DOCUMENTO TÉCNICO

Análisis estadístico para la determinación de los porcentajes de la medida.

1. Construcción de la base de datos (2010-2024)

Con el objetivo de identificar la cantidad de vehículos eléctricos registrado en los municipios de Colombia, se realizó una revisión exhaustiva de diversas fuentes de información. Entre ellas, se consultaron los boletines de vehículos publicados por la Asociación Nacional de Empresarios de Colombia (ANDI) y los datos disponibles en el portal de Datos abiertos del Min Tic, en relación con el registro de vehículos eléctricos en los municipios y distritos obligados a aplicar la medida.

A partir de esta revisión se consolidó una base de datos que recopila los registros de vehículos eléctricos entre los años 2010 y 2024, abarcando a 121 municipios del país.

2. Predicción de faltos faltantes del (2025-2030)

Con el fin de contar con una base de datos robusta que permitiera el desarrollo de estimaciones confiables, se implemento un modelo de predicción y entrenamiento de datos para proyectar el número de vehículos eléctricos en cada municipio entre los años 2025 y 2030.

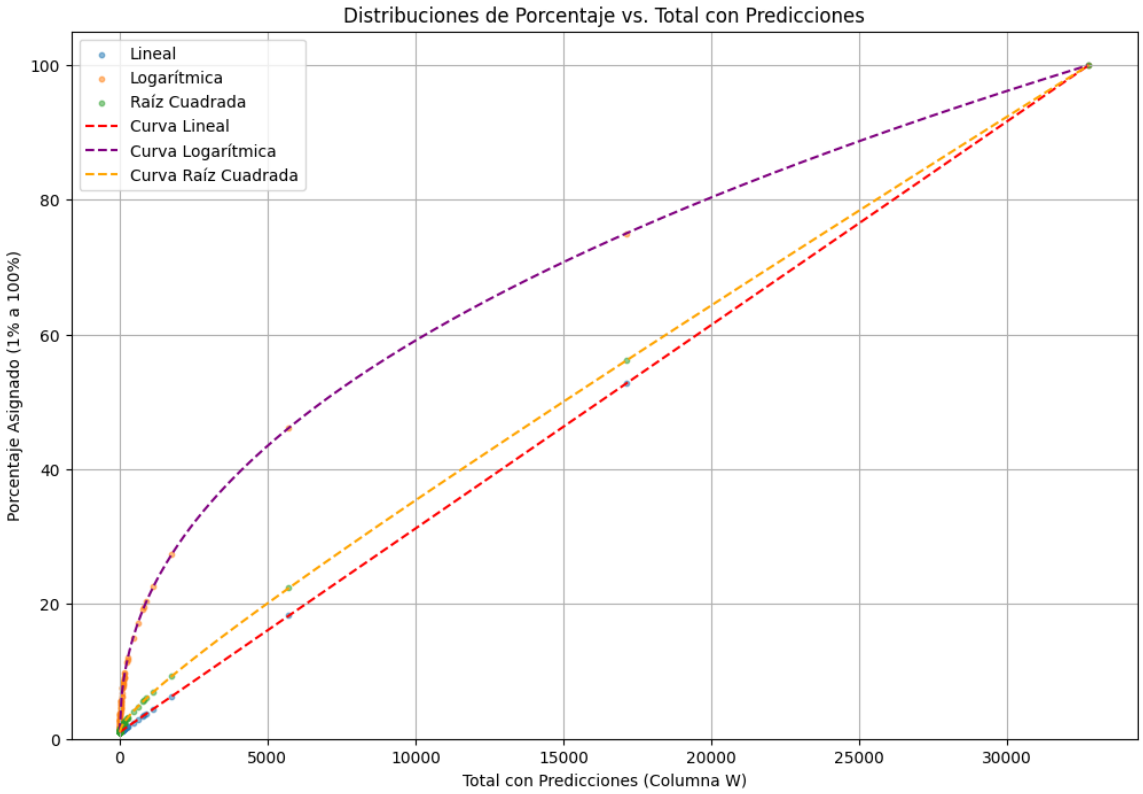
Esta predicción se desarrollo en lenguaje Python, utilizando los datos observados entre 2010 y 2024 sin modificar ni imputar los valores originales. Las predicciones permiten estimar con mayor precisión los porcentajes de adopción futuras de vehículos eléctricos. El código empleado para esta estimación se incluye al final del documento.

