Quin és el compilador més eficient pels problemes del Jutge, G++ o Clang?

RESUM:

Objectiu del treball: comparar G++ amb Clang per saber si aquest últim seria una alternativa més eficient que el primer per a utilitzar al grau.

Obtenció de dades: hem obtingut els programes generant 30 nombres aleatoris i seleccionant el programa corresponent a cada número del Jutge.

Resultats: Després de realitzar l'Interval de Confiança per saber quin compilador és millor (amb un 95% de confiança), hem obtingut el següent resultat: [0.858, 0.921]. Això vol dir que G++ tarda entre 0.858 i 0.921 vegades el que triga Clang. A més, hem aproximat les dades a un model lineal múltiple, i hem fet una predicció basada en aquest model, observant que és bastant semblant a les dades que vam recollir.

Discussió: encara que hem obtingut resultats clars, s'ha de comentar que en cap dels dos resultats d'abans es complien les premisses de validació, fet que ha estat un fort impediment per considerar els resultats del tot certs.

INTRODUCCIÓ:

Motivació: Un dels aspectes més importants que treballem al nostre grau universitari és l'eficiència. Ens han ensenyat que és igual d'important que un programa funcioni com que ho faci de forma eficient. Per aquest motiu, ens hem preguntat si el compilador amb el qual treballem actualment[1] és el millor en termes d'eficiència o si hi hauria una alternativa millor. Per això, hem decidit comparar G++ amb Clang, un compilador de C++ que també està inclòs al Jutge, però que mai hem utilitzat.

[1] G++ versió 7.5.0.

Objectius: Calcular els temps de compilació d'una mostra aleatòria de programes (que ja hem programat i que estan guardats al Jutge), compilats amb G++ i amb Clang, comparar els resultats i determinar quina de les dues opcions és millor en termes d'eficiència.

Hipòtesis: La nostra hipòtesi és que G++ és més eficient que Clang, per tant, aquest últim no és una alternativa millor per a utilitzar al nostre grau. Això significa que per saber si G++ és el millor compilador, s'hauria de buscar una altra opció que no fos Clang.

MÈTODES:

Com hem recollit les dades: Per a l'estudi hem utilitzat un ordinador amb sistema operatiu Ubuntu 22.04.3 LTS, amb un processador AMD Ryzen 5 5600 Hz i una RAM de 16 GB. Primer de tot, vam seleccionar trenta problemes del Jutge[2], que és la pàgina web on estan tots els problemes que hem programat fins ara. Per tal que fos totalment aleatori, vam assignar un número de forma ascendent a cada programa, començant pel número 1 i acabant en el número 450 (que correspon a l'últim programa), després, vam demanar a una IA[3] que generés trenta números no repetits que pertanyessin a l'interval [1,450], i així vam obtenir els programes que formarien part de la nostra mostra.

A l'hora d'obtenir el temps de compilació de cada programa, vam executar deu vegades la comanda de compilació juntament amb la comanda time (*p. ex: time G++ -o programa.exe programa.cc*), i després vam fer la mitjana d'aquestes deu compilacions, obtenint així el temps de compilació del programa (en ms) garantint que fos el més acurat possible. Aquest procés el vam repetir dues vegades per programa, una amb G++ i l'altre amb Clang.

A més, vam recollir el nombre de línies del codi i les llibreries incloses en el mateix.

[2] Jutge: https://jutge.org.
[3] ChatGPT: https://chat.openai.com/.

Mètodes emprats per l'anàlisi estadística: Per tal de dur a terme l'anàlisi estadística, hem utilitzat dos mètodes: l'obtenció d'un IC del 95% a partir de la mitjana de la diferència (per tal de veure quin compilador és millor) i el càlcul d'una predicció de totes les observacions a partir d'un model estadístic ajustat a les dades recollides.

