Informe Ml

Anderson Gomez Garcia

Juan David Pérez

Univalle

2025

Introducción

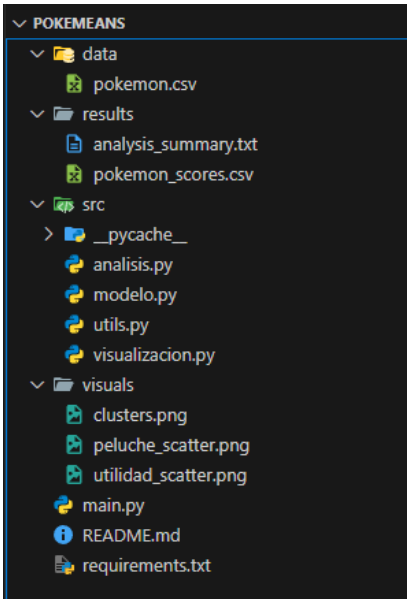
El proyecto Pokemeans aplica técnicas de aprendizaje no supervisado, especialmente el algoritmo K-Means, para analizar las características estadísticas de los Pokémon. El objetivo es agrupar a los Pokémon en diferentes grupos basados en sus atributos, como puntos de vida, ataque, defensa y velocidad, para identificar patrones y similitudes entre ellos.

El uso de clustering permite explorar la diversidad del universo Pokémon de una manera estructurada, facilitando la comprensión de cómo se relacionan entre sí según sus estadísticas. Esta técnica es útil para descubrir agrupaciones naturales sin necesidad de etiquetas previas, lo que ayuda a identificar roles o tipos similares dentro del conjunto de datos.

Además, el proyecto combina análisis de datos con visualizaciones que facilitan la interpretación de los resultados, mostrando cómo se distribuyen los Pokémon en los distintos clusters y destacando las características más relevantes de cada grupo.

Estructura

Para elaborar este proyecto se divido en 4 carpeteas principales datas, results, src y visuals; en donde data se encarga de tener guardado por así decirlo la base o el conjunto de datos usados. Results en el cual gguarda los resultados obtenidos, como modelos entrenados o métricas de evaluación. Visual el cual queda guardados los graficos en formato png y src donde está la lógica del programa.



Explicación de funcionalidades

Primero que nada, se genero un archivo de requirements en el cual se instalan todas las dependencias del programa las cuales son: pandas, matplotlib, seaborn y scikit-learn. Seguido a esto se inicia con la carga del proyecto por medio de src donde contamos con:

1. **Analisis.py**
   se encarga de realizar análisis estadísticos o evaluativos sobre los datos o los resultados obtenidos tras aplicar los algoritmos de clustering. Dependiendo de la implementación, el cual incluye un cálculo de métricas como la inercia, silueta, o coeficiente de Dunn para medir la calidad de los clusters generados, estadísticas descriptivas que resumen las características principales de cada cluster (por ejemplo, media de estadísticas de Pokémon en cada grupo). Aquí decidimos incluir 4 tipos del cluster secundarios los cuales son
   - Peluche_score el cual busca identificar qué tan "peluche" (débil o frágil) es un Pokémon, asignando mayor puntaje a los que tienen bajos valores en HP, Defensa y Velocidad. Se calcula usando la fórmula inversa: cuanto menor el valor de esas estadísticas, mayor será el puntaje

```python
def calcular_peluche_score(df):
    df = df.copy()
    df['peluche_score'] = (
        (1 / (df['HP'] + 1)) * 100 +
        (1 / (df['Defense'] + 1)) * 100 +
        (1 / (df['Speed'] + 1)) * 100
    )
    return df
```

   - Combate_score el cual se baa en la lógica de peluche_score pero enfocado a buscar por asi decirlo el pokemon perfecto

```python
def calcular_combate_score(df):
    df = df.copy()
    df['combate_score'] = (df['HP']*0.2 + df['Attack']*0.4 + df['Defense']*0.2 + df['Speed']*0.2)
    return df
```

