DEPENDENCIAS

In [2]:
```
import numpy as np
print(f"Version de Numpy: {np.__version__}")
import pandas as pd
print(f"Version de Pandas: {pd.__version__}")
import scipy
import scipy.stats as stats
print(f"Version de Scipy: {scipy.__version__}")
import statsmodels.api as sm
print(f"Version de Statsmodels: {sm.__version__}")
import statsmodels.formula.api as smf
import seaborn as sns
print(f"Version de Seaborn: {sns.__version__}")
import matplotlib
import matplotlib.pyplot as plt
print(f"Version de Matplotlib: {matplotlib.__version__}")
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import scikit_posthocs as sp
print(f"Version de Scikit posthocs: {sp.__version__}")
```

```
Version de Numpy: 1.26.4
Version de Pandas: 2.2.2
Version de Scipy: 1.13.1
Version de Statsmodels: 0.14.2
Version de Seaborn: 0.13.2
Version de Matplotlib: 3.9.2
Version de Scikit posthocs: 0.11.3
```

IMPORTO LOS DATOS Y LOS FILTRO

In [4]:
```
nombre_archivo = "datos.xlsx"
df = pd.read_excel(nombre_archivo)
df = df.loc[df["cep"] != "control"]
df = df.dropna(subset=["conc de bsps"])
print(df.info())
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 324 entries, 0 to 359
Data columns (total 13 columns):
 #   Column                                     Non-Null Count  Dtype  
---  ------                                     --------------  -----  
 0   cep                                        324 non-null    object 
 1   fecha inoculacion                          324 non-null    object 
 2   fecha medicion                             324 non-null    object 
 3   minutos transcurridos                      324 non-null    int64  
 4   lote                                       324 non-null    object 
 5   id u.e.                                    324 non-null    object 
 6   viabilidad (cnds germinados/cnds contados) 252 non-null    float64
 7   hora medicion                              324 non-null    int64  
 8   fuente de N                                324 non-null    object 
 9   conc de bsps                               324 non-null    float64
 10  log bsps                                   324 non-null    float64
 11  rotulo epp                                 162 non-null    float64
 12  peso biomasa seca                          162 non-null    float64
dtypes: float64(5), int64(2), object(6)
memory usage: 35.4+ KB
None
```

Out[4]:

	minutos transcurridos	viabilidad (cnds germinados/cnds contados)	hora medicion	conc de bsps	log bsps	rotulo epp	peso biomasa seca
count	324.000000	252.000000	324.000000	3.240000e+02	324.000000	162.000000	162.000000
mean	4313.148148	0.996143	72.000000	3.735719e+08	7.936603	98.049383	0.035414
std	1178.268952	0.009467	19.626229	4.724776e+08	0.867697	61.427006	0.008815
min	2836.000000	0.973000	48.000000	1.250000e+06	6.096910	0.000000	0.019000
25%	2881.000000	1.000000	48.000000	1.995000e+07	7.299664	45.250000	0.028000
50%	4314.000000	1.000000	72.000000	5.730000e+07	7.757709	85.500000	0.035000
75%	5749.000000	1.000000	96.000000	9.530000e+08	8.978865	155.750000	0.042750
max	5769.000000	1.000000	96.000000	1.360000e+09	9.134337	205.000000	0.060000

SELECCION DEL MEJOR MEDIO DE CULTIVO PARA CADA CEPA

AGRUPO LOS DATOS DE PRODUCCION DE BLASTOSPORAS POR CEPA Y POR MEDIO DE CULTIVO REALIZO TEST DE LEVENE (HOMOGENEIDAD DE VARIANZA) Y SHAPIRO WILK (NORMALIDAD)

