UNIVERSIDAD DE MURCIA

Segundo Informe

Avance del trabajo académicamente dirigido respecto al primer informe

Juan Luis Serradilla Tormos
UMU

Murcia, España
noviembre de 2024

Índice

1. Introducción

En este informe se explicarán todos los avances que se han realizado desde el primer informe del trabajo académicamente dirigido.

2. Enfoque del trabajo

Se ha concretado más el enfoque del trabajo. Este seguirá siendo sobre la comparación de las librerías Matplotlib, Seaborn y Plotly. Sin embargo, se han concretado las características a analizar de estas librerías.

- Facilidad de uso y curva de aprendizaje: Se comparará lo intuitivo que es de usar cada librería y lo fácil que es de usar para gente nueva. Se compará esta dificultad para diferentes tipos de gráficos, desde los más sencillos a los más complicados.
- Capacidad de visualización: Se comparará la cantidad de gráficos que pueden generarse con cada librería.
- Personalización: Se evaluará la cantidad de características disponibles para la personalización de los diferentes tipos de gráficos.
- Rendimiento: Se evaluará el rendimiento y velocidad de cómputo para grandes cantidades de datos.

Para poder realizar estas comparaciones se usarán los siguientes tipos de gráficos:

- Gráficos estadísticos: Se usarán diferentes tipos de gráficos estadísticos. Entre estos gráficos estadísticos encontraremos gráficos de dispersión, histogramas, hisrogramas 2D y gráficos de violín.
- Gráficos 3D: Se usarán gráficos 3D para poder comprobar las capacidades interactivas de la librería Plotly.

Para poder comprobar realizar los diferentes tipos de gráficos que se requieren para las comparaciones, se usarán los siguientes datasets conseguidos en Kaggle:

- Rendimiento escolar: Se usará un dataset con datos sobre alumnos de instituto y su rendimiento, para así poder realizar difernetes tipos de gráficos estadísticos.
- Diagnósticos de cáncer: Se usará un dataset con diferentes tipos de diagnósicos de cáncer y sus características, para así poder realizar análissi de variables numéricas relacionadas.
- Reproducciones de Spotify: Se usará un dataset con reproducciones de Spotify para poder realizar gráficos de líneas con las diferentes librerías.

3. Resultados

A continuación, se mostrarán los gráficos realizados con cada librería y se adjuntará el código correspondiente.

3.1. Gráficos de líneas

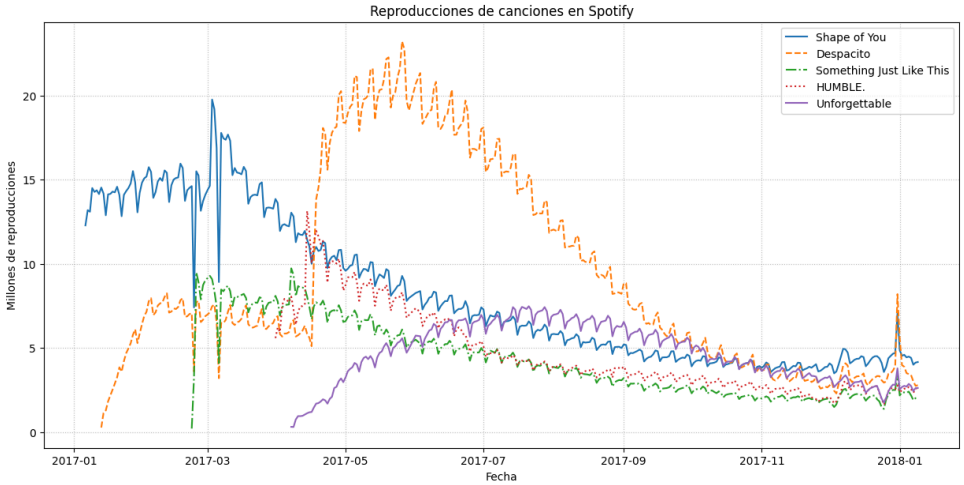

Figura 1: Gráfico de líneas con Matplotlib.

```
songs = data_spotify.columns
linestyle = ["-", "--", "-.", ":", "-"]
plt.figure(figsize=(15, 7))
for ls,song in zip(linestyle, songs):
    plt.plot(data_spotify[song] * 1e-6, label=song, ls=ls)
    plt.title("Reproducciones de canciones en Spotify")
    plt.ylabel("Millones de reproducciones")
    plt.xlabel("Fecha")
    plt.grid(ls=":", alpha=.5, color="grey")
    plt.legend()
plt.show()
```

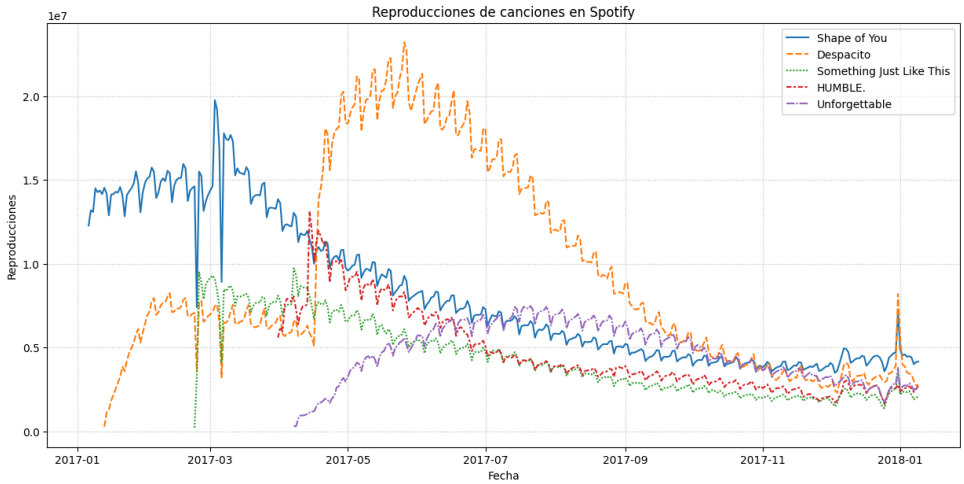

Figura 2: Gráfico de líneas con Seaborn.

```
plt.figure(figsize=(15, 7))
sns.lineplot(data=data_spotify)
plt.title("Reproducciones de canciones en Spotify")
plt.xlabel("Fecha")
plt.ylabel("Reproducciones")
plt.grid(ls=":", alpha=.5, color="grey")
plt.show()
```

