Creación de un dashboard

Motivación

Después de haber detallado el diseño de los cubos y los indicadores clave de rendimiento, el siguiente paso es la creación de un cuadro de mando (dashboard) que nos permitirá evaluar si estamos alcanzando los objetivos establecidos. Este cuadro de mando estará enfocado principalmente en los indicadores relacionados con la eficacia en la recopilación de datos de los partidos de fútbol.

El análisis se centrará en identificar patrones ocultos en estos datos, considerando diversas dimensiones. Estas dimensiones incluyen el aspecto temporal de los datos, como fechas y horas de los eventos; el tipo de evento (como pases, tiros, goles); el tipo de cámara utilizada para la recopilación de datos; y las características específicas de cada estadio.

Este enfoque integral y multidimensional nos ayudará a descubrir tendencias, anomalías o patrones consistentes, lo que a su vez facilitará la toma de decisiones informadas y la implementación de mejoras en el proceso de recopilación de datos.

Usuario final

El usuario final del cuadro de mando será un directivo de la empresa. Esta clase de persona se caracteriza por tener un conocimiento muy básico a nivel técnico en análisis de datos, pero también por la necesidad de obtener información rápida y clara para la toma de decisiones. El diseño del cuadro de mando debe ser intuitivo y enfocado en mostrar los KPIs más relevantes de manera que permitan una rápida interpretación y acción.

Propósito

El propósito del cuadro de mando es proporcionar una visión clara y actualizada del rendimiento de *Lemon Analytics* en cuanto a la recopilación y análisis de datos, segmentado por tipo de cliente. Los aspectos clave que busca mejorar en el proceso de negocio incluyen:

- **Eficiencia en la recopilación de datos**: Evaluar la precisión y rapidez con la

que se capturan los eventos, ayudando a identificar posibles áreas de mejora en la tecnología de captura.

- **Evolución temporal del rendimiento**: Visualizar si los indicadores han mejorado en una ventana de tiempo de unos 5 meses, lo cuál ayuda a identificar si los recientes cambios de la empresa han funcionado.

- **Distribución geográfica y desempeño del servicio**: Visualizar la cobertura y eficacia del servicio en diferentes regiones, lo que es crucial para estrategias de expansión y marketing.

Frecuencia

La actualización de los datos del cuadro de mando será periódica (push). Los datos se actualizarán de forma periódica, posiblemente al final de cada semana, para reflejar los últimos partidos y eventos analizados. Como las decisiones que se tomarán sobre este cuadro de mando son a largo plazo y cada actualización contendrá muchísimos datos (cada partido estimamos que tiene 1000 eventos y cada equipo juega un partido por semana), por lo que no tiene sentido que la frecuencia sea menor.

Adopción

La integración del cuadro de mando en el contexto de la empresa y su impacto en el proceso de negocio podrían seguir estos pasos:

- **Formación**: Los analistas de datos que lo creen deberán presentar el cuadro de mando a los usuarios finales, explicando su uso y beneficios. Además, sería conveniente proporcionar formación para garantizar su correcta interpretación y uso.

- **Integración en la toma de decisiones**: Una vez los usuarios finales controlen el cuadro de mando y sepan interpretar sus resultados, deberían empezar a utilizar este para tomar decisiones estratégicas y operativas. Esto puede incluir ajustar estrategias de captura de datos, rediseñar procesos de análisis, o tomar decisiones de expansión basadas en los datos geográficos.

- **Revisión y mejora**: Periódicamente, los usuarios finales se reunirán con los analistas de datos para transmitirles la utilidad del cuadro de mando, que ellos utilizarán para hacer ajustes y mejoras en el diseño o en la presentación de los datos.

Gráficas

Para conseguir el fin del cuadro de mando se usarán cuatro gráficas, que explicamos a continuación.

Evolución del porcentaje de eventos correctamente recopilados

Esta gráfica de línea muestra el porcentaje de eventos correctamente recopilados a lo largo de los últimos 5 meses. Se utiliza para evaluar la precisión en la recopilación de datos en función del tipo de cliente seleccionado mediante un menú desplegable. La línea azul facilita la visualización del rendimiento a lo largo del tiempo, permitiendo identificar tendencias y posibles áreas de mejora.

Característica	**Detalle**
Dimensiones	DatosEvento, Fecha
Medidas	Porcentaje de eventos correctamente recopilados
Filtrado/Interactividad	Selección de tipo de cliente mediante un menú desplegable
Estética	Gráfico de línea con líneas azules
Propósito	Evaluar la precisión en la recopilación de datos

: Características del gráfico de línea

: Gráfico de línea

Diferentes tipos de eventos y porcentaje de corrección

Esta gráfica de barras apiladas ofrece una visión detallada de la cantidad de eventos, diferenciando entre eventos correctos y fallados según el tipo de estos. Permite identificar qué tipos de eventos tienen mayor tasa de corrección y cuáles son más susceptibles a errores, ofreciendo una oportunidad para mejorar en áreas específicas. La interactividad

se mantiene a través de la selección del tipo de cliente, y la utilización de colores azul y rojo ayuda a diferenciar claramente entre eventos correctos y fallidos.