In [6]:
```
# Filtrar por cepa y evaluar normalidad y homocedasticidad
for cepa in df["cep"].unique():
    print(f"\nResultados para cepa {cepa}:")

    # Filtrar datos por cepa
```

```
    df_cepa = df[df["cep"] == cepa]

    # Evaluar normalidad para cada "fuente de N"
    print("\nPrueba de Shapiro-Wilk (normalidad):")
    normalidad = {}
    for fuente in df_cepa["fuente de N"].unique():
        datos = df_cepa[df_cepa["fuente de N"] == fuente]["conc de bsps"]
        stat, p_value = stats.shapiro(datos)
        normalidad[fuente] = p_value
        print(f"  {fuente}: p = {p_value:.4f}")

    # Evaluar homocedasticidad con prueba de Levene
    print("\nPrueba de Levene (homocedasticidad):")
    grupos = [df_cepa[df_cepa["fuente de N"] == fuente]["conc de bsps"] for fuente in df_cepa["fuente de N"].unique()]
    stat, p_value = stats.levene(*grupos)
    print(f"  p = {p_value:.4f}")

    # Interpretación
    if all(p > 0.05 for p in normalidad.values()):
        print("  → No se rechaza la normalidad en ningún grupo (p > 0.05)")
    else:
        print("  → Al menos un grupo no es normal (p < 0.05)")

    if p_value > 0.05:
        print("  → No se rechaza la homocedasticidad (varianzas iguales, p > 0.05)")
    else:
        print("  → Se rechaza la homocedasticidad (varianzas diferentes, p < 0.05)")
```

```
Resultados para cepa 350:

Prueba de Shapiro-Wilk (normalidad):
  levadura nutricional: p = 0.5266
  harina de soja: p = 0.0001
  pure de papas: p = 0.0000

Prueba de Levene (homocedasticidad):
  p = 0.0000
  → Al menos un grupo no es normal (p < 0.05)
  → Se rechaza la homocedasticidad (varianzas diferentes, p < 0.05)

Resultados para cepa 160:

Prueba de Shapiro-Wilk (normalidad):
  levadura nutricional: p = 0.7566
  harina de soja: p = 0.0010
  pure de papas: p = 0.0116

Prueba de Levene (homocedasticidad):
  p = 0.0000
  → Al menos un grupo no es normal (p < 0.05)
  → Se rechaza la homocedasticidad (varianzas diferentes, p < 0.05)

Resultados para cepa 416:

Prueba de Shapiro-Wilk (normalidad):
  levadura nutricional: p = 0.4683
  harina de soja: p = 0.0038
  pure de papas: p = 0.0173

Prueba de Levene (homocedasticidad):
  p = 0.0000
  → Al menos un grupo no es normal (p < 0.05)
  → Se rechaza la homocedasticidad (varianzas diferentes, p < 0.05)
```

No se cumplen los supuestos de homocedasticidad y normalidad para ningun grupo.

TEST DE KRUSKAL WALIS

El test de Kruskal Walis se utiliza como sustituto de ANOVA cuando los datos no provienen de distribuciones normales. Rankea los datos de mayor a menor, calculo el ranking promedio para cada grupo y lo compara.

In [8]:
```
# Recorrer cada cepa y aplicar la prueba de Kruskal-Wallis
for cepa in df["cep"].unique():
    print(f"\nResultados para cepa {cepa}:")

    # Filtrar datos por cepa
    df_cepa = df[df["cep"] == cepa]

    # Agrupar los datos por "fuente de N"
    grupos = [df_cepa[df_cepa["fuente de N"] == fuente]["conc de bsps"] for fuente in df_cepa["fuente de N"].unique()]

    # Aplicar la prueba de Kruskal-Wallis
    stat, p_value = stats.kruskal(*grupos)

    # Calcular grados de libertad
    df_numerator = len(grupos) - 1
    df_denominator = len(df_cepa) - len(grupos)

    # Mostrar resultados
    print(f"  Estadístico H = {stat:.4f}, p = {p_value:.4f}, df_numerator = {df_numerator}, df_denominator = {df_denominator}")
```

```
    # Interpretación del p-valor
    if p_value < 0.05:
        print("  → Se rechaza la hipótesis nula: hay diferencias significativas entre al menos dos grupos.")
    else:
        print("  → No se rechaza la hipótesis nula: no hay diferencias significativas entre los grupos.")
```

```
Resultados para cepa 350:
  Estadístico H = 92.8181, p = 0.0000, df_numerator = 2, df_denominator = 105
  → Se rechaza la hipótesis nula: hay diferencias significativas entre al menos dos grupos.

Resultados para cepa 160:
  Estadístico H = 71.3984, p = 0.0000, df_numerator = 2, df_denominator = 105
  → Se rechaza la hipótesis nula: hay diferencias significativas entre al menos dos grupos.

Resultados para cepa 416:
  Estadístico H = 93.9693, p = 0.0000, df_numerator = 2, df_denominator = 105
  → Se rechaza la hipótesis nula: hay diferencias significativas entre al menos dos grupos.
```

Habiendo comprobado que hay diferencias significativas entre al menos dos grupos para cada cepa, se realiza una prueba a posteriori para verificar diferencias significativas entre los grupos. La prueba de Mann-Whitney-Wilcoxon vuelve a rankear los grupos para comparar de a pares, distinto del propio test de K-W, equivaldria a usar datos diferentes, por lo que es deficiente aun con la correccion de Bonferroni.