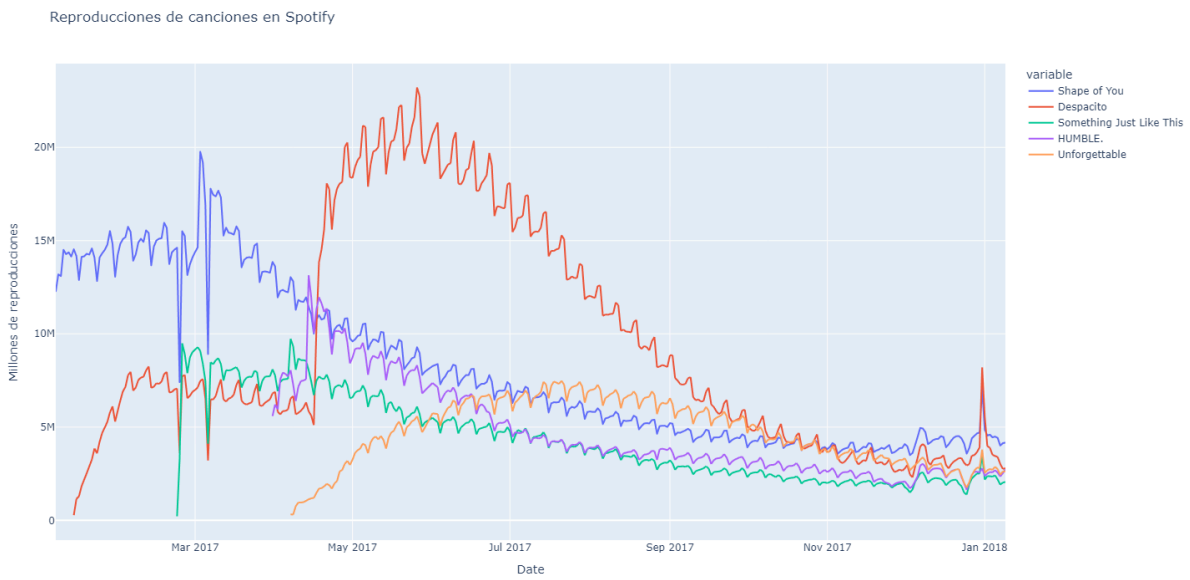

Figura 3: Gráfico de líneas con Plotly.

```
fig = px.line(data_spotify, title='Reproducciones de canciones en Spotify',
labels={"index": "Fecha", "value": "Millones de reproducciones"})
fig.update_layout(height=750, width=1450)
fig.show()
```

3.2. Histogramas

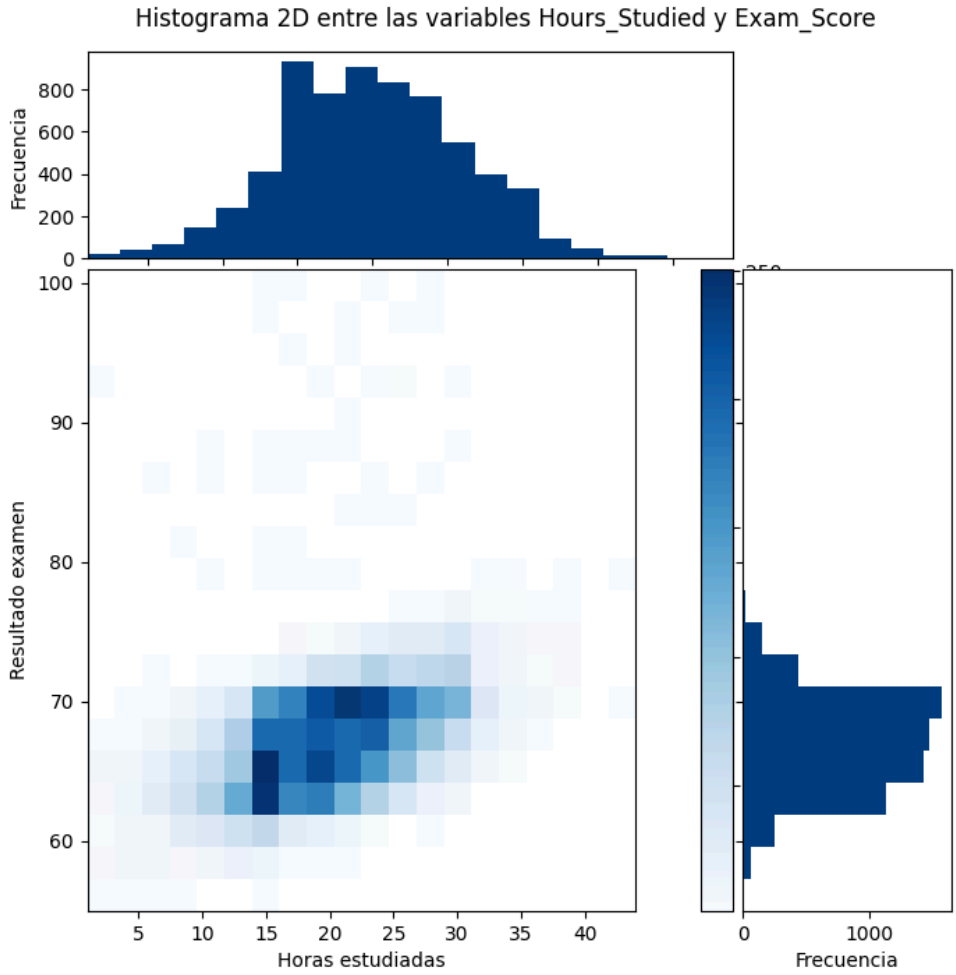

Figura 4: Histograma con Matplotlib.

```
# Crear la figura y los subgráficos
fig = plt.figure(figsize=(8, 8))
grid = fig.add_gridspec(4, 4, hspace=0.05, wspace=0.05)

# Crear el gráfico principal 2D (heatmap) en el centro
ax_main = fig.add_subplot(grid[1:4, 0:3])
hb = ax_main.hist2d(data_students.Hours_Studied, data_students.Exam_Score, bins=20,
cmap="Blues", cmin=1)
plt.colorbar(hb[3], ax=ax_main, orientation='vertical', fraction=0.05, pad=0.1,
label="Frecuencia")

# Crear el histograma marginal para el eje X (Hours_Studied)
ax_xhist = fig.add_subplot(grid[0, 0:3], sharex=ax_main)
ax_xhist.hist(data_students.Hours_Studied, bins=20, color="#004080")
ax_xhist.set_ylabel("Frecuencia")
ax_xhist.tick_params(axis="x", labelbottom=False)

# Crear el histograma marginal para el eje Y (Exam_Score)
ax_yhist = fig.add_subplot(grid[1:4, 3], sharey=ax_main)
ax_yhist.hist(data_students.Exam_Score, bins=20,
color="#004080", orientation='horizontal')
ax_yhist.set_xlabel("Frecuencia")
ax_yhist.tick_params(axis="y", labelleft=False)
```

```
# Etiquetas y título
ax_main.set_xlabel("Horas estudiadas")
ax_main.set_ylabel("Resultado examen")
fig.suptitle("Histograma 2D entre las variables Hours_Studied y Exam_Score", y=0.92)

plt.show()
```