Característica	**Detalle**
Dimensiones	DatosEvento
Medidas	Cantidad de eventos (correctos y fallidos)
Filtrado/Interactividad	Selección de tipo de cliente mediante un menú desplegable
Estética	Barras apiladas, colores azul y rojo
Propósito	Identificar tipos de eventos con mayor tasa de corrección

: Características de la gráfica de barras

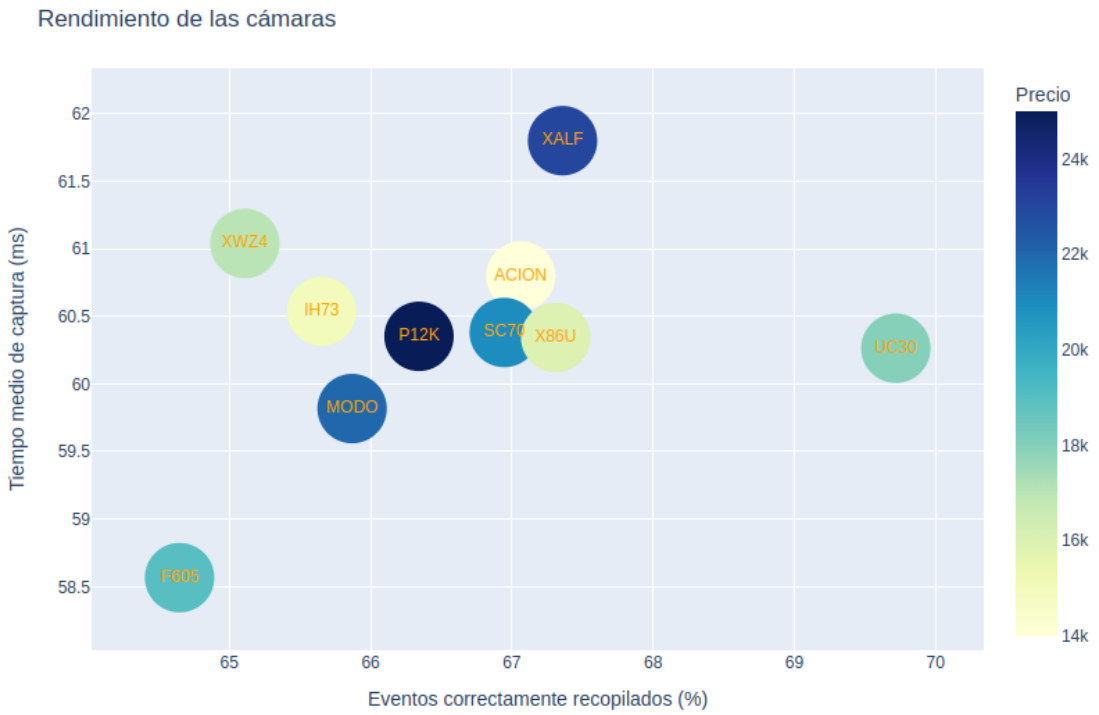
: Gráfica de barras

Rendimiento de las cámaras

El gráfico de burbujas presenta el rendimiento de las cámaras utilizadas en la recopilación de datos. Se enfoca en el tiempo medio de captura, el precio y la tasa de eventos correctos, diferenciando cada cámara por su tipo y abreviatura. Esta visualización permite comparar el rendimiento y coste de las diferentes cámaras, ayudando a tomar decisiones informadas sobre el equipo utilizado en la recopilación de datos. Las burbujas de diferentes colores, en un gradiente de azul, facilitan la comparación y el análisis.

Característica	**Detalle**
Dimensiones	Cámara, DatosEvento
Medidas	Tiempo medio de captura, Precio, Tasa de eventos correctos
Filtrado/Interactividad	Selección de tipo de cliente
Estética	Burbujas de diferentes tamaños y colores en gradiente de azul
Propósito	Comparar rendimiento y coste de cámaras