TEST DE DUNN

In [10]:
```
# Recorrer cada cepa y aplicar Kruskal-Wallis seguido de Dunn
for cepa in df["cep"].unique():
    print(f"\nResultados para cepa {cepa}:")

    # Filtrar datos por cepa
    df_cepa = df[df["cep"] == cepa]

    # Aplicar Kruskal-Wallis
    stat, p_value = stats.kruskal(*[df_cepa[df_cepa["fuente de N"] == fuente]["conc de bsps"] for fuente in df_cepa["fuente de N"].unique()])

    print(f"  Kruskal-Wallis: H = {stat:.4f}, p = {p_value:.4f}")

    if p_value < 0.05:
        print("  → Hay diferencias significativas, se procede con la prueba de Dunn.")

        # Aplicar la prueba de Dunn con corrección de Bonferroni
        dunn_pvals = sp.posthoc_dunn(df_cepa, val_col="conc de bsps", group_col="fuente de N", p_adjust="bonferroni")

        print("\n  Comparaciones post hoc (Dunn con Bonferroni):")
        print(dunn_pvals)
    else:
        print("  → No hay diferencias significativas, no se realiza prueba post hoc.")
```

```
Resultados para cepa 350:
  Kruskal-Wallis: H = 92.8181, p = 0.0000
  → Hay diferencias significativas, se procede con la prueba de Dunn.

  Comparaciones post hoc (Dunn con Bonferroni):
                     harina de soja  levadura nutricional  pure de papas
harina de soja             1.000000          1.725288e-06   1.083898e-05
levadura nutricional       0.000002          1.000000e+00   1.759314e-21
pure de papas              0.000011          1.759314e-21   1.000000e+00

Resultados para cepa 160:
  Kruskal-Wallis: H = 71.3984, p = 0.0000
  → Hay diferencias significativas, se procede con la prueba de Dunn.

  Comparaciones post hoc (Dunn con Bonferroni):
                     harina de soja  levadura nutricional  pure de papas
harina de soja          1.000000e+00          1.702157e-12   1.000000e+00
levadura nutricional    1.702157e-12          1.000000e+00   3.445514e-13
pure de papas           1.000000e+00          3.445514e-13   1.000000e+00

Resultados para cepa 416:
  Kruskal-Wallis: H = 93.9693, p = 0.0000
  → Hay diferencias significativas, se procede con la prueba de Dunn.

  Comparaciones post hoc (Dunn con Bonferroni):
                     harina de soja  levadura nutricional  pure de papas
harina de soja             1.000000          2.381015e-06   5.915253e-06
levadura nutricional       0.000002          1.000000e+00   9.666639e-22
pure de papas              0.000006          9.666639e-22   1.000000e+00
```

FUENTE DE NITROGENO: RESULTADOS

CEP 160: PURE DE PAPAS = HARINA DE SOJA < LEVADURA NUTRICIONAL

CEP 350: PURE DE PAPAS < HARINA DE SOJA < LEVADURA NUTRICIONAL

CEP 416: PURE DE PAPAS < HARINA DE SOJA < LEVADURA NUTRICIONAL

FILTRO EL DATAFRAME PARA "LEVADURA NUTRICIONAL"

In [13]:
```
df_lev_nut = df[df["fuente de N"] == "levadura nutricional"]
```

CHEQUEO DE SUPUESTOS

Chequeo que la concentracion de blastosporas para cada cepa entre los distintos tiempos de medicion cumplan con homocedasticidad (test de Levene) y con normalidad (test de Shapiro-Wilk)