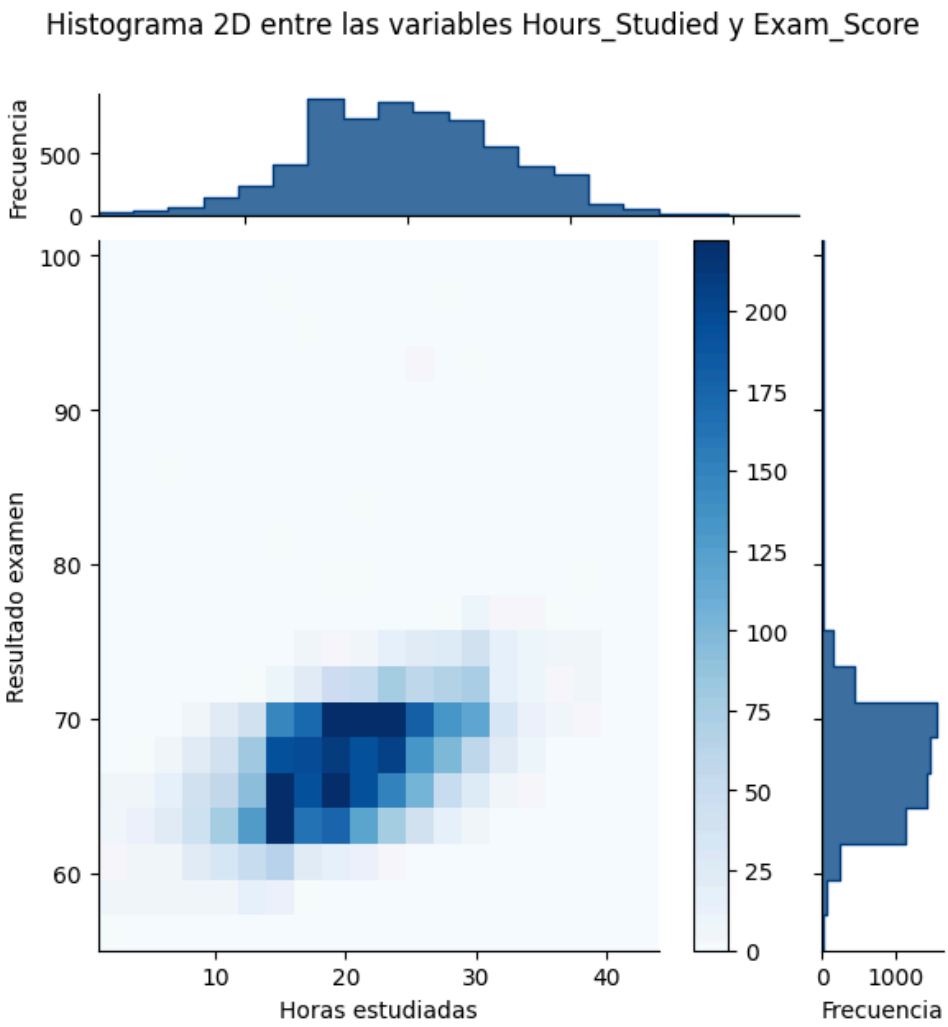

Figura 5: Histograma con Seaborn.

```
g = sns.JointGrid(data=data_students, x="Hours_Studied", y="Exam_Score",
marginal_ticks=True)
g.fig.suptitle("Histograma 2D entre las variables Hours_Studied y Exam_Score", y=1.05)
g.plot_joint(
    sns.histplot,
    cmap="Blues", pmax=.8, cbar=True, bins=20,
    thresh=None
)
g.plot_marginals(sns.histplot, element="step", color="#004080", bins=20)
g.ax_joint.set_xlabel("Horas estudiadas")
g.ax_joint.set_ylabel("Resultado examen")
g.ax_marg_x.set_ylabel("Frecuencia")
g.ax_marg_y.set_xlabel("Frecuencia")
g.ax_marg_x.yaxis.label.set_visible(True)
g.ax_marg_y.xaxis.label.set_visible(True)
plt.show()
```

