In [ ]:
```
# !pip install pandas
# !pip install xgboost
```

In [1]:
```
import os

import inspect
import time
import warnings
import calendar

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px
import seaborn as sns
from sklearn.metrics import mean_squared_log_error
from sklearn.preprocessing import LabelEncoder
import pytz
import scipy.stats as stats
import xgboost as xgb
from sklearn.model_selection import TimeSeriesSplit, RandomizedSearchCV, GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from xgboost import XGBRegressor
import optuna
from xgboost import plot_importance

warnings.simplefilter(action="ignore", category=FutureWarning)
warnings.filterwarnings("ignore")
```

In [2]:
```
def cal_rmlse(y_true, y_pred):
    """
    Calculate the Root Mean Logarithmic Squared Error (RMLSE).

    :param y_true: Array of true values
    :param y_pred: Array of predicted values
    :return: RMLSE value
    """
    # Ensure the predictions are positive
    y_pred = np.clip(y_pred, 0, None)

    # Calculate Mean Squared Logarithmic Error
    msle = mean_squared_log_error(y_true, y_pred)

    # Calculate the Root Mean Logarithmic Squared Error
    rmlse = np.sqrt(msle)

    return rmlse
```

Data Preparation

Importing the Datasets

In [3]:
```
def wrangle(filepath):
    # Read CSV file
    df = pd.read_csv(filepath)
    # Change the date to `datetime` type
    df["date"] = pd.to_datetime(df["date"])

    return df
```

In [4]:
```
train_data = wrangle('/kaggle/input/store-sales-time-series-forecasting/train.csv')
test_data = wrangle('/kaggle/input/store-sales-time-series-forecasting/test.csv')
```

In [5]:
```
oil=wrangle("/kaggle/input/store-sales-time-series-forecasting/oil.csv")
stores=pd.read_csv("/kaggle/input/store-sales-time-series-forecasting/stores.csv")
transactions=pd.read_csv("/kaggle/input/store-sales-time-series-forecasting/transactions.csv")
holidays=wrangle("/kaggle/input/store-sales-time-series-forecasting/holidays_events.csv")
```

In [6]:
```
# Aggregate data for training
train_data = train_data.groupby(['date', 'store_nbr', 'family']).agg({
        'sales': 'sum',
        'onpromotion': 'sum'
    }).reset_index()
# Aggregate data for testing
test_data = test_data.groupby(['date', 'store_nbr', 'family']).agg({
        'onpromotion': 'sum'
    }).reset_index()
```