In [15]:
```
# Recorrer cada cepa y aplicar los tests de Levene y Shapiro-Wilk
for cepa in df_lev_nut["cep"].unique():
    print(f"\nResultados para cepa {cepa}:")

    # Filtrar datos por cepa
    df_cepa = df_lev_nut[df_lev_nut["cep"] == cepa]

    # Test de Levene para homocedasticidad (varianza igual entre horas de medición)
    stat_levene, p_levene = stats.levene(*[df_cepa[df_cepa["hora medicion"] == hora]["conc de bsps"] for hora in df_cepa["hora medicion"].u

    print(f"  Test de Levene: W = {stat_levene:.4f}, p = {p_levene:.4f}")
    if p_levene < 0.05:
        print("  → Se rechaza la homocedasticidad (las varianzas no son iguales).")
    else:
        print("  → No se rechaza la homocedasticidad (las varianzas son similares).")

    # Test de Shapiro-Wilk para normalidad en cada tiempo de medición
    for hora in df_cepa["hora medicion"].unique():
        stat_shapiro, p_shapiro = stats.shapiro(df_cepa[df_cepa["hora medicion"] == hora]["conc de bsps"])

        print(f"  Shapiro-Wilk para hora {hora}: W = {stat_shapiro:.4f}, p = {p_shapiro:.4f}")
        if p_shapiro < 0.05:
            print("  → Se rechaza la normalidad para esta hora.")
        else:
            print("  → No se rechaza la normalidad para esta hora.")
```

```
Resultados para cepa 350:
  Test de Levene: W = 0.0983, p = 0.9066
  → No se rechaza la homocedasticidad (las varianzas son similares).
  Shapiro-Wilk para hora 48: W = 0.9337, p = 0.4207
  → No se rechaza la normalidad para esta hora.
  Shapiro-Wilk para hora 72: W = 0.9217, p = 0.3004
  → No se rechaza la normalidad para esta hora.
  Shapiro-Wilk para hora 96: W = 0.9530, p = 0.6815
  → No se rechaza la normalidad para esta hora.

Resultados para cepa 160:
  Test de Levene: W = 0.4890, p = 0.6176
  → No se rechaza la homocedasticidad (las varianzas son similares).
  Shapiro-Wilk para hora 48: W = 0.9127, p = 0.2308
  → No se rechaza la normalidad para esta hora.
  Shapiro-Wilk para hora 72: W = 0.8922, p = 0.1260
  → No se rechaza la normalidad para esta hora.
  Shapiro-Wilk para hora 96: W = 0.8986, p = 0.1523
  → No se rechaza la normalidad para esta hora.

Resultados para cepa 416:
  Test de Levene: W = 2.0750, p = 0.1416
  → No se rechaza la homocedasticidad (las varianzas son similares).
  Shapiro-Wilk para hora 48: W = 0.8625, p = 0.0526
  → No se rechaza la normalidad para esta hora.
  Shapiro-Wilk para hora 72: W = 0.9507, p = 0.6476
  → No se rechaza la normalidad para esta hora.
  Shapiro-Wilk para hora 96: W = 0.9331, p = 0.4141
  → No se rechaza la normalidad para esta hora.
```

In [16]:
```
# Iteramos sobre cada cepa en el dataframe filtrado
for cepa in df_lev_nut["cep"].unique():
    print(f"\nResultados para cepa {cepa}:")

    # Filtrar datos por cepa
    df_cepa = df_lev_nut[df_lev_nut["cep"] == cepa]

    # Agrupar los datos por 'hora medicion'
    grupos = [df_cepa[df_cepa["hora medicion"] == hora]["conc de bsps"] for hora in df_cepa["hora medicion"].unique()]

    # ANOVA de una vía
    F_stat, p_value = stats.f_oneway(*grupos)

    # Cálculo de grados de libertad
    df_numerator = len(grupos) - 1  # Número de niveles de 'hora medicion' - 1
    df_denominator = len(df_cepa) - len(grupos)  # Total de observaciones - número de niveles

    print(f"  ANOVA: F = {F_stat:.4f}, p = {p_value:.4f}, df_numerator = {df_numerator}, df_denominator = {df_denominator}")

    # Si el ANOVA es significativo (p < 0.05), proceder con Tukey HSD
    if p_value < 0.05:
        print("  → Diferencias significativas detectadas, se procede con el test de Tukey.")

        tukey = pairwise_tukeyhsd(df_cepa["conc de bsps"], df_cepa["hora medicion"])
        print(tukey)
    else:
        print("  → No se encontraron diferencias significativas, no se realiza el test de Tukey.")
```

```
Resultados para cepa 350:
  ANOVA: F = 0.5452, p = 0.5849, df_numerator = 2, df_denominator = 33
  → No se encontraron diferencias significativas, no se realiza el test de Tukey.

Resultados para cepa 160:
  ANOVA: F = 1.3573, p = 0.2714, df_numerator = 2, df_denominator = 33
  → No se encontraron diferencias significativas, no se realiza el test de Tukey.

Resultados para cepa 416:
  ANOVA: F = 2.6051, p = 0.0890, df_numerator = 2, df_denominator = 33
  → No se encontraron diferencias significativas, no se realiza el test de Tukey.
```