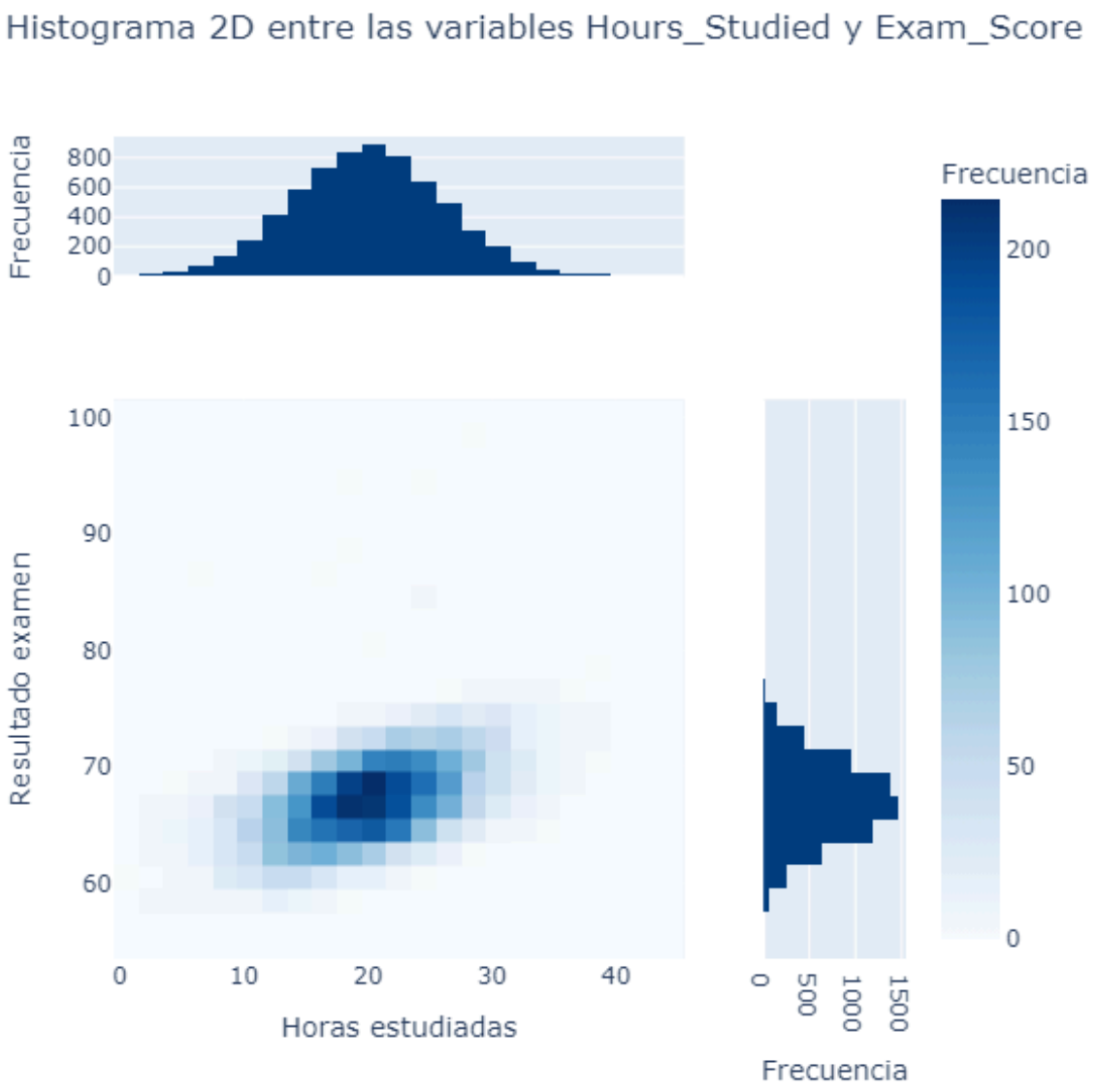

Figura 6: Histograma con Plotly.

```
fig = make_subplots(
    rows=2, cols=2,
    shared_xaxes=True, shared_yaxes=True,
    column_widths=[0.8, 0.2], row_heights=[0.2, 0.8],
    specs=[[{"type": "histogram"}, None],
           [{"type": "histogram2d"}, {"type": "histogram"}]]
)

# Histograma marginal en el eje X
fig.add_trace(
    go.Histogram(
        x=data_students.Hours_Studied,
        nbinsx=30,
        marker_color="#004080",
        showlegend=False,
    ),
    row=1, col=1
```

```
)

# Histograma 2D
fig.add_trace(
    go.Histogram2d(
        x=data_students.Hours_Studied,
        y=data_students.Exam_Score,
        colorscale="Blues",
        colorbar=dict(title="Frecuencia"),
        nbinsx=30, nbinsy=30
    ),
    row=2, col=1
)

# Histograma marginal en el eje Y
fig.add_trace(
    go.Histogram(
        y=data_students.Exam_Score,
        nbinsy=30,
        marker_color="#004080",
        showlegend=False,
        orientation="h"
    ),
    row=2, col=2
)

# Ajustes de ejes y títulos
fig.update_xaxes(title_text="Horas estudiadas", row=2, col=1)
fig.update_yaxes(title_text="Resultado examen", row=2, col=1)
fig.update_yaxes(title_text="Frecuencia", row=1, col=1)
fig.update_xaxes(title_text="Frecuencia", row=2, col=2)

# Título general
fig.update_layout(
    title_text="Histograma 2D entre las variables Hours_Studied y Exam_Score",
    title_x=0.5,
    showlegend=False,
    height=600, width=600
)

fig.show()
```

3.3. Gráfico de violín

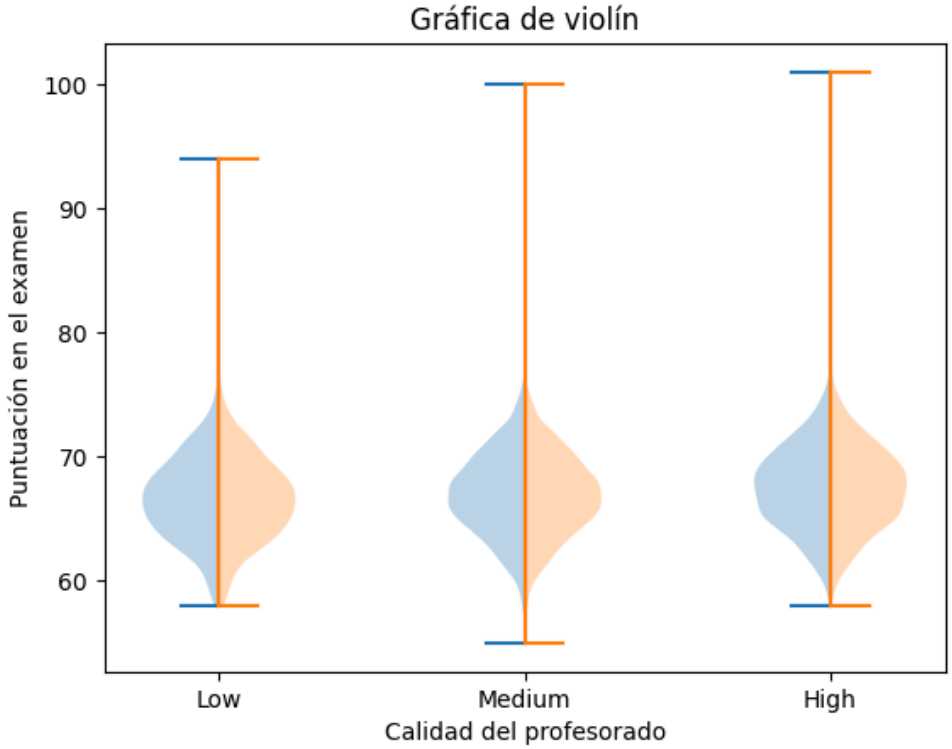

Figura 7: Violín con Matplotlib.

```
plot_data = [
    data_students[data_students["Teacher_Quality"] == "Low"]["Exam_Score"].values,
    data_students[data_students["Teacher_Quality"] == "Medium"]["Exam_Score"].values,
    data_students[data_students["Teacher_Quality"] == "High"]["Exam_Score"].values
]
plt.violinplot(plot_data, side="low")
plt.violinplot(plot_data, side="high")
plt.xticks(np.arange(1, 4), labels=["Low", "Medium", "High"])
plt.title("Gráfica de violín")
plt.xlabel("Calidad del profesorado")
plt.ylabel("Puntuación en el examen")
plt.show()
```

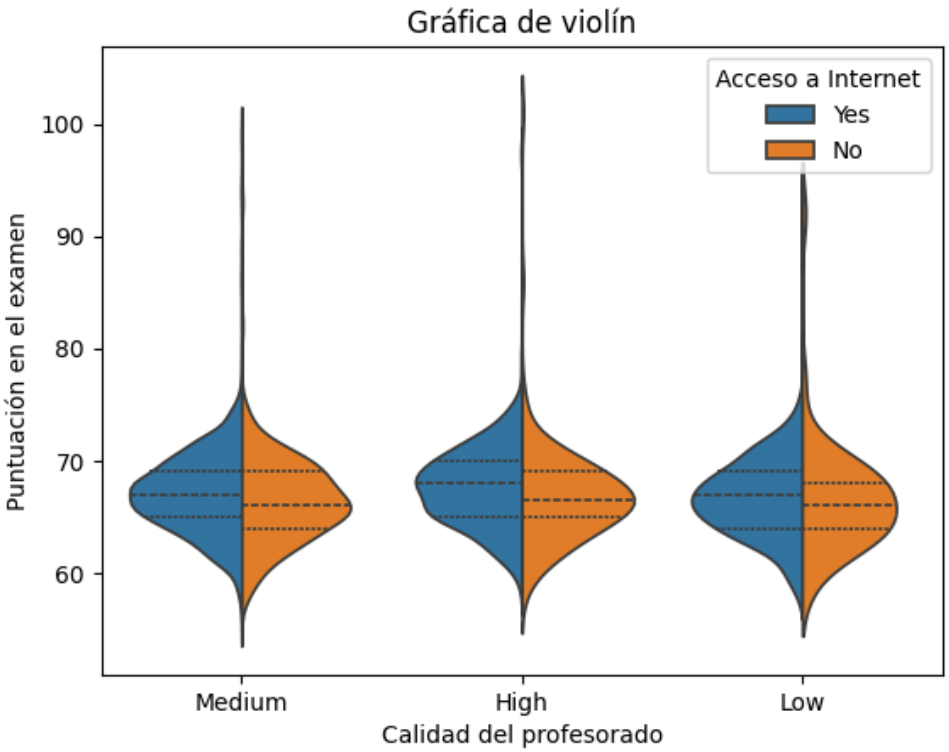

Figura 8: Violín con Seaborn.

```
sns.violinplot(data=data_students, x="Teacher_Quality", y="Exam_Score",
               hue="Internet_Access", split=True, inner="quart")
plt.title("Gráfica de violín")
plt.xlabel("Calidad del profesorado")
plt.ylabel("Puntuación en el examen")
legend = plt.gca().get_legend()
legend.set_title("Acceso a Internet")
plt.show()
```

