Procesamiento de Resultados de Simulación

June 24, 2024

1 Importación de Librerías

[37]:
```
import os

import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd
import numpy as np

from scipy.stats import kstest, levene, f_oneway, kruskal, rankdata, norm
from statsmodels.stats.multicomp import pairwise_tukeyhsd
```

2 Cargar Datos

Se carga el conjunto de datos

[52]:
```
# Se obtiene la ruta del directorio actual donde se encuentra el script
directorio_actual = os.getcwd()

# Se declara el nombre del archivo
nombre_archivo = 'resultados_simulacion.csv'

# Se combinan las rutas para obtener la ruta completa del archivo CSV
ruta_completa = os.path.join(directorio_actual, nombre_archivo)

# Se lee el archivo CSV
datos_tratados = pd.read_csv(ruta_completa, delimiter=';')

datos_tratados.head()
```

[52]:

	Pol_Control	Cantidad de Almacenamientos	Tiempo Total de Almacenamiento	\
0	PGA	3568	125544.6857	
1	PKA	3568	118285.8000	
2	PPA	3568	117694.9429	
3	PQA	3568	125836.0571	
4	PGA	3568	125532.4286	

```
   Cantidad de Recuperaciones  Tiempo Total de Recuperacion  \
0                        3600                    127445.3429
1                        3600                    130407.2571
2                        3600                    130434.2000
3                        3600                    127801.4000
4                        3600                    128065.2000

   Cantidad de Reubicaciones  Tiempo Total de Reubicacion  \
0                       1380                   56326.0286
1                       1896                   89183.4857
2                       1194                   58706.5429
3                       1378                   56127.1714
4                       1313                   53240.0571

   Cantidad de Reubicaciones en Almacenamiento  \
0                                          292
1                                          808
2                                           90
3                                          289
4                                          264

   Tiempo Total de Reubicaciones en Almacenamiento  \
0                                      10775.0857
1                                      37121.3143
2                                       4031.4857
3                                      10445.6286
4                                       9653.2571

   Cantidad de Reubicaciones en Recuperacion  …  \
0                                       1088  …
1                                       1088  …
2                                       1104  …
3                                       1089  …
4                                       1049  …

   Cantidad de Reubicaciones Regulares en Almacenamiento  \
0                                                292
1                                                808
2                                                 90
3                                                289
4                                                264

   Tiempo Total de Reubicaciones Regulares en Almacenamiento  \
0                                      10775.0857
1                                      37121.3143
2                                       4031.4857
```

3	10445.6286
4	9653.2571

	Cantidad de Reubicaciones Predeterminadas en Almacenamiento
0	0
1	0
2	0
3	0
4	0

	Tiempo Total de Reubicaciones Predeterminadas en Almacenamiento
0	0
1	0
2	0
3	0
4	0

	Cantidad de Reubicaciones Regulares en Recuperacion
0	1088
1	1088
2	1104
3	1089
4	1049

	Tiempo Total de Reubicaciones Regulares en Recuperacion
0	45550.9429
1	52062.1714
2	54675.0571
3	45681.5429
4	43586.8000

	Cantidad de Reubicaciones Predeterminadas en Recuperacion
0	0
1	0
2	0
3	0
4	0

	Tiempo Total de Reubicaciones Predeterminadas en Recuperacion
0	0
1	0
2	0
3	0
4	0

	Cantidad Total de Operaciones	Tiempo Total de Operaciones
0	7168	309316.0571

```
1                          7168                    337876.5429
2                          7168                    306835.6857
3                          7168                    309764.6286
4                          7168                    306837.6857

[5 rows x 21 columns]
```

3 Preparación de Datos

Se prepara el conjunto de datos con el fin de que pueda brindar información acerca de las políticas de reubicación desde el punto de vista de el tiempo de operación. Para ello se raaliza lo siguiente.

- Filtrar los datos considerando únicamente las columnas relevantes, que guardan los datos de tiempo de operación, almacenamiento, reubicación y recuperación.
- Tratar las entradas en la la columna de política de control, reduciendo la cadena de texto a una sola letra que representa la política de reubicación
- Redondear todas la entradas no enteras para facilitar visualización, además de que se consideran insignificantes las décimas de segundo en este caso.
- Organizar alfábéticamente la columna de políticas de control.

[39]:
```
# Se selecciona las columnas de tiempos de almacenamiento, reubicación y␣
 ↪recuperación
datos_tratados = datos_tratados[['Pol_Control', 'Tiempo Total de␣
 ↪Almacenamiento',
                                 'Tiempo Total de Recuperacion', 'Tiempo Total␣
 ↪de Reubicacion', 'Tiempo Total de Operaciones']].copy()

# Se trantan las entradas de la columna política de control
datos_tratados['Pol_Control'] = datos_tratados['Pol_Control'].apply(
    lambda x: x[1])

# Ordenar la columna de políticas de control alfabéticamente
datos_tratados = datos_tratados.sort_values(by='Pol_Control')
datos_tratados.reset_index(drop=True, inplace=True)

# Se cambia el nombre de la primera columna
datos_tratados = datos_tratados.rename(columns={
    'Pol_Control': 'Pol',
    'Tiempo Total de Almacenamiento': 'TTA',
    'Tiempo Total de Recuperacion': 'TTRc',
    'Tiempo Total de Reubicacion': 'TTRb',
    'Tiempo Total de Operaciones': 'TTO',
})

# Se redondean décimales
datos_tratados = datos_tratados.round()
```

```
# Se guardan los datos tratados
datos_tratados.to_csv('datos_tratados.csv', index=False)

datos_tratados.head()
```