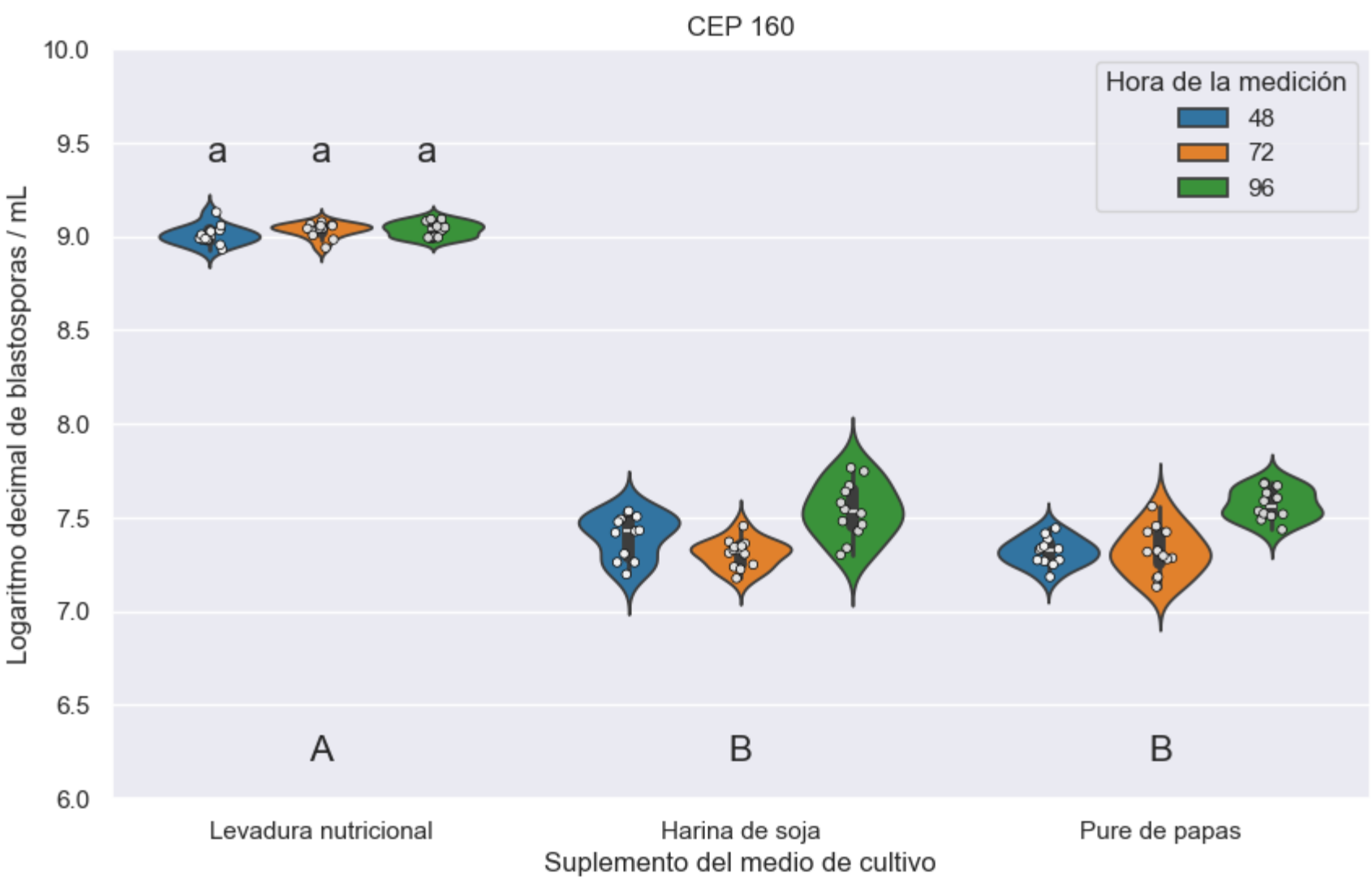
GRAFICOS

Realizo graficos de violin con puntos de distribucion para cada cepa, presentando los resultados de los test con Compact Letter Display (Piepho 2004)