Per fer l'IC del 95%, primer vam decidir quin IC calcularíem. Com el nostre estudi es basa en dades aparellades, vam decidir fer l'IC de la mitjana de la diferència de les nostres variables resposta. Per fer això, primer vam calcular la diferència de les variables i vam comprovar que complís les premisses de validació: aleatorietat de la mostra i normalitat. La premissa d'aleatorietat ja la vam poder afirmar gràcies al mètode utilitzat per recollir les dades. La normalitat la vam avaluar amb les instruccions qqline i qqnorm de la diferència de respostes. Quan les vam validar sobre la diferència, vam observar que no seguia una distribució normal. Per tal d'intentar arreglar-ho, vam aplicar una transformació logarítmica (fent la diferència del logaritme dels temps de G++ i del logaritme del temps de Clang), fet que va millorar una mica les premisses, però no va ser suficient perquè les validés completament. Encara això, vam seguir calculant l'IC amb la variable transformada. En obtenir el resultat, vam desfer la transformació i ens va sortir un interval que, en comptes d'indicar la diferència dels temps, indicava una ràtio entre el temps de G++ i el de Clang[4]. Per tant, si dins el nostre interval un extrem era major que 1 i l'altre menor que 1, això volia dir que no podíem afirmar que un compilador fos més ràpid que l'altre. Si els dos extrems de l'interval eren menors a 1 volia dir que trigava menys G++, i si els dos extrems eren majors a 1 Clang era més ràpid, tot això amb un 95% de confiança.

A l'hora de fer la predicció d'acord amb un model estadístic, primer vam haver de decidir quin era el millor model per aproximar les nostres dades. D'acord amb les nostres variables, el model adequat era el model lineal múltiple, ja que calculàvem el temps de compilació de G++ segons el temps de compilació en Clang, les línies de codi i les llibreries. El model aproximat va ser el següent:

- $Y_{G++} = \beta_0 + \beta_1 \cdot Y_{Clang} + \beta_2 \cdot LINIES\_CODI + \beta_3 \cdot LLIBRERIES\_CODI$

[4] En fer log(temps_g++) - log(temps_clang) equival a log(temps_g++/temps_clang) i, en desfer el logaritme fent l'exponencial ens queda temps_g++ / temps_clang.

on β_0 indicava el temps base de compilació de G++ (en cas que els altres coeficients fossin 0, fet impossible) i β_x indicava el coeficient amb el qual es veia afectat el temps de compilació de G++ per cada categoria.

Per saber si el model era vàlid per fer la predicció, vam haver de validar les següents premisses: linealitat, mostra aleatòria, normalitat dels residus i homoscedasticitat dels residus.

Per validar la linealitat i l'homoscedasticitat dels residus, vam fer un gràfic on apareixia els residus enfront de les prediccions del model i vam observar si els punts del gràfic seguien la línia de la predicció, i si ho feien amb una distància més o menys igual per a tots els punts. Si es complia la primera observació, es validava la linealitat i si es complia la segona, es validava l'homoscedasticitat. La normalitat la vam validar mitjançant les mateixes instruccions (qqline i qqnorm) que vam usar per a l'IC de la diferència. Finalment, l'aleatorietat de les dades la vam comprovar comparant els residus amb l'ordre de recollida. Si s'observava un patró a mesura que augmentava l'ordre de recollida, la mostra no era aleatòria, i si no s'observava cap patró, sí que era aleatòria.

Un cop vam intentar validar les respostes, vam observar que el model al qual havíem ajustat les dades només complia l'aleatorietat de la mostra. Com al mètode anterior, vam aplicar una transformació logarítmica al temps de G++ i al de Clang. En comprovar les premisses, tampoc es complien, però s'ajustaven una mica més. Per tant, vam fer la predicció per a cada observació en funció del model amb la transformació, i després de desfer-la, vam obtenir les prediccions dels temps per a cada programa que havíem compilat segons el model obtingut.