[39]:

	Pol	TTA	TTRc	TTRb	TTO
0	G	125545.0000	127445.0000	56326.0000	309316.0000
1	G	125412.0000	127773.0000	56432.0000	309617.0000
2	G	125146.0000	127374.0000	55443.0000	307963.0000
3	G	125319.0000	127679.0000	58086.0000	311084.0000
4	G	124523.0000	126892.0000	56728.0000	308143.0000

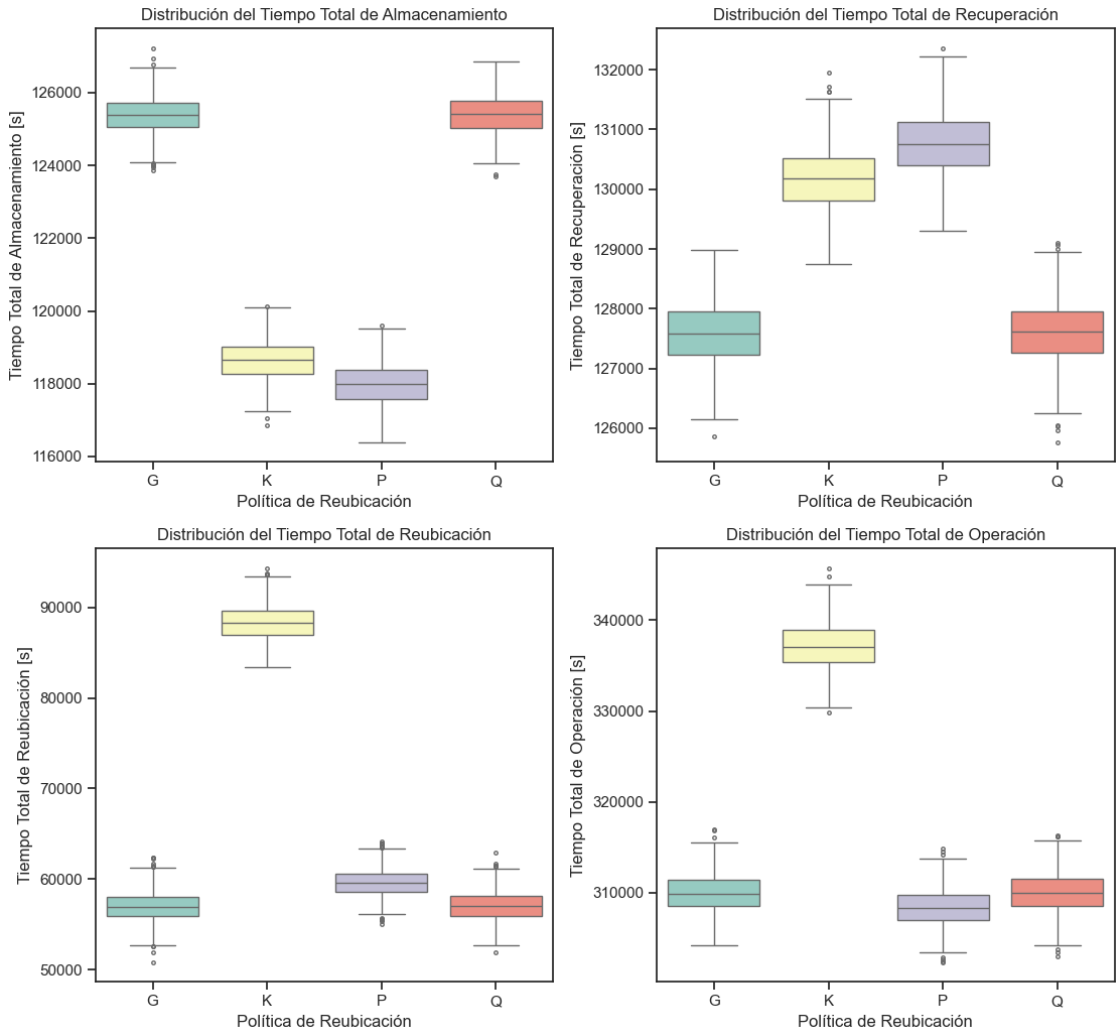
4 Análisis Descriptivo

Se espera que análisis descriptivo proporcione una visión general de las características de los datos recopilados. Se realizarán las siguientes actividades:

- **Cálculo de Estadísticas Descriptivas**: Se calcularán medidas de tendencia central (media, mediana) y de dispersión (desviación estándar, rango, percentiles) para cada variable y cada política de reubicación. Estas estadísticas ayudarán a entender la distribución y variabilidad de los datos.
- **Visualización de Datos**: Se crean un gráficos de cajas y para visualizar la distribución de los datos.

