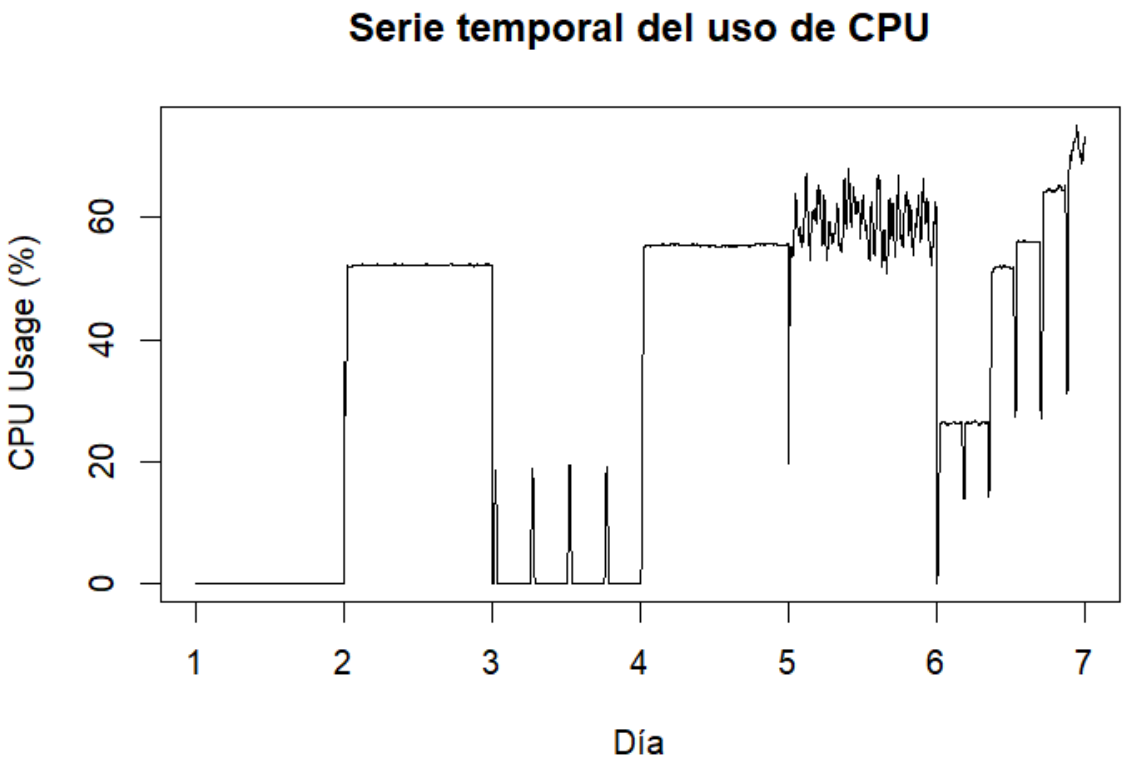
Predicción con ARIMA

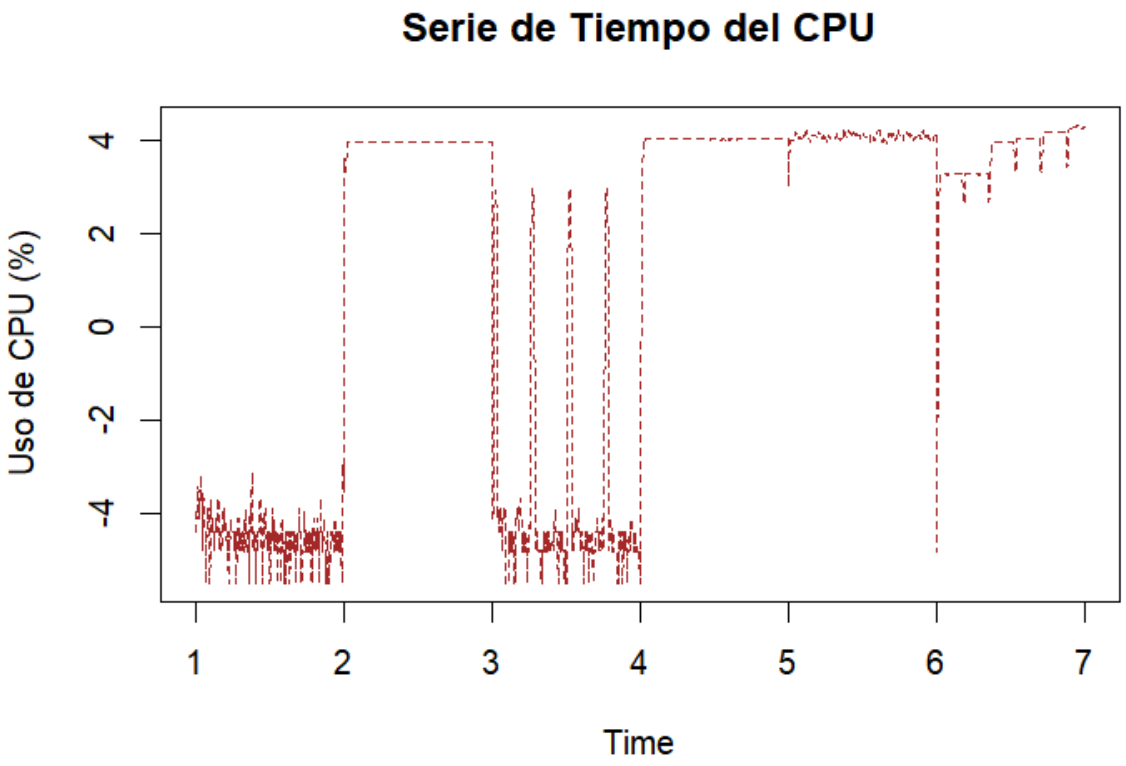
```
attach(tesita_zabbix)
names(tesita_zabbix)
install.packages("tseries")
install.packages("astsa")
install.packages("forecast")
install.packages("foreign")
install.packages("quantmod")

library(astsa)
library(tseries)
library(lubridate)
library(tidyverse)
library(forecast)

cpu ← tesita_zabbix$cpu_usage
cpu_ts ← ts(cpu, start = c(1), frequency = 720)
cpu_ts
plot(cpu_ts)
plot(cpu_ts, main="Serie temporal del uso de CPU", ylab="CPU Usage
(%)", xlab="Día")
```

