# XGBoost (Extreme Gradient Boosting)

In [3]:
```
import numpy as np
import pandas as pd
from IPython.display import display, HTML
import importlib

import plotly.express as px
import matplotlib.pyplot as plt
import statsmodels.api as sm

from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolu
import xgboost as xgb


import warnings
import os
import pickle

warnings.filterwarnings("ignore")
pd.options.display.float_format = '{:,.2f}'.format
```

In [4]:
```
df_main = pd.read_excel("https://raw.githubusercontent.com/carrenogf/MCD-Series-
df_main = df_main.sort_values("FECHA",ascending=True)
df_main.set_index("FECHA", inplace=True)
df_copa = df_main["CHU_COPA_AJUST"].dropna()
df_recprop = df_main["CHU_REC_PROPIOS_AJUST"].dropna()
df_regal = df_main["CHU_REGALIAS_AJUST"].dropna()
dataframes = [df_copa, df_recprop, df_regal]
titulos = ["CHU_COPA_AJUST", "CHU_REC_PROPIOS_AJUST", "CHU_REGALIAS_AJUST"]
```

In [5]:
```
def extract_time_features(index):
    return pd.Series({
        'dayofweek': index.dayofweek,
        'quarter': index.quarter,
        'month': index.month,
        'year': index.year,
        'dayofyear': index.dayofyear,
        'dayofmonth': index.day,
        'weekofyear': index.isocalendar().week
    })


def add_lags(df, titulo):
    target_map = df[titulo].to_dict()
    df['lag1'] = (df.index - pd.Timedelta('364 days')).map(target_map) # df_1['P
    df['lag2'] = (df.index - pd.Timedelta('728 days')).map(target_map) # df_1['P
    df['lag3'] = (df.index - pd.Timedelta('1092 days')).map(target_map) # df_1['
    return df
```

In [6]:
```
time_features = dataframes[0].index.to_series().apply(extract_time_features)
dataframes[0] = pd.concat([dataframes[0], time_features], axis=1)
dataframes[0] = add_lags(dataframes[0], titulos[0])

time_features = dataframes[1].index.to_series().apply(extract_time_features)
```

```
dataframes[1] = pd.concat([dataframes[1], time_features], axis=1)
dataframes[1] = add_lags(dataframes[1], titulos[1])

time_features = dataframes[2].index.to_series().apply(extract_time_features)
dataframes[2] = pd.concat([dataframes[2], time_features], axis=1)
dataframes[2] = add_lags(dataframes[2], titulos[2])
```

In [5]:
```
dataframes[2].head()
```

Out[5]:

| FECHA | CHU_REGALIAS_AJUST | dayofweek | quarter | month | year | dayofyear | dayofmo |
|---|---|---|---|---|---|---|---|
| **2016-04-08** | 212,159.00 | 4 | 2 | 4 | 2016 | 99 | |
| **2016-04-14** | 26,246.00 | 3 | 2 | 4 | 2016 | 105 | |
| **2016-04-15** | 16,002,725.00 | 4 | 2 | 4 | 2016 | 106 | |
| **2016-04-20** | 5,582.00 | 2 | 2 | 4 | 2016 | 111 | |
| **2016-04-29** | 11,066,374.00 | 4 | 2 | 4 | 2016 | 120 | |

In [7]:
```
# TRAIN TEST
dataframes_train = []
dataframes_test = []

# for i in range(3):
#     train = dataframes[i].iloc[:round(len(dataframes[0])*.8)]
#     test = dataframes[i].iloc[round(len(dataframes[0])*.8):]
#     dataframes_train.append(train)
#     dataframes_test.append(test)

train_copa = dataframes[0].iloc[:round(len(dataframes[0])*.8)]
test_copa = dataframes[0].iloc[round(len(dataframes[0])*.8):]
print(f"Coparticipacion: train({train_copa.shape}), test({test_copa.shape})")

train_recursos = dataframes[1].iloc[:round(len(dataframes[1])*.8)]
test_recursos = dataframes[1].iloc[round(len(dataframes[1])*.8):]
print(f"Recursos: train({train_recursos.shape}), test({test_recursos.shape})")

train_regalias = dataframes[2].iloc[:round(len(dataframes[2])*.8)]
test_regalias = dataframes[2].iloc[round(len(dataframes[2])*.8):]
print(f"Regalias: train({train_regalias.shape}), test({test_regalias.shape})")

dataframes_train = [ train_copa, train_recursos, train_regalias ]
dataframes_test = [ test_copa, test_recursos, test_regalias ]
```

```
Coparticipacion: train((1275, 11)), test((319, 11))
Recursos: train((1626, 11)), test((406, 11))
Regalias: train((460, 11)), test((115, 11))
```