[40]:
```
# Configuración de estilo para gráficos
sns.set_theme(style='ticks')

# Definir las dimensiones de la figura
altura_grafico = 7  # Altura del gráfico en pulgadas
# Anchura del gráfico basada en una proporción agradable
anchura_grafico = altura_grafico * 1.618

# Crear una figura con subplots para cada métrica
fig, axs = plt.subplots(2, 2, figsize=(anchura_grafico, altura_grafico*1.5))

# Métricas para analizar
metricas = ['TTA', 'TTRc', 'TTRb', 'TTO']
titulos = [
    'Distribución del Tiempo Total de Almacenamiento',
    'Distribución del Tiempo Total de Recuperación',
    'Distribución del Tiempo Total de Reubicación',
    'Distribución del Tiempo Total de Operación'
]
ylabel = [
    'Tiempo Total de Almacenamiento [s]',
    'Tiempo Total de Recuperación [s]',
    'Tiempo Total de Reubicación [s]',
```

```
    'Tiempo Total de Operación [s]'
]

# Iterar sobre las métricas y crear los gráficos de cajas
for i, metrica in enumerate(metricas):
    sns.boxplot(x='Pol', y=metrica, hue='Pol', data=datos_tratados,
                ax=axs[i//2, i % 2], palette='Set3', fliersize=2.5,␣
 ↪dodge=False, legend=False)
    axs[i//2, i % 2].set_title(titulos[i])
    axs[i//2, i % 2].set_ylabel(ylabel[i])
    axs[i//2, i % 2].set_xlabel('Política de Reubicación')

# Ajustar el layout
plt.tight_layout()

# Mostrar los gráficos
plt.show()
```



[41]:
```python
# Crear la tabla de medidas de tendencia central y dispersión
tabla_estadisticas_descr = pd.DataFrame({
    'Política': [],
    'Métrica': [],
    'Media': [],
    'Mediana': [],
    'Desviación Estándar': [],
    'Rango Intercuartílico (IQR)': []
})

# Lista de métricas y políticas
metricas = ['TTA', 'TTRc', 'TTRb', 'TTO']
politicas = datos_tratados['Pol'].unique()

# Generar las estadísticas por cada política y métrica
for politica in politicas:
    for metrica in metricas:
        # Filtrar los datos por política y métrica
        datos_filtrados = datos_tratados[datos_tratados['Pol']
                                         == politica][metrica]

        # Calcular medidas estadísticas
        media = datos_filtrados.mean()
        mediana = datos_filtrados.median()
        desviacion = datos_filtrados.std()
        Q1 = datos_filtrados.quantile(0.25)
        Q3 = datos_filtrados.quantile(0.75)
        iqr = Q3 - Q1

        # Crear un DataFrame temporal con los resultados
        temp_df = pd.DataFrame({
            'Política': [politica],
            'Métrica': [metrica],
            'Media': [media],
            'Mediana': [mediana],
            'Desviación Estándar': [desviacion],
            'Rango Intercuartílico (IQR)': [iqr]
        })

        # Concatenar el DataFrame temporal con la tabla principal
        tabla_estadisticas_descr = pd.concat(
            [tabla_estadisticas_descr, temp_df], ignore_index=True)

        # Redondear, para mejor visualización
```

```
        tabla_estadisticas_descr = tabla_estadisticas_descr.round(2)

# Mostrar la tabla generada
tabla_estadisticas_descr
```

[41]:

	Política	Métrica	Media	Mediana	Desviación Estándar	Rango Intercuartílico (IQR)
0	G	TTA	125402.3900	125388.0000	522.7200	673.7500
1	G	TTRc	127587.2100	127577.0000	501.7100	720.2500
2	G	TTRb	56982.3600	56936.5000	1658.1700	2174.0000
3	G	TTO	309971.9500	309855.0000	2133.0900	2900.7500
4	K	TTA	118662.8400	118653.0000	539.1800	727.0000
5	K	TTRc	130178.4600	130182.0000	519.9300	714.5000
6	K	TTRb	88344.4000	88292.0000	1918.4400	2643.2500
7	K	TTO	337185.7000	337025.5000	2530.1700	3555.5000
8	P	TTA	117986.5900	118002.5000	552.0700	793.0000
9	P	TTRc	130757.7100	130747.5000	520.5700	735.0000
10	P	TTRb	59614.2900	59543.5000	1437.9100	1915.2500
11	P	TTO	308358.5700	308291.0000	1972.6500	2719.5000
12	Q	TTA	125404.6300	125405.0000	529.6600	751.2500
13	Q	TTRc	127600.9500	127623.5000	523.9900	692.2500
14	Q	TTRb	56991.0100	57005.5000	1610.0300	2178.2500
15	Q	TTO	309996.5700	309961.5000	2180.2500	2983.7500

5 Comprobación de Supuestos para el ANOVA

5.1 Prueba de Normalidad (Kolmogorov-Smirnov)

Esta prueba verifica si los datos siguen una distribución normal. La prueba se realiza para cada combinación de política y métrica (TTA, TTRc, TTRb, TTO). Un valor p mayor que 0.05 indica

que los datos no se desvían significativamente de una distribución normal.