1. En esta primera parte creamos una serie temporal en la cual indicamos que tenemos 720 observaciones por día, lo que equivale a un registro cada 2 minutos pero en este caso fue cada 5 segundos durante 2 horas. Se puede observar los “Periodos Planos” (CPU estable), “Picos bruscos” (uso intenso de CPU) y cambios de nivel.



2. Aplicamos el logaritmo natural a cada valor de nuestra serie temporal con el propósito de suavizar fluctuaciones grandes en los datos, reducir la varianza de los valores altos y hacer que la serie sea más estable (ARIMA). En nuestro caso, como el uso de CPU puede variar desde 0% hasta más del 70%, la escala logarítmica **reduce las diferencias extremas** entre periodos de baja y alta carga.

```
#Transformación logarítmica
serielog=log(cpu_ts)
serielog
plot(serielog,lty="dashed",ylab = "Uso de CPU (%)", col = "brown", main =
"Serie de Tiempo del CPU")
#Verificamos que si es una serie estacionaria
adf.test(serielog,alternative = "stationary")
```



3. OBSERVACIÓN: Recordando que el objetivo de nuestro poryecto es predecir el estado o pico más alto del CPU (uso máximo) para anticipar sobrecargas o eventos críticos en el servidor.

En este tipo de series:

- Los **valores bajos** (0% o idle) son importantes porque marcan períodos de descanso.
- Los **valores altos o picos** (por ejemplo, 60–80%) son lo que el modelo debe detectar y anticipar.

Por tanto, **no podemos distorsionar la escala original** más de lo necesario.

Un poco de teoría:
Qué pasa si mantienes el `log(cpu_ts + 1)`

Ventajas	Desventajas
Elimina los `-Inf`.	Suaviza demasiado los picos grandes.
Mejora la estabilidad numérica.	Reduce la amplitud de los valores altos, afectando la detección de picos.

Ventajas	Desventajas
Útil para modelos que asumen normalidad.	No ideal para predecir valores máximos absolutos.

El logaritmo **aplasta los valores grandes**: un pico de 80% y uno de 60% se vuelven más parecidos, porque el log reduce las diferencias relativas.

Esto **va en contra** de tu objetivo (detectar los máximos reales del CPU).

3.1 Ahora si siguiendo la lógica mantendremos la escala original para predecir los picos reales del CPU (%) así que envés de transformar con log solo limpiaremos los outliers sin distorsionar la escala y eliminaremos la tendencia.

```
#Limpiamos outliers sin distorcionar la escala, luego se elimina la tendencia
cpu_clean ← tsclean(cpu_ts)       # limpia outliers sin distorsionar escala
cpu_diff  ← diff(cpu_clean, 1)    # elimina tendencia
#Prueba de ADF para ver si es estacionaria
adf.test(cpu_diff)
```

```
> #Limpiamos outliers sin distorcionar la escala, luego se elimina la tendencia
> cpu_clean <- tsclean(cpu_ts)        # limpia outliers sin distorsionar escala
> cpu_diff  <- diff(cpu_clean, 1)     # elimina tendencia
> #Prueba de ADF para ver si es estacionaria
> adf.test(cpu_diff)

        Augmented Dickey-Fuller Test

data:  cpu_diff
Dickey-Fuller = -18.308, Lag order = 16, p-value = 0.01
alternative hypothesis: stationary
```

```
Aviso:
In adf.test(cpu_diff) : p-value smaller than printed p-value
```