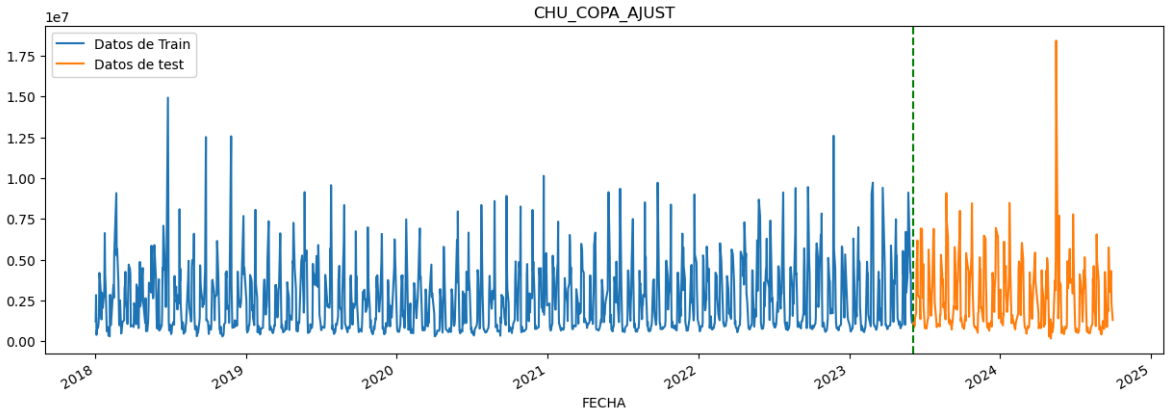
In [8]:
```
train_recursos.head()
```

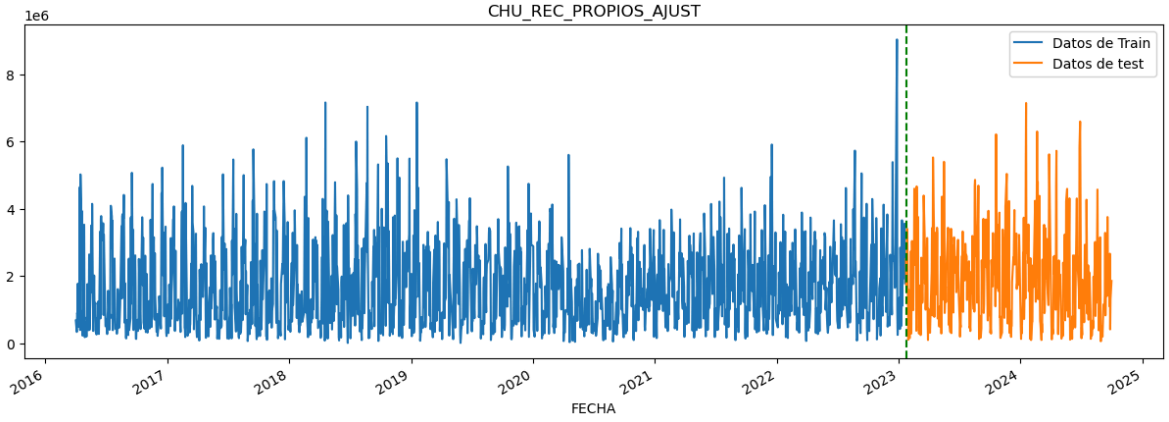
Out[8]:

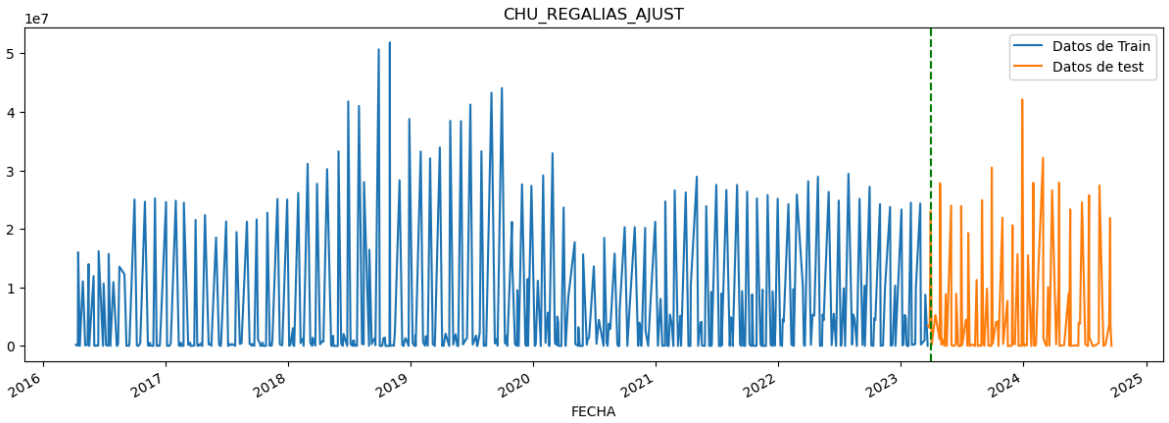
| FECHA | CHU_REC_PROPIOS_AJUST | dayofweek | quarter | month | year | dayofyear | dayof |
|---|---|---|---|---|---|---|---|
| **2016-04-01** | 679233 | 4 | 2 | 4 | 2016 | 92 | |
| **2016-04-04** | 339379 | 0 | 2 | 4 | 2016 | 95 | |
| **2016-04-05** | 903634 | 1 | 2 | 4 | 2016 | 96 | |
| **2016-04-06** | 858197 | 2 | 2 | 4 | 2016 | 97 | |
| **2016-04-07** | 1774956 | 3 | 2 | 4 | 2016 | 98 | |

In [8]:

```
for i in range(len(dataframes)):
    # Separamos los datos en train y test
    fig, ax = plt.subplots(figsize=(15, 5))

    dataframes_train[i][titulos[i]].plot(ax=ax, label='Datos de Train', title=f'
    dataframes_test[i][titulos[i]].plot(ax=ax, label='Datos de test')
    ax.axvline(f'{dataframes_test[i].index[i]}', color='green', ls='--')
    ax.legend(['Datos de Train', 'Datos de test'])
    plt.show()
```






In [9]:
```
FEATURES = ['dayofweek', 'quarter', 'month', 'year', 'dayofyear' , 'dayofmonth',
TARGET = 'CHU_COPA_AJUST'
```

In [10]:
```
X_train = dataframes_train[0][FEATURES]
y_train = dataframes_train[0][TARGET]

X_test = dataframes_test[0][FEATURES]
y_test = dataframes_test[0][TARGET]
```

In [9]:
```
import optuna
import lightgbm as lgb
import numpy as np
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error



def objective(trial, df):
    param = {
        "n_estimators": trial.suggest_int("n_estimators", 100, 500),  # Reducido
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.1, log=Tru
        "max_depth": trial.suggest_int("max_depth", 3, 7),  # Reducido de 3 a 7
        "min_child_weight": trial.suggest_int("min_child_weight", 1, 5),  # Redu
        "subsample": trial.suggest_float("subsample", 0.6, 1.0),  # Reducido de
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.6, 1.0),
        "gamma": trial.suggest_float("gamma", 0, 0.5),  # Reducido de 0 a 0.5
        "reg_alpha": trial.suggest_float("reg_alpha", 0.0, 0.5),  # Regularizaci
        "reg_lambda": trial.suggest_float("reg_lambda", 0.5, 1.5),  # Regulariza
        "random_state": 42,
        "objective": "reg:squarederror",
        "early_stopping_rounds": 50,
    }

    test_size = 100
    tss = TimeSeriesSplit(n_splits=5, test_size=252, gap=1) # 252 son los dias h
    scores = []
    df = df.sort_index()

    for train_idx, val_idx in tss.split(df):
        train = df.iloc[train_idx]
        test = df.iloc[val_idx]

        FEATURES = ['dayofyear', 'dayofweek', 'quarter', 'month', 'year',
                    'lag1', 'lag2', 'lag3']
        TARGET = df.columns[0]
```

```
        # Verificar características y dropna
        # assert all(feature in train.columns for feature in FEATURES), "Faltan
        # assert all(feature in test.columns for feature in FEATURES), "Faltan c

        X_train = train[FEATURES]
        y_train = train[TARGET]

        X_test = test[FEATURES]
        y_test = test[TARGET]

        # if X_train.empty or X_test.empty:
        #     raise ValueError("X_train o X_test está vacío después de dropna")

        model = xgb.XGBRegressor(**param)
        model.fit(
            X_train, y_train,
            eval_set=[(X_test, y_test)]
        )

        y_pred = model.predict(X_test)
        rmse = mean_squared_error(y_test, y_pred, squared=False)
        scores.append(rmse)

    return np.mean(scores)
```