RESULTATS:

Descriptiva de les dades:

- Numèrica: La nostra variable d'interès (Y) és numèrica contínua, ja que pot ser qualsevol valor numèric dins d'un rang, per tant, observarem la tendència central (amb la mitjana i la mediana), la dispersió mitjançant la desviació estàndard i l'IQR[5]. Els resultats obtinguts són:

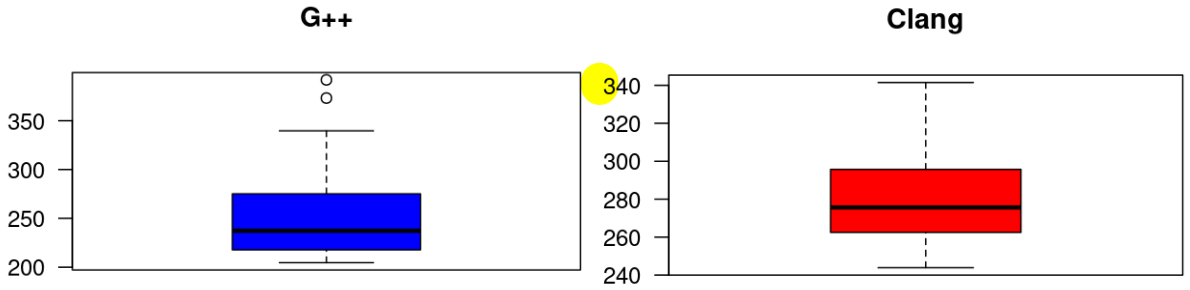
[5] L'IQR és el Rang Interquartílic, que és la diferència entre el primer i tercer quartil. És una mesura de la dispersió estadística.

	Mitjana	Mediana	Desviació std.	IQR
G++	254.607	237.250	49.111	59.375
Clang	283.110	275.650	26.720	36.025

*Totes les dades estan en ms

Observem que la mitjana del temps de compilació en G++ és menor que amb Clang. Veiem també que la mediana de G++ és bastant més baixa que la seva mitjana. Això vol dir que generalment els temps de compilació en G++ seran encara més baixos que la mitjana calculada. Amb Clang també passa, però la diferència és menor. La desviació estàndard és menor per Clang, això vol dir que els temps que trigui un programa a compilar amb aquest compilador seran més propers a la mitjana que en el cas de G++. L'IQR representa la diferència o distància entre el tercer quartil i el primer quartil, cosa que ens indica com estan de dispersos els valors. Com podem veure, l'IQR de Clang és menor, indicant que hi ha menys variància entre els quartils. Això té sentit, ja que, com hem comentat, la seva desviació estàndard també és menor que en G++.

- Gràfica: Com hem dit, la nostra variable d'interès és numèrica contínua, per tant, farem la descriptiva gràfica de les dades amb un boxplot, per observar els indicadors robustos (mediana i IQR), i un histograma), per veure la distribució corresponent. Els gràfics observats són[6]:



A la gràfica de G++ podem veure la mediana en aproximadament 240 ms. La majoria de valors inferiors a la mediana es concentren per sobre dels 220 ms. D'altra banda, la majoria de valors per sobre la mediana són inferiors a 280 ms. Per últim, trobem dos valors atípics. En Clang veiem la mediana en 275 ms. Els valors inferiors es concentren per sobre els 260 ms i els superiors per sota el 295 ms.

[6] L'histograma es pot consultar a l'Annex 1.1.

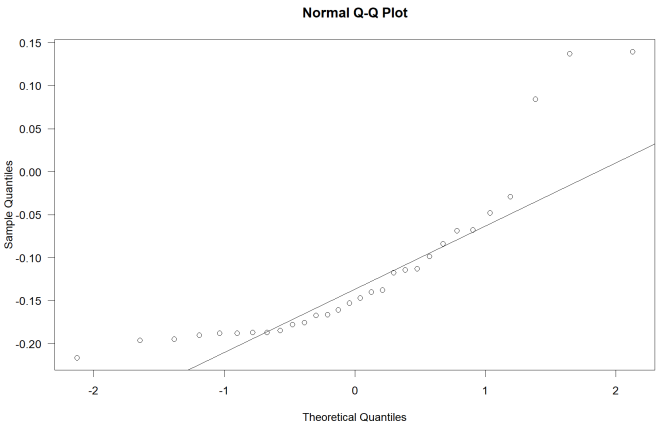
En Clang no podem considerar cap valor atípic. Comparant tots dos podem veure que tot i que els quartils de G++ estan en un temps inferior a Clang, els valors obtinguts amb Clang són més propers a la mediana que els obtinguts en G++.