El elemento estadístico de Dickey-Fuller da un valor de -18.308, ello es negativo así que es una evidencia de que es estacionaria, y el p-value es 0.01, en realidad el mensaje "smaller than printed p-value" significa que es aún menor, típicamente ***p < 0.01****. Así que r****echazamos la hipótesis nula (*** H_0 ***)*** *→ tu serie* ***es estacionaria***

3.1 Por curiosidad aplicamos el adf solo para el cpu_clean antes del "cpu_diff", y vemos que no es estacionaria, es por eso que aplicamos una diferencia para que sea estacionaria, ósea sin diferenciar no es estacionaria

Valor	Interpretación
p-value = 0.7836	Muy alto (mayor a 0.05).
Conclusión	❌ **No se rechaza H_0** → la serie **NO es estacionaria**.

```
> #Limpiamos outliers sin distorcionar la escala, luego se elimina la tendencia
> cpu_clean <- tsclean(cpu_ts)        # limpia outliers sin distorsionar escala
> #Prueba de ADF para ver si es estacionaria
> adf.test(cpu_clean)

        Augmented Dickey-Fuller Test

data:  cpu_clean
Dickey-Fuller = -1.5154, Lag order = 16, p-value = 0.7836
alternative hypothesis: stationary
```

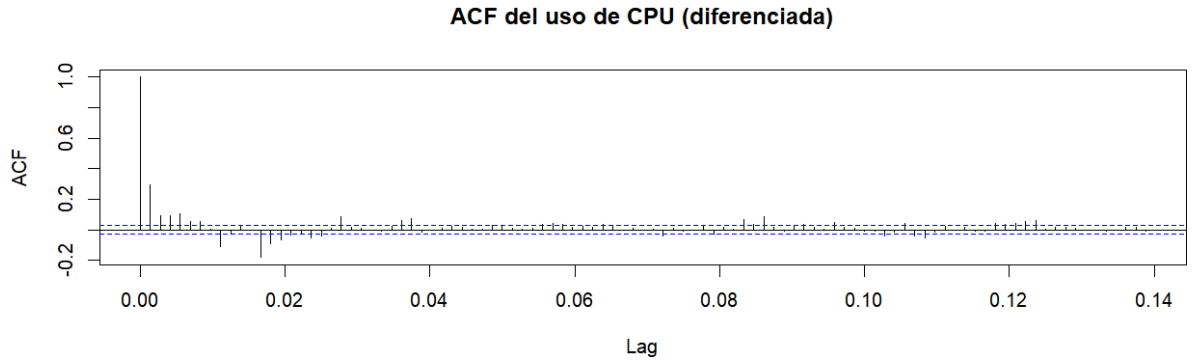
3.2 Después de aplicar esto: cpu_diff ← diff(cpu_clean, 1), vemos que nuestra serie ya no tiene tendencia y su media y varianza se estabilizaron

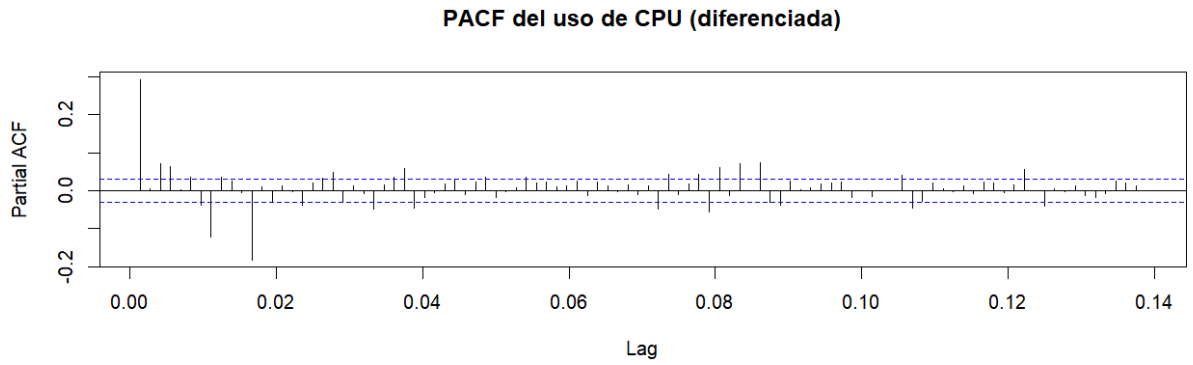
```
> cpu_diff  <- diff(cpu_clean, 1)     # elimina tendencia
> adf.test(cpu_diff)

        Augmented Dickey-Fuller Test

data:  cpu_diff
Dickey-Fuller = -18.308, Lag order = 16, p-value = 0.01
alternative hypothesis: stationary
```

```
Aviso:
In adf.test(cpu_diff) : p-value smaller than printed p-value
```