In [ ]:
```
# Ejecutar la optimización con Optuna
# df = dataframes_train[0].copy() ### ACACAACACACACACACACA
df = dataframes[0].copy() # ACA TORTUGA NINJA
study = optuna.create_study(direction="minimize")
study.optimize(lambda trial: objective(trial, df), n_trials=100)


copa_best_params = study.best_params
copa_rmse = study.best_value
# Imprimir los mejores parámetros y el mejor RMSE
print("Mejores parámetros para COPA:", study.best_params)
print("Mejor RMSE promedio para COPA:", study.best_value)
```

RMSE promedio para COPA: 1718582.9603018877

In [ ]:
```
# Ejecutar la optimización con Optuna
#df = dataframes_train[1].copy()
df = dataframes[1].copy()
study = optuna.create_study(direction="minimize")
study.optimize(lambda trial: objective(trial, df), n_trials=100)



rec_propios_best_params = study.best_params
rec_propios_rmse = study.best_value
# Imprimir los mejores parámetros y el mejor RMSE
print("Mejores parámetros para REC_PROPIOS:", study.best_params)
print("Mejor RMSE promedio para REC_PROPIOS:", study.best_value)
```

In [12]:
```
def objective_2(trial, df):

    FEATURES = ['dayofweek', 'quarter', 'month', 'year', 'dayofyear' , 'dayofmon
    TARGET = df.columns[0]
```

```
    X_train = df[FEATURES]
    y_train = df[TARGET]

    X_test = df[FEATURES]
    y_test = df[TARGET]

    param = {
        "n_estimators": trial.suggest_int("n_estimators", 100, 500),  # Reducido
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.1, log=Tru
        "max_depth": trial.suggest_int("max_depth", 3, 7),  # Reducido de 3 a 7
        "min_child_weight": trial.suggest_int("min_child_weight", 1, 5),  # Redu
        "subsample": trial.suggest_float("subsample", 0.6, 1.0),  # Reducido de
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.6, 1.0),
        "gamma": trial.suggest_float("gamma", 0, 0.5),  # Reducido de 0 a 0.5
        "reg_alpha": trial.suggest_float("reg_alpha", 0.0, 0.5),  # Regularizaci
        "reg_lambda": trial.suggest_float("reg_lambda", 0.5, 1.5),  # Regulariza
        "random_state": 42,
        "objective": "reg:squarederror",
        "early_stopping_rounds": 50,
    }

    model = xgb.XGBRegressor(**param)
    model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_test, y_test)])









    # Puedes usar la métrica de tu preferencia aquí
    y_pred = model.predict(X_test)
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    return rmse
```

In [ ]:
```
# Ejecutar la optimización con Optuna
df = dataframes[2].copy()
study = optuna.create_study(direction="minimize")
study.optimize(lambda trial: objective_2(trial, df), n_trials=100)



regalias_best_params = study.best_params
regalias_rmse = study.best_value
# Imprimir los mejores parámetros y el mejor RMSE
print("Mejores parámetros para REGALIAS:", study.best_params)
print("Mejor RMSE promedio para REGALIAS:", study.best_value)
```

In [14]:
```
FEATURES = ['dayofweek', 'quarter', 'month', 'year', 'dayofyear' , 'dayofmonth',
TARGET = ["CHU_COPA_AJUST", "CHU_REC_PROPIOS_AJUST", "CHU_REGALIAS_AJUST"]

X_train_COPA = dataframes_train[0][FEATURES]
y_train_COPA = dataframes_train[0][TARGET[0]]
X_test_COPA = dataframes_test[0][FEATURES]
y_test_COPA = dataframes_test[0][TARGET[0]]

X_train_REC_PROPIOS = dataframes_train[1][FEATURES]
y_train_REC_PROPIOS = dataframes_train[1][TARGET[1]]
```

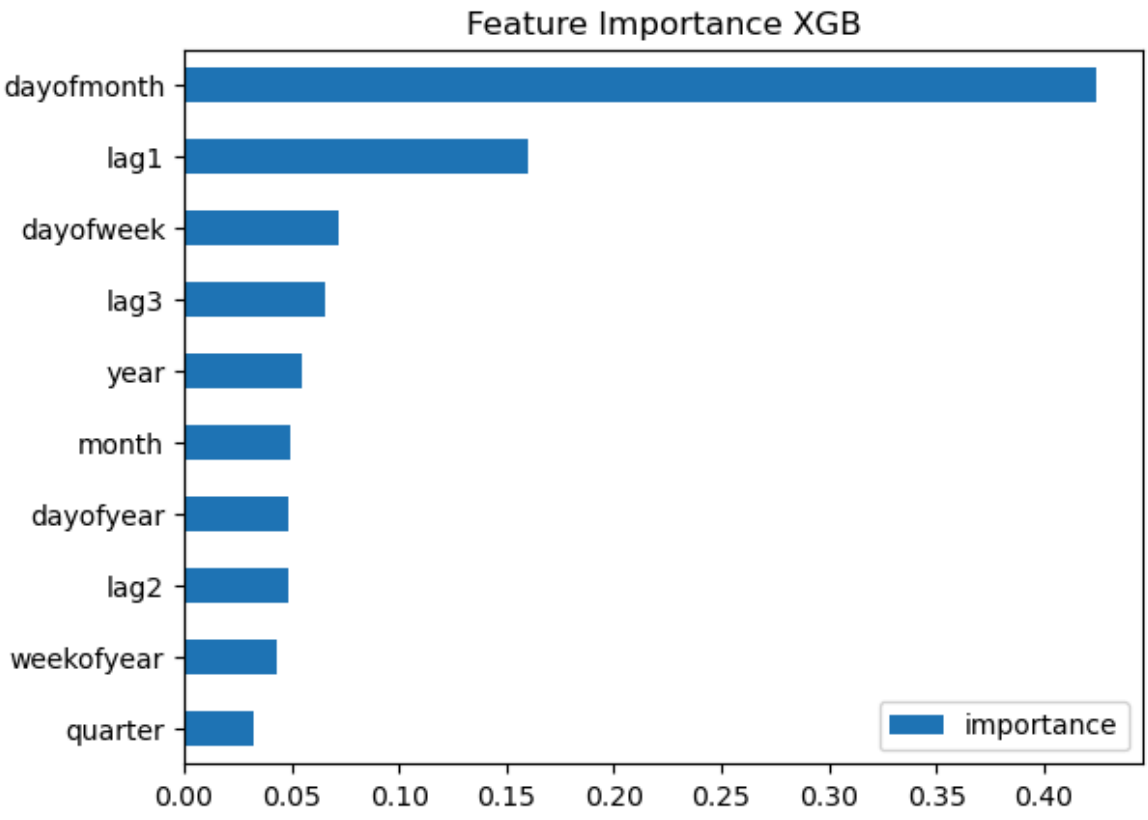
```
X_test_REC_PROPIOS = dataframes_test[1][FEATURES]
y_test_REC_PROPIOS = dataframes_test[1][TARGET[1]]

X_train_REGALIAS = dataframes_train[2][FEATURES]
y_train_REGALIAS = dataframes_train[2][TARGET[2]]
X_test_REGALIAS = dataframes_test[2][FEATURES]
y_test_REGALIAS = dataframes_test[2][TARGET[2]]
```