Resultats bloc C:

- Validació premisses: Les dues premisses a comprovar són que la mostra sigui aleatòria i la normalitat de les dades. La primera premissa és vàlida, ja que els programes compilats amb C++ i Clang han estat escollits amb nombres aleatoris. Per la premissa de normalitat, hem fet qqnorm i qqline de la diferència de les variables, però el resultat no s'ajusta a una normal[7].

Per aquest motiu, hem aplicat una transformació logarítmica a les dades, hem tornat a calcular la diferència (amb els logaritmes) i hem replicat el qqnorm i qqline.

El resultat tampoc s'ajusta a una distribució normal, però ho fa una mica més, per això, farem l'estudi amb la transformació.



- IC de la diferència de mitjanes: Per calcular l'interval de confiança del 95% vam utilitzar les següents comandes al R:

```
#IC 95% de la diferencia de mitjanes amb log
diferencia_log <- log(dades$`TEMPS_G`) - log(dades$`TEMPS_CLANG`)
mitj_dif_log <- mean(diferencia_log)
n <- length(diferencia_log)
t <- qt(0.975,n-1)
se_log <- sd(diferencia_log) / sqrt(n)
IC <- c(mitj_dif_log - t*se_log, mitj_dif_log + t*se_log)
IC
exp(IC)
```

Com hem comentat a l'apartat de mètodes, a l'hora de desfer la transformació logarítmica, l'IC resultant correspon a una ràtio entre G++ i Clang. L'interval de confiança del 95% que hem obtingut és [0.858, 0.921].

[7] Es pot consultar a l'Annex 1.2.

- Interpretació: Hem vist que les nostres dades no segueixen una distribució normal, i tot i que després d'aplicar logaritmes es normalitzen una mica seguim sense poder afirmar que es compleix la premissa d'aquest apartat. Si interpretem les dades obtingudes veiem que amb un 95% de confiança podem dir que G++ serà sempre més ràpid que Clang. Això és perquè els dos extrems de l'interval de confiança ens ha quedat més petits que 1. A més a més, com que aquest interval ens indica una ràtio, ens permet afirmar, amb un 95% de confiança, que el temps que triga un programa a compilar en G++ serà entre 0.858 i 0.921 vegades el temps que triga a compilar en Clang de mitjana.

Resultats Bloc D:

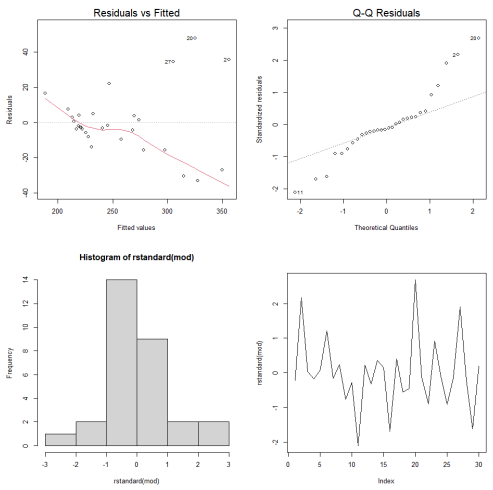
- Model ajustat i estimacions del model: La variable resposta del temps de compilació de G++ s'ajusta a un model lineal múltiple amb les variables de temps de compilació de Clang, les línies de codi del programa i les llibreries que conté, com hem mencionat a l'apartat de mètodes.