: Características del gráfico de burbujas



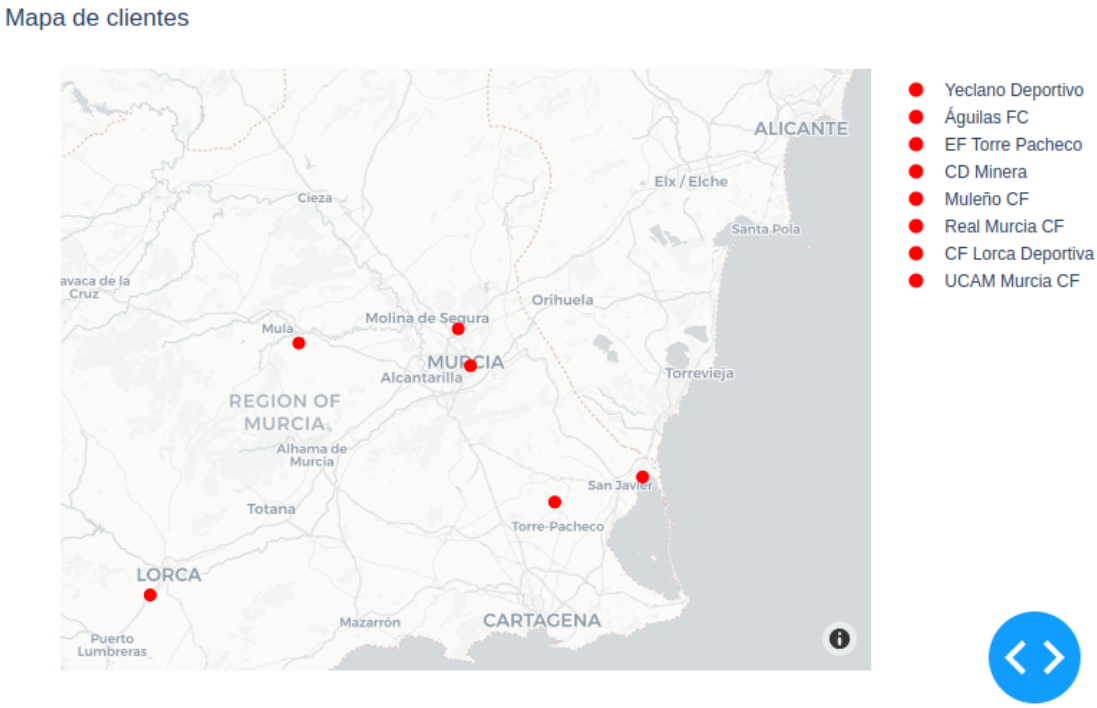
: Gráfico de burbujas

Distribución geográfica de clientes

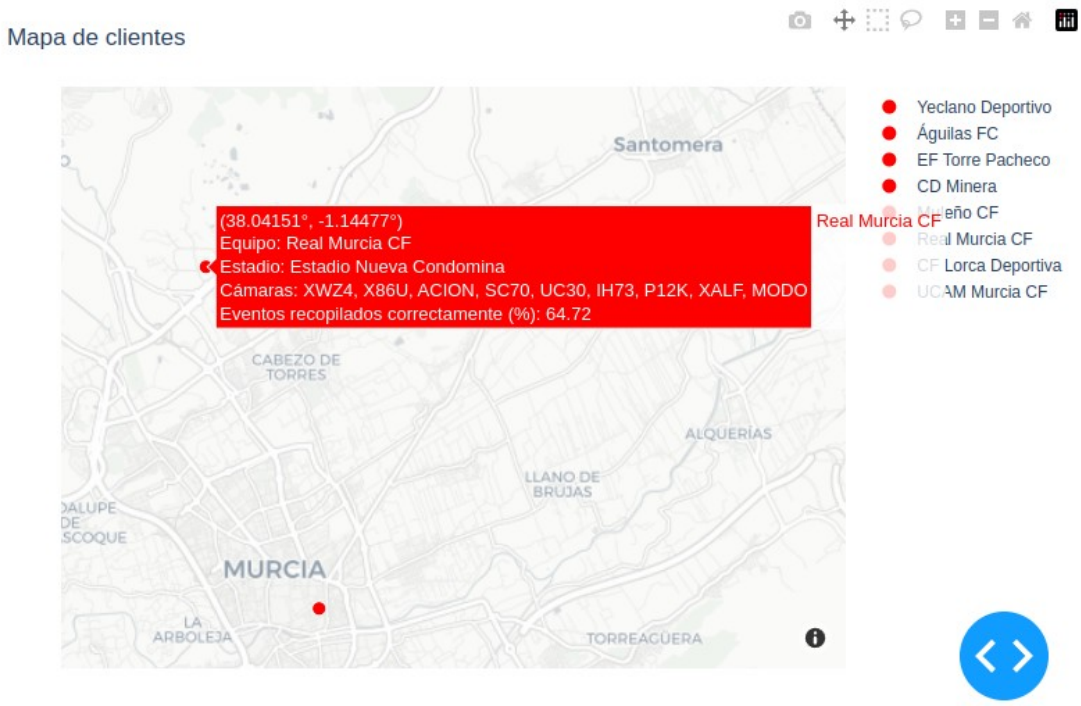
Mapa interactivo que muestra la distribución geográfica de los clientes y el rendimiento de nuestras cámaras en sus estadios. Utiliza las coordenadas del estadio de cada equipo para ubicarlo en el mapa y presenta la tasa de eventos correctamente recopilados en cada ubicación. Los marcadores rojos con información detallada permiten una visión detallada de los datos y el rendimiento por regiones.

Característica	**Detalle**
Dimensiones	Cliente, Cámara, Estadio
Medidas	Tasa de Eventos Correctamente Recopilados
Filtrado/Interactividad	Selección de tipo de cliente
Estética	Mapa interactivo, marcadores rojos y tooltips
Propósito	Visualizar distribución geográfica y rendimiento por regiones