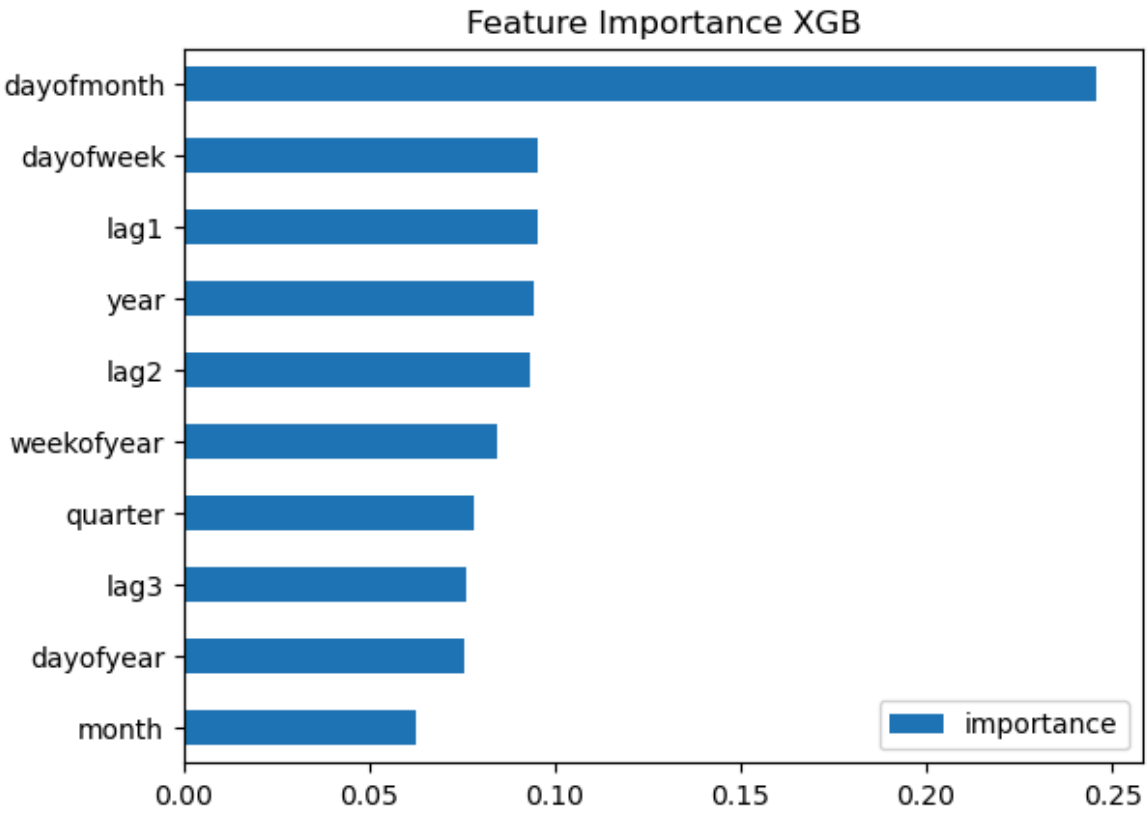
In [ ]:
```
xgb_copa = xgb.XGBRegressor(**copa_best_params )
xgb_copa.fit(X_train_COPA, y_train_COPA,
        eval_set=[(X_train_COPA, y_train_COPA), (X_test_COPA, y_test_COPA)])
```

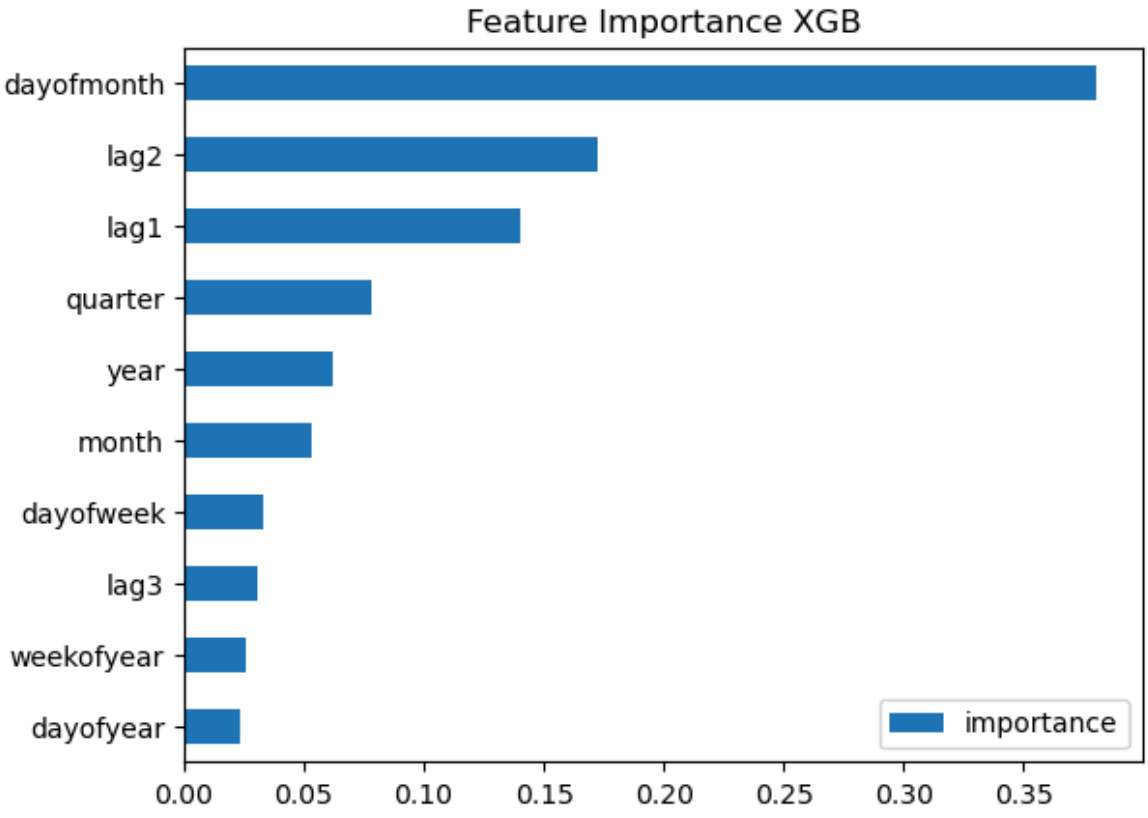
In [ ]:
```
xgb_rec_propios =  xgb.XGBRegressor(**rec_propios_best_params )
xgb_rec_propios.fit(X_train_REC_PROPIOS, y_train_REC_PROPIOS,
        eval_set=[(X_train_REC_PROPIOS, y_train_REC_PROPIOS), (X_test_REC_PROPIO
```