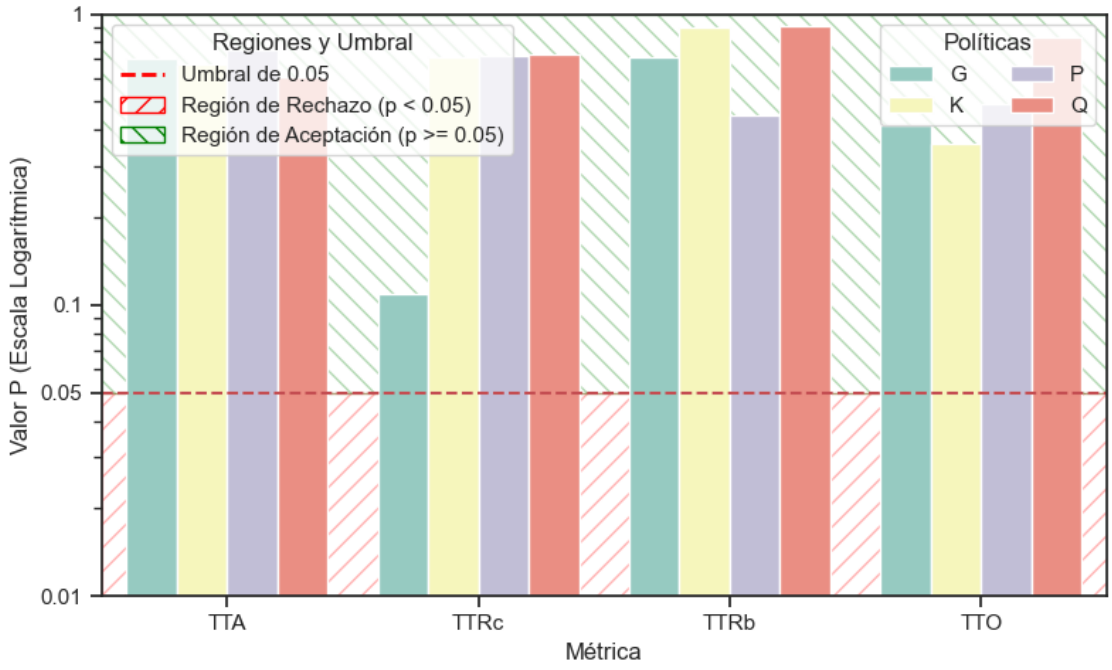
5.2 Prueba de Homocedasticidad (Levene)

Esta prueba evalúa si las variaciones entre los grupos son homogéneas. Se realiza para cada métrica comparando entre las diferentes políticas. Un valor p mayor que 0.05 sugiere que las varianzas son iguales entre los grupos.

[42]:
```
# Realizar la prueba de normalidad Kolmogorov-Smirnov para cada combinación de␣
↪política y métrica
resultados_ks = []

for politica in datos_tratados['Pol'].unique():
    for metrica in ['TTA', 'TTRc', 'TTRb', 'TTO']:
        # Estandarizar los datos para la prueba KS
        datos_estandarizados = (datos_tratados[datos_tratados['Pol'] ==␣
↪politica][metrica] - datos_tratados[datos_tratados['Pol']
                                == politica][metrica].mean()) /␣
↪datos_tratados[datos_tratados['Pol'] == politica][metrica].std()
        estadistico, valor_p = kstest(datos_estandarizados, 'norm')
        resultados_ks.append(
            {'Política': politica, 'Métrica': metrica, 'Valor P': valor_p})

# Convertir los resultados a un DataFrame
resultados_ks_df = pd.DataFrame(resultados_ks)

# Definir las dimensiones de la figura
altura_grafico = 5  # Altura del gráfico en pulgadas
# Anchura del gráfico basada en una proporción agradable
anchura_grafico = altura_grafico * 1.618

# Crear el gráfico de barras con las dimensiones ajustadas
plt.figure(figsize=(anchura_grafico, altura_grafico))

# Sombrear la región de rechazo (p < 0.05) con trama
plt.fill_between(x=[-0.5, len(resultados_ks_df['Métrica'].unique()) *␣
↪len(datos_tratados['Pol'].unique()) - 0.5],
                 y1=0.001, y2=0.05, color='none', edgecolor='red', hatch='//',␣
↪alpha=0.3, label='Región de Rechazo (p < 0.05)')

# Sombrear la región de aceptación (p >= 0.05) con trama
plt.fill_between(x=[-0.5, len(resultados_ks_df['Métrica'].unique()) *␣
↪len(datos_tratados['Pol'].unique()) - 0.5],
                 y1=0.05, y2=1, color='none', edgecolor='green', hatch='\\\\',␣
↪alpha=0.3, label='Región de Aceptación (p >= 0.05)')

# Configurar el gráfico usando seaborn con el tema 'ticks' y la paleta 'Set3'
```

```
sns.set_theme(style="ticks")
grafico_barras = sns.barplot(data=resultados_ks_df, x='Métrica',
                             y='Valor P', hue='Política', palette='Set3',␣
 ↪alpha=1.0, dodge=True)

# Configurar el eje Y en escala logarítmica
plt.yscale('log')

# Definir los ticks específicos en el eje Y para la escala logarítmica
plt.yticks([0.01, 0.05, 0.1, 1], ['0.01', '0.05', '0.1', '1'])

# Ajustar los límites del eje Y para que el punto máximo sea 1
plt.ylim(0.01, 1)

# Añadir línea de umbral de significancia
umbral_line = plt.axhline(
    y=0.05, color='r', linestyle='--', label='Umbral de 0.05')

plt.xlabel('Métrica')
plt.ylabel('Valor P (Escala Logarítmica)')
# plt.title('Prueba de Normalidad Kolmogorov-Smirnov por Métrica y Política')

# Obtener manejadores y etiquetas para las regiones y el umbral
handles_prueba = [
    plt.Line2D([0], [0], color='red', lw=2,
               linestyle='--', label='Umbral de 0.05'),
    plt.Rectangle((0, 0), 1, 1, facecolor='none', edgecolor='red',
                  hatch='//', label='Región de Rechazo (p < 0.05)'),
    plt.Rectangle((0, 0), 1, 1, facecolor='none', edgecolor='green',
                  hatch='\\\\', label='Región de Aceptación (p >= 0.05)')
]

# Crear la leyenda para las regiones y el umbral en la parte inferior izquierda
prueba_legend = plt.legend(
    handles=handles_prueba, loc='upper left', frameon=True, title="Regiones y␣
 ↪Umbral")

# Obtener manejadores y etiquetas de la leyenda de políticas
handles_politicas, labels_politicas = grafico_barras.get_legend_handles_labels()

# Filtrar las entradas que no son de políticas (remover las entradas␣
 ↪relacionadas con las tramas y el umbral)
# Esto incluye eliminar la primera, segunda y última entrada
# Remover la primera y segunda (tramas) y la última (umbral)
handles_politicas = handles_politicas[2:-1]
# Remover la primera y segunda (tramas) y la última (umbral)
labels_politicas = labels_politicas[2:-1]
```