3. Distribución y estimación de los porcentajes de acometida.

A partir del total de vehículos eléctricos registrados entre 2010 y 2025, se idéntico un rango de valores entre 1 (mínimo) y 32,761 (máximo). Este último corresponde al valor proyectado para el año 2030. Con base en estos datos, se evaluaron distintas metodologías para distribuir y estimar los porcentajes de acometidas eléctricas por municipio. Se realizaron pruebas estadísticas comparativas entre distribuciones lineales, logarítmicas y de raíz cuadrada, con el fin de identificar la transformación más adecuada para representar la relación entre el número total de vehículos eléctricos y el porcentaje estimado de parqueaderos.

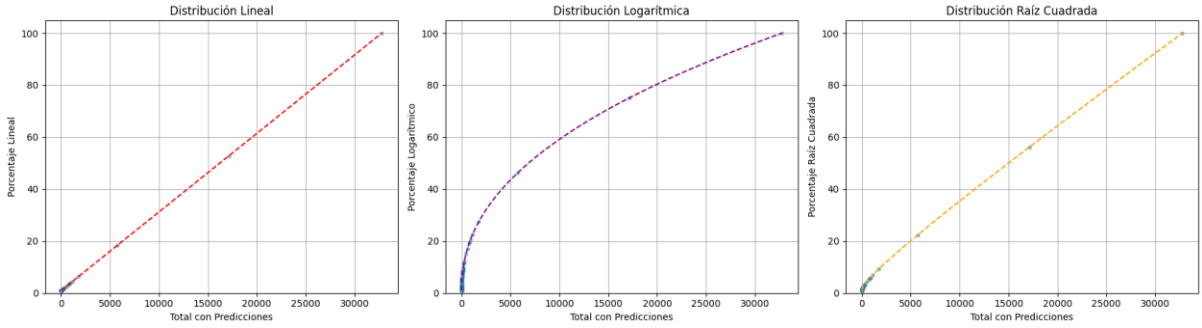
A continuación, se presentan los resultados de estas pruebas:

Imagen 1: Comparativa por distribución



Fuente: Elaboración propia con base en datos de vehículos eléctricos (2024-2030)

Imagen 2: Distribución lineal, logarítmica y raíz cuadrada



Fuente: Elaboración propia con base en datos de vehículos eléctricos (2024-2030)

Evaluación de la bondad de ajuste de las distribuciones:

Se evaluó el coeficiente de determinación R^2 para tres tipos de distribución: lineal., logarítmica y raíz cuadrada, esto con el fin de identificar cual de estas mostraba un mejor ajuste entre el total de vehículos eléctricos y el porcentaje estimado de acometidas eléctricas

Los resultados obtenidos fueron los siguientes:

R^2para Distribución Lineal: 1.0000

R^2 para Distribución Logarítmica: 0.8675

R^2 para Distribución Raíz Cuadrada: 0.9954

Al analizar los coeficientes de determinación R^2 asociados a las diferentes distribuciones, se denota que la distribución lineal presenta mejor ajuste en la bondad del R^2 =1.000. Sin embargo, al revisar la coherencia de los porcentajes estimados en función del comportamiento esperado, se evidencia que esta distribución no refleja adecuadamente la forma real de la progresión de los datos. Específicamente, la distribución lineal no captura de manera precisa los porcentajes en los rangos intermedios y superiores.

En este orden, se opta por utilizar la distribución de raíz cuadrada, la cual, aunque presenta un R^2 =0.9954 esta ofrece una representación adecuada y lógica con el crecimiento porcentual en infraestructura de acometidas eléctricas. Esta elección permite captura de forma realista las diferencias relativas de los porcentajes entre los municipios con pocos y muchos vehículos eléctricos.

4. Ecuación de la distribución de raíz cuadrada en la estimación de los porcentajes:

La anterior distribución de raíz cuadrada se encuentra seguida por la siguiente formula:

Ecuación I: Principal

$$\text{Porcentaje} = \left[1 + \left(\sqrt{X} - 1\right) \cdot \frac{9}{\sqrt{M} - 1}\right]^2$$

Donde:

X: es el valor original por convertir (cantidad de vehículos eléctricos en un municipio)

M: el valor máximo posible en el conjunto de vehículos eléctricos

√X: raíz cuadrada de X, para las diferencias entre valores grandes y pequeños

9: se da √100- √1 =10-1 = 9 el rango de transformación entre 1% y 100%

Nota: Se usa 1 como mínimo porque √ 0 un rango no intuitivo. El rango de salida es siempre de 1% a 100%.

