Predicciones de partidos de La Liga

Hecho por:

- Juan Pablo Blanco Lemus
- Diego Mercado Coello
- Carlos Arturo Torres Sánchez

LaLiga Santander

Tecnologías y herramientas

- Fbref
- Git
- Sklearn
- Mlflow
- Dagshub
- Prefect
- Fast api
- StreamLit
- Docker

Introducción

- Crear un modelo que intente predecir quién ganará, empatará o perderá en un partido.
- Muchos factores influyen: rendimiento de los equipos, localidad, recuperaciones, etc.
- Analizar los datos y encontrar patrones.

Objetivos

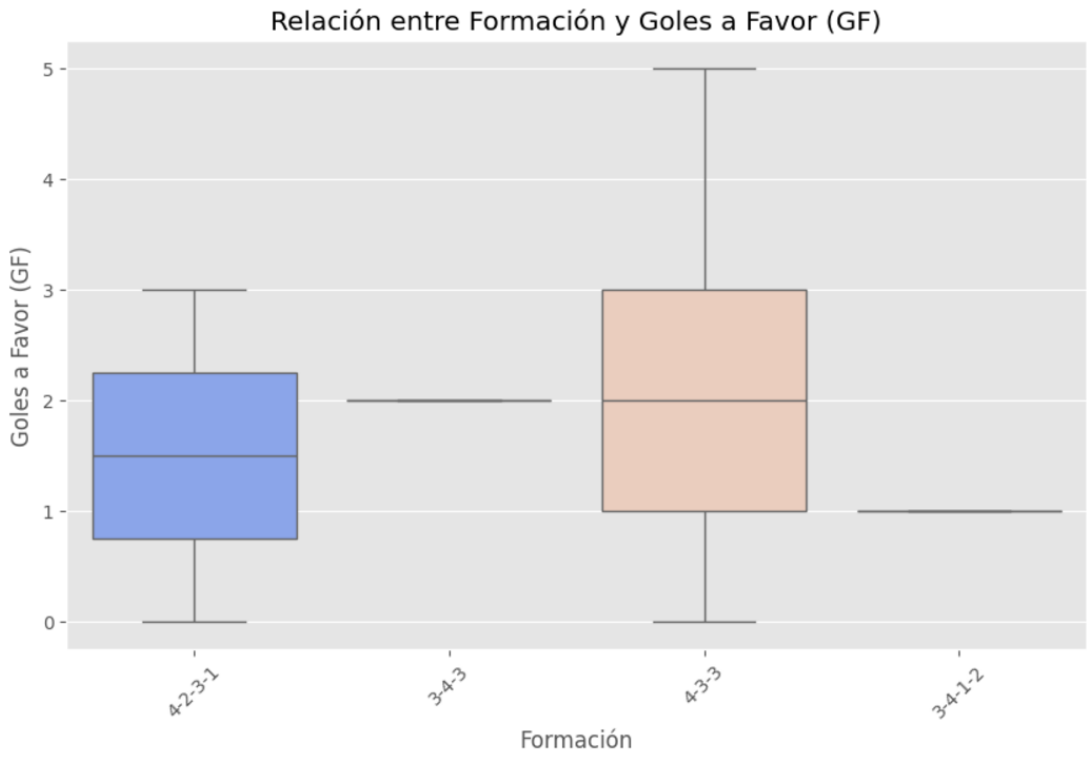
- Desarrollar un flujo de trabajo complete en el que el resultado final sea la UI en un ambiente y formato reproducible
- Aplicar los conocimientos adquiridos durante la materia.
- Poder utilizer el product o final para aplicarlo en las apuestas en el mundo real

Extracción de datos

- Página web FBref
- Web scraping
- Datos actualizados y detallados sobre equipos y jugadores en formato de tabla.