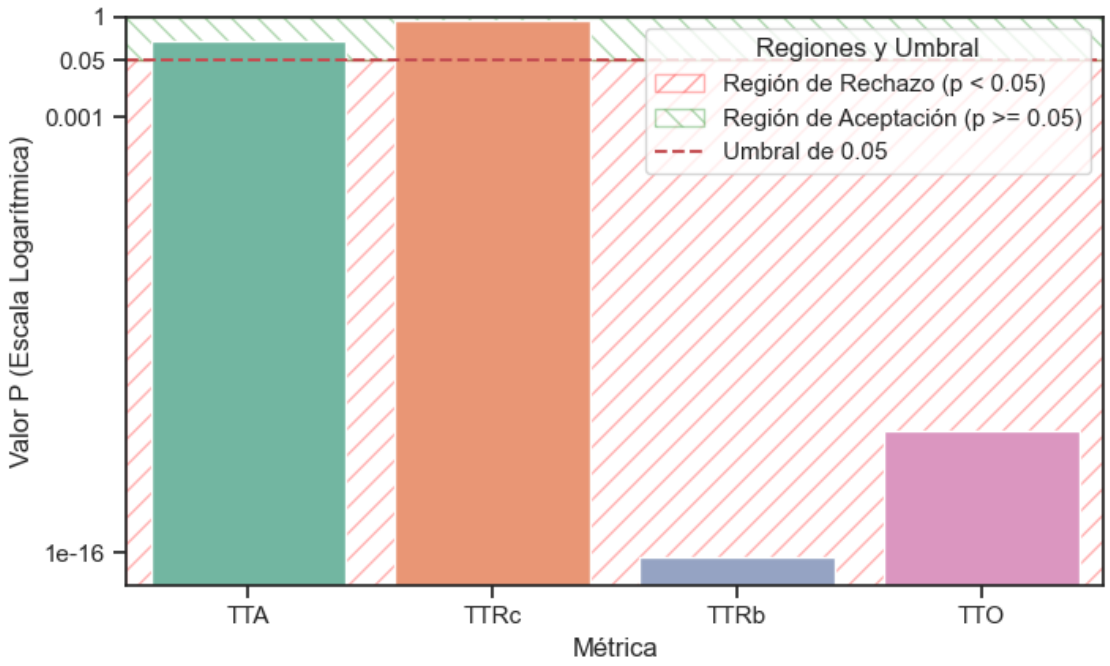
```
# Crear la leyenda para las políticas en la parte inferior derecha
politicas_legend = plt.legend(
    handles=handles_politicas, labels=labels_politicas, ncol=2, frameon=True,␣
 ↪title="Políticas")

# Añadir la leyenda de prueba al gráfico
plt.gca().add_artist(prueba_legend)

plt.tight_layout()

# Mostrar el gráfico
plt.show()
```



[43]:
```
# Realizar la prueba de homocedasticidad de Levene para cada métrica comparando␣
 ↪entre políticas
resultados_levene = []

# Políticas y Métricas a evaluar
politicas = datos_tratados['Pol'].unique()
metricas = ['TTA', 'TTRc', 'TTRb', 'TTO']

# Realizar la prueba de Levene
for metrica in metricas:
```

```
    # Obtener los datos por política para la métrica actual
    grupos = [datos_tratados[datos_tratados['Pol'] == politica][metrica]
              for politica in politicas]
    estadistico, valor_p = levene(*grupos)
    resultados_levene.append({'Métrica': metrica, 'Valor P': valor_p})

# Convertir los resultados a un DataFrame
resultados_levene_df = pd.DataFrame(resultados_levene)

# Definir las dimensiones de la figura
altura_grafico = 4.5  # Altura del gráfico en pulgadas
# Anchura del gráfico basada en una proporción agradable
anchura_grafico = altura_grafico * 1.618

# Crear el gráfico de barras con las dimensiones ajustadas
plt.figure(figsize=(anchura_grafico, altura_grafico))

# Sombrear la región de rechazo (p < 0.05) con trama
plt.fill_between(x=[-0.5, len(metricas)-0.5], y1=0, y2=0.05, color='none',
                 edgecolor='red', hatch='//', alpha=0.3, label='Región de␣
 ↪Rechazo (p < 0.05)')

# Sombrear la región de aceptación (p >= 0.05) con trama
plt.fill_between(x=[-0.5, len(metricas)-0.5], y1=0.05, y2=1, color='none',
                 edgecolor='green', hatch='\\\\', alpha=0.3, label='Región de␣
 ↪Aceptación (p >= 0.05)')

# Configurar el gráfico usando seaborn con el tema 'ticks' y la paleta 'Set3'
sns.set_theme(style="ticks")
grafico_barras = sns.barplot(data=resultados_levene_df, x='Métrica',
                             hue='Métrica', y='Valor P', palette='Set2',␣
 ↪alpha=1.0)

# Configurar el eje Y en escala logarítmica
plt.yscale('log')

# Definir los ticks específicos en el eje Y para la escala logarítmica
plt.yticks([1e-16, 0.001, 0.05, 1], ['1e-16', '0.001', '0.05', '1'])

# Ajustar los límites del eje Y para que el punto mínimo sea 0.0001 y el máximo␣
 ↪sea 1
plt.ylim(1e-17, 1)

# Añadir línea de umbral de significancia
plt.axhline(y=0.05, color='r', linestyle='--', label='Umbral de 0.05')

plt.xlabel('Métrica')
```

```
plt.ylabel('Valor P (Escala Logarítmica)')
# plt.title('Prueba de Homocedasticidad de Levene por Métrica entre Políticas')

# Ajustar la posición de la leyenda dentro del gráfico y darle un poco de␣
 ↪transparencia
plt.legend(loc='upper right', frameon=True, title="Regiones y Umbral")

plt.tight_layout()

# Mostrar el gráfico
plt.show()
```