   - mejores_generacion el cual iltra los Pokémon legendarios y devuelve el Pokémon con mejor combate_score para cada generación.
   - mejores_generacion_normales: Similar al anterior, pero excluye además variantes especiales como Mega evoluciones o formas regionales para enfocarse en Pokémon "normales".

```python
def mejores_generacion(df):
    df = df.copy()
    if 'combate_score' not in df.columns:
        df = calcular_combate_score(df)
    df = df[~df['Legendary']]
    mejores = df.loc[df.groupby('Generation')['combate_score'].idxmax()]
    return mejores[['Name', 'Generation', 'combate_score']]

def mejores_generacion_normales(df):
    mejores = []
    for gen in sorted(df['Generation'].unique()):
        sub = df[(df['Generation'] == gen) &
                 (~df['Legendary']) &
                 (~df['Name'].str.contains('Mega|X|Y|Alola|Galar|Hisui|Gigantamax', case=False))]

        if not sub.empty:
            mejor = sub.loc[sub['combate_score'].idxmax()]
            mejores.append(mejor)

    return pd.DataFrame(mejores)
```

Hay varias funciones para visualizar datos con Matplotlib, algunas generan gráficos guardados en archivo (ruta_salida), y otras además los muestran en pantalla.

- Gráficos de Pokémon “peluche” (graficar_pokemon_peluche): Muestra todos los Pokémon en un scatterplot donde el eje X es peluche_score y el eje Y es combate_score. Los Pokémon con mayor peluche_score (más débiles) se destacan en rosa.

```python
def graficar_pokemon_peluche(df, ruta_salida, top_n=10):
    df_peluches = df.sort_values(by=['peluche_score', 'combate_score'], ascending=[False, True])
    top_peluches = df_peluches.head(top_n)

    plt.figure(figsize=(10, 6))
    plt.scatter(df['peluche_score'], df['combate_score'], alpha=0.3, color='gray', label='Todos')
    plt.scatter(top_peluches['peluche_score'], top_peluches['combate_score'], color='hotpink', label='Top Peluches')

    for i, row in top_peluches.iterrows():
        plt.text(row['peluche_score'], row['combate_score'], row['Name'], fontsize=8, color='hotpink')

    plt.xlabel('Puntuación Peluche (mayor = más peluche)')
    plt.ylabel('Puntuación Combate (menor = menos útil)')
    plt.title('Pokémon más “peluche”')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(ruta_salida)
    plt.close()

    plt.figure(figsize=(10, 6))
    plt.scatter(df['peluche_score'], df['combate_score'], alpha=0.3, color='gray', label='Todos')
    plt.scatter(top_peluches['peluche_score'], top_peluches['combate_score'], color='hotpink', label='Top Peluches')

    for i, row in top_peluches.iterrows():
        plt.text(row['peluche_score'], row['combate_score'], row['Name'], fontsize=8, color='hotpink')

    plt.xlabel('Puntuación Peluche (mayor = más peluche)')
    plt.ylabel('Puntuación Combate (menor = menos útil)')
    plt.title('Pokémon más “peluche”')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

- Gráficos de Pokémon útiles (graficar_pokemon_utiles): Similar, pero destaca los Pokémon con mejor combate_score en verde.

```python
def graficar_pokemon_utiles(df, ruta_salida, top_n=10):
    df_utiles = df.sort_values(by=['combate_score', 'peluche_score'], ascending=[False, True])
    top_utiles = df_utiles.head(top_n)

    plt.figure(figsize=(10, 6))
    plt.scatter(df['peluche_score'], df['combate_score'], alpha=0.3, color='gray', label='Todos')
    plt.scatter(top_utiles['peluche_score'], top_utiles['combate_score'], color='seagreen', label='Top Útiles')

    for _, row in top_utiles.iterrows():
        plt.text(row['peluche_score'], row['combate_score'], row['Name'], fontsize=8, color='seagreen')