En fer el summary del model obtenim les següents estimacions[8]:

	Estimació	**Error estàndard**
Intercept	-245.114	59.787
Clang	1.786	0.256
Línies de Codi	-0.239	0.278
Llibreries	0.629	7.615

Amb $R^2 = 0.867$

- Validació premisses: Les quatre premisses a validar en aquest model són: linealitat, mostra aleatòria, normalitat dels residus i homoscedasticitat (variabilitat homogènia) dels residus.



La linealitat no podem validar-la, ja que els punts de la primera gràfica no s'ajusten a un pla o hiperplà, en canvi, sí que podem afirmar la premissa de mostra aleatòria, ja que a la

[8] Es pot consultar a l'Annex 2.1.

quarta gràfica no es pot veure cap patró o tendència de les dades, cosa que implica la independència entre observacions. La premissa de normalitat no podem validar-la, ja que els punts de la segona gràfica no s'ajusten a la recta i, per tant, els residus no segueixen una distribució normal. Finalment, l'homoscedasticitat tampoc la podem afirmar, ja que com es pot veure a la primera gràfica, al principi hi ha molta concentració de punts i després estan més dispersos.

- Explicació de les transformacions: Com que el nostre model no ha validat les premisses, hem hagut d'aplicar una transformació logarítmica per tal de millorar-ho. Els logaritmes els hem aplicat a les variables de temps de G++ i temps de Clang. Ho hem fet de la següent manera:

```
g_log <- log(dades$TEMPS_G)
clang_log <- log(dades$TEMPS_CLANG)
mod_log <- lm(g_log ~ clang_log + LINIES_CODI + LLIBRERIES_CODI, data = dades)
```

En validar les premisses d'aquest nou model[9], veiem que la situació no millora molt respecte al model sense logaritmes, ja que no es compleixen cap de les premisses menys l'aleatorietat. Encara això, com s'ajusta millor, hem decidit seguir amb el model amb la transformació logarítmica.

- Interpretació: com hem explicat abans, el nostre model lineal múltiple segueix la següent estimació:

- $Y_{G++} = \beta_0 + \beta_1 \cdot Y_{Clang} + \beta_2 \cdot LINIES\_CODI + \beta_3 \cdot LLIBRERIES\_CODI$

Per calcular els coeficients de cada variable, utilitzarem la funció summary del R, que ens indica quins són aquests coeficients a l'apartat *Estimate*. Aquesta és la sortida del summary[10]:

	Estimació	**Error estàndard**
Intercept	-4.771	1.323
Clang	1.825	0.242
Línies de Codi	-0.001	0.001
Llibreries	0.010	0.024

Amb $R^2 = 0.895$

[9] Es pot consultar a l'Annex 2.2.
[10] Aquest summary és del model després de fer la transformació logarítmica. Es pot consultar a l'Annex 3.1

Per tant, segons el model que hem ajustat, per tal d'obtenir el temps de compilació (en logaritme) del programa en G++ hauríem d'aplicar la següent fórmula:

- $Y_{G++} = -4.771 + 1.825 \cdot T\_CLANG - 0.001 \cdot LIN\_CODI + 0.010 \cdot LLIBR\_CODI$

La fórmula ens indica que el temps base de compilació d'un programa de G++ segons el temps de compilació en Clang, les línies de codi i les llibreries seria de -4.771 ms. Aquest cas és impossible, ja que un codi mai tindrà 0 línies, 0 llibreries i tardarà 0 ms a compilar amb Clang. Encara això, és normal que aquest temps base sigui negatiu, perquè hem vist que G++ tarda menys que Clang a compilar, per tant, si tarda 0 ms a compilar en Clang, ha de tardar menys de 0 ms en G++. Pels altres coeficients, es pot interpretar que el temps de compilació del programa en G++ disminuirà en 0.001 ms per cada línia de codi i que, en canvi, augmentarà en 0.010 ms per cada llibreria i en 1.825 ms per cada ms que tardi a compilar en Clang.
Aquests coeficients poden ser una mica contradictoris (sobretot en el cas de les línies), ja que sona estrany que el temps disminueixi per cada línia de codi, però això pot estar a causa que el model no valida les premisses, o a què hi ha una part del model que no està explicat per aquestes variables. Això ho podem saber gràcies al R^2, que és un coeficient del model que determina la ràtio entre la variabilitat explicada pel model i la total. Com més gran sigui, millor representa el model la relació entre les variables. En aquest cas, el R^2 equival a 0.895, per tant, hi ha quasi un 10% de la variable que no està explicat per aquest motiu.