4. Ahora verificaremos la estructura del ACF y PACF para estimar los parámetros p y q

```
#saber cuantas medias móviles y cuantos auto-regresivos vamos a utilizar en nuestro modelo ARIMA
par(mfrow=c(2,1), mar=c(4,4,4,1)+.1)
acf(cpu_diff, lag.max = 100, main="ACF del uso de CPU (diferenciada)")
pacf(cpu_diff, lag.max = 100, main="PACF del uso de CPU (diferenciada)")
#Por si quieres restaurar para que no aparezca la gráfica 2 en 1 en los siguientes codes:
par(mfrow=c(1,1))
```





En ello se puede observar que en el ACF tiene un pico muy fuerte en lag 1 (~0.9), después las barras caen rápidamente dentro del rango de confianza (líneas azules), no hay picos picos importantes en lags mayores y en cuanto al PACF también tiene un solo pico significativo en lag 1 y los demás son pequeños casi nulos a excepción de uno que llega a alcanzar a -0.2

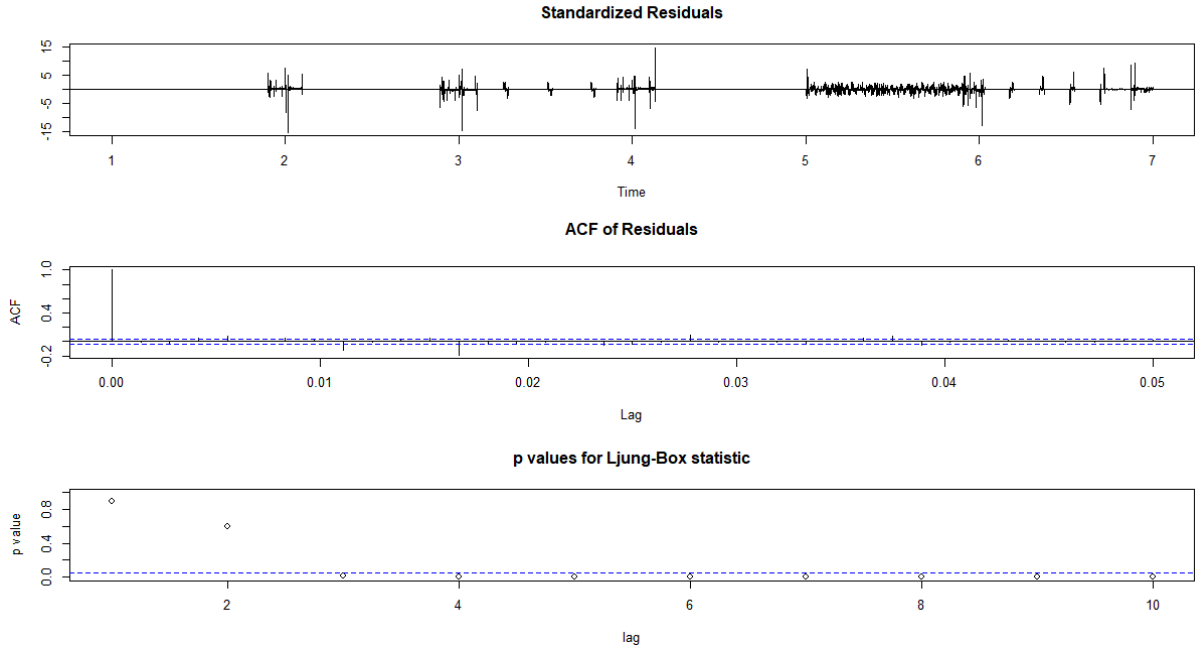
Ello nos indica que para empezar podemos usar este modelo ARIMA(1,1,0)

Usamos el "lag.max = 100" para ver con más detalle los primeros rezagos para detectar p y q, no es necesario colocar de 1000 o 2000 porque eso es buscar estacionalidad y ya sabes que es estacionaria

5. Ahora que ya sabemos que tenemos que usar un P, entrenaremos el primer modelo ARIMA(1,1,0) y veremos que tal se nos da

```
# Ajuste con p=1, d=1, q=0 sobre la serie limpia (no la diferenciada)
modelo1 ← arima(cpu_clean, order = c(1,1,0))
summary(modelo1)
tsdiag(modelo1)
```

```
> modelo1 <- arima(cpu_clean, order = c(1,1,0))
> summary(modelo1)

Call:
arima(x = cpu_clean, order = c(1, 1, 0))

Coefficients:
         ar1
      0.2933
s.e.  0.0145

sigma^2 estimated as 0.5604:  log likelihood = -4877.88,  aic = 9759.77

Training set error measures:
                      ME      RMSE       MAE       MPE     MAPE      MASE         ACF1
Training set 0.01199971 0.7485154 0.2560158 -21.81571 53.31804 0.9317517 -0.002034827
```



El modelo ARIMA (1,1,0) se ajusta bien, con un coeficiente significativo y un AIC razonable

Elemento	Significado
ar1 = 0.2933	El valor actual de CPU depende en un **29.3 % del valor anterior (rezago 1)**. En otras palabras, hay **memoria temporal leve**: si el CPU sube, es probable que el siguiente instante también esté un poco alto.
s.e. (error estándar) = 0.0145	Es muy bajo → el parámetro **es altamente significativo**.
σ^2 = 0.5604	Varianza del error. Mientras más pequeña, mejor ajuste (aunque depende de la escala del CPU).