    plt.xlabel('Puntuación Peluche (menor = menos peluche)')
    plt.ylabel('Puntuación Combate (mayor = más útil)')
    plt.title('Pokémon más útiles en combate')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(ruta_salida)
    plt.close()

    plt.figure(figsize=(10, 6))
    plt.scatter(df['peluche_score'], df['combate_score'], alpha=0.3, color='gray', label='Todos')
    plt.scatter(top_utiles['peluche_score'], top_utiles['combate_score'], color='seagreen', label='Top Útiles')

    for _, row in top_utiles.iterrows():
        plt.text(row['peluche_score'], row['combate_score'], row['Name'], fontsize=8, color='seagreen')

    plt.xlabel('Puntuación Peluche (menor = menos peluche)')
    plt.ylabel('Puntuación Combate (mayor = más útil)')
    plt.title('Pokémon más útiles en combate')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

- Mejores Pokémon por generación (graficar_mejores_generacion): Muestra para cada generación el Pokémon con mayor combate_score y destaca también el mejor Pokémon "normal" (no variante) con colores diferentes y anotaciones con sus nombres.

```python
def graficar_mejores_generacion(df, ruta_salida):
    df_filtrado = df[~df['Legendary']]
    mejores = df_filtrado.loc[df_filtrado.groupby('Generation')['combate_score'].idxmax()]
    mejores_normales = mejores_generacion_normales(df)

    plt.figure(figsize=(10, 6))
    plt.scatter(df_filtrado['Generation'], df_filtrado['combate_score'],
                color='lightgray', alpha=0.5, label='Otros Pokémon')

    colores_absolutos = plt.cm.viridis(mejores['Generation'] / mejores['Generation'].max())
    colores_normales = plt.cm.plasma(mejores_normales['Generation'] / mejores_normales['Generation'].max())

    plt.scatter(mejores['Generation'], mejores['combate_score'],
                color=colores_absolutos, label='Mejor por generación', s=100)
    for _, row in mejores.iterrows():
        plt.text(row['Generation'], row['combate_score'] + 2, row['Name'],
                 ha='center', fontsize=8, color='black', fontweight='bold')

    plt.scatter(mejores_normales['Generation'], mejores_normales['combate_score'],
                color=colores_normales, marker='s', label='Mejor normal', s=80)
    for _, row in mejores_normales.iterrows():
        plt.text(row['Generation'], row['combate_score'] - 5, row['Name'],
                 ha='center', fontsize=8, color='blue')

    plt.title("Mejor Pokémon por generación")
    plt.xlabel("Generación")
    plt.ylabel("Puntuación de combate")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(ruta_salida)
    plt.close()

    plt.figure(figsize=(10, 6))
    plt.scatter(df_filtrado['Generation'], df_filtrado['combate_score'],
                color='lightgray', alpha=0.5, label='Otros Pokémon')

    plt.scatter(mejores['Generation'], mejores['combate_score'],
                color=colores_absolutos, label='Mejor por generación', s=100)
    for _, row in mejores.iterrows():
        plt.text(row['Generation'], row['combate_score'] + 2, row['Name'],
                 ha='center', fontsize=8, color='black', fontweight='bold')

    plt.scatter(mejores_normales['Generation'], mejores_normales['combate_score'],
                color=colores_normales, marker='s', label='Mejor normal', s=80)
    for _, row in mejores_normales.iterrows():
```