: Características del gráfico de mapa

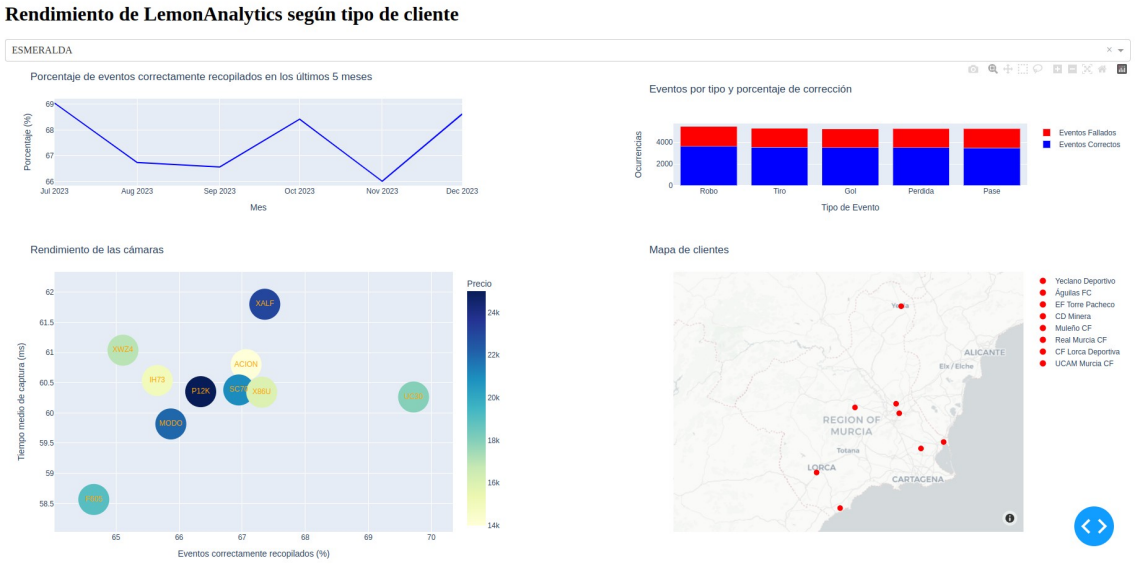


: Gráfico de mapa



: Detalles de gráfico de mapa

Distribución final del Dashboard



: Dashboard final

Generación de datos sintéticos

Para generar el conjunto de datos, se ha implementado un script de Python que combina aleatoriedad controlada con entradas manuales específicas. Utilizando la biblioteca Faker para simular datos aleatorios como los nombres de los jugadores, la hora de los partidos y la categoría de los equipos. Sin embargo, algunos campos, como los nombres de los equipos y las cámaras, se han especificado manualmente para mantener un nivel de consistencia y precisión. Además, se han tenido que definir funciones y estructuras de datos para mantener cierta coherencia en el conjunto de datos (como que un equipo siempre tenga el mismo estadio o que un jugador sólo pueda pertenecer a un equipo).