Elemento	Significado
Log likelihood = −4877.88	Valor usado internamente para comparar modelos.
AIC = 9759.77	Métrica de calidad del modelo (cuanto menor, mejor). La usaremos para comparar con el auto-ARIMA.

En cuanto a las métricas de error el modelo no tiene sesgo, tiene errores pequeños y los residuos son independientes

Métrica	Significado	Interpretación
ME (Mean Error)	Promedio de los residuos	0.012 ≈ 0 → sin sesgo sistemático 👍
RMSE (Root Mean Square Error)	Error cuadrático medio	0.75 → error promedio de ±0.75 % en el CPU (muy bajo considerando 0–100 %)
MAE (Mean Absolute Error)	Error absoluto medio	0.256 → predicciones bastante ajustadas.
MPE / MAPE	Error porcentual medio / absoluto	El MAPE alto (53 %) se debe a los momentos en que el CPU es muy bajo (cerca de cero, cualquier pequeño error se amplifica).
MASE	Error estandarizado	0.93 < 1 → el modelo predice mejor que un pronóstico ingenuo (por ejemplo, “usar el valor anterior”).
ACF1	Autocorrelación de los residuos	~0 → los residuos son prácticamente ruido blanco.

Diagnóstico del “tsdiag” es que los gráficos de residuos estandarizados (arriba), oscilan alrededor de cero, no hay tendencia asíq ue es buena señal y algunos picos asilados prueban que hay momentos de carga anómala de CPU, en cuanto al segundo del ACF todas las barras están dentro de las líneas azules, no hay autocorrelación significativa que es que si hay residuos hay ruido blanco y en la tercera los p-values están por encima de 0.05 (solo 2)

5.1 Un poco de teoría, sobre los p-values de Ljung-Box, en el eje X están los lags del 1 al 10, y en el Y el p-value, las líneas de azules marcar el umbral de 0.05 (nivel de significacia), en mi gráfico solo 2 puntos están por encima del umbral, y los demás debajo, quiere decir que todavia hay algo de autocrrelación en los residuos, ósea el modelo no ha capturado completamente toda la dinámica temporal, recordar que los primeros lags son importantes (1,2 y 3) ya ls lags altos como (8,9,10) si están debajo, puede ser un ruido normal.

Situación	Interpretación
La mayoría o todos los puntos están **por encima** de la línea azul (p > 0.05)	✅ Los residuos son ruido blanco → modelo adecuado.
Muchos puntos están **por debajo** (p < 0.05)	⚠️ Hay autocorrelación → el modelo no explica bien los datos.