5. Ejemplo práctico:

Porcentaje = [1 + (√X - 1) * (9 / (√M - 1))]^2
Donde:
X = 147
M = 32761
√147 ≈ 12.124
√32761 ≈ 181.009

Calculo:

Porcentaje = [1 + (12.124 - 1) * (9 / (181.009 - 1))]^2
= [1 + (11.124) * (9 / 180.009)]^2
= [1 + 11.124 * 0.049997]^2
= [1 + 0.5562]^2
= (1.5562)^2 ≈ 2.4227

Resultado:

Porcentaje ≈ 2.42%

6. Código de programación utilizado para la estimación en las diferentes fases:

Para las predicciones y entrenamiento de los datos se usó el siguiente código:

"(…)

```
import pandas as pd
import numpy as np
```

```
if 'df_entrenamiento_2' not in locals():
    print("El DataFrame 'df_entrenamiento_2' no ha sido cargado.")
else:
    historic_years = [str(year) for year in range(2010, 2025)]
    prediction_years = [str(year) for year in range(2025, 2031)]
    all_years = [str(year) for year in range(2010, 2031)]
DataFrame
    relevant_cols = ['Municipio'] + historic_years
    df_relevant = df_entrenamiento_2[relevant_cols].copy()
    for year in historic_years:
        # Manejar posibles errores en la conversión a numérico
        df_relevant[year] = pd.to_numeric(df_relevant[year], errors='coerce')
    # Preparar el DataFrame para las predicciones
    # Rellenar las columnas de predicción con NaN para que tengan el mismo
formato
    for year in prediction_years:
         df_entrenamiento_2[year] = np.nan # Asegurarse de que existen las
columnas de predicción
    # Iterar a través de cada fila (municipio) para hacer la predicción
    for index, row in df_relevant.iterrows():
        municipio = row['Municipio']
        # Extraer los datos históricos para el municipio actual
         # Filtrar los años históricos para eliminar NaN y ceros (si se considera
apropiado)
        # Aquí solo eliminamos NaN para basarnos en los datos existentes
        historic_data = row[historic_years].dropna()

        # Crear pares de (año, valor) para el modelo
        # Usamos el índice del año como variable independiente (tiempo)
        if not historic_data.empty:
            # Usar el índice del año como base temporal
            # Ajustamos el año_base para que el primer año sea 0
            # Consideramos que el año es el valor de la columna
                              X_train = np.array([int(year) for year in
historic_data.index]).reshape(-1, 1)
            y_train = historic_data.values

            # Entrenar un modelo de regresión lineal simple
            # Puedes experimentar con otros modelos si es necesario
            model = LinearRegression()
            model.fit(X_train, y_train)

            # Preparar los datos de entrada para la predicción (años futuros)
            X_pred = np.array([int(year) for year in prediction_years]).reshape(-
1, 1)

            # Realizar la predicción
            predictions = model.predict(X_pred)
```

```
            # Asignar las predicciones de vuelta al DataFrame original (df_entrenamiento_2)
            # Asegurarse de que las predicciones no sean negativas (ya que representa un conteo)
            # Se puede redondear al entero más cercano o superior
            for i, year in enumerate(prediction_years):
                # Asignar al DataFrame original
                df_entrenamiento_2.loc[index, year] = max(0, round(predictions[i])) # No valores negativos, redondear a entero
        else:
            # Si no hay datos históricos para un municipio, las predicciones serán NaN o 0
            # Aquí las dejamos como estaban (NaN si no se rellenaron antes) o asignamos 0
            print(f"Advertencia: No hay datos históricos válidos para el municipio '{municipio}'. No se realizaron predicciones.")
            # Opcional: Rellenar los años de predicción con 0 o un valor por defecto
            # for year in prediction_years:
            #     df_entrenamiento_2.loc[index, year] = 0

    # Mostrar el DataFrame con las predicciones
    print("\nDataFrame con predicciones para 2025-2030:")
    # Mostrar solo las columnas de municipio y los años de 2010 a 2030
            print(df_entrenamiento_2[['Municipio'] + historic_years + prediction_years].head())

    # Save the DataFrame to an Excel file
    output_filename = 'predicciones_vehiculos_completas.xlsx'
    # Seleccionar solo las columnas deseadas antes de guardar
      df_to_save = df_entrenamiento_2[['Municipio'] + historic_years + prediction_years]
    df_to_save.to_excel(output_filename, index=False)


(…)"
```