[45]:
```
# Umbral de significancia
umbral_significancia = 0.05

# Evaluar normalidad por la prueba de Kolmogorov-Smirnov
resultados_ks = []

for politica in datos_tratados['Pol'].unique():
    for metrica in ['TTA', 'TTRc', 'TTRb', 'TTO']:
        # Estandarizar los datos para la prueba KS
        datos_estandarizados = (datos_tratados[datos_tratados['Pol'] ==␣
 ↪politica][metrica] - datos_tratados[datos_tratados['Pol']
```

```
                                == politica][metrica].mean()) /␣
 ↪datos_tratados[datos_tratados['Pol'] == politica][metrica].std()
        estadistico, valor_p = kstest(datos_estandarizados, 'norm')
        resultados_ks.append({'Política': politica, 'Métrica': metrica, 'Valor␣
 ↪P': round(
            valor_p, 4), 'Normalidad': 'Cumple' if valor_p >=␣
 ↪umbral_significancia else 'No Cumple'})

# Convertir los resultados de normalidad a un DataFrame
resultados_ks_df = pd.DataFrame(resultados_ks)

# Evaluar homocedasticidad por la prueba de Levene
resultados_levene = []

for metrica in ['TTA', 'TTRc', 'TTRb', 'TTO']:
    # Obtener los datos por política para la métrica actual
    grupos = [datos_tratados[datos_tratados['Pol'] == politica][metrica]
              for politica in datos_tratados['Pol'].unique()]
    estadistico, valor_p = levene(*grupos)
    resultados_levene.append({'Métrica': metrica, 'Valor P': round(
        valor_p, 4), 'Homocedasticidad': 'Cumple' if valor_p >=␣
 ↪umbral_significancia else 'No Cumple'})

# Convertir los resultados de homocedasticidad a un DataFrame
resultados_levene_df = pd.DataFrame(resultados_levene)

# Configurar pandas para mostrar los números sin notación científica y␣
 ↪redondear a 4 decimales
pd.set_option('display.float_format', '{:.4f}'.format)

# Mostrar resultados
print("\nResultados de la Prueba de Normalidad Kolmogorov-Smirnov por Política␣
 ↪y Métrica")
print(resultados_ks_df)

print("\nResultados de la Prueba de Homocedasticidad de Levene por Métrica")
print(resultados_levene_df)
```

```
Resultados de la Prueba de Normalidad Kolmogorov-Smirnov por Política y Métrica
   Política Métrica  Valor P Normalidad
0         G     TTA   0.7002     Cumple
1         G    TTRc   0.1092     Cumple
2         G    TTRb   0.7098     Cumple
3         G     TTO   0.4166     Cumple
4         K     TTA   0.6748     Cumple
5         K    TTRc   0.7084     Cumple
```

```
6          K    TTRb   0.9005     Cumple
7          K     TTO   0.3598     Cumple
8          P     TTA   0.7509     Cumple
9          P    TTRc   0.7170     Cumple
10         P    TTRb   0.4476     Cumple
11         P     TTO   0.4927     Cumple
12         Q     TTA   0.6101     Cumple
13         Q    TTRc   0.7255     Cumple
14         Q    TTRb   0.9048     Cumple
15         Q     TTO   0.8328     Cumple

Resultados de la Prueba de Homocedasticidad de Levene por Métrica
  Métrica  Valor P Homocedasticidad
0     TTA   0.1851           Cumple
1    TTRc   0.7740           Cumple
2    TTRb   0.0000        No Cumple
3     TTO   0.0000        No Cumple
```

6 Análisis de Varianza

6.1 ANOVA

Se realiza un análisis de varianza (ANOVA) para las métricas TTA y TTRc. Esta prueba determina si hay diferencias significativas en las medias de estas métricas entre las diferentes políticas de reubicación. Si el valor p es menor que 0.05, se rechaza la hipótesis nula de que todas las medias son iguales.

6.2 Prueba de Kruskal-Wallis

Para las métricas TTRb y TTO, que no cumplen con la suposición de normalidad, se utiliza la prueba de Kruskal-Wallis, una alternativa no paramétrica al ANOVA. Similarmente, un valor p menor que 0.05 indica diferencias significativas entre las políticas.