- Comparación Pokémon útiles vs peluche (graficar_utiles_vs_peluches y graficar_utiles_vs_peluches_3d): Muestra en 2D y 3D la comparación entre los Pokémon más útiles y los más peluche, utilizando sus puntuaciones y en 3D además la estadística de velocidad para dar otra dimensión al análisis.

```python
def graficar_utiles_vs_peluches_3d(df, ruta_salida, top_n=10):
    df = df.copy()
    top_peluches = df.sort_values(by='peluche_score', ascending=False).head(top_n)
    top_utiles = df.sort_values(by='combate_score', ascending=False).head(top_n)

    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')

    ax.scatter(df['peluche_score'], df['combate_score'], df['Speed'],
               color='#665c5d', s=80, alpha=0.8, label='Resto de Pokémon')

    ax.scatter(top_peluches['peluche_score'], top_peluches['combate_score'], top_peluches['Speed'],
               color='orange', s=110, label=f'Top {top_n} Peluches')

    ax.scatter(top_utiles['peluche_score'], top_utiles['combate_score'], top_utiles['Speed'],
               color='#e51a4c', s=110, label=f'Top {top_n} Útiles')

    for _, row in top_peluches.iterrows():
        ax.text(row['peluche_score'], row['combate_score'], row['Speed'],
                row['Name'], color='orange', fontsize=9)

    for _, row in top_utiles.iterrows():
        ax.text(row['peluche_score'], row['combate_score'], row['Speed'],
                row['Name'], color='#e51a4c', fontsize=9)

    ax.set_xlabel('Puntuación Peluche')
    ax.set_ylabel('Puntuación Combate')
    ax.set_zlabel('Velocidad (Speed)')
    ax.set_title(f'Top {top_n} Pokémon más útiles vs más peluches (3D)')
    ax.legend()
    plt.tight_layout()
    plt.savefig(ruta_salida)
    plt.show()
```

2. Cluster principal

2.1. Modelo: este archivo contiene la implementación principal del algoritmo de clustering y la lógica que procesa los datos de Pokémon para agruparlos. Este es el núcleo del análisis no supervisado del proyecto. Sin este archivo, no se realizaría la agrupación que permite descubrir patrones en los datos.

Funciones principales:

- Implementa el algoritmo K-Means para agrupar Pokémon basándose en sus atributos estadísticos.
- Contiene funciones para elegir el número óptimo de clusters, probablemente usando el método del codo (elbow method).
- Asigna cada Pokémon a un cluster y devuelve los resultados para su análisis o visualización.

```python
def entrenar_kmeans(datos_escalados, num_clusters):
    modelo = KMeans(n_clusters=num_clusters, random_state=42, n_init=10)
    etiquetas = modelo.fit_predict(datos_escalados)
    return modelo, etiquetas

def resumen_clusters(datos, etiquetas, columnas):
    datos = datos.copy()
    datos['cluster'] = etiquetas
    resumen = ""

    descripciones = {
        0: "Pokémon rápidos y ofensivos",
        1: "Pokémon defensivos y lentos",
        2: "Pokémon balanceados"
    }

    for c in sorted(datos['cluster'].unique()):
        descripcion = descripciones.get(c, "Cluster desconocido")
        grupo = datos[datos['cluster'] == c]
        resumen += f"\nCluster {c}: {descripcion}\n"
        for col in columnas:
            promedio = grupo[col].mean()
            resumen += f"  - Promedio {col}: {promedio:.2f}\n"
        resumen += f"  - Tamaño del cluster: {len(grupo)}\n"
    return resumen

def graficar_clusters(datos_originales, etiquetas, columna_x, columna_y, ruta_guardado):
    datos_originales = datos_originales.copy()
    datos_originales['Cluster'] = etiquetas

    nombres_clusters = {
        0: "Pokémon rápidos y ofensivos",
        1: "Pokémon defensivos y lentos",
        2: "Pokémon balanceados"
    }
    datos_originales['cluster_nombre'] = datos_originales['Cluster'].map(nombres_clusters)
```

2.2. Utils: este archivo reúne funciones auxiliares que apoyan al resto del código, para no repetir lógica y facilitar la organización; facilitando la modularidad y limpieza del código principal, haciendo más sencillo reutilizar funciones comunes.

Funciones típicas:

- Carga y preprocesamiento de datos (normalización, manejo de valores faltantes).
- Funciones para guardar y cargar archivos, resultados o modelos.
- Métodos para calcular métricas o preparar datos para visualización.

