim t_{n-2}$$

Using this equation as well as the sample correlation matrix of Applicant data, we calculated the test statistics for variable pairs (LC, SC), (SC, SMS), and (LC, SMS), respectively. Then we obtained p-values for each test statistic. Part of the code is shown below:

In [21]:
```
#hypothesis test (LC,SC)
r1=applicant1_R.loc['LC','SC']
test_stat1=r1*np.sqrt(len(applicant1)-2)/np.sqrt(1-r1**2)
test_stat1
```

Out[21]: 9.286167554204495

In [22]:
```
pval1=stats.t.sf(abs(test_stat1), df=len(applicant1)-2)*2
pval1 #reject H0
```

Out[22]: 4.0639656387803185e-12

We obtained the following results:

	test statistic	p-value
(LC, SC)	9.2861	4.0639×10^{-12}
(SC, SMS)	9.0315	9.3688×10^{-12}
(LC, SMS)	9.6456	1.2659×10^{-12}

Since each p-value is smaller than 0.05, for each case we can reject the null hypothesis. Thus, at a 0.05 level of significance, the correlations between the variable pairs (LC, SC), (SC, SMS), and (LC, SMS), respectively are significant.

3, Let us find confidence intervals at significance level $\alpha = 0.05$. Thus, we use the critical value $z = 1.96$ for our calculations.

(a) Fisher's method

$$\rho = \tanh\left(\text{invtanh}(r) \pm z_{\alpha/2} / \sqrt{n-3}\right)$$

```
In [27]: #3
         #Let us find confidence intervals for population correlation at significance level a=0.05
         #Thus we use the critical value z=1.96 for our calculations
         #(a) Fisher's Method
         #95% confidence interval for (LC,SC)
         LB=np.tanh(np.arctanh(r1))-1.96/np.sqrt(len(applicant1)-3)
         UB=np.tanh(np.arctanh(r1))+1.96/np.sqrt(len(applicant1)-3)
         Fisher_CI1=[LB,UB]
         Fisher_CI1

Out[27]: [0.5153654674900736, 1.0997245656100185]
```

The 95% confidence intervals obtained for the variable pairs (LC, SC), (SC, SMS), and (LC, SMS) are

[0.5153, 1.0997),

[0.5094, 1.0918],

[0.5258, 1.1102], respectively.

(b) Ruben's method

Let $u = z_{\alpha/2} = 1.96$ ($\because \alpha = 0.05$) and $r^* = \dfrac{r}{\sqrt{1-r^2}}$

Let $a = 2n - 3 - u^2$

$b = r^* \times \sqrt{(2n-3)(2n-5)}$

$c = (2n - 5 - u^2) \times r^{*2} - 2u^2$

Set $ay^2 - 2by + c = 0$

The roots of the equation are

$$y_1 = \frac{2b - \sqrt{4b^2 - 4ac}}{2a}, \quad y_2 = \frac{2b + \sqrt{4b^2 - 4ac}}{2a}$$

$$\rho = \left[\frac{y_1}{\sqrt{1 + y_1^2}}, \frac{y_2}{\sqrt{1 + y_2^2}} \right]$$

We defined a function that returns the confidence interval obtained by Ruben's method.

In [30]:
```
#(b) Ruben's Method
def Ruben(n,u,r):
    r_star=r/np.sqrt(1-r**2)
    a=2*n-3-u**2
    b=r_star*np.sqrt((2*n-3)*(2*n-5))
    c=(2*n-5-u**2)*r_star**2-2*u**2

    y1=(2*b-np.sqrt(4*b**2-4*a*c))/(2*a)
    y2=(2*b+np.sqrt(4*b**2-4*a*c))/(2*a)

    LB=y1/np.sqrt(1+y1**2)
    UB=y2/np.sqrt(1+y2**2)
    CI=[LB,UB]

    return CI
```

The 95% confidence intervals obtained for the variable pairs (LC, SC), (SC, SMS), and (LC, SMS) are

[0.6743, 0.8861],

[0.6621, 0.8812],

[0.6907, 0.8925], respectively.

We can observe that the confidence intervals obtained using Ruben's method have a smaller size. Ruben's method is more accurate than Fisher's method.

4. First, we use the mean substitution method for missing values in all variables except for 'rep78'.

In [35]:
```
mean_sub=auto.fillna(value=auto3.mean())
mean_sub
```

Out[35]:

	model	origin	price	mpg	rep78	rep77	hroom	rseat	trunk	weight	length	turn	displa	gratio
0	AMC CONCORD	A	4099	22	3.0	2.00000	2.5	27.5	11	2930	186	40	121	3.58
1	AMC PACER	A	4749	17	3.0	1.00000	3.0	25.5	11	3350	173	40	258	2.53
2	AMC SPIRIT	A	3799	22	NaN	3.19697	3.0	18.5	12	2640	168	35	121	3.08
3	AUDI 5000	E	9690	17	5.0	2.00000	3.0	27.0	15	2830	189	37	131	3.20
4	AUDI FOX	E	6295	23	3.0	3.00000	2.5	28.0	11	2070	174	36	97	3.70
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Then, we use the regression method with 'rep78' as our response variable and other numeric variables as our predictor variables.

```
In [36]: import statsmodels.formula.api as sm
```

```
In [37]: #Use regression method
         model = sm.ols(formula="rep78 ~ price + mpg + rep77 + hroom + rseat + trunk + weight + length + turn + displa + gratio",
         model.params
```

```
Out[37]: Intercept    5.115108
         price        0.000043
         mpg         -0.006684
         rep77        0.629205
         hroom        0.023721
         rseat        0.010905
         trunk        0.041055
         weight       0.000259
         length      -0.010368
         turn        -0.073785
         displa      -0.003038
         gratio      -0.020136
         dtype: float64
```

By using the rows of data with a missing value for 'rep78' after mean substitution, we can predict new values for 'rep78' using our regression model.

```
In [39]: prediction=pd.DataFrame(model.predict(missing), columns=['rep78'])
         prediction
```

Out[39]:

	rep78
2	3.838687
9	3.213037
52	4.106392
56	3.365051
62	2.946001

After filling in our missing values with predicted values, we compared the mean and standard deviation of 'rep78' variable before and after imputation. The computed values are shown below:

```
In [41]: #Mean and standard deviation before imputation
         print(auto['rep78'].mean(),auto['rep78'].std())
```

```
3.4057971014492754 0.989932270109041
```

```
In [42]: #Mean and standard deviation after imputation
         print(regression['rep78'].mean(),regression['rep78'].std())
```

```
3.411745510352649 0.9620434775947883
```

After imputation, the mean slightly increased, whereas, the standard deviation slightly decreased.