Explore

Train and Test Data

In [7]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000888 entries, 0 to 3000887
Data columns (total 5 columns):
 #   Column       Dtype
---  ------       -----
 0   date         datetime64[ns]
 1   store_nbr    int64
 2   family       object
 3   sales        float64
 4   onpromotion  int64
dtypes: datetime64[ns](1), float64(1), int64(2), object(1)
memory usage: 114.5+ MB
```

The date changed to `datetime` object for easier manipulation of time series data.

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

Out[8]:

	date	store_nbr	family	sales	onpromotion
0	2013-01-01	1	AUTOMOTIVE	0.0	0
1	2013-01-01	1	BABY CARE	0.0	0
2	2013-01-01	1	BEAUTY	0.0	0
3	2013-01-01	1	BEVERAGES	0.0	0
4	2013-01-01	1	BOOKS	0.0	0

In [9]:
```
test_data.head()
```

Out[9]:

	date	store_nbr	family	onpromotion
0	2017-08-16	1	AUTOMOTIVE	0
1	2017-08-16	1	BABY CARE	0
2	2017-08-16	1	BEAUTY	2
3	2017-08-16	1	BEVERAGES	20
4	2017-08-16	1	BOOKS	0

In [10]:
```
train_data.isnull().sum()
```

Out[10]:
```
date           0
store_nbr      0
family         0
sales          0
onpromotion    0
dtype: int64
```

I believe that we are incredibly lucky that there seems to be no missing values

In [11]:
```
train_data.nunique()
```

Out[11]:
```
date             1684
store_nbr          54