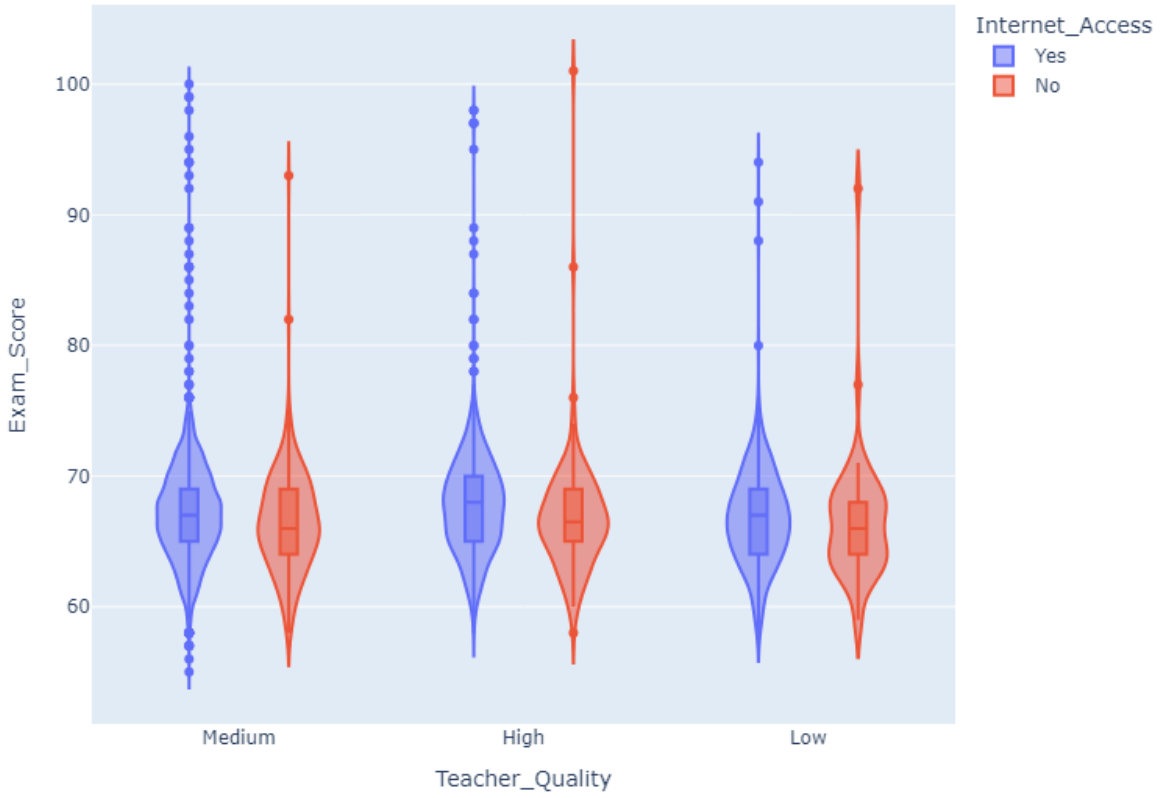

Figura 9: Violín con Plotly.

```
fig = px.violin(data_students, x="Teacher_Quality", y="Exam_Score",
color="Internet_Access", box=True)
fig.update_layout(height=600, width=775)
fig.show()
```

3.4. Gráficos de parejas

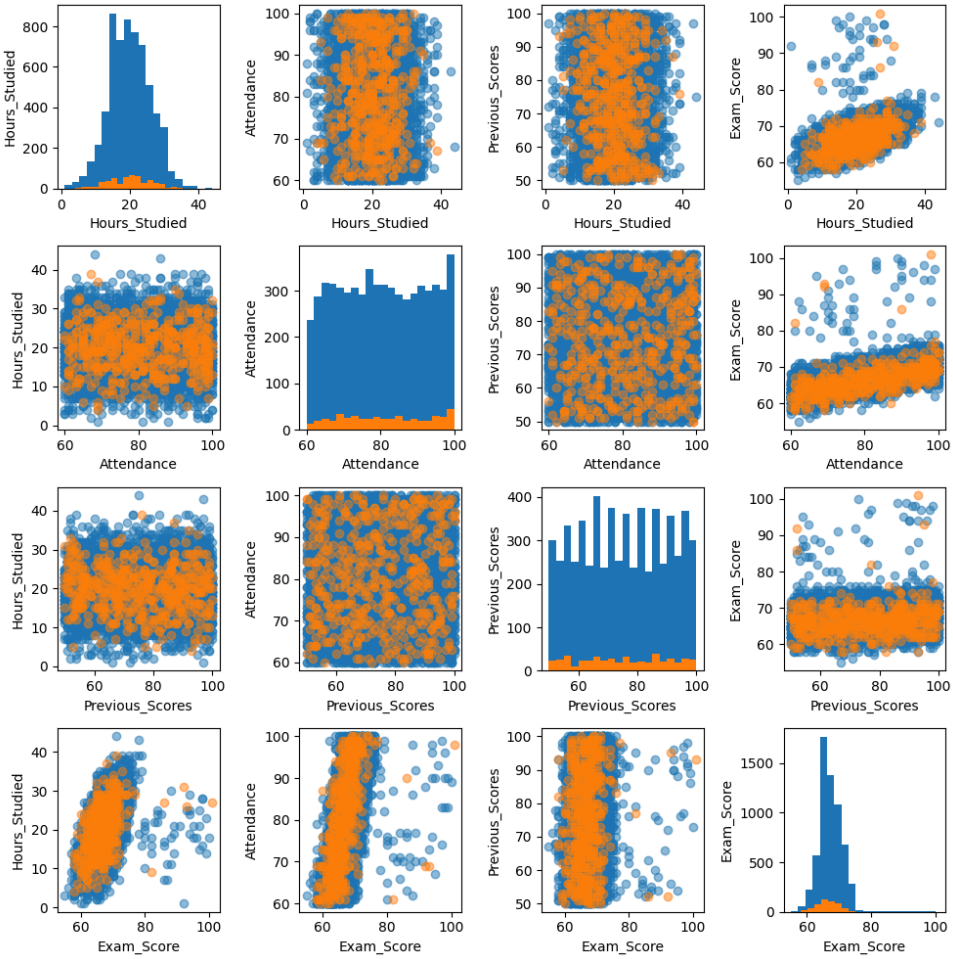

Figura 10: Gráficos de parejas con Matplotlib.

```
numerical_columns = ["Hours_Studied", "Attendance", "Previous_Scores", "Exam_Score"]
categorical = "Internet_Access"
plot_dataset = data_students[numerical_columns]
fig, axes = plt.subplots(len(numerical_columns), len(numerical_columns), figsize=(10,
10))
for cat in ["Yes", "No"]:
    plot_dataset_filt = plot_dataset[data_students[categorical] == cat]
    for x,x_label in enumerate(numerical_columns):
        for y,y_label in enumerate(numerical_columns):
            ax = axes[x, y]
            if x == y:
                ax.hist(plot_dataset_filt[x_label], bins=20)
            else:
                  ax.scatter(plot_dataset_filt[x_label], plot_dataset_filt[y_label],
alpha=0.5)
            ax.set_xlabel(x_label)
            ax.set_ylabel(y_label)
plt.tight_layout()
plt.show()
```