[46]:
```python
# Filtrar los datos relevantes
datos_anova = datos_tratados[['Pol', 'TTA', 'TTRc']]
datos_kruskal_ttrb = datos_tratados[['Pol', 'TTRb']]
datos_kruskal_tto = datos_tratados[['Pol', 'TTO']]

# Agrupar los datos por política para el ANOVA y la prueba de Kruskal-Wallis
grupos_tta = [grupo['TTA'].values for nombre,
              grupo in datos_anova.groupby('Pol')]
grupos_ttrc = [grupo['TTRc'].values for nombre,
               grupo in datos_anova.groupby('Pol')]
grupos_ttrb = [grupo['TTRb'].values for nombre,
               grupo in datos_kruskal_ttrb.groupby('Pol')]
grupos_tto = [grupo['TTO'].values for nombre,
              grupo in datos_kruskal_tto.groupby('Pol')]
```

```
# Realizar ANOVA para TTA usando f_oneway
anova_tta_stat, anova_tta_p_value = f_oneway(*grupos_tta)

# Realizar ANOVA para TTRc usando f_oneway
anova_ttrc_stat, anova_ttrc_p_value = f_oneway(*grupos_ttrc)

# Realizar la prueba de Kruskal-Wallis para TTRb
kruskal_ttrb_stat, kruskal_ttrb_p_value = kruskal(*grupos_ttrb)

# Realizar la prueba de Kruskal-Wallis para TTO
kruskal_tto_stat, kruskal_tto_p_value = kruskal(*grupos_tto)

# Crear un DataFrame con los resultados y las interpretaciones
resultados_anova_kruskal = pd.DataFrame({
    'Métrica': ['TTA', 'TTRc', 'TTRb', 'TTO'],
    'Prueba Utilizada': ['ANOVA', 'ANOVA', 'Kruskal-Wallis', 'Kruskal-Wallis'],
    'Estadístico de Prueba': [anova_tta_stat, anova_ttrc_stat,␣
 ↪kruskal_ttrb_stat, kruskal_tto_stat],
    'Valor p': [anova_tta_p_value, anova_ttrc_p_value, kruskal_ttrb_p_value,␣
 ↪kruskal_tto_p_value],
    'Interpretación': [
        "Rechaza H0" if anova_tta_p_value < 0.05 else "Acepta H0",
        "Rechaza H0" if anova_ttrc_p_value < 0.05 else "Acepta H0",
        "Rechaza H0" if kruskal_ttrb_p_value < 0.05 else "Acepta H0",
        "Rechaza H0" if kruskal_tto_p_value < 0.05 else "Acepta H0"
    ]
})

# Mostrar los resultados
print("Resultados del ANOVA para TTA y TTRc, y de la prueba de Kruskal-Wallis␣
 ↪para TTRb y TTO:")
resultados_anova_kruskal
```

```
Resultados del ANOVA para TTA y TTRc, y de la prueba de Kruskal-Wallis para TTRb
y TTO:
```

[46]:

	Métrica	Prueba Utilizada	Estadístico de Prueba	Valor p	Interpretación
0	TTA	ANOVA	58399.7808	0.0000	Rechaza H0
1	TTRc	ANOVA	10525.5642	0.0000	Rechaza H0
2	TTRb	Kruskal-Wallis	2928.4249	0.0000	Rechaza H0
3	TTO	Kruskal-Wallis	2447.5272	0.0000	Rechaza H0

7 Pruebas Post Hoc

7.1 Prueba de Tukey HSD

Después de un ANOVA significativo, se utiliza la prueba de Tukey HSD para realizar comparaciones múltiples entre las medias de las políticas. Esta prueba ayuda a identificar cuáles políticas difieren

significativamente entre sí.

8 Prueba de Dunn

Después de una prueba de Kruskal-Wallis significativa, se utiliza la prueba de Dunn con corrección de Bonferroni para realizar comparaciones múltiples. Esto ayuda a identificar diferencias específicas entre las políticas para métricas que no cumplen con la suposición de normalidad.

[47]:
```
# Realizar la prueba de Tukey HSD para TTA
tukey_TTA = pairwise_tukeyhsd(
    endog=datos_tratados['TTA'], groups=datos_tratados['Pol'], alpha=0.05)

# Realizar la prueba de Tukey HSD para TTRb
tukey_TTRc = pairwise_tukeyhsd(
    endog=datos_tratados['TTRc'], groups=datos_tratados['Pol'], alpha=0.05)

# Convertir los resultados a un dataframe para una mejor visualización
resultados_TTA = pd.DataFrame(
    data=tukey_TTA.summary().data[1:], columns=tukey_TTA.summary().data[0])
resultados_TTRc = pd.DataFrame(
    data=tukey_TTRc.summary().data[1:], columns=tukey_TTRc.summary().data[0])

# Devolver los dataframes de resultados
print('Resultados Tukey para TTA')
print(resultados_TTA, '\n')
print('Resultados Tukey para TTRc')
print(resultados_TTRc)
```

```
Resultados Tukey para TTA
  group1 group2   meandiff  p-adj      lower      upper  reject
0      G      K -6739.5540 0.0000 -6801.1637 -6677.9443    True
1      G      P -7415.7970 0.0000 -7477.4067 -7354.1873    True
2      G      Q     2.2380 0.9997   -59.3717    63.8477   False
3      K      P  -676.2430 0.0000  -737.8527  -614.6333    True
4      K      Q  6741.7920 0.0000  6680.1823  6803.4017    True
5      P      Q  7418.0350 0.0000  7356.4253  7479.6447    True

Resultados Tukey para TTRc
  group1 group2   meandiff  p-adj      lower      upper  reject
0      G      K  2591.2470 0.0000  2531.8671  2650.6269    True
1      G      P  3170.5000 0.0000  3111.1201  3229.8799    True
2      G      Q    13.7440 0.9337   -45.6359    73.1239   False
3      K      P   579.2530 0.0000   519.8731   638.6329    True
4      K      Q -2577.5030 0.0000 -2636.8829 -2518.1231    True
5      P      Q -3156.7560 0.0000 -3216.1359 -3097.3761    True
```