In [ ]:
```
xgb_regalias =  xgb.XGBRegressor(**regalias_best_params )
xgb_regalias.fit(X_train_REGALIAS, y_train_REGALIAS,
        eval_set=[(X_train_REGALIAS, y_train_REGALIAS), (X_test_REGALIAS, y_test
```

In [20]:
```
models = [xgb_copa, xgb_rec_propios, xgb_regalias]
i = 0
for model in models:

    fi = pd.DataFrame(data=model.feature_importances_,
             index=model.feature_names_in_,
             columns=['importance'])
    fi.sort_values('importance').plot(kind='barh', title='Feature Importance XGB
    plt.show()
```



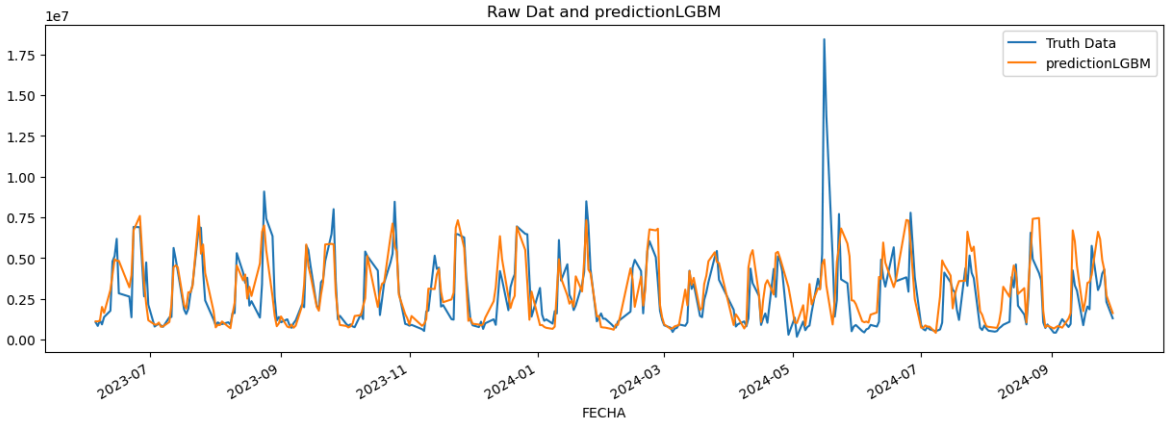


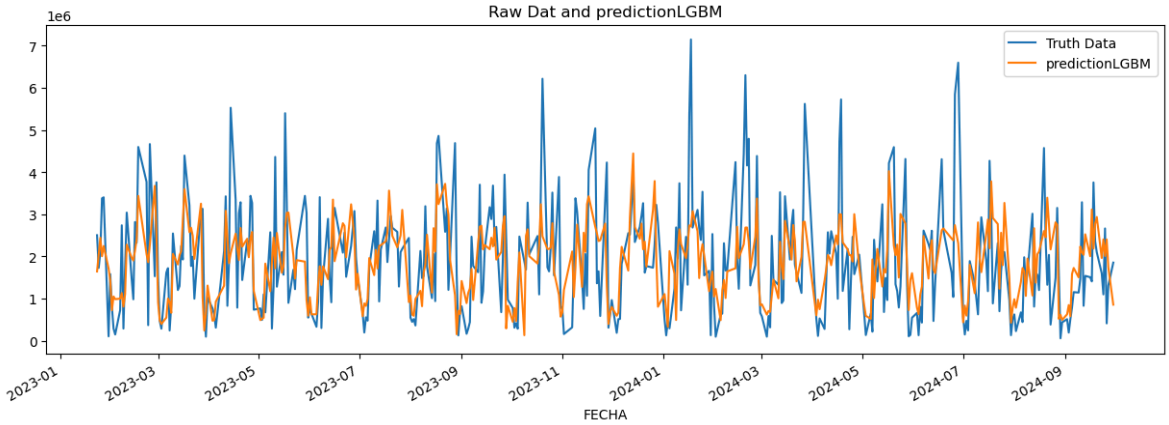



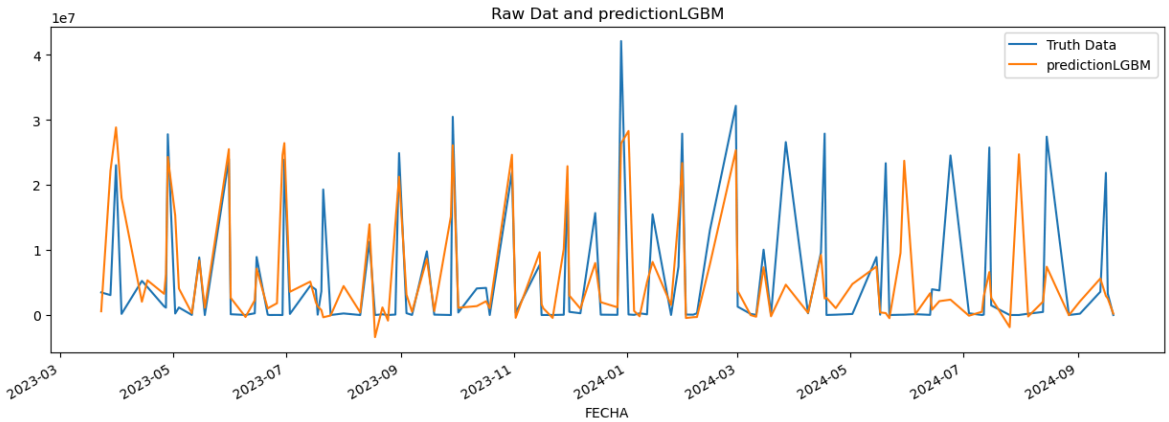

In [21]:
```
def plotear_predicciones(df_aux, model, X_test):
    df = df_aux.copy()
    test = df.copy()
    test['predictionLGBM'] = model.predict(X_test)
    df = df.merge(test[['predictionLGBM']], how='left', left_index=True, right_i
    ax = df[df.columns[0]].plot(figsize=(15, 5))
    df['predictionLGBM'].plot(ax=ax)
    plt.legend(['Truth Data', 'predictionLGBM'])
```

```
    ax.set_title('Raw Dat and predictionLGBM')
    plt.show()
```