- Ús per fer predicció: Una vegada ja tenim ajustat el model amb la transformació i hem validat les premisses (tot i que no es compleixen, continuem amb l'estudi amb aquest model) hem realitzat una predicció sobre el model per així estimar els temps de compilació de la mostra segons el model escollit. Per fer-ho utilitzem la següent comanda a R:

```
#Predicció
g_log <- log(dades$TEMPS_G)
clang_log <- log(dades$TEMPS_CLANG)
mod_log <- lm(g_log ~ clang_log + LINIES_CODI + LLIBRERIES_CODI, data = dades)
exp(predict(mod_log))
```

Per desfer la transformació logarítmica, hem fet l'exponencial de la funció predict, per així obtenir l'aproximació del temps (en ms) de cada observació segons el model que hem aproximat.
Quan executem les instruccions, el resultat és el següent:

```
> exp(predict(mod_log))
       1        2        3        4        5        6        7        8        9       10       11
216.7229 355.6464 215.1596 239.8757 270.4819 244.9282 220.7724 221.3622 229.1996 266.1270 323.5499
      12       13       14       15       16       17       18       19       20       21       22
220.8486 224.5980 234.2464 213.9476 312.9143 210.6224 257.5500 228.4544 323.4662 218.5014 295.5590
      23       24       25       26       27       28       29       30
193.0920 245.0731 276.7242 220.1680 303.0367 221.6926 351.6843 266.9624
```

Comparant la sortida del R amb les dades recollides inicialment podem observar que, en general, gairebé tots els valors són semblants a les dades, cosa que ens indica que és una bona predicció i, conseqüentment, el model escollit és correcte. Encara això, com el model no és 100% determinista (el R quadrat no és 1), hi ha valors que difereixen.

DISCUSSIÓ:

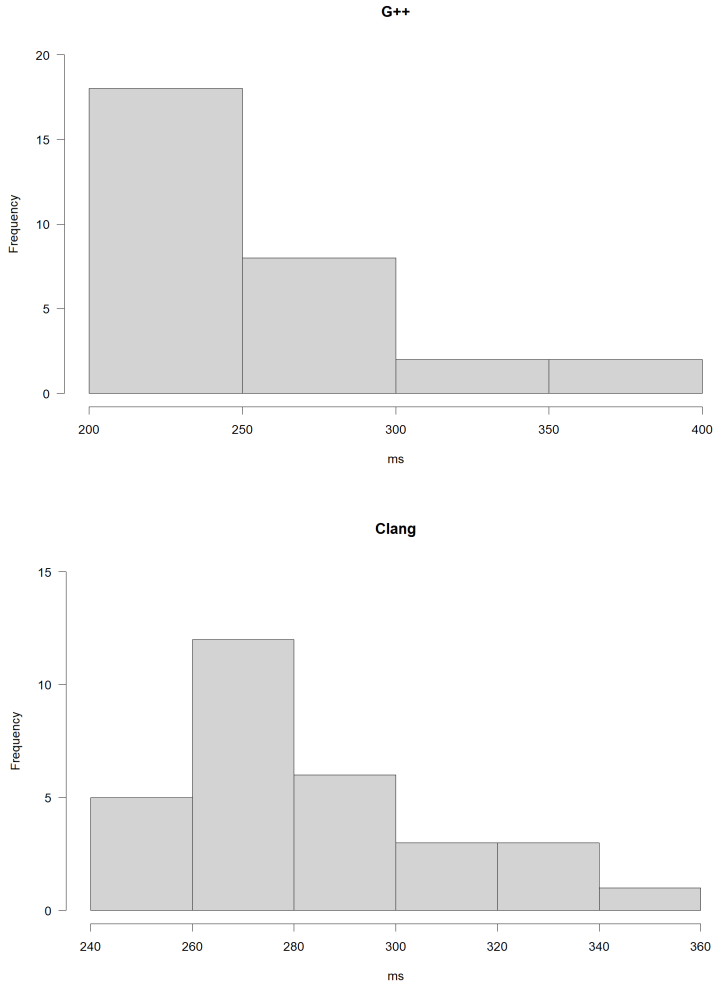
- Interpretació dels resultats: Segons els resultats obtinguts, podem afirmar que per a tota la mostra estudiada es compleix la que era la nostra hipòtesi inicial, que G++ és més eficient que Clang. Així ho demostra l'IC obtingut, on indica que la ràtio entre G++ i Clang és menor que 1 amb un 95% de confiança. A més, com el valor 1 no està dins de l'interval, podem descartar definitivament que G++ i Clang triguin el mateix. El model aproximat també corrobora la nostra hipòtesi, ja que si ens fixem, el coeficient base del model és negatiu, el que vol dir que el temps de G++ sempre serà menor al temps de Clang. A més, la predicció feta amb aquest model és molt semblant a la recollida de dades, fet que demostra que el model s'aproxima bastant a la realitat (encara que no al 100%).