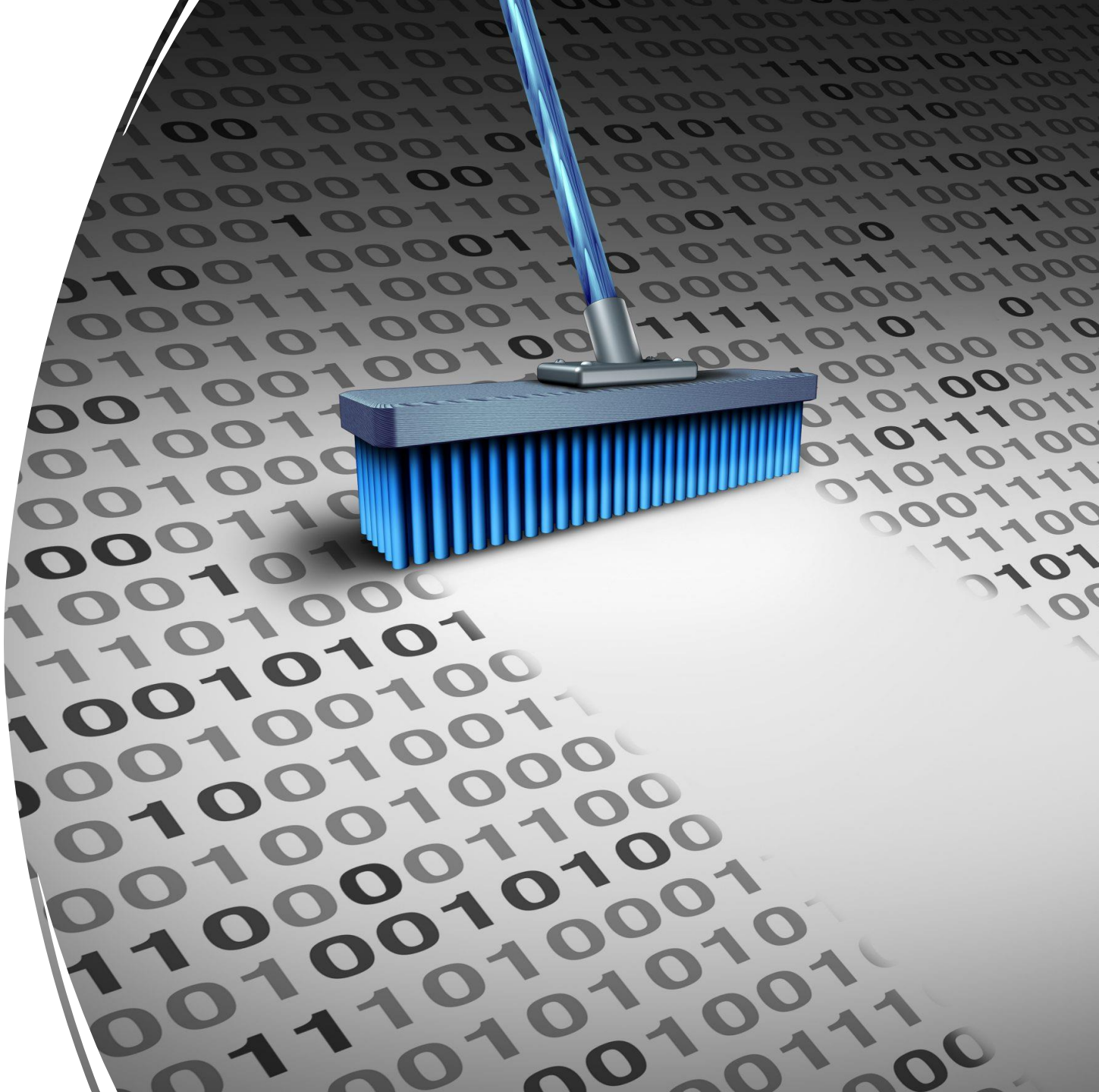
EDA

- Exploramos el dataset
- Identificamos patrones
- Ayudo a seleccionar características importantes para el modelo
- Detección de errores en el dataset



Data wrangling

- Limpieza de datos
- Creación de nuevas columnas
- Encoding de variables categóricas
- Cambio de nombre de variables