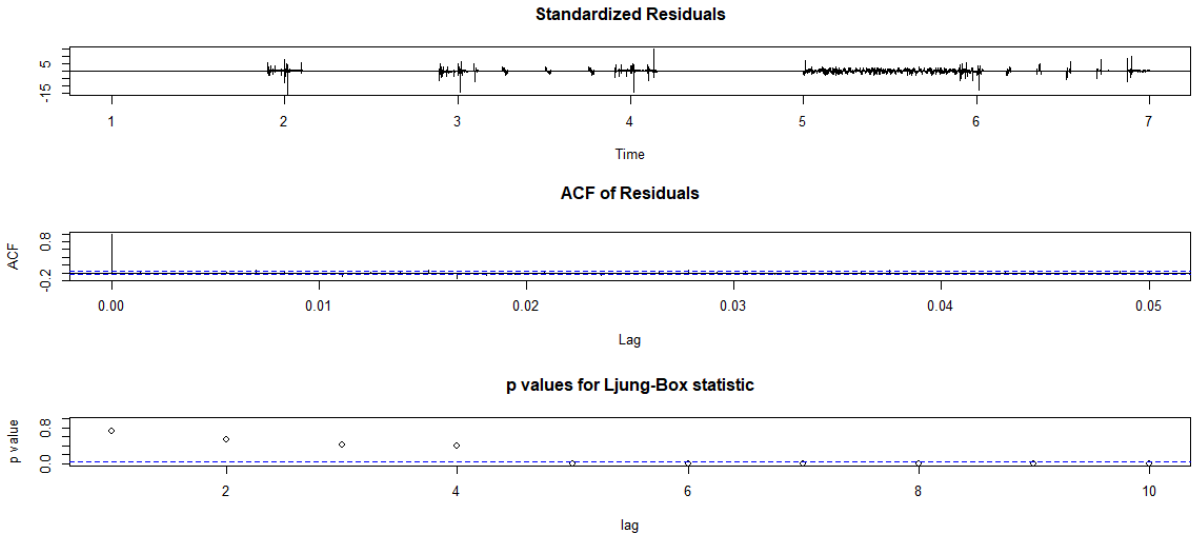
5.2 Ahora veremos como aumentar p o q

- Cuando aumentar p:
  - Si la PACF muestra picos significativos como en el lag 1, 2 o 3....
  - Si los residuos muestran autocorrelación en los primeros lags (gráfico Ljung-Box o ACF de residuos)
- Cuando aumentar q:
  - Si la ACF muestra picos significativos después de la diferencia.
  - Si los residuos muestran correlación a través de los errores pasados.

Como mi modelo actual, en cuanto a mis risudios todavia tienen algo de autocrrelación, probraremos añadiendo un componente MA(1) = ARIMA(1,1,1)

Parámetro	Qué representa	Qué controla
p	Términos autoregresivos (AR)	Relación con valores **pasados** de la serie
q	Términos de media móvil (MA)	Relación con **errores pasados (ruido)**
d	Diferenciaciones	Tendencia (ya la tenemos en 1)

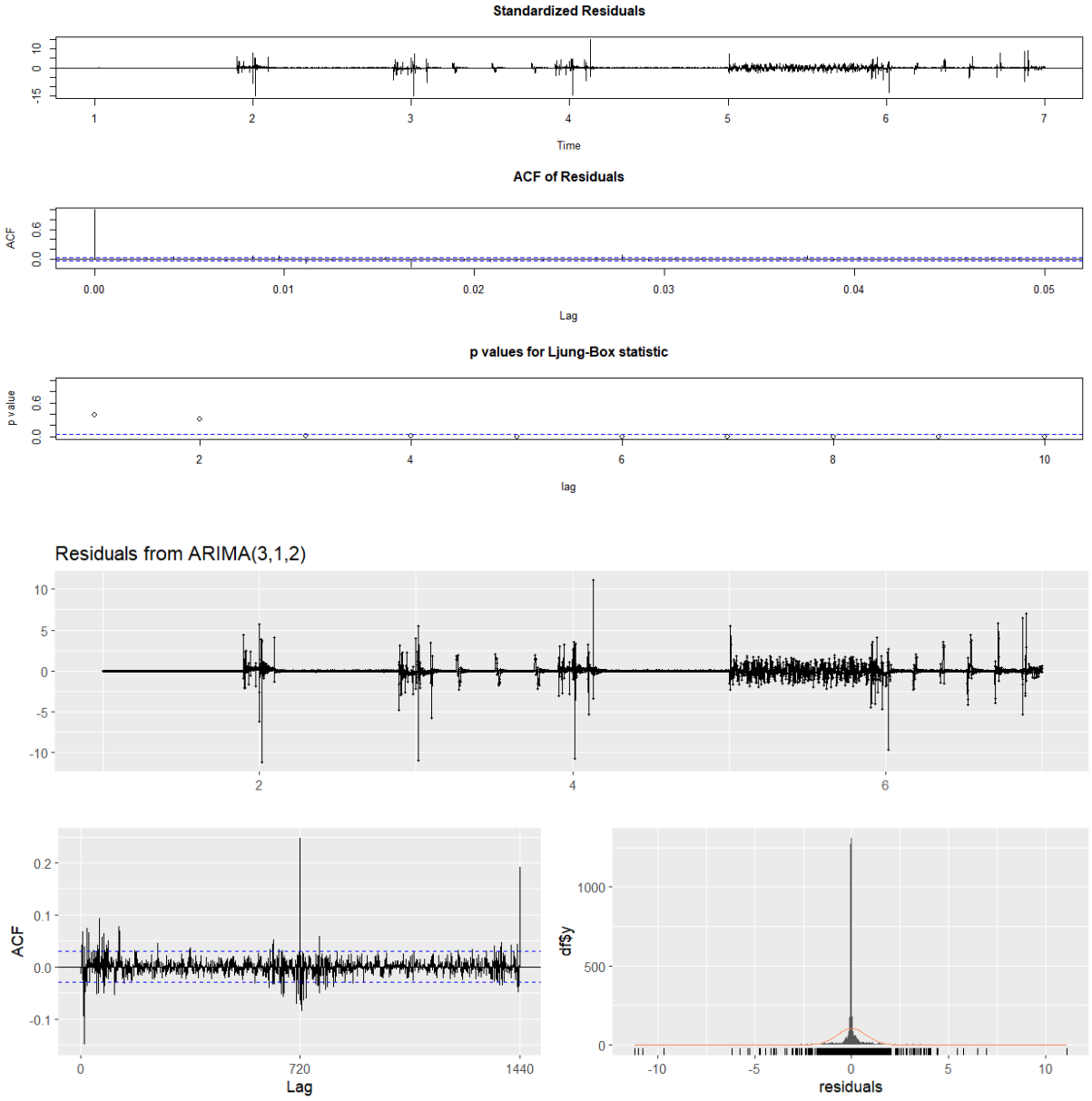
5.3 Hice la prueba en un modelo5 con ARIMA(3,1,3) tiene 4 círculos sobre la línea, hay algo de autocorrelación pero vemos que al menos 4 resaltan.




```
> summary(modelo5)

Call:
arima(x = cpu_clean, order = c(3, 1, 3))

Coefficients:
          ar1      ar2     ar3     ma1     ma2      ma3
      -0.1103  -0.5109  0.4874  0.3968  0.6311  -0.2515
s.e.   0.0628   0.0985  0.0555  0.0694  0.1082   0.0818

sigma^2 estimated as 0.552:  log likelihood = -4845.44,  aic = 9704.89

Training set error measures:
                     ME      RMSE       MAE       MPE     MAPE      MASE         ACF1
Training set 0.01085596 0.7429043 0.2611338 -16.34122 67.65037 0.9503786 0.005260669
> tsdiag(modelo5)
```