Código

Código del dashboard

```
import dash
from dash import dcc
from dash import html
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import datetime
```

```
import geopandas as gpd
from dash.dependencies import Input, Output

geojson_file = 'spain-provinces.geojson'
gdf = gpd.read_file(geojson_file)


df = pd.read_csv('datos_futbol.csv')

# Convertir la columna 'Fecha del Partido' a tipo fecha
df['Fecha del Partido'] = pd.to_datetime(df['Fecha del Partido'])

app = dash.Dash(__name__)

# Obtener una lista de tipos de cliente unicos
tipos_de_cliente = df['Categoria Equipo'].unique()

colores = px.colors.qualitative.Set1

app.layout = html.Div([
   html.H1('Rendimiento de LemonAnalytics segun tipo de cliente'),

   # Dropdown para seleccionar el tipo de cliente
   dcc.Dropdown(
      id='cliente-dropdown',
      options=[{'label': cliente, 'value': cliente} for cliente in
          tipos_de_cliente],
      value=tipos_de_cliente.tolist()[0], # Valor predeterminado al primer tipo
           de cliente
      multi=False
   ),

   html.Div([
      dcc.Graph(id='line-chart', style={'width': '100%', 'height': '30vh'})
   ], style={'width': '45%', 'float': 'left'}),


   html.Div([
      dcc.Graph(id='bar-chart', style={'width': '100%', 'height': '30vh'})
   ], style={'width': '45%', 'float': 'right'}),


html.Div([
   dcc.Graph(id='bubble-camera', style={'width': '100%', 'height': '60vh'})
], style={'width': '45%', 'float': 'left'}),


html.Div([
   dcc.Graph(id='spain-map', style={'width': '100%', 'height': '60vh'})
```

```
],style={'width': '45%', 'float': 'right'})


])

@app.callback(
   Output('line-chart', 'figure'),
   [Input('cliente-dropdown', 'value')]
)
def update_graph(selected_cliente):
   fig = px.line()

   filtered_df = df[df['Categoria Equipo'] == selected_cliente]

   today = datetime.date.today()
   five_months_ago = today - pd.DateOffset(months=5)

   filtered_df = filtered_df[filtered_df['Fecha del Partido'] >=
       five_months_ago]

   porcentaje_correctos_mes = filtered_df.groupby(filtered_df['Fecha del
       Partido'].dt.to_period('M')).apply(lambda x: (1 - x['Corregido'].sum() /
        len(x)) * 100)

   df_porcentaje = pd.DataFrame({
      'Mes': porcentaje_correctos_mes.index.to_timestamp(),
      'Porcentaje': porcentaje_correctos_mes.values
   })

   fig.add_scatter(x=df_porcentaje['Mes'], y=df_porcentaje['Porcentaje'], mode=
       'lines', name=selected_cliente, line=dict(color='blue'))

   fig.update_layout(
      title=f'Porcentaje de eventos correctamente recopilados en los ultimos 5
          meses',
      xaxis_title='Mes',
      yaxis_title='Porcentaje (%)'
   )

   fig.update_xaxes(
      tickmode='array',
      tickvals=df_porcentaje['Mes'],
      ticktext=[fecha.strftime('%b %Y') for fecha in df_porcentaje['Mes']])

   return fig


@app.callback(
   Output('bar-chart', 'figure'),
```

```
    [Input('cliente-dropdown', 'value')]
)
def update_bar_chart(selected_cliente):
    filtered_df = df[df['Categoria Equipo'] == selected_cliente]
    eventos_fallados = filtered_df[filtered_df['Corregido'] == 1]
    eventos_correctos = filtered_df[filtered_df['Corregido'] == 0]

    conteo_eventos_fallados = eventos_fallados['Tipo de Evento'].value_counts()
    conteo_eventos_correctos = eventos_correctos['Tipo de Evento'].value_counts
        ()

    bar_fig = go.Figure()
    bar_fig.add_trace(go.Bar(x=conteo_eventos_correctos.index, y=
        conteo_eventos_correctos.values, name='Eventos Correctos', marker_color=
        'blue'))
    bar_fig.add_trace(go.Bar(x=conteo_eventos_fallados.index, y=
        conteo_eventos_fallados.values, name='Eventos Fallados', marker_color='
        red'))

    bar_fig.update_layout(
        title=f'Eventos por tipo y porcentaje de correccion',
        xaxis_title='Tipo de Evento',
        yaxis_title='Ocurrencias',
        barmode='stack' # Para apilar las barras
    )

    return bar_fig


@app.callback(
    Output('bubble-camera', 'figure'),
    [Input('cliente-dropdown', 'value')]
)
def update_bubble_chart(selected_cliente):

    filtered_df = df[df['Categoria Equipo'] == selected_cliente]
    tipos_de_camara = filtered_df['Camara'].unique()
    resultados = []

    for tipo_camara in tipos_de_camara:
        camara_df = filtered_df[filtered_df['Camara'] == tipo_camara].copy()

        formato = "%H:%M:%S"
        # Calcular el tiempo medio de captura para esta camara
        camara_df['Hora de Recoleccion'] = pd.to_datetime(camara_df['Hora de
            Recoleccion'], format = formato)
        camara_df['Hora del Evento'] = pd.to_datetime(camara_df['Hora del Evento'
            ], format = formato)
```

```
        camara_df['tiempo_captura'] = (camara_df['Hora de Recoleccion'] -
            camara_df['Hora del Evento']).dt.total_seconds()
        camara_df['tiempo_captura'] = camara_df['tiempo_captura'].apply(lambda x:
             x if x >= 0 else x + 86400) # Sumar 24 horas si es negativo

        tiempo_medio_captura = camara_df['tiempo_captura'].mean()

        tasa_eventos_correctos = (1 - camara_df['Corregido'].mean())*100

        precio = camara_df['Precio'].iloc[0]
        abreviatura = camara_df['Camara abreviatura'].iloc[0]

        resultados.append({'Tipo de Camara': tipo_camara, 'Tiempo Medio de
            Captura': tiempo_medio_captura, 'Precio': precio, 'Abreviatura':
            abreviatura,'Tasa de eventos correctos': tasa_eventos_correctos})

    fig = px.scatter(resultados,
                 x='Tasa de eventos correctos',
                 y='Tiempo Medio de Captura',
                 text='Abreviatura',
                 color='Precio',
                 color_continuous_scale='YlGnBu',
                 hover_data=['Tipo de Camara', 'Abreviatura'])

    fig.update_traces(textfont=dict(color='orange'), marker=dict(size=50))

    fig.update_layout(
        title='Rendimiento de las camaras',
        xaxis_title='Eventos correctamente recopilados (%)',
        yaxis_title='Tiempo medio de captura (ms)',
        showlegend=False
    )

    return fig

@app.callback(
    Output('spain-map', 'figure'),
    [Input('cliente-dropdown', 'value')]
)
def update_spain_map(selected_cliente):
    fig = px.choropleth_mapbox(
        gdf,
        geojson=gdf.geometry,
        color=None,
        mapbox_style="carto-positron",
        center={"lat": 37.98, "lon": -1.13},
    )
```

```
    filtered_df = df[df['Categoria Equipo'] == selected_cliente]
    equipos_unicos = filtered_df['Equipo Local'].unique()
    coordenadas_equipos = filtered_df[['Equipo Local', 'Coordenada X', '
        Coordenada Y']].drop_duplicates()

    for index, row in coordenadas_equipos.iterrows():
        equipo = row['Equipo Local']
        coordenada_x = row['Coordenada X']
        coordenada_y = row['Coordenada Y']

        estadio = df[df['Equipo Local'] == equipo]['Estadio'].iloc[0]
        camaras = df[df['Estadio'] == estadio]['Camara abreviatura'].unique()

        tasa_eventos_corregidos = (1 - df[(df['Estadio'] == estadio)]['Corregido'
            ].mean())*100

        texto_marcador = f"Equipo: {equipo}<br>Estadio: {estadio}<br>Camaras: {',
            '.join(camaras)}<br>Eventos recopilados correctamente (%): {
            tasa_eventos_corregidos:.2f}"

        fig.add_trace(
            go.Scattermapbox(
                lat=[coordenada_y],
                lon=[coordenada_x],
                mode='markers',
                marker=dict(size=10, color='red'),
                text=texto_marcador,
                name=equipo
            )
        )

        fig.update_layout(
        title='Mapa de clientes'
    )

    return fig


if __name__ == '__main__':
    app.run_server(debug=True)
```