Letras mayúsculas para mostrar diferencias significativas entre suplementos para los medios de cultivo (Dunn test with Bonferroni corrections, P<0.05)

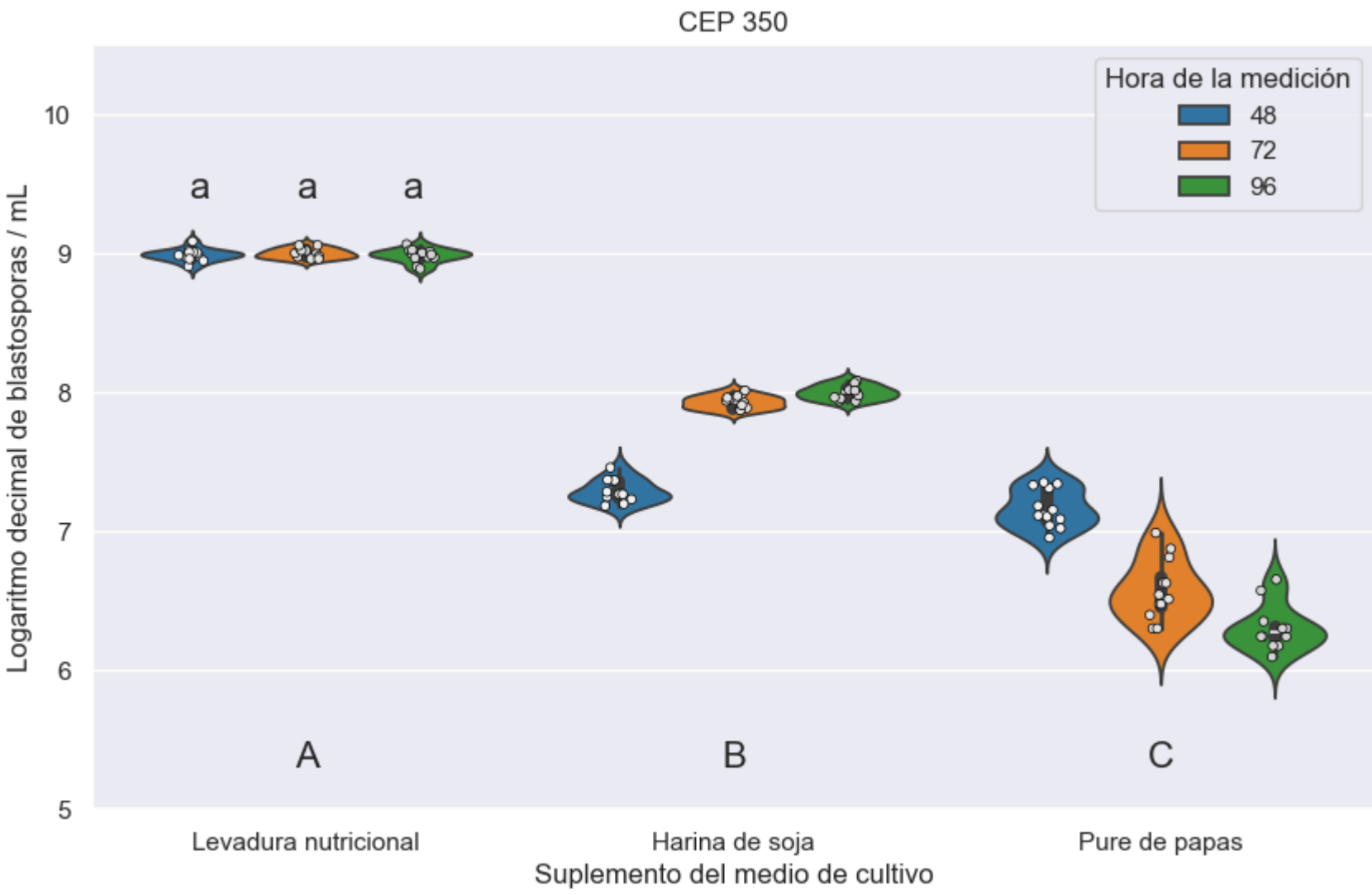
Letras minúsculas para mostrar diferencias significativas entre los distintos tiempos en el mejor suplemento de medio de cultivo (ANOVA no encontro diferencias significativas, no se realiza el test HSD de Tukey)