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler

def cargar_datos(ruta_archivo):
    """Carga un archivo CSV de Pokémon desde la ruta especificada."""
    try:
        return pd.read_csv(ruta_archivo)
    except Exception as e:
        print(f"Error cargando los datos: {e}")
        return None

def limpiar_datos(df):
    """Limpia los datos eliminando filas con valores nulos."""
    df = df.dropna()
    return df

def seleccionar_caracteristicas(df, columnas):
    """Selecciona columnas específicas para el análisis."""
    return df[columnas]

def escalar_datos(df):
    """Escala los datos con StandardScaler (media = 0, desviación estándar = 1)."""
    scaler = StandardScaler()
    datos_escalados = scaler.fit_transform(df)
    return datos_escalados
```

2.3. Visualización: este archivo contiene las funciones que generan gráficos y visualizaciones para interpretar los resultados del clustering**, también** permite transformar datos complejos en imágenes intuitivas, facilitando la interpretación y comunicación de los hallazgos del análisis.

Funciones comunes:

- Gráficos de dispersión (scatterplots) para mostrar los clusters en dos dimensiones.
- Gráficos de barras para representar distribución de Pokémon en cada cluster.
- Visualizaciones que ayudan a comparar grupos de Pokémon (ej. mejores vs. “peluches”).

```python
def graficar_clusters(df, columnas, etiquetas, ruta="visuals/clusters.png"):
    """
    Dibuja un scatterplot de los clusters según dos columnas.
    """
    col_x, col_y = columnas[1], columnas[2]
    df['cluster'] = etiquetas

    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=df, x=col_x, y=col_y, hue='cluster', palette='Set2', s=60)
    plt.title(f'Clusters de Pokémon por {col_x} y {col_y}')
    plt.xlabel(col_x.capitalize())
    plt.ylabel(col_y.capitalize())
    plt.legend(title='Cluster')
    plt.tight_layout()
    plt.savefig(ruta)
    plt.close()

def guardar_resumen_clusters(df, columnas, num_clusters, ruta):
    """
    Guarda estadísticas de cada cluster en un archivo de texto.
    """
    with open(ruta, "w", encoding="utf-8") as f:
        f.write(f"Resumen del análisis de clustering (K-Means, {num_clusters} clusters):\n\n")
        for i in range(num_clusters):
            cluster_df = df[df['cluster'] == i]
            f.write(f"Cluster {i}:\n")
            for col in columnas:
                promedio = cluster_df[col].mean()
                f.write(f"  - Promedio {col}: {promedio:.2f}\n")
            f.write(f"  - Tamaño del cluster: {len(cluster_df)}\n\n")
```

3. Aplicación del K-means

Antes de aplicar K-Means, los datos deben estar en un formato numérico adecuado y generalmente normalizados o estandarizados para que todas las variables tengan igual peso. En el caso de Pokémon, se usan atributos como: puntos de Vida, ataque, defensa, velocidad y otros atributos numéricos. Después se extrae un array de características con estas estadísticas para todos los Pokémon.

Luego El número de clusters K debe definirse antes de correr K-Means. Para elegir un buen K, se usa comúnmente el método del codo (elbow method):

- Se calcula el modelo K-Means para varios valores de K (por ejemplo, de 1 a 10).
- Para cada K se calcula la inercia, que mide la suma de las distancias cuadráticas dentro de cada cluster.
- Se grafica la inercia versus K, y se busca el punto donde la reducción de inercia se vuelve menos significativa (el "codo").
- Ese punto sugiere el número óptimo de clusters.
-

4. Conclusión

La implementación demuestra cómo preparar y normalizar datos, seleccionar el número óptimo de clusters con el método del codo, y visualizar los resultados para facilitar su interpretación. Esto permite una mejor comprensión de la diversidad y roles de los Pokémon dentro de su universo, mostrando agrupaciones que pueden reflejar características comunes como fuerza, velocidad o resistencia.

Además, el proyecto integra funciones auxiliares y visualizaciones que enriquecen el análisis, haciendo que los resultados sean accesibles tanto para expertos en ciencia de datos como para aficionados al mundo Pokémon.