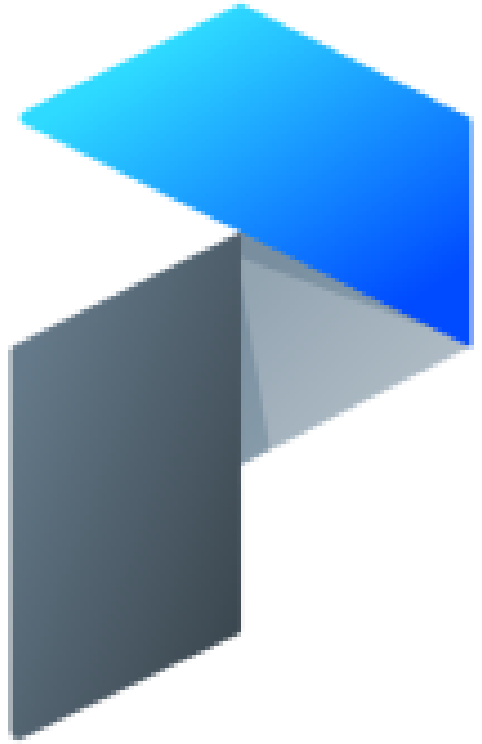


Model registry

- Escalamiento de datos
- Diferentes modelos con Gridsearch
- Mlflow (runs, experimentos y modelos)
- Dagshub

Training pipeline

- Prefect
- Orquestracion del flujo de trabajo
- Tasks y flows
- Monitores de cada task
- Automatización



```python
@flow(name="Pipeline de Entrenamiento y Registro de Modelos")  1 usage  arturotowers
def pipeline_entrenamiento(jornada_actual: int):
    file_path ="LaLiga Dataset 2023-2024.xlsx"
    jornada_actual = 12
    # Inicializamos MLflow y DagsHub
    dagshub.init(url="https://dagshub.com/arturotowers/Proyecto_LaLiga", mlflow=True)
    MLFLOW_TRACKING_URI = mlflow.get_tracking_uri()
    mlflow.set_tracking_uri(MLFLOW_TRACKING_URI)
    mlflow.set_experiment(experiment_name="arturo-prefect-experiment")

    # Ejecutar las tareas de flujo
    print("Ejecutando tarea: Actualizar dataset")
    df = actualizar_dataset(file_path,jornada_actual)

    print("Ejecutando tarea: Preparar datos para prediccion")
    df_prediccion = preparar_datos_prediccion(jornada_actual)

    # Cargamos y procesamos el dataset
    print("Ejecutando tarea: Cargando y procesando el dataset")
    X_train, X_test, y_train, y_test = cargar_procesar_dataset(df)

    print("Ejecutando tarea: hyper-parameter tuning")
    best_params_xgb, best_params_rf = hyper_parameter_tunning(X_train, X_test, y_train, y_test)

    print("Ejecutando tarea: train best models")
    train_best_model(X_train, X_test, y_train, y_test, best_params_xgb, best_params_rf)

    print("Ejecutando tarea: register best model")
    register_best_model()

    print("Flujo completado con éxito.")
```

Modelo champion y challenger

- XGBoost
- SoftProb
- 84 % de Accuracy

Despliegue del modelo

- Un cliente envia un JSON con el numero de la jornada
- Se valilda que la jornada sea un entero
- Llama a la función predict para generar las predicciones
- Se devuelve la predicción en un formato JSON