In [94]:
```
paleta_gris = sns.light_palette("gray", n_colors=9)
plt.figure(figsize=(10, 6))
sns.set()
sns.violinplot(data=df[(df["cep"] == 160)], x="fuente de N", y="log bsps", hue="hora medicion", palette="tab10",
               cut=3, gap=0.1, inner="box", density_norm= "width")
sns.stripplot(data=df[(df["cep"] == 160)], x="fuente de N", y="log bsps", hue="hora medicion", palette=paleta_gris[:3],
              linewidth=.5, edgecolor= "black", size=4, jitter=True, dodge= True, legend= False)
plt.text(0, 6.2, "A", fontsize=16, ha='center')
plt.text(1, 6.2, "B", fontsize=16, ha='center')
plt.text(2, 6.2, "B", fontsize=16, ha='center')
plt.text(0.25, 9.4, "a", fontsize=16, ha='center')
plt.text(0, 9.4, "a", fontsize=16, ha='center')
plt.text(-0.25, 9.4, "a", fontsize=16, ha='center')
plt.xlabel('Suplemento del medio de cultivo')
plt.ylabel('Logaritmo decimal de blastosporas / mL')
plt.legend(title="Hora de la medición", loc="upper right")
plt.ylim([6, 10])
ax = plt.gca()
ax.set_xticks([0, 1, 2])
ax.set_xticklabels(["Levadura nutricional", "Harina de soja", "Pure de papas"])
plt.title("CEP 160")
plt.show()
```



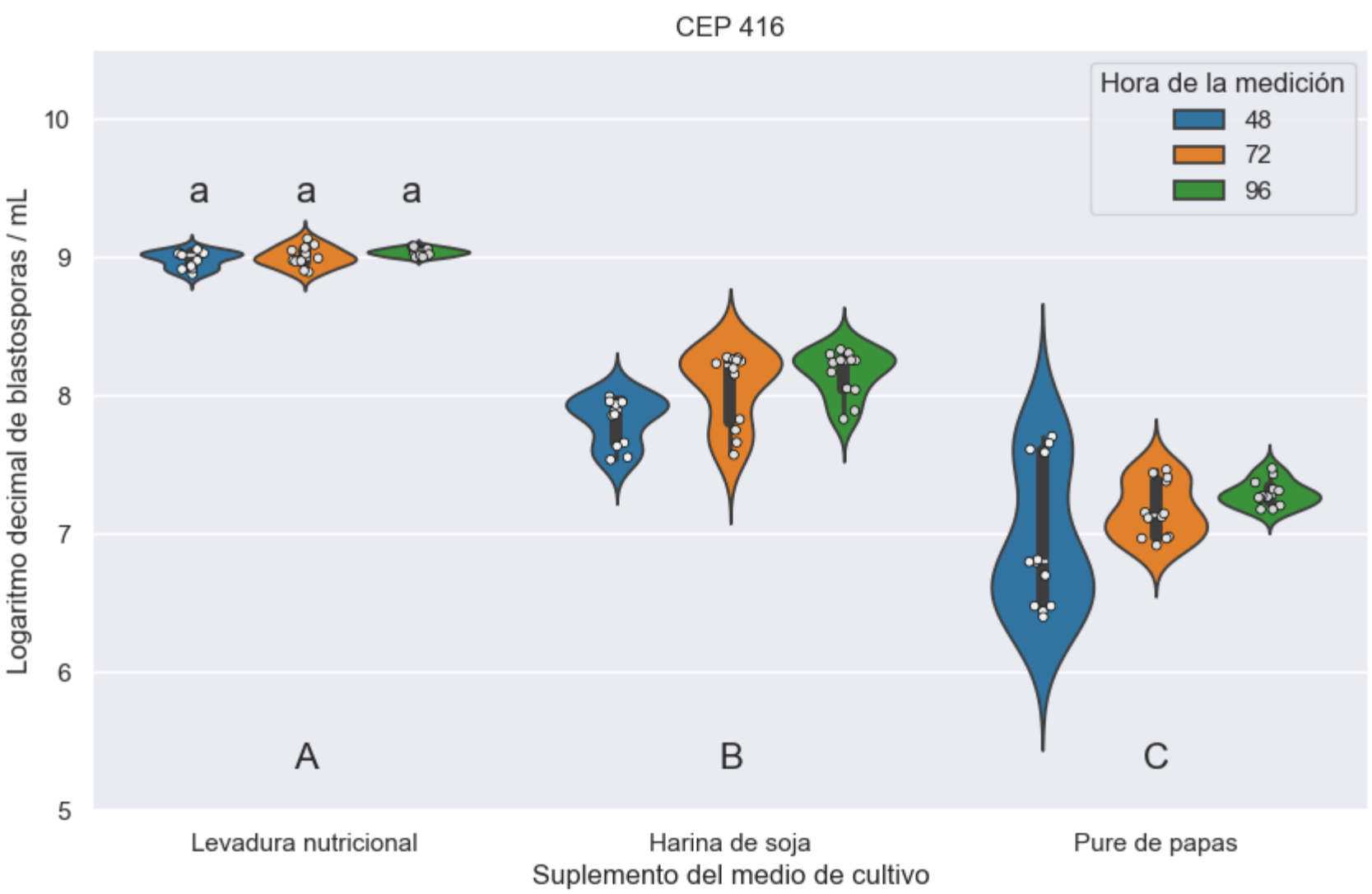
In [106…
```
plt.figure(figsize=(10, 6))
sns.set()
sns.violinplot(data=df[(df["cep"] == 350)], x="fuente de N", y="log bsps", hue="hora medicion", palette="tab10",
               cut=3, gap=0.1, inner="box", density_norm= "width")
sns.stripplot(data=df[(df["cep"] == 350)], x="fuente de N", y="log bsps", hue="hora medicion", palette=paleta_gris[:3],
              linewidth=.5, edgecolor= "black", size=4, jitter=True, dodge= True, legend= False)
plt.text(0, 5.3, "A", fontsize=16, ha='center')
plt.text(1, 5.3, "B", fontsize=16, ha='center')
plt.text(2, 5.3, "C", fontsize=16, ha='center')
plt.text(0.25, 9.4, "a", fontsize=16, ha='center')
plt.text(0, 9.4, "a", fontsize=16, ha='center')
plt.text(-0.25, 9.4, "a", fontsize=16, ha='center')
plt.xlabel('Suplemento del medio de cultivo')
```

```
plt.ylabel('Logaritmo decimal de blastosporas / mL')
plt.legend(title="Hora de la medición", loc="upper right")
plt.ylim([5, 10.5])
ax = plt.gca()
ax.set_xticks([0, 1, 2])
ax.set_xticklabels(["Levadura nutricional", "Harina de soja", "Pure de papas"])
plt.title("CEP 350")
plt.show()
```