Para estimar la distribución se usó el siguiente código:

"(…)

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
import numpy as np
# Asegurarse de que df_entrenamiento_2 está cargado y tiene la columna 'Total con predicciones'
if 'df_entrenamiento_2' not in locals() or 'Total con predicciones' not in df_entrenamiento_2.columns:
    print("El DataFrame 'df_entrenamiento_2' o la columna 'Total con predicciones' no está disponible.")
else:
    # Limpiar y convertir la columna 'Total con predicciones' a numérica
    # Rellenar valores no numéricos o vacíos con NaN, y luego eliminar filas con NaN
    df_dist = df_entrenamiento_2.copy()
    df_dist['Total con predicciones'] = pd.to_numeric(df_dist['Total con predicciones'], errors='coerce')
    df_dist = df_dist.dropna(subset=['Total con predicciones'])

    df_dist = df_dist[df_dist['Total con predicciones'] >= 0].copy()

    # Obtener los valores de la columna
    valores = df_dist['Total con predicciones'].values

    # Eliminar el valor 0 si existe, ya que puede causar problemas con la escala logarítmica y raíz cuadrada de 0 es 0
    # Para propósitos de mapeo a porcentajes del 1 al 100%, el 0 no es un valor objetivo común
    # Si 0 debe mapearse a 1%, se puede manejar por separado. Aquí asumimos que los valores a mapear son > 0.
    # O si 0 debería mapearse a 0%, ajustamos la lógica. El prompt pide 1 al 100%.
    # Vamos a considerar que el valor mínimo (1) corresponde al 1% y el máximo al 100%.
    # Si el valor mínimo real es 0, tendremos que ajustar el mapeo para la lineal.

    min_val = valores.min() if valores.size > 0 else 0
    max_val = valores.max() if valores.size > 0 else 1 # Evitar división por cero si solo hay un valor

    # Ajuste para mapear el rango [min_val, max_val] al rango [1, 100]
    # Si min_val es 0 y max_val > 0, ajustamos la escala para que 0 mapee cerca del 1%.
    # Una forma simple es sumar 1 a todos los valores antes de la transformación y luego ajustar la escala de porcentaje.
    # O simplemente escalar el rango [min_val, max_val] a [1, 100].

    # Rango objetivo de porcentajes
    min_percent = 1
```

```
    max_percent = 100

    # Si el rango de valores es cero (todos los valores son iguales), no podemos escalar.
    if max_val == min_val:
        df_dist['Porcentaje_Lineal'] = (min_percent + max_percent) / 2 # O asignar un valor por defecto
        df_dist['Porcentaje_Logaritmica'] = (min_percent + max_percent) / 2
        df_dist['Porcentaje_RaizCuadrada'] = (min_percent + max_percent) / 2
        print("Todos los valores en 'Total con predicciones' son iguales. No se puede realizar una distribución significativa.")
    else:
        # Distribución Lineal
        # Escalar linealmente el rango [min_val, max_val] al rango [min_percent, max_percent]
        # Formula: nuevo_valor = min_nuevo + (valor_original - min_original) * (max_nuevo - min_nuevo) / (max_original - min_original)
        df_dist['Porcentaje_Lineal'] = min_percent + (valores - min_val) * (max_percent - min_percent) / (max_val - min_val)