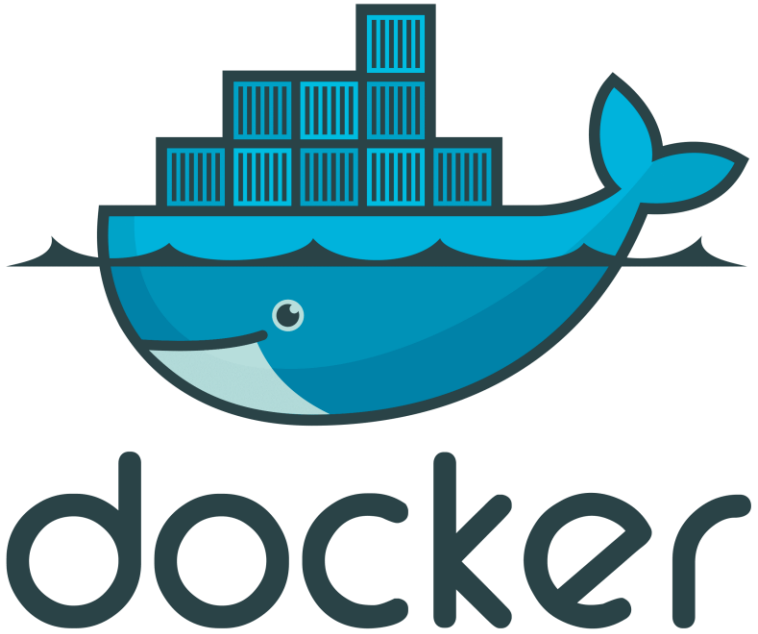
Despliegue de la UI

- Se define una interfaz con Streamlit con un título y una barra lateral para la entrada de la jornada
- El usuario presiona el botón predict para enviar la solicitud al backend
- Se realiza una solicitud POST a la API del modelo que está dentro de un contenedor de Docker
- Si la predicción es exitosa, se presentan los resultados en una tabla.
- Si ocurre un error se muestra un mensaje que dice “Error en la predicción”

Streamlit

Contenerización y Despliegue con Docker

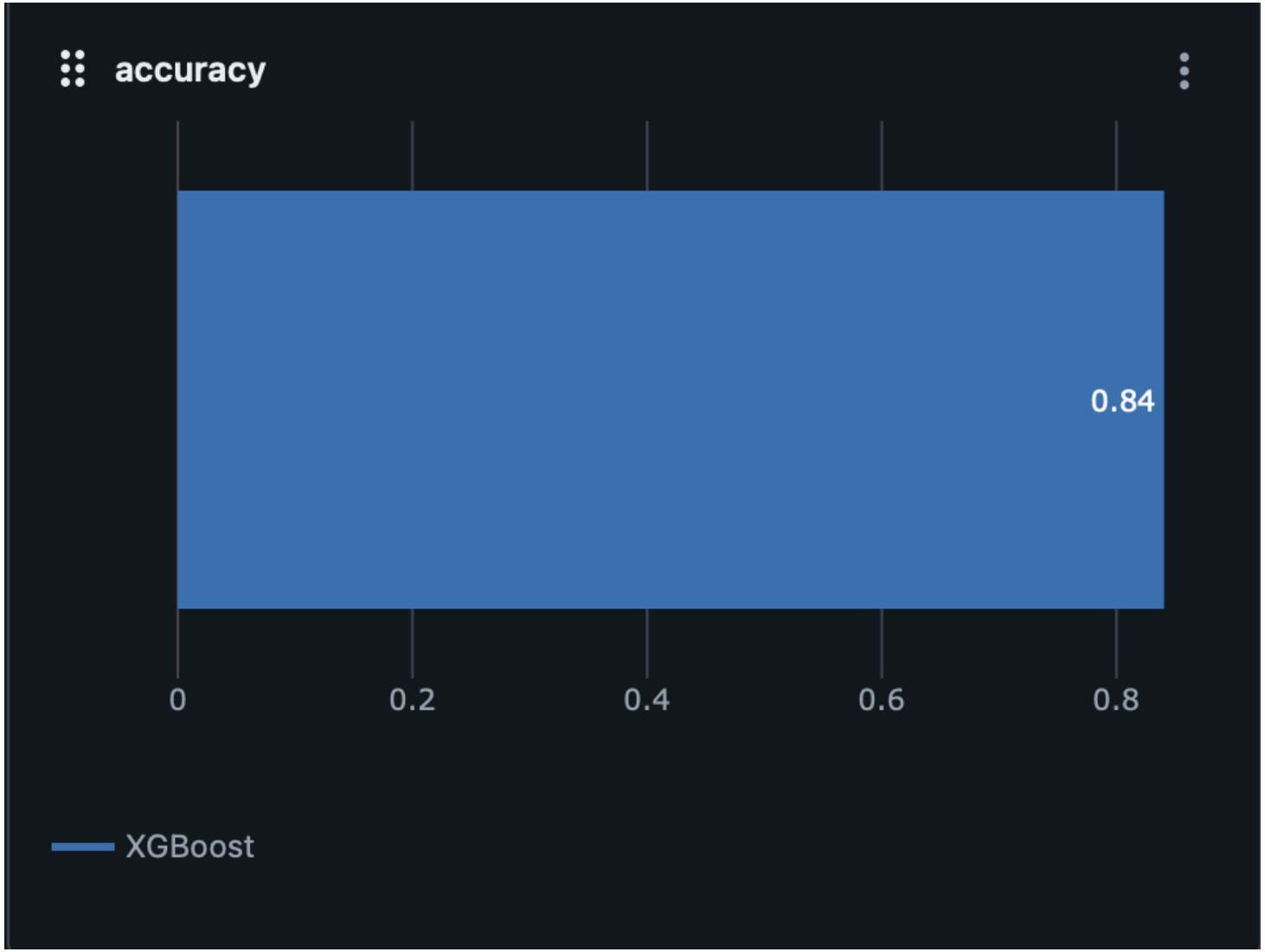
- **Dividimos la aplicación en dos servicios**: un backend (FastAPI) y un frontend (Streamlit).
- **Creamos imágenes Docker** para cada servicio usando sus respectivos Dockerfiles.
- **Orquestamos los contenedores** con Docker Compose, permitiendo que trabajen juntos y se comuniquen en una red compartida.