5.4 El modelo 6 con el que finalmente nos quedaremos es ARIMA(3,1,2) en la cual a pesar de tener solo los 2 primeros círculos por encima de la línea azul, su AIC, BIC y sigma2 es mejor que todos los demás modelos

```
> summary(modelo6)

Call:
arima(x = cpu_clean, order = c(3, 1, 2))

Coefficients:
         ar1      ar2     ar3      ma1     ma2
      1.1200  -1.1780  0.3187  -0.8244  0.9066
s.e.  0.0235   0.0231  0.0144   0.0205  0.0214

sigma^2 estimated as 0.5507:  log likelihood = -4840.08,  aic = 9692.16

Training set error measures:
                     ME      RMSE      MAE       MPE     MAPE      MASE         ACF1
Training set 0.01160669 0.7419747 0.26228 -20.93627 78.34432 0.9545499 -0.01334574
```



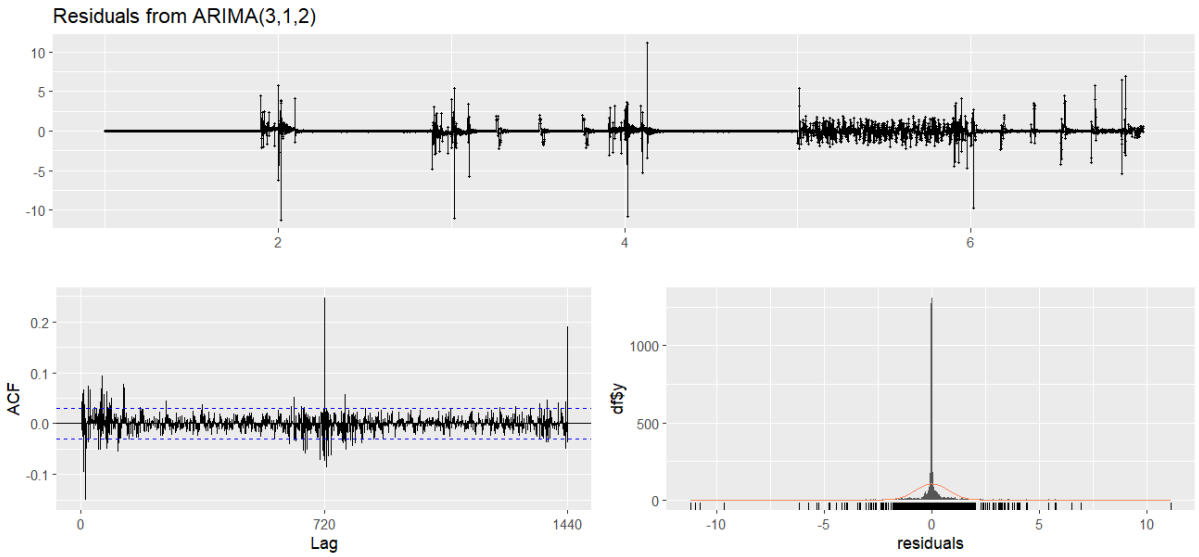
Ahora tenemos duda con cual quedarnos, recordemos que estamos modelando una métrica real de CPU es naturalmente ruidosa y variable (no una serie económica estable), por eso es que existe cierta autocorrelación residual es normal y aceptable, ya que los picos de CPU pueden depender de procesos externos que no se capturan con ARIMA puro, así que siguiente nuestro objetivo de predecir el comportamiento general y detectar picos futuros, el modelo mas eficiente es el de AIC (3,1,2)

💡 El modelo ARIMA(3,1,2) presenta el menor valor de AIC (9692.16) y una varianza residual ($\sigma^2 = 0.5507$) inferior a las demás configuraciones.
Aunque los valores p del test de Ljung–Box son bajos (indicando cierta autocorrelación residual), esto es esperable debido a la naturaleza irregular y dependiente de eventos externos del uso de CPU.
Por tanto, se considera que el modelo ARIMA(3,1,2) logra el mejor equilibrio entre ajuste estadístico y capacidad predictiva realista para anticipar picos de carga.

