Documentación: Aplicación de Benchmarking y Análisis de Jugadoras

Visión General

Esta aplicación web integral está diseñada para analizar jugadoras de fútbol femenino de las cinco principales ligas europeas. La plataforma proporciona estadísticas detalladas de las jugadoras, herramientas de análisis comparativo y funciones de identificación de talento, con un enfoque particular en el Atlético de Madrid, pero cubriendo jugadoras de las principales ligas femeninas, incluyendo Liga F (España), WSL (Inglaterra), Frauen-Bundesliga (Alemania), Serie A (Italia) y D1 Féminine (Francia).

Requisitos del Sistema

- Python 3.7+
- Streamlit
- Pandas
- NumPy
- Matplotlib
- Seaborn
- scikit-learn
- SciPy

Fuentes de Datos

La aplicación utiliza varios archivos CSV almacenados en una estructura de directorios:

- `data/data_gold/df_keepers_gold_1.csv`: Estadísticas específicas de porteras
- `data/data_gold/df_players_gold_1.csv`: Estadísticas de jugadoras de campo
- `data/players_info/df_230_players_info.csv`: Información adicional de jugadoras
- `data/players_info/big/df_players_info_global.csv`: Detalles ampliados de jugadoras
- `data/teams_info/big/df_teams_info_global.csv`: Información de equipos incluyendo escudos
- `data/players_info/atm_pics.csv`: Fotos de jugadoras del Atlético de Madrid

Estructura de la Aplicación

La aplicación sigue un diseño modular con varias páginas, cada una con propósitos analíticos distintos:

1. Página Principal (`Main_page.py`)

La página de inicio que muestra el título de la aplicación, los logotipos de las cinco principales ligas de fútbol femenino e instrucciones de navegación.

2. Análisis de Equipo (`Equipo_propio.py`)

Centrado en las jugadoras del Atlético de Madrid con estadísticas individuales detalladas:

- Perfiles de jugadoras con fotos e información básica
- Métricas de rendimiento con análisis comparativo
- Métricas con código de colores que muestran el rendimiento en relación con los promedios de la liga y la posición

3. Búsqueda de Jugadoras (`Buscar_jugadoras.py`)

Una herramienta completa de búsqueda de jugadoras con múltiples opciones de filtrado:

- Filtro por liga, club, posición y año de nacimiento
- Perfiles detallados de jugadoras con fotos y escudos de clubes
- Extensas métricas de rendimiento con comparaciones con la liga
- Representación visual de métricas en relación con los promedios por posición

4. Comparación de Jugadoras (`Comparar_jugadoras completo.py`)

Análisis comparativo avanzado entre dos jugadoras:

- Visualización de perfiles lado a lado
- Índices de similitud utilizando distancia euclidiana y métodos basados en percentiles
- Gráficos de radar que visualizan fortalezas relativas
- Métricas detalladas organizadas en categorías macro, meso y micro
- Comparaciones de percentiles que muestran la posición relativa dentro de grupos por posición

5. Análisis de Reemplazo de Jugadoras (`Reemplazar_jugadora final.py`)

Herramienta sofisticada para identificar jugadoras similares utilizando aprendizaje automático:

- PCA (Análisis de Componentes Principales) para reducción de dimensionalidad
- Clustering K-means para identificar grupos de jugadoras similares
- Visualizaciones de clusters de jugadoras
- Comparaciones detalladas de métricas entre la jugadora seleccionada y posibles reemplazos
- Análisis DAFO (Debilidades, Amenazas, Fortalezas, Oportunidades) para las jugadoras evaluadas

6. Identificación de Talentos Emergentes (`Talentos_emergentes.py`)

Herramienta especializada para descubrir y evaluar jóvenes talentos menores de 23 años:

- Índices de talento personalizados por posición
- Rankings de talento con opciones de filtrado
- Tarjetas detalladas de jugadoras con puntuaciones estandarizadas
- Análisis de distribución por edad
- Comparaciones con jugadoras del Atlético de Madrid
- Interpretación de índices de talento específicos por posición

Características Principales

Métricas Específicas por Posición

La plataforma utiliza métricas adaptadas para diferentes posiciones:

- **Porteras (GK)**: Porcentaje de paradas, porterías a cero, métricas de goles esperados, estadísticas de distribución
- **Defensas (DF)**: Entradas, intercepciones, bloqueos, precisión de pase, posicionamiento defensivo
- **Centrocampistas (MF)**: Pases, métricas de creación, contribuciones defensivas, acciones progresivas
- **Delanteras (FW)**: Goles, goles esperados, eficiencia de tiro, posicionamiento ofensivo

Herramientas Analíticas

1. **Análisis Comparativo**:
    - Comparaciones con promedios de liga
    - Comparaciones con promedios por posición
    - Rankings por percentiles
2. **Métodos de Visualización**:
    - Gráficos de radar para perfiles de jugadoras
    - Gráficos de barras para comparaciones de métricas
    - Visualizaciones de clustering
    - Análisis de distribución por edad
3. **Integración de Aprendizaje Automático**:
    - Análisis de Componentes Principales (PCA) para reducción de dimensionalidad
    - Clustering K-means para similitud de jugadoras
    - Cálculos de distancia euclidiana para emparejamiento de jugadoras
4. **Índices Personalizados**:
    - Índices compuestos específicos por posición
    - Índices defensivos, creativos y basados en posesión
    - Evaluación de potencial para jugadoras jóvenes

Detalles de Implementación

Procesamiento de Datos

1. **Carga de Datos**:

```
@st.cache_data
def cargar_datos():
    # Carga de diferentes archivos CSV
    df_keepers = pd.read_csv("data/data_gold/df_keepers_gold_1.csv")
    df_players = pd.read_csv("data/data_gold/df_players_gold_1.csv")
    # Carga de datos adicionales...

    # Estandarizar nombres de columnas y combinar dataframes
    df_combined = pd.concat([df_keepers, df_players],
ignore_index=True)
```

```
    return df_combined, df_players_info, df_teams_info, df_atm_photos
```

2. **Métricas Específicas por Posición**:

```
position_metrics = {
    'GK': ['MP', 'Starts', 'Min', 'GA', 'GA90', 'SoTA', 'Save%',
'CS%', ...],
    'DF': ['MP', 'Starts', 'Min', 'Min%', 'Tkl/90', 'Tkl%', 'Blocks',
...],
    'MF': ['MP', 'Starts', 'Min', 'Min%', 'Gls', 'Ast', 'G+A',
'SCA90', ...],
    'FW': ['MP', 'Starts', 'Min', 'Min%', 'Gls', 'Ast', 'G+A',
'SoT/90', ...]
}
```

3. **Categorización de Métricas**:

```
metrics_by_level = {
    'macro': position_metrics.get(player_position, {}).get('macro',
[]),
    'meso': position_metrics.get(player_position, {}).get('meso', []),
    'micro': position_metrics.get(player_position, {}).get('micro',
[])
}
```

Cálculo de Similitud

1. **Similitud Basada en Distancia Euclidiana**:

```
def calcular_similitud(metrics1, metrics2, position_metrics,
position):
    # Obtener métricas para la posición
    relevant_metrics = []
    for level_metrics in position_metrics.get(position, {}).values():
        relevant_metrics.extend(level_metrics)

    # Encontrar métricas comunes
    common_metrics = [m for m in relevant_metrics if m in metrics1 and
m in metrics2]

    # Crear vectores y normalizar
    vector1 = np.array([metrics1.get(m, 0) for m in common_metrics])
    vector2 = np.array([metrics2.get(m, 0) for m in common_metrics])

    # Calcular distancia euclidiana normalizada
    euclidean_distance = np.sqrt(np.sum((normalized_vector1 -
```

```
normalized_vector2) ** 2))
    max_possible_distance = np.sqrt(len(common_metrics))

    # Convertir a porcentaje de similitud
    similarity = (1 - (euclidean_distance / max_possible_distance)) *
100

    return max(0, similarity)
```

2. **Similitud Basada en Percentiles**:

```
def calcular_similitud_percentiles(df_player1, df_player2,
metrics_list, position):
    # Calcular percentiles para cada jugadora
    for metric in common_metrics:
        # Obtener valores de las jugadoras
        player1_value = df_player1[metric].iloc[0]
        player2_value = df_player2[metric].iloc[0]

        # Calcular percentiles comparados con el grupo de posición
        percentile1 = stats.percentileofscore(metric_values,
player1_value)
        percentile2 = stats.percentileofscore(metric_values,
player2_value)

        # Calcular similitud basada en diferencia de percentiles
        diff = abs(percentile1 - percentile2)
        similarity = 100 - diff  # 100 significa percentiles idénticos
        similarities.append(similarity)

    # Similitud promedio a través de todas las métricas
    avg_similarity = sum(similarities) / len(similarities)

    return avg_similarity
```