In [108…

```
plt.figure(figsize=(10, 6))
sns.set()
sns.violinplot(data=df[(df["cep"] == 416)], x="fuente de N", y="log bsps", hue="hora medicion", palette="tab10",
               cut=3, gap=0.1, inner="box", density_norm= "width")
sns.stripplot(data=df[(df["cep"] == 416)], x="fuente de N", y="log bsps", hue="hora medicion", palette=paleta_gris[:3],
              linewidth=.5, edgecolor= "black", size=4, jitter=True, dodge= True, legend= False)
plt.text(0, 5.3, "A", fontsize=16, ha='center')
plt.text(1, 5.3, "B", fontsize=16, ha='center')
plt.text(2, 5.3, "C", fontsize=16, ha='center')
plt.text(0.25, 9.4, "a", fontsize=16, ha='center')
plt.text(0, 9.4, "a", fontsize=16, ha='center')
plt.text(-0.25, 9.4, "a", fontsize=16, ha='center')
plt.xlabel('Suplemento del medio de cultivo')
plt.ylabel('Logaritmo decimal de blastosporas / mL')
plt.legend(title="Hora de la medición", loc="upper right")
plt.ylim([5, 10.5])
ax = plt.gca()
ax.set_xticks([0, 1, 2])
ax.set_xticklabels(["Levadura nutricional", "Harina de soja", "Pure de papas"])
plt.title("CEP 416")
plt.show()
```


CEP 416
Hora de la medición
48
72
96
Logaritmo decimal de blastosporas / mL
10
9
8
7
6
5
a
a
a
A
B
C
Levadura nutricional
Harina de soja
Pure de papas
Suplemento del medio de cultivo