        # Distribución Logarítmica
        # Escalar logarítmicamente el rango [min_val, max_val] al rango [log(min_percent), log(max_percent)]
        # Luego invertir la transformación logarítmica para obtener porcentajes en el rango [min_percent, max_percent]
        # Para evitar log(0) si min_val es 0, sumamos una pequeña constante o mapeamos 0 a un valor específico.
        # Ya filtramos valores <= 0 si existen, pero si el min_val >= 1, podemos usar log directamente.
        # Si min_val es 1, log(1) = 0. Queremos mapear 1 a 1%.
        # Una forma es mapear log(valores + 1) al rango [log(min_percent + 1), log(max_percent + 1)].
        # O mapear log(valores - min_val + 1) al rango [log(1), log(max_val - min_val + 1)], y luego escalar.
        # Vamos a usar una transformación que mapee [min_val, max_val] a [log(min_percent), log(max_percent)]
        # y luego aplicar np.exp para obtener el porcentaje.
        # Ajustamos la base logarítmica para que se ajuste al rango de porcentajes.
        # Mapear log(valores) en [log(min_val), log(max_val)] a [log(min_percent), log(max_percent)]
        # Asegurarse de que los valores sean > 0 para logaritmo
        valores_log = np.log(valores[valores > 0])
        min_log_val = np.log(min_val) if min_val > 0 else np.log(1) # Usar log(1) si min_val es 0 (después de filtrado >0, min_val >=1)
        max_log_val = np.log(max_val)

        if max_log_val == min_log_val:
```

```
            # Todos los valores logarítmicos son iguales (ocurre si todos los valores originales son iguales)
            df_dist['Porcentaje_Logaritmica'] = (min_percent + max_percent) / 2
        else:
            # Escalar log(valores) linealmente al rango [log(min_percent), log(max_percent)]
            log_percent_scaled = np.log(min_percent) + (valores_log - min_log_val) * (np.log(max_percent) - np.log(min_percent)) / (max_log_val - min_log_val)
            # Convertir de vuelta de la escala logarítmica a la escala de porcentaje
            porcentaje_log = np.exp(log_percent_scaled)
            # Asignar de vuelta al DataFrame, manejando los valores originales que eran <= 0 si los hubo (aunque filtramos >0)
            # Creamos una serie temporal para alinear con el índice original
            porcentaje_log_series = pd.Series(porcentaje_log, index=df_dist[valores > 0].index)
            df_dist['Porcentaje_Logaritmica'] = np.nan # Inicializar la columna
            df_dist.loc[valores > 0, 'Porcentaje_Logaritmica'] = porcentaje_log_series

        # Distribución Raíz Cuadrada
            # Escalar sqrt(valores) linealmente al rango [sqrt(min_percent), sqrt(max_percent)]
            # Luego elevar al cuadrado para obtener porcentajes en el rango [min_percent, max_percent]
            # Asegurarse de que los valores sean >= 0 para raíz cuadrada. Ya filtramos < 0.
        valores_sqrt = np.sqrt(valores)
        min_sqrt_val = np.sqrt(min_val)
        max_sqrt_val = np.sqrt(max_val)

        if max_sqrt_val == min_sqrt_val:
            # Todos los valores de raíz cuadrada son iguales (ocurre si todos los valores originales son iguales)
            df_dist['Porcentaje_RaizCuadrada'] = (min_percent + max_percent) / 2
        else:
            # Escalar sqrt(valores) linealmente al rango [sqrt(min_percent), sqrt(max_percent)]
            sqrt_percent_scaled = np.sqrt(min_percent) + (valores_sqrt - min_sqrt_val) * (np.sqrt(max_percent) - np.sqrt(min_percent)) / (max_sqrt_val - min_sqrt_val)
            # Convertir de vuelta elevando al cuadrado
            df_dist['Porcentaje_RaizCuadrada'] = sqrt_percent_scaled**2

    # Asegurarse de que los porcentajes estén en el rango [1, 100]
```

```
        df_dist['Porcentaje_Lineal'] = np.clip(df_dist['Porcentaje_Lineal'], min_percent, max_percent)
                                df_dist['Porcentaje_Logaritmica'] = np.clip(df_dist['Porcentaje_Logaritmica'], min_percent, max_percent)
                                df_dist['Porcentaje_RaizCuadrada'] = np.clip(df_dist['Porcentaje_RaizCuadrada'], min_percent, max_percent)

    print("\nPrimeras 5 filas con las nuevas columnas de porcentajes:")
            print(df_dist[['Total con predicciones', 'Porcentaje_Lineal', 'Porcentaje_Logaritmica', 'Porcentaje_RaizCuadrada']].head())

    # Calcular R2 para cada distribución simulada
    # Para calcular R2, necesitamos tener "valores reales" para los porcentajes.
    # El problema es que no hay "valores reales" de porcentaje, solo tenemos los valores de 'Total con predicciones'.
    # El R2 se calcula típicamente comparando valores predichos con valores reales.
    # Aquí estamos creando diferentes "transformaciones" o "asignaciones" de los valores a porcentajes.
    # No podemos calcular un R2 en el sentido tradicional (bondad de ajuste de un modelo a datos reales).
    # Lo que sí podemos analizar es cómo cada transformación "distribuye" los valores en el rango de porcentajes.
    # Por ejemplo, una distribución logarítmica asignará porcentajes más altos a valores bajos y se aplanará para valores altos.
    # Una distribución lineal asignará porcentajes de manera uniforme.
    # Una distribución de raíz cuadrada tendrá un comportamiento intermedio.