[48]:
```
def dunn_test(df, metric, group_col):
    # Preparar los datos
    groups = df[group_col].unique()
    data = {group: df[metric][df[group_col]
                              == group].values for group in groups}

    # Número de comparaciones
    num_comparisons = len(groups) * (len(groups) - 1) / 2

    # Calcular las medias de los rangos
    all_values = np.concatenate(list(data.values()))
    ranks = rankdata(all_values)
    group_ranks = {group: ranks[i * len(data[group]):(i + 1) * len(data[group])]
                   for i, group in enumerate(groups)}
    rank_means = {group: np.mean(group_ranks[group]) for group in groups}

    # Tamaño de muestra por grupo
    group_sizes = {group: len(data[group]) for group in groups}

    # Calcular el estadístico de Dunn
    results = []
    for i, group1 in enumerate(groups):
        for j, group2 in enumerate(groups):
            if i < j:
                mean_diff = rank_means[group1] - rank_means[group2]
                var_sum = (len(all_values) * (len(all_values) + 1) / 12) * \
                    (1/group_sizes[group1] + 1/group_sizes[group2])
                z = mean_diff / np.sqrt(var_sum)
                p = 2 * (1 - norm.cdf(abs(z)))
                results.append([group1, group2, z, p])

    # Corrección de Bonferroni
    bonferroni_results = []
    for result in results:
        bonferroni_p = min(result[3] * num_comparisons, 1.0)
        bonferroni_results.append(
            [result[0], result[1], result[2], bonferroni_p])

    # Convertir a DataFrame para mostrar los resultados
    result_df = pd.DataFrame(bonferroni_results, columns=[
                             f'Grupo 1', f'Grupo 2', f'Estadístico Z',␣
 ↪f'P-valor (Bonferroni)'])
    return result_df


# Aplicar el test de Dunn con corrección de Bonferroni para las métricas TTRb y␣
 ↪TTO
```

```
dunn_ttrb = dunn_test(datos_tratados, 'TTRb', 'Pol')
dunn_tto = dunn_test(datos_tratados, 'TTO', 'Pol')


dunn_ttrb, dunn_tto
```

[48]:
```
(  Grupo 1 Grupo 2  Estadístico Z  P-valor (Bonferroni)
 0       G       K       -46.3196                0.0000
 1       G       P       -22.6387                0.0000
 2       G       Q        -0.1453                1.0000
 3       K       P        23.6810                0.0000
 4       K       Q        46.1743                0.0000
 5       P       Q        22.4934                0.0000,
   Grupo 1 Grupo 2  Estadístico Z  P-valor (Bonferroni)
 0       G       K       -34.8887                0.0000
 1       G       P        11.9581                0.0000
 2       G       Q        -0.4491                1.0000
 3       K       P        46.8467                0.0000
 4       K       Q        34.4396                0.0000
 5       P       Q       -12.4071                0.0000)
```

9 Cálculo de Porcentajes Relativos

Este bloque calcula los porcentajes relativos para cada métrica y política de reubicación en comparación con el valor medio mínimo. Esto proporciona una visión clara de cómo se comparan las diferentes políticas en relación con la métrica con el valor medio mínimo, facilitando la interpretación de la eficiencia relativa de cada política.

[50]:
```
# Función para calcular los porcentajes relativos para cada métrica
def calcular_porcentajes_relativos(df, metrica):
    # Agrupar por 'Pol' y calcular la media para la métrica especificada
    valores_medios = df.groupby("Pol")[metrica].mean()

    # Encontrar el valor medio mínimo y usarlo como base para calcular los␣
 ↪porcentajes
    valor_medio_minimo = valores_medios.min()

    # Calcular el porcentaje relativo para cada política
    porcentajes_relativos = (valores_medios / valor_medio_minimo) - 1

    return porcentajes_relativos.reset_index().rename(
        columns={metrica: f"{metrica}_Relativo"}
    )


# Métricas a analizar
metricas = ["TTA", "TTRc", "TTRb", "TTO"]
```

```
# Calcular los porcentajes relativos para cada métrica
porcentajes_relativos = {}
for metrica in metricas:
    porcentajes_relativos[metrica] = calcular_porcentajes_relativos(
        datos_tratados, metrica
    )

# Fusionar todos los dataframes de porcentajes relativos en un solo dataframe
resultado = porcentajes_relativos["TTA"]
for metrica in metricas[1:]:
    resultado = resultado.merge(porcentajes_relativos[metrica], on="Pol")

print("Diferencias relativas")
resultado
```

```
Diferencias relativas
```

[50]:

	Pol	TTA_Relativo	TTRc_Relativo	TTRb_Relativo	TTO_Relativo
0	G	0.0629	0.0000	0.0000	0.0052
1	K	0.0057	0.0203	0.5504	0.0935
2	P	0.0000	0.0248	0.0462	0.0000
3	Q	0.0629	0.0001	0.0002	0.0053