In [22]:
```
models = [xgb_copa, xgb_rec_propios, xgb_regalias]
X_test = [X_test_COPA, X_test_REC_PROPIOS, X_test_REGALIAS]

for i in range(len(dataframes)):
    plotear_predicciones(dataframes_test[i], models[i], X_test[i])
```




In [ ]:
```
import seaborn as sns

def plot_train_test_predictions(dataframes_train, dataframes_test, predictions_t

    num_series = len(dataframes_train)
    fig, axes = plt.subplots(1, num_series, figsize=(20, 6), sharey=True)
    sns.set(style="whitegrid")

    for i in range(num_series):
        ax = axes[i]
```

```
        # Filtrar datos desde la fecha indicada (si se especifica)
        train = dataframes_train[i][start_date:] if start_date else dataframes_t
        test = dataframes_test[i][start_date:] if start_date else dataframes_tes
        pred = predictions_test[i][start_date:] if start_date else predictions_t
        # Graficar series
        #sns.lineplot(data=train, label='Train', ax=ax, color='#3477eb')
        #sns.lineplot(data=test, label='Test', ax=ax, color='green')
        #sns.lineplot(data=pred, ax=ax, color='#f72525', linestyle='--')
        ax.plot(train, label='Train', color='#3477eb')
        ax.plot(test, label='Test', color='green')
        ax.plot(pred, label='Predicciones', color='red', linestyle='--')

        # Configuración del gráfico
        ax.set_title(series_names[i], fontsize=14)
        ax.set_xlabel('Fecha')
        ax.set_ylabel('Valor')
        ax.legend(loc='best')
        ax.grid(True)

    plt.tight_layout()
    plt.show()
```

In [40]:

```
import matplotlib.pyplot as plt
import seaborn as sns

def plot_train_test_predictions(dataframes_train, dataframes_test, predictions_t
    """
    Grafica las series de entrenamiento, prueba y predicciones.

    Parámetros:
    - dataframes_train: Lista de dataframes con los datos de entrenamiento.
    - dataframes_test: Lista de dataframes con los datos de prueba.
    - predictions_test: Lista de arreglos con las predicciones.
    - series_names: Lista de nombres para las series (uno por gráfico).
    - target_column: Nombre de la columna objetivo a graficar.
    - start_date: Fecha de inicio para filtrar las series (opcional).
    """
    sns.set(style="whitegrid")
    num_series = len(dataframes_train)
    fig, axes = plt.subplots(1, num_series, figsize=(20, 6), sharey=True)

    for i in range(num_series):
        ax = axes[i]

        # Filtrar datos desde la fecha indicada
        train = dataframes_train[i][titulos[i]][start_date:] if start_date else
        test = dataframes_test[i][titulos[i]][start_date:] if start_date else da
        pred = predictions_test[i]  # Asumimos que ya tiene las predicciones ali

        # Graficar series
        ax.plot(train.index, train.values, label='Train', color='#3477eb')
        ax.plot(test.index, test.values, label='Test', color='green')
        ax.plot(test.index, pred, label='Predicciones', color='red', linestyle='

        # Configuración del gráfico
        ax.set_title(series_names[i], fontsize=14)
        ax.set_xlabel('Fecha')
        ax.set_ylabel('Valor')
        ax.legend(loc='best')
        ax.grid(True)
```

```
    plt.tight_layout()
    plt.show()
```

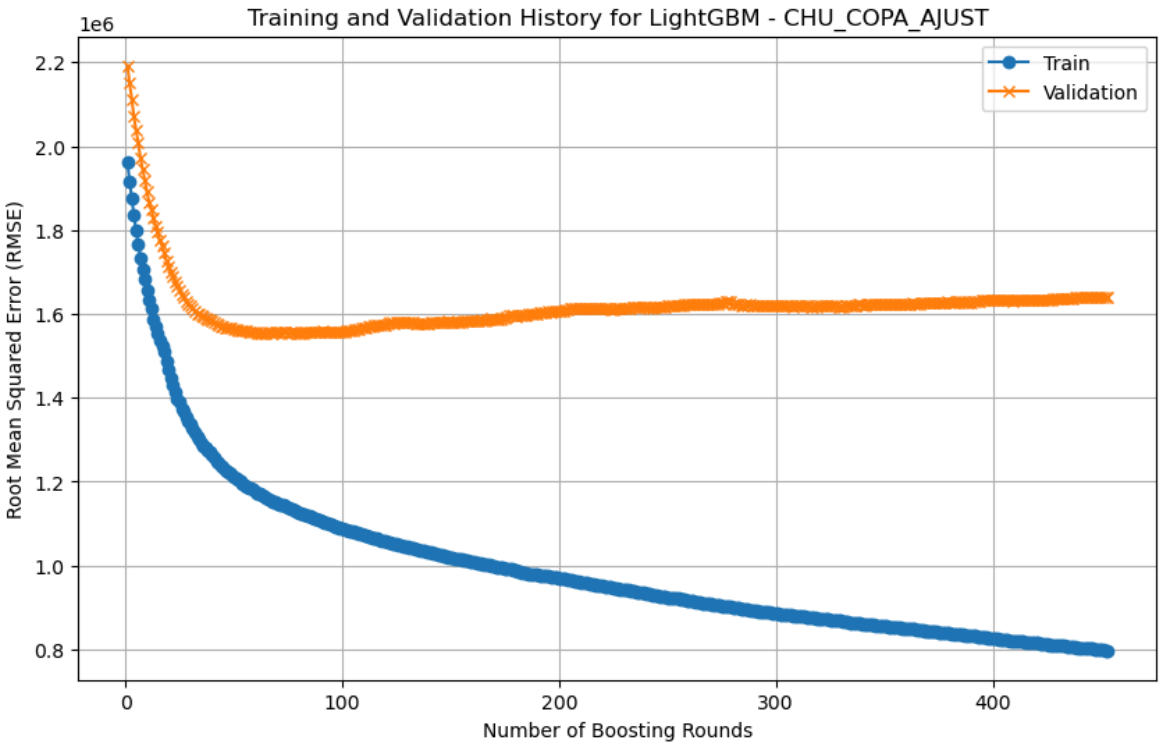
In [42]:
```
# models = [lgb_copa, lgb_rec_propios, lgb_regalias]
# X_test = [X_test_COPA, X_test_REC_PROPIOS, X_test_REGALIAS]
# X_train = [X_train_COPA, X_train_REC_PROPIOS, X_train_REGALIAS]
# y_pred_copa = lgb_copa.predict(X_test_COPA)
# y_pred_rec_propios = lgb_rec_propios.predict(X_test_REC_PROPIOS)
# y_pred_regalias = lgb_regalias.predict(X_test_REGALIAS)
# y_pred = [y_pred_copa, y_pred_rec_propios, y_pred_regalias]


# plot_train_test_predictions(
#     dataframes_train,       # Lista de dataframes de entrenamiento
#     dataframes_test,        # Lista de dataframes de prueba
#     y_pred,         # Lista de arreglos con las predicciones
#     titulos,        # Lista de nombres para los gráficos
#     target_column='hola',  # Nombre de la columna objetivo
#     start_date='2023-10-01'          # Fecha de inicio opcional
# )
```