Código de la generación de datos

```
    import random
import csv
from faker import Faker
from datetime import datetime, timedelta, date
```

```
fake = Faker('es_ES')

equipos = [
    "Real Murcia CF",
    "FC Cartagena",
    "UCAM Murcia CF",
    "Lorca FC",
    "Yeclano Deportivo",
    "CF Lorca Deportiva",
    "Mar Menor FC",
    "CD Minera",
    "CD Algar",
    "UD Los Garres",
    "Muleno CF",
    "Ciudad de Cieza",
    "CD Bullense",
    "Caravaca CF",
    "Bala Azul CF",
    "Aguilas FC",
    "CF La Union",
    "Racing Murcia FC",
    "EDF Logrono",
    "EF Alhama",
    "Moratalla CF",
    "CD Plus Ultra",
    "FC Pinatar Arena",
    "CD Almazora",
    "CD Lumbreras",
    "CD Jumilla",
    "Olimpico de Totana",
    "CF Beniel",
    "EF Torre Pacheco",
    "CD Alquerias"
]

# Diccionario para asociar cada equipo con su lista de jugadores
jugadores_por_equipo = {equipo: [fake.name_male() for _ in range(20)] for
    equipo in equipos}

# Lista de tipos de eventos
tipos_evento = ["Pase", "Robo", "Tiro", "Perdida", "Gol"]

categorias = ["DIAMANTE", "ESMERALDA", "ORO", "PLATA", "BRONCE"]
categorias_equipos = {equipo: random.choice(categorias) for equipo in equipos}

camaras_por_categoria = {
    "DIAMANTE": [
        "Sony HDC-4300",
        "Grass Valley LDX 86",
```

```
        "Panasonic AK-UC4000",
        "Ikegami HK-430",
        "Canon XF705",
        "Blackmagic URSA Broadcast",
        "AJA RovoCam",
        "Red Digital Cinema DSMC2 HELIUM 8K S35",
        "ARRI AMIRA",
        "Canon C300 Mark III"
    ],
    "ESMERALDA": [
        "Sony PXW-Z450",
        "Grass Valley LDX 86 Universe",
        "Panasonic AK-UC3000",
        "Ikegami HDK-73",
        "Canon XF605",
        "Blackmagic URSA Mini Pro 12K",
        "AJA Cion",
        "RED KOMODO 6K",
        "ARRI ALEXA LF",
        "Canon EOS C70"
    ],
    "ORO": [
        "Sony PXW-X500",
        "Grass Valley LDX 86 WorldCam",
        "Panasonic AK-HC5000",
        "Ikegami HDK-99",
        "Canon XF405",
        "Blackmagic Pocket Cinema Camera 6K",
        "AJA Mini-Converters",
        "RED RANGER MONSTRO 8K VV",
        "ARRI ALEXA Mini LF",
        "Canon XF605"
    ],
    "PLATA": [
        "Sony HXC-FB80",
        "Grass Valley LDX C82",
        "Panasonic AW-HR140",
        "Ikegami HDK-95C",
        "Canon XA55",
        "Blackmagic Pocket Cinema Camera 4K",
        "AJA Ki Pro Ultra 12G",
        "RED DSMC2 HELIUM 8K S35",
        "ARRI AMIRA UHD",
        "Canon XF305"
    ],
    "BRONCE": [
        "Sony HXR-NX80",
        "Grass Valley LDX 82",
        "Panasonic AG-CX350",
```

```
        "Ikegami HDK-73C",
        "Canon Vixia HF G21",
        "Blackmagic Micro Cinema Camera",
        "AJA Mini-Connect",
        "RED RANGER HELIUM 8K S35",
        "ARRI ALEXA Mini",
        "Canon Vixia HF G21"
    ]
}


precios_camara = {
    "Sony HDC-4300": 20000,
    "Grass Valley LDX 86": 18000,
    "Panasonic AK-UC4000": 22000,
    "Ikegami HK-430": 19000,
    "Canon XF705": 21000,
    "Blackmagic URSA Broadcast": 15000,
    "AJA RovoCam": 8000,
    "Red Digital Cinema DSMC2 HELIUM 8K S35": 25000,
    "ARRI AMIRA": 24000,
    "Canon C300 Mark III": 23000,
    "Sony PXW-Z450": 17000,
    "Grass Valley LDX 86 Universe": 16000,
    "Panasonic AK-UC3000": 18000,
    "Ikegami HDK-73": 15000,
    "Canon XF605": 19000,
    "Blackmagic URSA Mini Pro 12K": 25000,
    "AJA Cion": 14000,
    "RED KOMODO 6K": 22000,
    "ARRI ALEXA LF": 23000,
    "Canon EOS C70": 21000,
    "Sony PXW-X500": 14000,
    "Grass Valley LDX 86 WorldCam": 13000,
    "Panasonic AK-HC5000": 15000,
    "Ikegami HDK-99": 12000,
    "Canon XF405": 16000,
    "Blackmagic Pocket Cinema Camera 6K": 22000,
    "AJA Mini-Converters": 8000,
    "RED RANGER MONSTRO 8K VV": 24000,
    "ARRI ALEXA Mini LF": 25000,
    "Canon XF605": 19000,
    "Sony HXC-FB80": 9000,
    "Grass Valley LDX C82": 8000,
    "Panasonic AW-HR140": 10000,
    "Ikegami HDK-95C": 7500,
    "Canon XA55": 11000,
    "Blackmagic Pocket Cinema Camera 4K": 18000,
    "AJA Ki Pro Ultra 12G": 7000,
```

```
    "RED DSMC2 HELIUM 8K S35": 21000,
    "ARRI AMIRA UHD": 20000,
    "Canon XF305": 17000,
    "Sony HXR-NX80": 3500,
    "Grass Valley LDX 82": 3000,
    "Panasonic AG-CX350": 4000,
    "Ikegami HDK-73C": 3200,
    "Canon Vixia HF G21": 4200,