Índices de Talento

1. **Definición de Índices**:

```
talent_indices = {
    'GK': {
        'Índice de Potencial Defensivo': {
            'metricas': ['Save%', 'CS%', 'PSxG-GA'],
            'pesos': [0.4, 0.3, 0.3],
            'min_threshold': {'MP': 5, 'Min': 270},
            'descripcion': 'Evalúa la capacidad fundamental para
evitar goles...'
        },
        # Otros índices...
```

```
    },
    # Otras posiciones...
}
```

2. **Cálculo de Índices**:

```
def calcular_indices_talento(df, edad_maxima=23):
    # Filtrar jugadoras jóvenes
    año_actual = 2025
    df_jovenes = df[df['Born'] >= (año_actual - edad_maxima)].copy()

    # Calcular índices para cada posición
    for posicion, indices in talent_indices.items():
        df_posicion = df_jovenes[df_jovenes['Posición Principal'] ==
posicion].copy()

        for nombre_indice, definicion in indices.items():
            metricas = definicion['metricas']
            pesos = definicion['pesos']
            min_threshold = definicion['min_threshold']

            # Aplicar umbrales mínimos
            df_valido = df_posicion.copy()
            for metrica, valor in min_threshold.items():
                df_valido = df_valido[df_valido[metrica] >= valor]

            # Calcular valores normalizados y suma ponderada
            for _, row in df_valido.iterrows():
                valor_indice = 0
                for i, metrica in enumerate(metricas):
                    valor_norm = (row[metrica] - min_vals[metrica]) /
(max_vals[metrica] - min_vals[metrica])
                    valor_indice += valor_norm * pesos[i]

                # Escalar a 0-100
                valor_final = valor_indice * 100

                # Almacenar resultados
                resultados[posicion][nombre_indice][jugadora] = {
                    'valor': valor_final,
                    # Datos adicionales...
                }
```

Despliegue y Navegación

La aplicación utiliza el sistema de navegación de Streamlit con un menú lateral:

```
import streamlit as st
from utils import display_logo
```

```
st.set_page_config(page_title="Análisis de Jugadoras", layout="wide")

main_page = st.Page("pages/Main_page.py", title="Inicio", icon="🏠")
page_1= st.Page("pages/Equipo_propio.py", title="Equipo Propio",
icon="🇲🇦")
page_2= st.Page("pages/Buscar_jugadoras.py", title="Buscar Jugadoras",
icon="🔎")
page_3 = st.Page("pages/Comparar_jugadoras completo.py", title="Comparar
Jugadoras", icon="📊")
page_4 = st.Page("pages/Reemplazar_jugadora final.py", title="Reemplazar
Jugadoras", icon="🔄")
page_5 = st.Page("pages/Talentos_emergentes.py", title="Talentos
emergentes", icon="🌟")

pg = st.navigation([main_page, page_1, page_2, page_3, page_4, page_5])
pg.run()
```

Ejecución de la Aplicación

Para ejecutar la aplicación localmente:

1. Asegúrate de que todas las dependencias están instaladas:

   ```
   pip install streamlit pandas numpy matplotlib seaborn scikit-learn
   scipy
   ```

2. Verifica que todos los archivos de datos están en la estructura de directorios correcta

3. Lanza la aplicación:

   ```
   streamlit run app.py
   ```

Notas Adicionales

- La aplicación está diseñada con el Atlético de Madrid como equipo focal, pero incluye capacidades de análisis para todas las principales ligas europeas femeninas
- Todas las métricas están normalizadas en relación con los grupos de posición para comparaciones justas
- El sistema de identificación de talento está calibrado para jugadoras menores de 23 años con umbrales personalizados para cada posición
- La aplicación utiliza un diseño responsivo que funciona en varios tamaños de pantalla

Esta documentación proporciona una visión general completa de la Plataforma de Análisis de Fútbol Femenino, explicando su estructura, características y detalles de implementación.