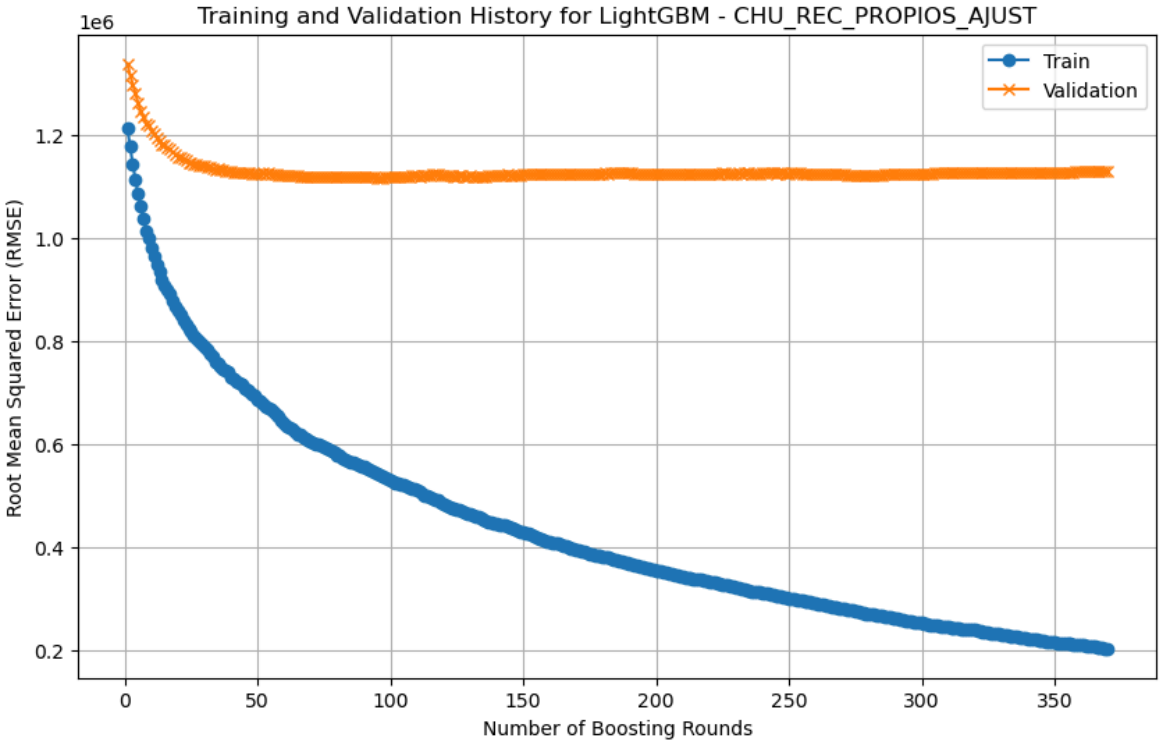
In [23]:
```
from sklearn.metrics import mean_absolute_error as mae

models = [xgb_copa, xgb_rec_propios, xgb_regalias]
X_test = [X_test_COPA, X_test_REC_PROPIOS, X_test_REGALIAS]
y_test = [y_test_COPA, y_test_REC_PROPIOS, y_test_REGALIAS]

for i in range(len(models)):
    rmse_score = np.sqrt(mean_squared_error(y_test[i], model.predict(X_test[i])))
    mae_score = mae(y_test[i], model.predict(X_test[i]))
    print(f'RMSE en conjunto de Test Modelo XGB: {rmse_score:0.2f}')
    print(f'MAE en conjunto de Test Modelo XGB: {mae_score:0.2f}')
    print('--------------')
```

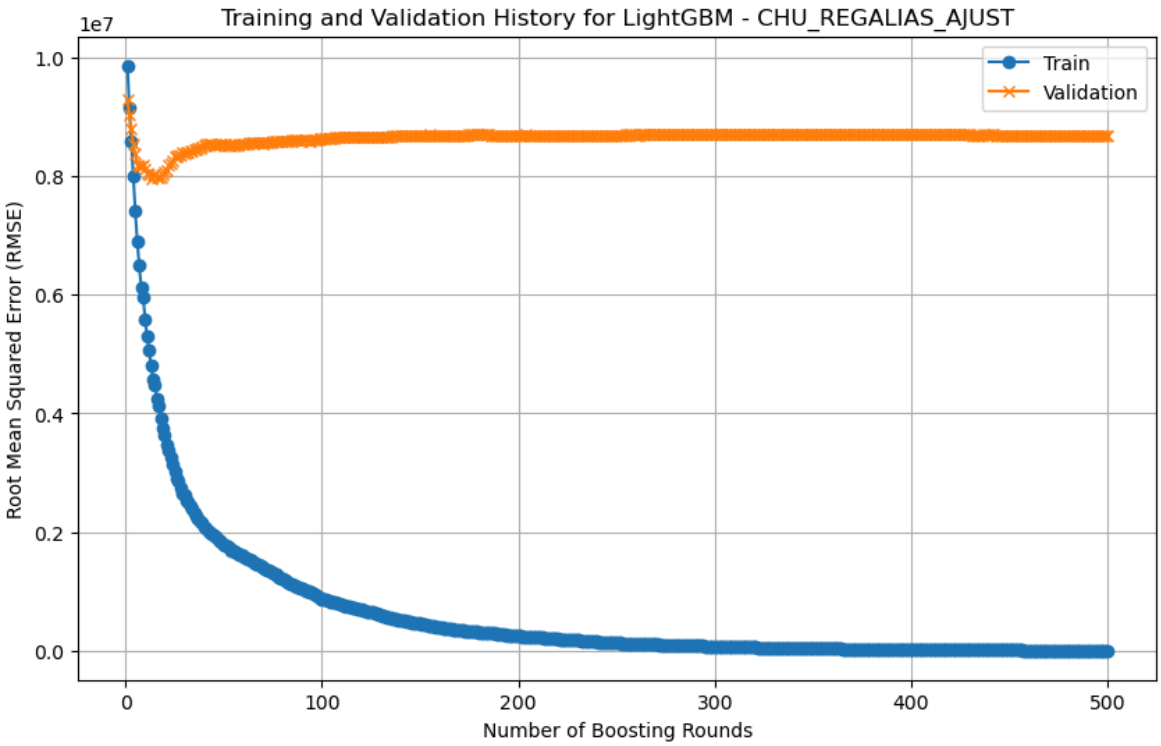
```
RMSE en conjunto de Test Modelo XGB: 6190399.85
MAE en conjunto de Test Modelo XGB: 3598181.03
--------------
RMSE en conjunto de Test Modelo XGB: 6773735.25
MAE en conjunto de Test Modelo XGB: 4069939.65
--------------
RMSE en conjunto de Test Modelo XGB: 8688363.20
MAE en conjunto de Test Modelo XGB: 5021455.02
--------------
```