- Limitacions i aspectes a millorar de l'estudi: Una de les principals limitacions de l'estudi ha estat que les dades recollides no segueixen una distribució normal. A més, aplicant logaritmes tampoc millorava. Això ha provocat que no puguem afirmar amb certesa la correctesa de l'interval de confiança o del model al qual hem aproximat les dades, ja que per fer l'IC o per aplicar el model corresponent cal afirmar que les dades s'ajustin a una distribució normal, a més d'altres premisses.
Un altre aspecte a tenir en compte és que inicialment només teníem una variable Z que mesurava les línies de codi del programa que estàvem compilant. No obstant això, recollint les dades ens vam adonar que hi havia altre factor que afectava el temps de compilació encara més que les línies del programa: les llibreries que inclou. Per solucionar aquest problema, vam decidir afegir un altre variable Z_2 que mesurés les llibreries del codi.

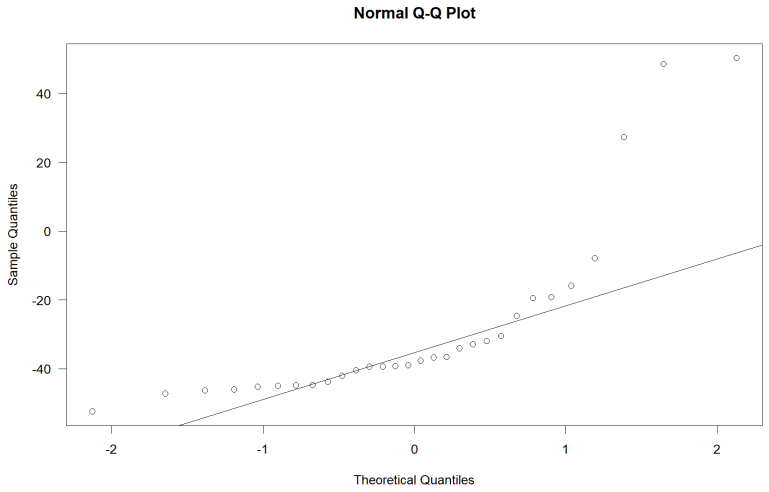
Tot i això, també vam veure que no totes les llibreries incloses afectaven de la mateixa manera, ja que diferents llibreries provocaven augments bastant diferents en el temps total.

- Generabilitat: Cal destacar que la mostra aleatòria ha estat obtinguda de la població de problemes que hem fet fins ara al grau, és a dir, els problemes de PRO1, PRO2 i EDA. Això significa que aquest estudi no és extrapolable a la població del 100% dels problemes del jutge, ja que no es tenen en compte problemes de cursos més avançats. Encara això, el nostre estudi és extrapolable a tots els problemes de les assignatures d'aquestes assignatures, que és la població a la qual hem dirigit el nostre estudi.