    # Lo que podemos mostrar es la relación entre 'Total con predicciones' y cada una de las distribuciones de porcentaje.
    # Visualizar estas relaciones es la mejor manera de entender cuál "ajusta" mejor la idea de cómo queremos que los valores se mapeen a porcentajes.

    # Generar valores de ejemplo para graficar las curvas de distribución
    # Usaremos un rango de valores que cubra el rango real de 'Total con predicciones'
    x_range = np.linspace(min_val, max_val, 500)

    # Calcular los porcentajes correspondientes para cada distribución
    # Distribución Lineal
      y_linear = min_percent + (x_range - min_val) * (max_percent - min_percent) / (max_val - min_val)
    y_linear = np.clip(y_linear, min_percent, max_percent) # Asegurar que está en el rango [1, 100]

    # Distribución Logarítmica
    # Similar al cálculo anterior, pero sobre el rango continuo x_range
    x_range_log = np.log(x_range[x_range > 0])
```

```
    min_log_val_plot = np.log(min_val) if min_val > 0 else np.log(1)
    max_log_val_plot = np.log(max_val)

    y_logarithmic = np.full_like(x_range, np.nan, dtype=float) # Inicializar con NaN
    if max_log_val_plot != min_log_val_plot:
            log_percent_scaled_plot = np.log(min_percent) + (x_range_log - min_log_val_plot) * (np.log(max_percent) - np.log(min_percent)) / (max_log_val_plot - min_log_val_plot)
        y_logarithmic[x_range > 0] = np.exp(log_percent_scaled_plot)
         y_logarithmic = np.clip(y_logarithmic, min_percent, max_percent) # Asegurar que está en el rango [1, 100]
        # Manejar el caso de valores <= 0 si x_range los incluye (aunque x_range empieza en min_val que es >=0)
         # Si min_val es 0, log(0) es indefinido. Si nuestro rango de valores empieza en 1, no hay problema.
         # Si empieza en 0, el mapeo logarítmico para 0 es problemático si queremos mapear a 1%.
          # Aquí, como x_range empieza en min_val >= 0, si min_val=0, x_range[0]=0, log(0) es inf.
        # Ajustamos para que el primer punto si min_val=0 sea 1%.
        if min_val == 0:
            y_logarithmic[0] = min_percent

    # Distribución Raíz Cuadrada
    y_sqrt = np.sqrt(min_percent) + (np.sqrt(x_range) - np.sqrt(min_val)) * (np.sqrt(max_percent) - np.sqrt(min_percent)) / (np.sqrt(max_val) - np.sqrt(min_val))
    y_sqrt = y_sqrt**2
    y_sqrt = np.clip(y_sqrt, min_percent, max_percent) # Asegurar que está en el rango [1, 100]

    # Graficar las distribuciones
    plt.figure(figsize=(12, 8))
     plt.scatter(df_dist['Total con predicciones'], df_dist['Porcentaje_Lineal'], label='Lineal', alpha=0.5, s=10)
                        plt.scatter(df_dist['Total con predicciones'], df_dist['Porcentaje_Logaritmica'], label='Logarítmica', alpha=0.5, s=10)
                        plt.scatter(df_dist['Total con predicciones'], df_dist['Porcentaje_RaizCuadrada'], label='Raíz Cuadrada', alpha=0.5, s=10)

    # Graficar las curvas teóricas de las distribuciones
    plt.plot(x_range, y_linear, color='red', linestyle='--', label='Curva Lineal')
    # Solo graficar la curva logarítmica si x_range incluye valores > 0 donde la función es definida
    if np.any(x_range > 0) and max_log_val_plot != min_log_val_plot:
              plt.plot(x_range[x_range > 0], y_logarithmic[x_range > 0], color='purple', linestyle='--', label='Curva Logarítmica')
```