Resultados

- El modelo proporciona la probabilidad de victoria local, empate o victoria visitante.

- Predició correctamente el resultado en el 84% de los casos.



	perder	empatar	ganar	Anfitrion	Rival
0	0.1	0.4254	0.4746	Getafe	Valladolid
1	0.3348	0.5795	0.0858	Valencia	Betis
2	0.0072	0.0611	0.9317	Atlético Madrid	Alavés
3	0.5772	0.179	0.2439	Las Palmas	Mallorca
4	0.0279	0.0629	0.9091	Girona	Espanyol
5	0.9334	0.0442	0.0224	Celta Vigo	Barcelona
6	0.2123	0.5903	0.1975	Osasuna	Villarreal
7	0.204	0.3253	0.4708	Sevilla	Rayo Vallecano
8	0.6679	0.2974	0.0347	Leganés	Real Madrid
9	0.046	0.212	0.742	Athletic Club	Real Sociedad
10	0.4134	0.5136	0.073	Valladolid	Getafe
11	0.1639	0.5464	0.2897	Betis	Valencia
12	0.9028	0.0862	0.0111	Alavés	Atlético Madrid
13	0.4028	0.1244	0.4728	Mallorca	Las Palmas
14	0.9334	0.048	0.0186	Espanyol	Girona
15	0.0517	0.1453	0.803	Barcelona	Celta Vigo
16	0.1766	0.429	0.3944	Villarreal	Osasuna
17	0.2339	0.6401	0.126	Rayo Vallecano	Sevilla
18	0.0429	0.5809	0.3763	Real Madrid	Leganés
19	0.6054	0.3539	0.0407	Real Sociedad	Athletic Club

Modelo vs Realidad

Dificultades y Aprendizajes

- Tuvimos dificultad para automatizar el pipeline al desglozar todas las tasks.
- Hubo problemas en las dependencias y lo solucionamos ingresando a los artefactos de mlflow.
- Importando el modelo desde Mlflow

Mejoras a Futuro

- Ampliar las predicciones para más ligas.
- Agregar más datos para ver si hay más variables que afecten como el clima, lesiones y otros factores externos.
- Despliegue en la nube utilizando servicios como AWS

Gracias por su atención