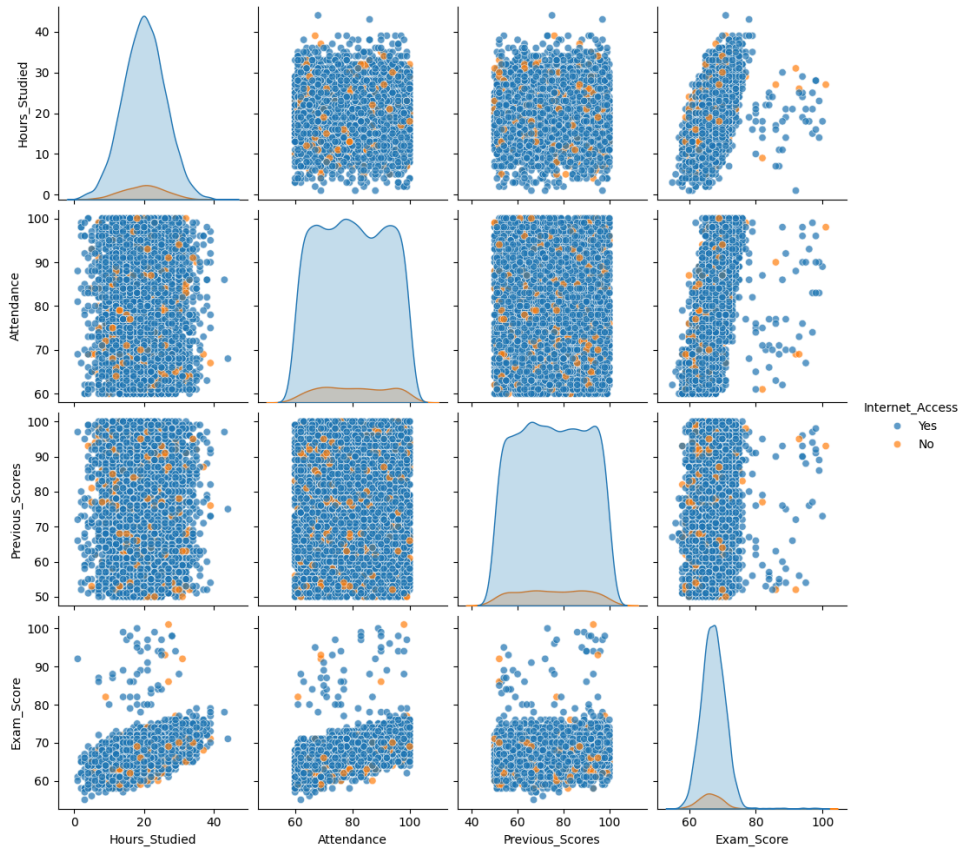

Figura 11: Gráficos de parejas con Seaborn.

```
numerical_columns = ["Hours_Studied", "Attendance", "Previous_Scores", "Exam_Score"]
categorical = "Internet_Access"
plot_dataset = data_students[numerical_columns]
numerical_columns.append(categorical)
plot_dataset = data_students[numerical_columns]
sns.pairplot(plot_dataset, hue=categorical, plot_kws={"alpha": 0.7})
plt.title("Gráfico de parejas")
plt.show()
```