    "Blackmagic Micro Cinema Camera": 7000,
    "AJA Mini-Connect": 2500,
    "RED RANGER HELIUM 8K S35": 9000,
    "ARRI ALEXA Mini": 8500,
    "Canon Vixia HF G21": 4200
}

abreviaturas_camara = {
    "Sony HDC-4300": "S4300",
    "Grass Valley LDX 86": "LDX86",
    "Panasonic AK-UC4000": "PUC4",
    "Ikegami HK-430": "IH430",
    "Canon XF705": "F705",
    "Blackmagic URSA Broadcast": "URSA",
    "AJA RovoCam": "voC",
    "Red Digital Cinema DSMC2 HELIUM 8K S35": "SMC2",
    "ARRI AMIRA": "MIRA",
    "Canon C300 Mark III": "00M3",
    "Sony PXW-Z450": "XWZ4",
    "Grass Valley LDX 86 Universe": "X86U",
    "Panasonic AK-UC3000": "UC30",
    "Ikegami HDK-73": "IH73",
    "Canon XF605": "F605",
    "Blackmagic URSA Mini Pro 12K": "P12K",
    "AJA Cion": "ACION",
    "RED KOMODO 6K": "MODO",
    "ARRI ALEXA LF": "XALF",
    "Canon EOS C70": "SC70",
    "Sony PXW-X500": "XWX5",
    "Grass Valley LDX 86 WorldCam": "86WC",
    "Panasonic AK-HC5000": "HC50",
    "Ikegami HDK-99": "IH99",
    "Canon XF405": "F405",
    "Blackmagic Pocket Cinema Camera 6K": "PC6K",
    "AJA Mini-Converters": "AMC",
    "RED RANGER MONSTRO 8K VV": "RNRO",
    "ARRI ALEXA Mini LF": "AMLF",
    "Canon XF605": "F605",
    "Sony HXC-FB80": "CFB8",
    "Grass Valley LDX C82": "XC82",
    "Panasonic AW-HR140": "HR140",
```

```
    "Ikegami HDK-95C": "IH95C",
    "Canon XA55": "CXA55",
    "Blackmagic Pocket Cinema Camera 4K": "BPC4K",
    "AJA Ki Pro Ultra 12G": "AKTRA",
    "RED DSMC2 HELIUM 8K S35": "RMC2",
    "ARRI AMIRA UHD": "AUHD",
    "Canon XF305": "CX305",
    "Sony HXR-NX80": "SX80",
    "Grass Valley LDX 82": "GX82",
    "Panasonic AG-CX350": "PX50",
    "Ikegami HDK-73C": "IH73C",
    "Canon Vixia HF G21": "CF21",
    "Blackmagic Micro Cinema Camera": "BMCC",
    "AJA Mini-Connect": "AMC",
    "RED RANGER HELIUM 8K S35": "RL8K",
    "ARRI ALEXA Mini": "AANI",
    "Canon Vixia HF G21": "CV21"
}


def generar_camaras_por_equipo():
    camaras_equipo = {}
    for equipo, categoria in categorias_equipos.items():
        camaras_disponibles = camaras_por_categoria[categoria]
        num_camaras = random.randint(1, len(camaras_disponibles))
        camaras_seleccionadas = random.sample(camaras_disponibles, num_camaras)
        camaras_equipo[equipo] = camaras_seleccionadas
    return camaras_equipo


# Diccionario para asociar cada equipo con su estadio
estadios_por_equipo = {
    "Real Murcia CF": "Estadio Nueva Condomina",
    "FC Cartagena": "Estadio Cartagonova",
    "UCAM Murcia CF": "Estadio La Condomina",
    "Lorca FC": "Estadio Francisco Artes Carrasco",
    "Yeclano Deportivo": "Estadio Municipal de Yecla",
    "CF Lorca Deportiva": "Estadio Mundial 82",
    "Mar Menor FC": "Estadio Pitin",
    "CD Minera": "Estadio Polideportivo Jose Antonio Perez",
    "CD Algar": "Estadio Polideportivo Municipal El Acueducto",
    "UD Los Garres": "Estadio El Pitin",
    "Muleno CF": "Estadio Municipal de Mula",
    "Ciudad de Cieza": "Estadio Municipal La Arboleja",
    "CD Bullense": "Estadio El Mayayo",
    "Caravaca CF": "Estadio Municipal El Morao",
    "Bala Azul CF": "Estadio Municipal de Bala Azul",
    "Aguilas FC": "Estadio El Rubial",
    "CF La Union": "Estadio Sanchez Luengo",
```

```
    "Racing Murcia FC": "Estadio de La Nueva Condomina",
    "EDF Logrono": "Estadio Las Gaunas",
    "EF Alhama": "Estadio Municipal Jose Kubala",
    "Moratalla CF": "Estadio Municipal Santiago El Mayor",
    "CD Plus Ultra": "Estadio Municipal San Javier",
    "FC Pinatar Arena": "Estadio Municipal de San Pedro del Pinatar",
    "CD Almazora": "Estadio El Palmar",
    "CD Lumbreras": "Estadio Lumbreras",
    "CD Jumilla": "Estadio Municipal de Jumilla",
    "Olimpico de Totana": "Estadio Municipal de Totana",
    "CF Beniel": "Estadio Municipal de Beniel",
    "EF Torre Pacheco": "Estadio El Limonar",
    "CD Alquerias": "Estadio Municipal de Alquerias"
}

coordenadas_por_equipo = {'Real Murcia CF': [-1.14477,38.04151],
 'FC Cartagena': [-0.99605, 37.6096],
 'UCAM Murcia CF': [-1.12135, 37.98581],
 'Lorca FC': [-1.73474, 37.63926 ],
 'Yeclano Deportivo': [-1.10677, 38.60847],
 'CF Lorca Deportiva': [-1.73478, 37.63929],
 'Mar Menor FC': [-0.83084, 37.80925],
 'CD Minera': [-0.79171, 37.81788],
 'CD Algar': [-0.88532, 37.62209],
 'UD Los Garres': [-0.84, 37.812],
 'Muleno CF': [-1.45, 38.02],