In [24]:
```
models = [xgb_copa, xgb_rec_propios, xgb_regalias]

for i in range(len(models)):

    resultsXGB = models[i].evals_result()
    train_error = resultsXGB['validation_0']['rmse']
    val_error = resultsXGB['validation_1']['rmse']

    # Número de rondas de boosting
    epoch = range(1, len(train_error) + 1)

    # Crear el gráfico
    plt.figure(figsize=(10, 6))
    plt.plot(epoch, train_error, label='Train', marker='o')
```

```
    plt.plot(epoch, val_error, label='Validation', marker='x')
    plt.xlabel('Number of Boosting Rounds')
    plt.ylabel('Root Mean Squared Error (RMSE)')
    plt.title(f'Training and Validation History for LightGBM - {titulos[i]}')
    plt.legend()
    plt.grid()
    plt.show()
```



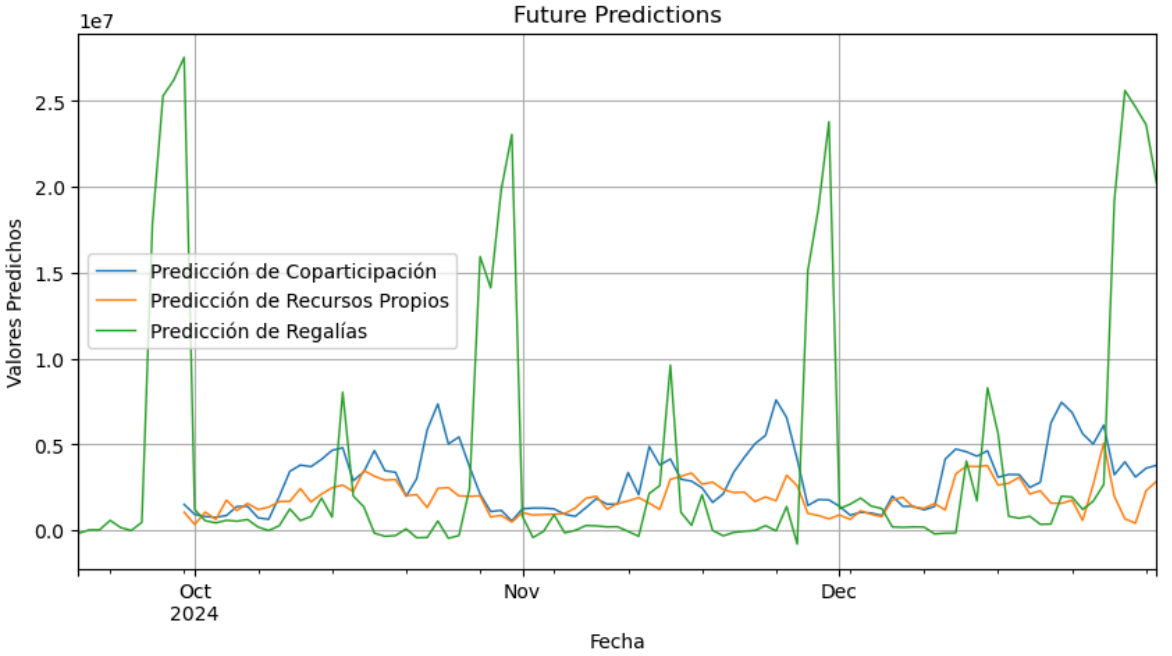






In [38]:

```python
import matplotlib.pyplot as plt

models = [xgb_copa, xgb_rec_propios, xgb_regalias]
titles = ["Predicción de Coparticipación", "Predicción de Recursos Propios", "Pr

plt.figure(figsize=(10, 5))  # Crear una figura para graficar

for i in range(len(models)):
    df_aux = dataframes[i].copy()
    df_aux.drop(columns=['dayofweek', 'quarter', 'month', 'year', 'dayofyear',
                         'dayofmonth', 'weekofyear', 'lag1', 'lag2', 'lag3'], in

    # Generar datos futuros
    future = pd.date_range(dataframes[i].index.max(), '2024-12-31', freq='1d')
    future_df = pd.DataFrame(index=future)
    future_df['isFuture'] = True
    df_aux['isFuture'] = False
    df_and_future = pd.concat([df_aux, future_df])

    # Agregar características de tiempo y lags
    time_features = df_and_future.index.to_series().apply(extract_time_features)
    df_and_future = pd.concat([df_and_future, time_features], axis=1)
    df_and_future = add_lags(df_and_future, titulos[i])

    # Predicciones futuras
    future_w_features = df_and_future.query('isFuture').copy()
    future_w_features['pred'] = models[i].predict(future_w_features[FEATURES])

    # Graficar las predicciones
    future_w_features['pred'].plot(
        ms=1,
        lw=1,
        label=titles[i]  # Etiqueta para la leyenda
    )

# Personalizar el gráfico
```

```
plt.title("Future Predictions")
plt.xlabel("Fecha")
plt.ylabel("Valores Predichos")
plt.legend()  # Mostrar leyenda
plt.grid(True)
plt.show()
```




In [ ]:
```
param_grid = {
    'n_estimators': [100, 500], # Numero de arboles a incluir en el modelo
    'max_depth': [3, 5, 10],   # Profundidad maxima de cada arbol
    'learning_rate': [0.01, 0.1]
}

# Lags used as predictors
lags_grid = [48, 72]

results_grid = grid_search_forecaster(
                    forecaster         = forecaster,    # Se pasa el objeto Forec
                    y                  = data.loc[:end_validation, 'users'], #  s
                    param_grid         = param_grid,
                    lags_grid          = lags_grid,
                    steps              = 36,  # numero de pasos en el futuro que
                    refit              = False, # no se ajustara el modelo con lo
                    metric             = 'mean_squared_error',
                    initial_train_size = len(data_train), # se utilizara todos lo
                    fixed_train_size   = False, # en cada iteracion de la busqued
                    return_best        = True, # la función devolvera el mejor mo
                    verbose            = False
                )
```