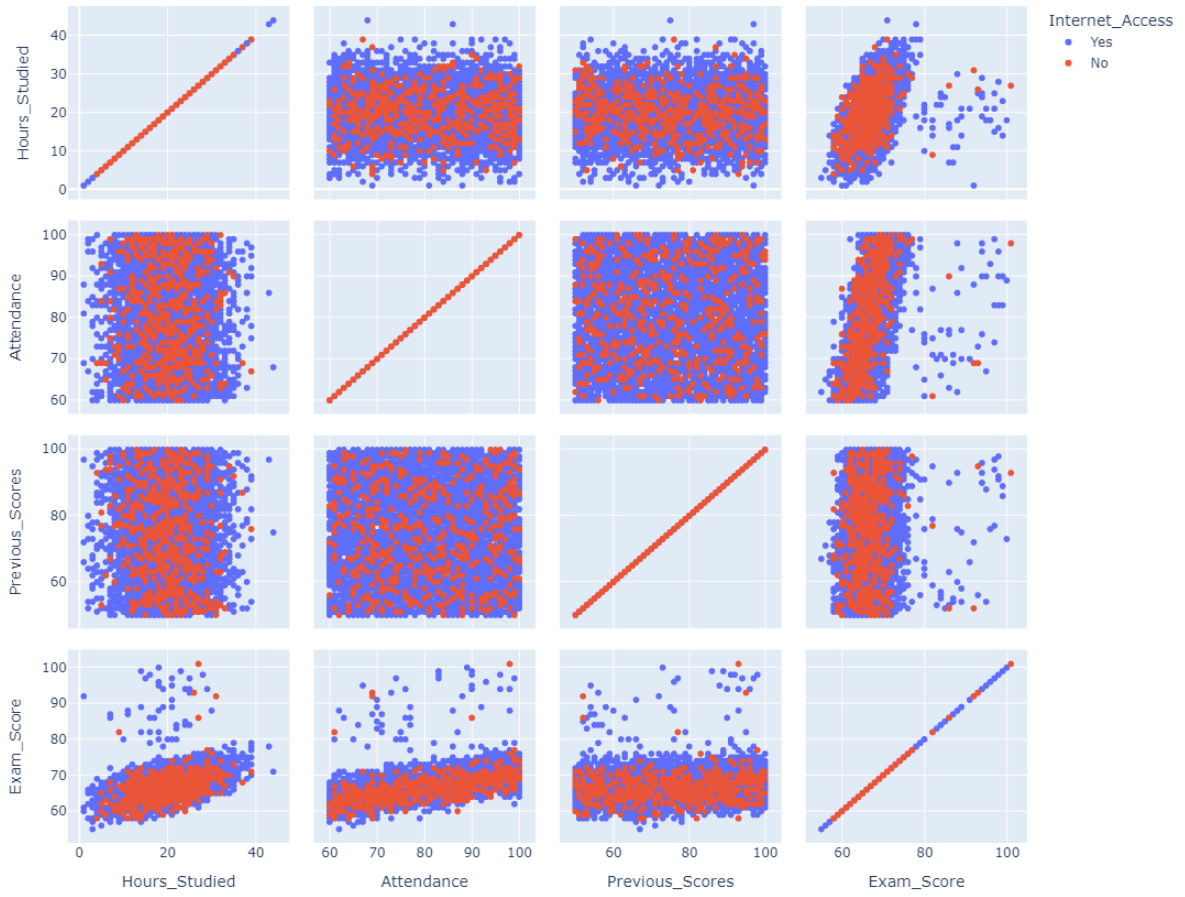

Figura 12: Gráficos de parejas con Plotly.

```
numerical_columns = ["Hours_Studied", "Attendance", "Previous_Scores", "Exam_Score"]
categorical = "Internet_Access"
dataset_columns = numerical_columns.copy()
dataset_columns.append(categorical)
plot_dataset = data_students[dataset_columns]
fig = px.scatter_matrix(plot_dataset, dimensions=numerical_columns, color=categorical)
fig.update_layout(height=900, width=1100)
fig.show()
```

3.5. Gráficos de dispersión 3D

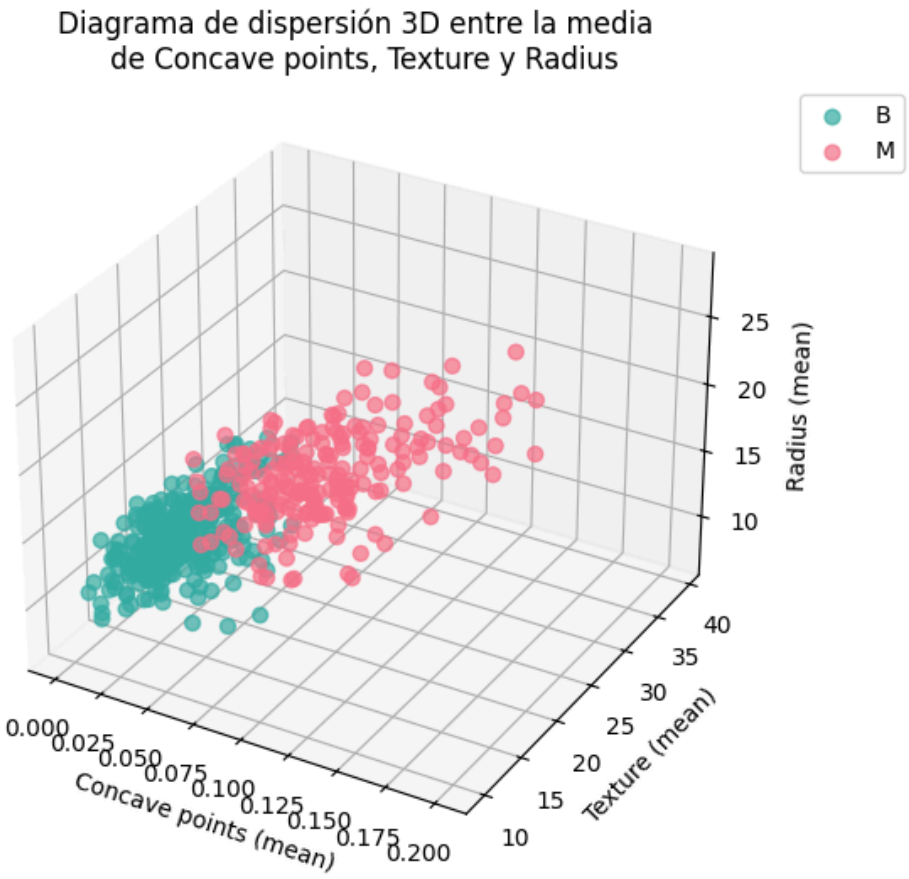

Figura 13: Gráficos de dispersión 3D con Matplotlib.

```
# Crear la figura y el eje 3D
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(projection='3d')

# Crear el colormap con dos colores para las categorías "M" y "B"
cmap = ListedColormap(sns.color_palette("husl", 2).as_hex())

# Filtrar los datos por categorías para cada grupo
data_B = data_cancer[data_cancer["Diagnosis"] == "B"]
data_M = data_cancer[data_cancer["Diagnosis"] == "M"]

# Graficar puntos para categoría "B"
sc_B = ax.scatter(
    data_B["Concave points (mean)"],
    data_B["Texture (mean)"],
    data_B["Radius (mean)"],
    s=40, color=cmap(1), label="B", marker='o', alpha=0.7
)

# Graficar puntos para categoría "M"
sc_M = ax.scatter(
    data_M["Concave points (mean)"],
    data_M["Texture (mean)"],
    data_M["Radius (mean)"],
    s=40, color=cmap(0), label="M", marker='o', alpha=0.7
)
```

```
# Etiquetas de los ejes
ax.set_xlabel('Concave points (mean)')
ax.set_ylabel('Texture (mean)')
ax.set_zlabel('Radius (mean)')
ax.set_title("Diagrama de dispersión 3D entre la media \n de Concave points, Texture
y Radius")

# Añadir la leyenda
plt.legend(bbox_to_anchor=(1.05, 1), loc=2)

# Guardar la figura
plt.savefig("scatter_hue", bbox_inches='tight')
plt.show()
```

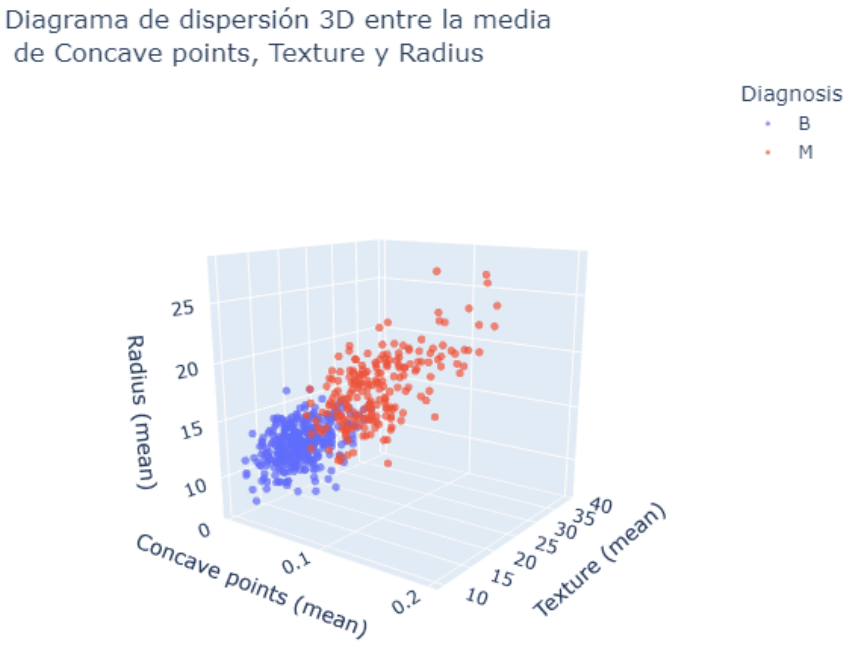

Figura 14: Gráficos de dispersión 3D con Plotly.

```
fig = px.scatter_3d(data_cancer,
                            x="Concave points (mean)", y="Texture (mean)", z="Radius
(mean)", color="Diagnosis",
                    opacity=.7)
fig.update_traces(marker_size=3)
fig.update_layout(
    title_text="Diagrama de dispersión 3D entre la media <br> de Concave points, Texture
y Radius",
    height=600, width=600
)
fig.show()
```

4. Análisis

4.1. Matplotlib

- **Sencillez:** Al ser una librería básica de visualización de datos, es fácil de implementar para gráficos sencillos. Sin embargo, cuando se quieren realizar gráficos más complejos o con más personalizaciones, la implementación de esta librería pasa a ser complicada. Esto se debe a que hay que especificar cada nivel de detalle y, para realiar gráficos complejos, hay que saber como utilizar y combinar las opciones básicas que tiene la librería.