 'Ciudad de Cieza': [-1.41, 38.24],
 'CD Bullense': [-1.67, 38.04],
 'Caravaca CF': [-1.86, 38.10],
 'Bala Azul CF': [-1.26, 37.56],
 'Aguilas FC': [-1.56, 37.43],
 'CF La Union': [-0.88, 37.60],
 'Racing Murcia FC': [-1.13, 37.99],
 'EDF Logrono': [-2.45, 42.45],
 'EF Alhama': [-1.41, 37.85],
 'Moratalla CF': [-1.89, 38.18],
 'CD Plus Ultra': [-1.06, 38.00],
 'FC Pinatar Arena': [-0.77, 37.86],
 'CD Almazora': [-0.05, 39.94],
 'CD Lumbreras': [-1.83, 37.5],
 'CD Jumilla': [-1.32, 38.47],
 'Olimpico de Totana': [-1.50, 37.77],
 'CF Beniel': [-1.0, 38.04],
 'EF Torre Pacheco': [-0.96, 37.78],
 'CD Alquerias': [-1.03, 38.01]}


def generar_fechas_unicas(num_fechas):
```

```
    fechas = [fake.date_between_dates(date(2020, 1, 1), date(2023, 12, 31)) for
        _ in range(num_fechas)]
    return fechas

def generar_horas_inicio_unicas(num_fechas):
    horasInicio = [fake.time(pattern="%H:%M:%S") for _ in range(num_fechas)]
    return horasInicio


def generar_correccion(categoria_equipo):
    if categoria_equipo == "DIAMANTE":
        return random.choice([False, False, False, False, True])
    elif categoria_equipo == "ESMERALDA":
        return random.choice([False, False, True])
    elif categoria_equipo == "ORO":
        return random.choice([False, True])
    elif categoria_equipo == "PLATA":
        return random.choice([False, False, True, True, True])
    elif categoria_equipo == "BRONCE":
        return random.choice([False, True, True])
    else:
        return random.choice([False, True])


# Generar datos sinteticos
def generar_datos_sinteticos(num_eventos):
    datos_sinteticos = []
    camaras_por_equipo = generar_camaras_por_equipo()


    for _ in range(num_eventos):
        partido_id = fake.random_int(min=0, max=999)
        equipo_local = equipos[partido_id % len(equipos)]
        categoria_equipo = categorias_equipos[equipo_local]
        jugador = random.choice(jugadores_por_equipo[equipo_local])
        tipo_evento = random.choice(tipos_evento)
        fecha_partido = fechas[partido_id]
        hora_evento = (datetime.strptime(horasInicio[partido_id], "%H:%M:%S") +
            timedelta(minutes=random.randint(1, 120))).strftime("%H:%M:%S")
        hora_recoleccion = (datetime.strptime(hora_evento, "%H:%M:%S") +
            timedelta(seconds=random.randint(1, 120))).strftime("%H:%M:%S")
        corregido = generar_correccion(categoria_equipo)
        hora_revision = (datetime.strptime(hora_recoleccion, "%H:%M:%S") +
            timedelta(minutes=random.randint(10, 120))).strftime("%H:%M:%S")
        camara = camaras_por_equipo[equipo_local][random.randint(0, len(
            camaras_por_equipo[equipo_local])-1 )]
        camara_abr = abreviaturas_camara[camara]
        precio = precios_camara[camara]
        estadio = estadios_por_equipo[equipo_local]
```

```
        coordenada_x = coordenadas_por_equipo[equipo_local][0]
        coordenada_y = coordenadas_por_equipo[equipo_local][1]


        datos_sinteticos.append([
            partido_id,
            equipo_local,
            categoria_equipo,
            jugador,
            tipo_evento,
            fecha_partido,
            hora_evento,
            hora_recoleccion,
            corregido,
            hora_revision,
            camara,
            camara_abr,
            precio,
            estadio,
            coordenada_x,
            coordenada_y
        ])

    return datos_sinteticos

# Guardar datos en un archivo CSV
def guardar_datos_csv(datos, nombre_archivo):
    with open(nombre_archivo, 'w', newline='') as archivo_csv:
        escritor_csv = csv.writer(archivo_csv)
        escritor_csv.writerow(["Partido", "Equipo Local","Categoria Equipo", "
            Jugador", "Tipo de Evento", "Fecha del Partido", "Hora del Evento", "
            Hora de Recoleccion", "Corregido", "Hora de Revision", "Camara","
            Camara abreviatura", "Precio", "Estadio", "Coordenada X", "Coordenada
             Y"])
        escritor_csv.writerows(datos)

num_partidos = 1000
num_datos = 100000

fechas = generar_fechas_unicas(num_partidos)
horasInicio = generar_horas_inicio_unicas(num_partidos)
datos_sinteticos = generar_datos_sinteticos(num_datos)

nombre_archivo = "datos_futbol.csv"
guardar_datos_csv(datos_sinteticos, nombre_archivo)
print(f"Los datos han sido guardados en el archivo {nombre_archivo}.")
```