5.5 Para respaldar y comparar usaremos la función AUTO-ARIMA y vemos que irónicamente nos da el mismo modelo es óptimo según el criterio AICc.

```
y ← cpu_clean
#AutoArima para comparar
modelo7_auto ← auto.arima(
  y,
  seasonal      = FALSE,        # evita SARIMA con periodo 720
  stepwise      = FALSE,        # búsqueda global (mejor)
  approximation  = FALSE,         # sin aproximaciones
  ic            = "aicc",       # usamos AICc para comparar
  allowmean     = FALSE,        # consistente con tus ARIMA(d>=1) sin const
ante
  allowdrift    = TRUE          # permite drift si d=1 (lo decidirá el algoritmo)
)
summary(modelo7_auto)

# 3) Diagnóstico de residuos (mejor que tsdiag para objetos 'forecast')
checkresiduals(modelo7_auto)
```

```
> summary(modelo7_auto)
Series: y
ARIMA(3,1,2)

Coefficients:
         ar1      ar2     ar3      ma1     ma2
      1.1200  -1.1780  0.3187  -0.8244  0.9066
s.e.  0.0235   0.0231  0.0144   0.0205  0.0214

sigma^2 = 0.5513:  log likelihood = -4840.08
AIC=9692.16   AICc=9692.18   BIC=9730.38

Training set error measures:
                     ME      RMSE      MAE       MPE     MAPE         MASE         ACF1
Training set 0.01160669 0.7419747 0.26228 -20.93627 78.34432 0.007835907 -0.01334574
> # 3) Diagnóstico de residuos (mejor que tsdiag para objetos 'forecast')
> checkresiduals(modelo7_auto)

        Ljung-Box test

data:  Residuals from ARIMA(3,1,2)
Q* = 1891, df = 859, p-value < 2.2e-16

Model df: 5.   Total lags used: 864
```



5.5.1

Gráfico superior (residuos en el tiempo)

- La mayoría de los residuos están **centrados en 0**,

  salvo algunos picos aislados (que son normales en series de CPU real).
- No hay tendencia visible ni patrón cíclico → **bien modelado**.

ACF de residuos

- Las barras están dentro del intervalo azul →

  no hay autocorrelación significativa en los residuos.

- Esto significa que el modelo **ya capturó casi toda la dependencia temporal**.

Histograma

- Los residuos siguen una forma cercana a normal (ligera asimetría por los picos extremos del CPU, lo cual es esperable).

Aclarar que aunque nos de un p-value debajo de 0.05 en la prueba de Ljung-Box test en un contexto teórico es una alerta, ya que los residuos no son completamente ruido blanco, pero en nuestro caso al ser datos reales de CPU eso no invalida el modelo, ya que nuestra serie no es una serie económica o controlada, sino una serie de comportamiento físico caótico (uso de CPU minuto a minuto), en este tipo de señales siempre hay microdependencias locales y picos abruptos (procesos del sistema, tareas, hilos, etc), el test inerpreta esas pequeñas correlaciones naturales como "no ruido blanco" incluso si el modelo es corecto. Para un mejor énfasis nos enfocamos en la ACF visual de los residuos, si las barras están dentro de las líneas azules, el modelo es aceptable y si lo está

No asustarse por el MASE, porque vemos que es el mismo, pero en diferente escala ya que el inicio usa el naive simple, y el auto-arima usa uno mas estandarizado del forecast

```
# Para ver la diferencia de la escala del MASE
  # 1. Predicciones dentro de la muestra
pred ← fitted(modelo7_auto)

  # 2. Calcular MAE del modelo
mae_model ← mean(abs(cpu_clean - pred))

  # 3. Calcular MAE del modelo naive (y_{t} - y_{t-1})
naive_pred ← cpu_clean[-length(cpu_clean)]
mae_naive ← mean(abs(diff(cpu_clean)))

  # 4. Calcular MASE manual
mase_manual ← mae_model / mae_naive
mase_manual
```

6. Ahora con respecto a la predicción haremos una comparativa entre el modelo ARIMA (3,1,2) que es el mejor, y con el auto-arima pero con drift

6.1 Con el modelo 6 (3,1,2)