ANNEX:



Annex 1.1: Histogrames



Annex 1.2: Validació de la normalitat de la diferència sense transformació logarítmica

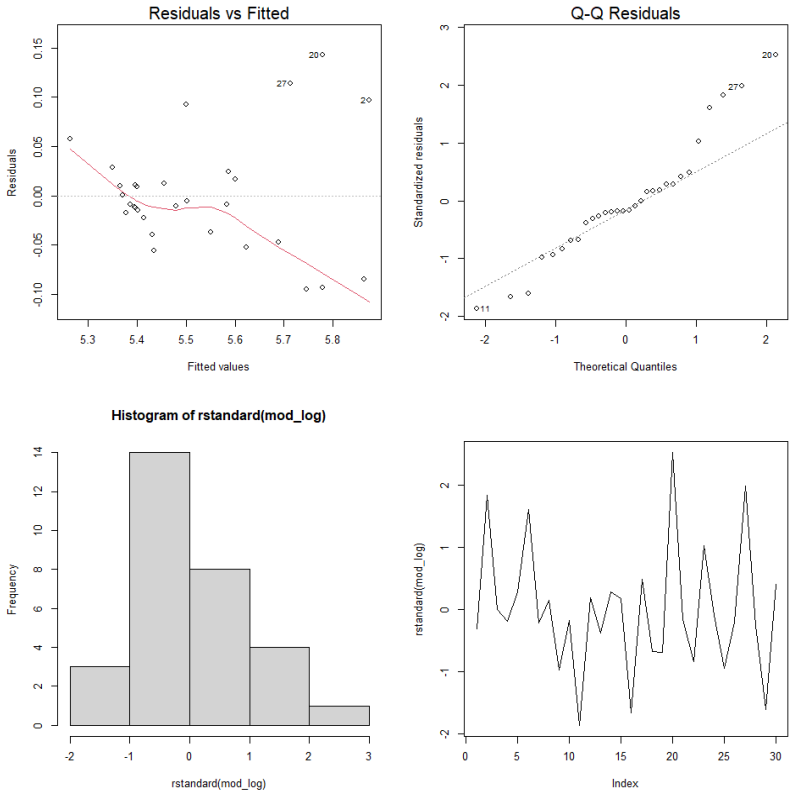
```
Call:
lm(formula = TEMPS_G ~ TEMPS_CLANG + LINIES_CODI + LLIBRERIES_CODI,
    data = dades)

Residuals:
    Min      1Q  Median      3Q     Max
-33.156  -7.640  -2.421   4.121  47.776

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     -245.1137    59.7872  -4.100  0.00036 ***
TEMPS_CLANG        1.7857     0.2563   6.966 2.14e-07 ***
LINIES_CODI       -0.2390     0.2779  -0.860  0.39767
LLIBRERIES_CODI    0.6294     7.6154   0.083  0.93477
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 18.95 on 26 degrees of freedom
Multiple R-squared:  0.8666,    Adjusted R-squared:  0.8512
F-statistic: 56.28 on 3 and 26 DF,  p-value: 1.666e-11
```

Annex 2.1: Sortida de la comanda summary del model sense transformació.



Annex 2.2: Validació de premisses del model lineal múltiple amb transformació logarítmica.

```
Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)     -4.7712871  1.3237502  -3.604   0.0013 **
clang_log        1.8253866  0.2423683   7.531 5.38e-08 ***
LINIES_CODI     -0.0007322  0.0008894  -0.823   0.4178
LLIBRERIES_CODI  0.0095311  0.0244446   0.390   0.6998
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.0602 on 26 degrees of freedom
Multiple R-squared:  0.8953,    Adjusted R-squared:  0.8833
F-statistic: 74.14 on 3 and 26 DF,  p-value: 7.184e-13
```

Annex 3.1: Sortida de la comanda summary del model sense transformació.