  Esto se puede observar en la Figura 4, donde para generar un gráfico complejos de histograma 2D con histogramas 1D en los ejes marginales ha habido que realizar una personalización compleja.

- **Personalización:** La librería Matplotlib ofrece una gran personalización de sus gráficos. A pesar de ser complicado llegar a un nivel alto de personalización en un gráfico, la librería ofrece muchas opciones y estas pueden combinarse y detallarse de muchas formas diferentes.

  Por ejemplo, en la Figura 1, se han podido personalizar muchas características del gráfico: títulos, etiquetas de los ejes, leyenda, gridlines, magnitud de los ejes, etc.

- **Capacidad de visualización:** Aunque sencillos, la librería Matplotlib ofrece una gran cantidad de tipos de gráficos. Sin embargo, al ser una librería de visualización básica y general, hay gráficos como el gráfico por parejas de Figura 10, que no ofrece en su repertorio de funciones.

  Por ejemplo, en la Figura 10, no se tenía una funcón de Matplotlib para generar un gráfico de parejas. Por ende, se ha tenido que realizar una malla de subplots y un bucle para poder realizar el gráfico.

- **Rendimiento:** El rendimiento de la librería Matplotlib es bastante bueno. En la Figura 10, donde se tinen que representar una gran cantidad de datos, se ha tardado entre 1 y 2 s en realizar el gráfico.

4.2. Seaborn

- **Sencillez:** La librería Seaborn, al ser una librería especializada en gráficos estadísticos, es muy fácil de implementar para este tipo de visualizaciones. Como se puede ver en la Figura 8, con unas pocas líneas de código se ha podido realizar un gráfico de violín con muchas personalizaciones.
- **Personalización:** La librería Seaborn, gracias a que se desarrolla en Matplotlib, tiene una buena personalizaciones, compartiendo muchas funciones de decoración. Sin embargo, al ser una librería de visualización especializada, tiene menos funciones de personalización que Matplotlib.

  Por ejemplo, se puede ver como en Figura 2, no se ha podido realizar la modificación de la magnitud del eje y.

- **Capacidad de visualización:** La librería Seaborn ofrece muchas funciones para realizar gráficos estadísticos. Como se puede ver en la Figura 11 y la Figura 5, la librería tiene funciones de generación de gráficos estadísticos que librerías como Matplotlib no.

  Sin embargo, la liberría no posee soporte de gráficos 3D.

- **Rendimiento:** El rendimiento de la liberría Seaborn para grandes cantidades de datos no es realmente bueno. Por ejemplo, para generar el gráfico Figura 11, se ha tenido que esperar entre 5 y 10 s, que en comparación con otras librerías es un rendimiento muy bajo.

4.3. Plotly

- **Sencillez:** La libería Plotly es una librería de visualización de datos interactiva. La forma de generar sus gráficos no es tan intuitiva como en Matplotlib, sin embargo, a la hora de generar gráficos más complicados pero que la librería tiene por defecto, sí que llega a ser más sencilla de implementar.

  Se puede ver que para generar un gráfico estadístico complicado que la propia librería no posee, se han requerido muchas lineas de código (Figura 6). Por otro lado, para realizar un gráfico de violín que sí posee la librería (Figura 9), se han requerido muy pocas líneas de código.

- **Personalización:** La librería Plotly no ofrece tantas opciones de personalización. Los gráficos que realiza ya suelen estar bastante pulidos, por lo que no suele hacer falta aplicarle muchas personalizaciones. Sin embargo, sí posee las personalizaciones básicas necesarias, como cambio de título, etiquetas de los ejes, leyenda, gridlines, etc.

  Por ejemplo, se puede ver como en Figura 3, no ha hecho falta realizar una gran personalización para obtener el gráfico que se deseaba.

- **Capacidad de visualización:** La librería Plotly posee una gran cantidad de gráficos. Es cierto que a hay algunso gráficos estadísticos que no posee pero que Seaborn sí, pero de la misma forma tiene siporte de gráficos 3D, que con otras librerías es muy complicado de implementar o incluso imposible. Podemos ver esta implementación 3D en la Figura 14.

- **Rendimiento:** El rendimiento de la librería Plotly es muy bueno. Aunque tarda un poco más en generar la interfaz interactivas de las gráficas, tarda mucho menos en preparar los gráficos. Para cantidades pequeñas de datos hace que se tarde por lo general un poco más pero, para gráficos de muchos datos como en la Figura 12, es bastante rápido, tardando menos de medio segundo en realizar el gráfico.

5. Conclusiones

A continuación, se enumerarán las fortalezas y desventajas de cada librería.

- La libería Matplotlib es una librería muy potente en cuanto a capacidades, ya que ofrece una gran personalizaciones, haciendo que aunque no posea algunos tipos de gráficos, estos puedan realizarse si se domina bien la librería. Sin embargo, como desventaja es que es difícil dominar esta librería a alto nivel y realizar estas personalizaciones tan complejas.
- La librería Seaborn es una librería con una gran cantidad de gráficos estadísticos, siendo muy fácil hacer visualizaciones complejas de este tipo de gráficos. Sin embargo, como desventaja tiene su poca flexibilidad a la hora de personalizarla, ya que si no tiene un gráfico implementado como función, no se puede realizar.
- La librería Plotly es una librería de gráficas interactivas con una gran cantidad de funciones implementadas. La interactividad de esta librería hace que los gráficos sean muchos más fáciles de comprender. Además, posee un apartado 3D muy fácil de usar, realizando potentes gráficas interactivas. Sin embargo, no posee tampoc una gran personalización, costando realizar los gráficos que no posee.

Como conclusión, para las personas que quieran meterse en la visualización de datos con Python, es recomendable empezar con Matplotlib si se quiere abarcar un ámbito más general. En el caso de tener interés solo en gráficos estadísticos, Seaborn es la mejor opción, resultando muy sencilla de usar. Finalmente, si lo que se quiere son gráficos interactivos, gráficos 3D o gráficos que no requieren mucha personalización, se recomienda la librería Plotly, que ofrece gráficos interactivos muy detallados sin necesidad de aplicar personalizaciones.