```
    elif min_val == 0: # Si min_val es 0, la curva logarítmica empieza en el origen (0, 1) si mapeamos 0 a 1%
                plt.plot(x_range, y_logarithmic, color='purple', linestyle='--', label='Curva Logarítmica (Ajustada)')

     plt.plot(x_range, y_sqrt, color='orange', linestyle='--', label='Curva Raíz Cuadrada')

    plt.xlabel('Total con Predicciones (Columna W)')
    plt.ylabel('Porcentaje Asignado (1% a 100%)')
    plt.title('Distribuciones de Porcentaje vs. Total con Predicciones')
    plt.legend()
    plt.grid(True)
    plt.xscale('linear') # Eje x lineal
    plt.ylim(0, 105) # Un poco de margen en el eje y
    plt.show()

     # Mostrar las gráficas de las distribuciones individualmente para mayor claridad
    plt.figure(figsize=(18, 5))

    plt.subplot(1, 3, 1)
     plt.scatter(df_dist['Total con predicciones'], df_dist['Porcentaje_Lineal'], alpha=0.5, s=10)
    plt.plot(x_range, y_linear, color='red', linestyle='--')
    plt.xlabel('Total con Predicciones')
    plt.ylabel('Porcentaje Lineal')
    plt.title('Distribución Lineal')
    plt.grid(True)
    plt.ylim(0, 105)

    plt.subplot(1, 3, 2)
                    plt.scatter(df_dist['Total con predicciones'], df_dist['Porcentaje_Logaritmica'], alpha=0.5, s=10)
    if np.any(x_range > 0) and max_log_val_plot != min_log_val_plot:
              plt.plot(x_range[x_range > 0], y_logarithmic[x_range > 0], color='purple', linestyle='--')
    elif min_val == 0:
        plt.plot(x_range, y_logarithmic, color='purple', linestyle='--')
    plt.xlabel('Total con Predicciones')
    plt.ylabel('Porcentaje Logarítmico')
    plt.title('Distribución Logarítmica')
    plt.grid(True)
    plt.ylim(0, 105)

    plt.subplot(1, 3, 3)
                    plt.scatter(df_dist['Total con predicciones'], df_dist['Porcentaje_RaizCuadrada'], alpha=0.5, s=10)
```

```
    plt.plot(x_range, y_sqrt, color='orange', linestyle='--')
    plt.xlabel('Total con Predicciones')
    plt.ylabel('Porcentaje Raíz Cuadrada')
    plt.title('Distribución Raíz Cuadrada')
    plt.grid(True)
    plt.ylim(0, 105)

    plt.tight_layout()
    plt.show()

    # Opcional: Agregar las columnas de porcentaje al DataFrame original y actualizar la hoja de Google Sheets
    # Esto podría ser útil si quieres tener estos porcentajes disponibles en tu hoja.
    # Asegúrate de alinear por índice si es necesario.
                                    #                 df_entrenamiento_2                 =
df_entrenamiento_2.merge(df_dist[['Porcentaje_Lineal',
'Porcentaje_Logaritmica', 'Porcentaje_RaizCuadrada']],
    #                                                 left_index=True, right_index=True,
how='left')

    # if 'df_entrenamiento_2' in locals():
    #     try:
    #         # Accede a la hoja "Datos de entrenamiento 2"
      #             entrenamiento_ws_2 = spreadsheet.worksheet("Datos de
entrenamiento 2")
     #           print(f"\nPreparando para actualizar la hoja con porcentajes:
{entrenamiento_ws_2.title}")

    #         # Convierte el DataFrame actualizado a una lista de listas
     #           # Asegúrate de que el DataFrame contenga todas las columnas
originales + las nuevas de porcentaje
                #                                   data_to_update_percent       =
[df_entrenamiento_2.columns.values.tolist()]                                      +
df_entrenamiento_2.values.tolist()

    #         # Actualiza la hoja de Google Sheets
    #        entrenamiento_ws_2.update([df_entrenamiento_2.columns.tolist()]
+ df_entrenamiento_2.values.tolist())

    #         print("\nLa hoja 'Datos de entrenamiento 2' ha sido actualizada con
las columnas de porcentaje.")

    #     except Exception as e:
    #         print(f"\nOcurrió un error al actualizar la hoja con porcentajes: {e}")
(…)"
```

Dichos códigos corresponden al lenguaje de programación Python.