family             33
sales          379610
onpromotion       362
dtype: int64
```

It seems like we need bar charts/histograms to better explore store_nbr and family and need boxplots and histograms for exploring sales

In [12]:
```
train_data[["sales", "onpromotion"]].describe()
```

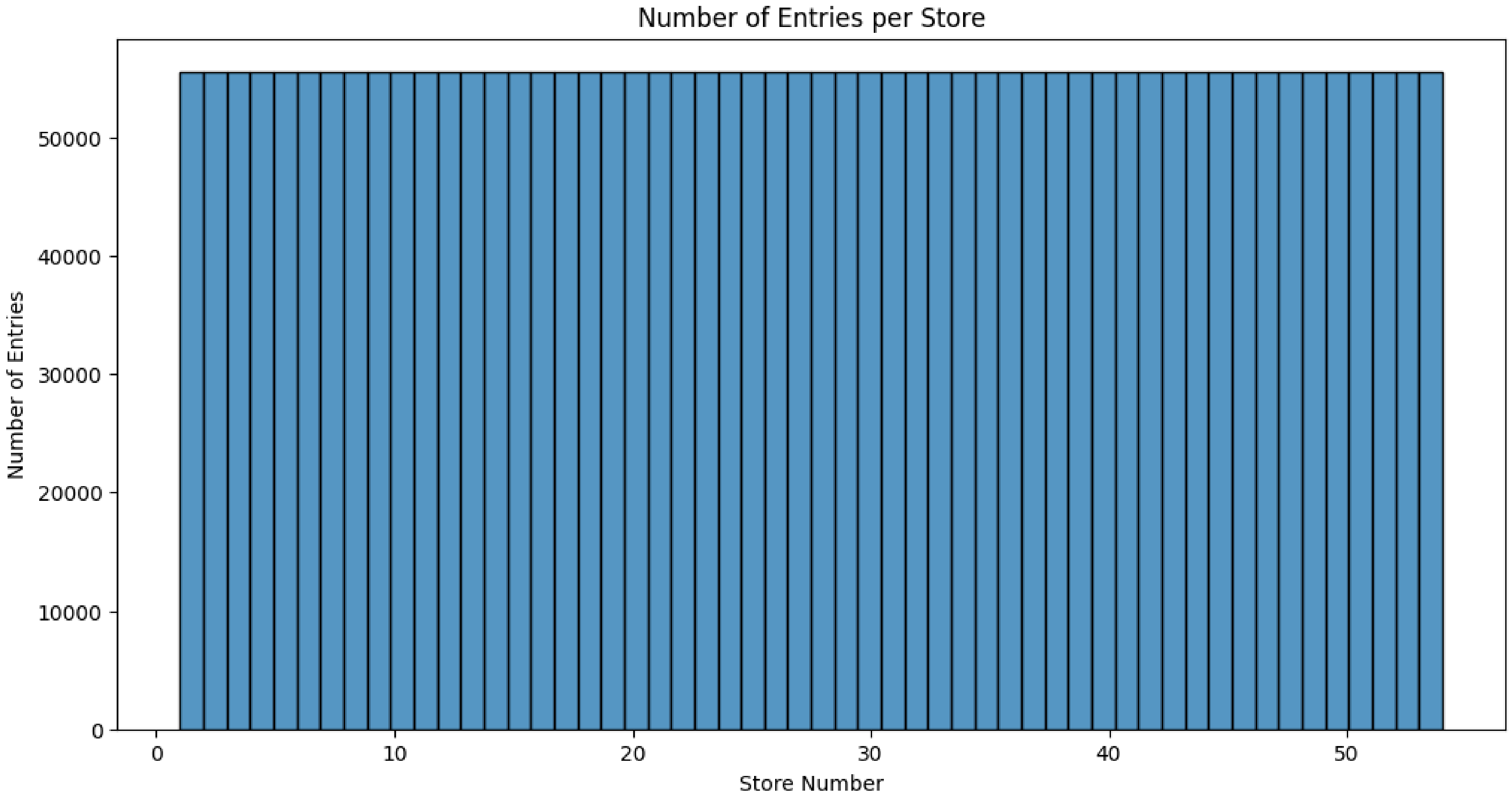
Out[12]:

	sales	onpromotion
count	3.000888e+06	3.000888e+06
mean	3.577757e+02	2.602770e+00
std	1.101998e+03	1.221888e+01
min	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00
50%	1.100000e+01	0.000000e+00
75%	1.958473e+02	0.000000e+00
max	1.247170e+05	7.410000e+02

As we can see, although the dataset has no failed data point (no data point falling under 0), the mean for both `sales` and `onpromotion` is much larger than the median (the 50% quartile). To understand this, we need to see how house sizes are distributed in our dataset. Let's look at two ways to visualize the distribution: a histogram and a boxplot.

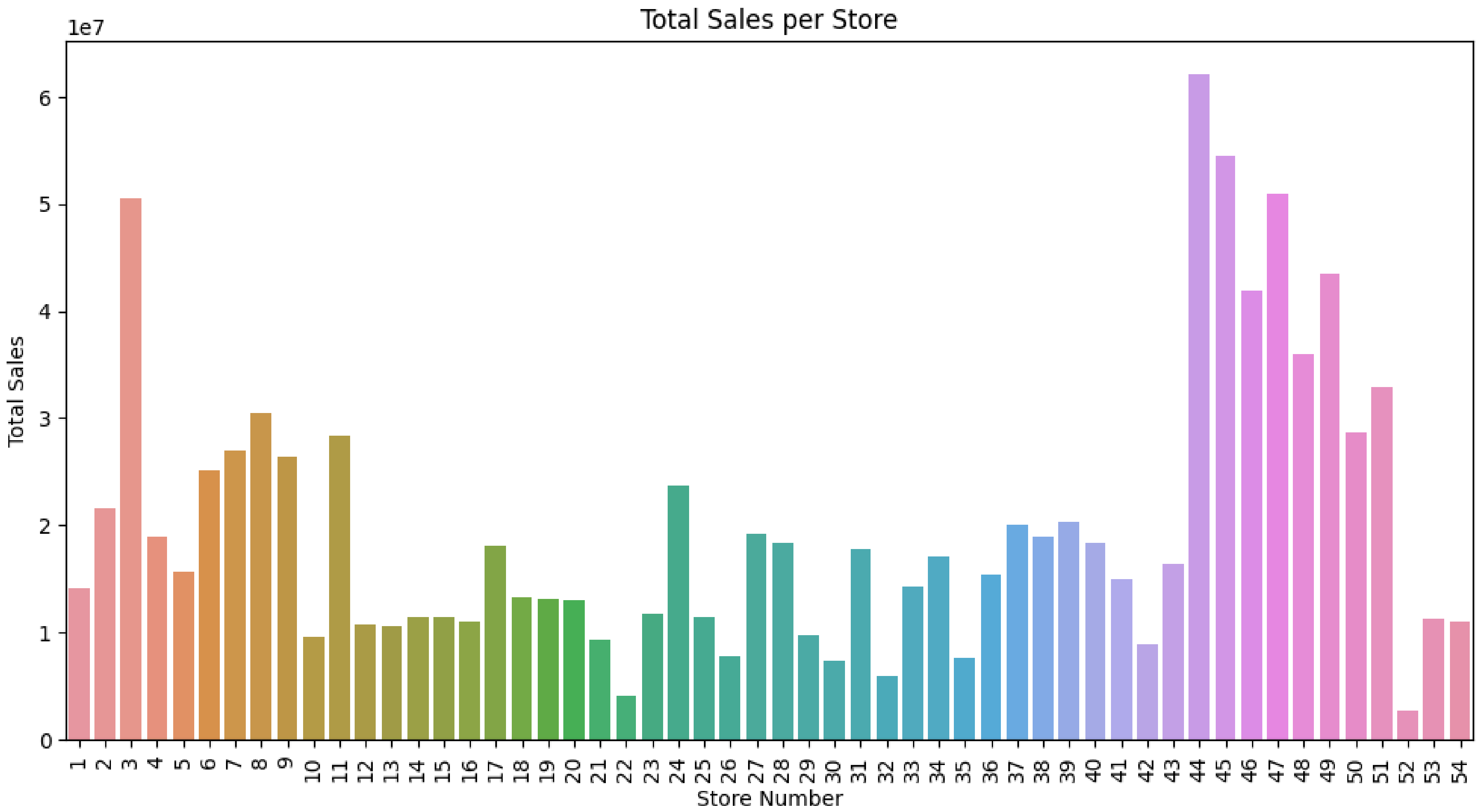
store_nbr

In [13]:
```
plt.figure(figsize=(12, 6))

sns.histplot(train_data["store_nbr"], bins=len(train_data["store_nbr"].unique()), kde=False)
plt.title("Number of Entries per Store")
plt.xlabel("Store Number")
plt.ylabel("Number of Entries")
plt.show()
```



Data collection was evenly distributed across all stores.

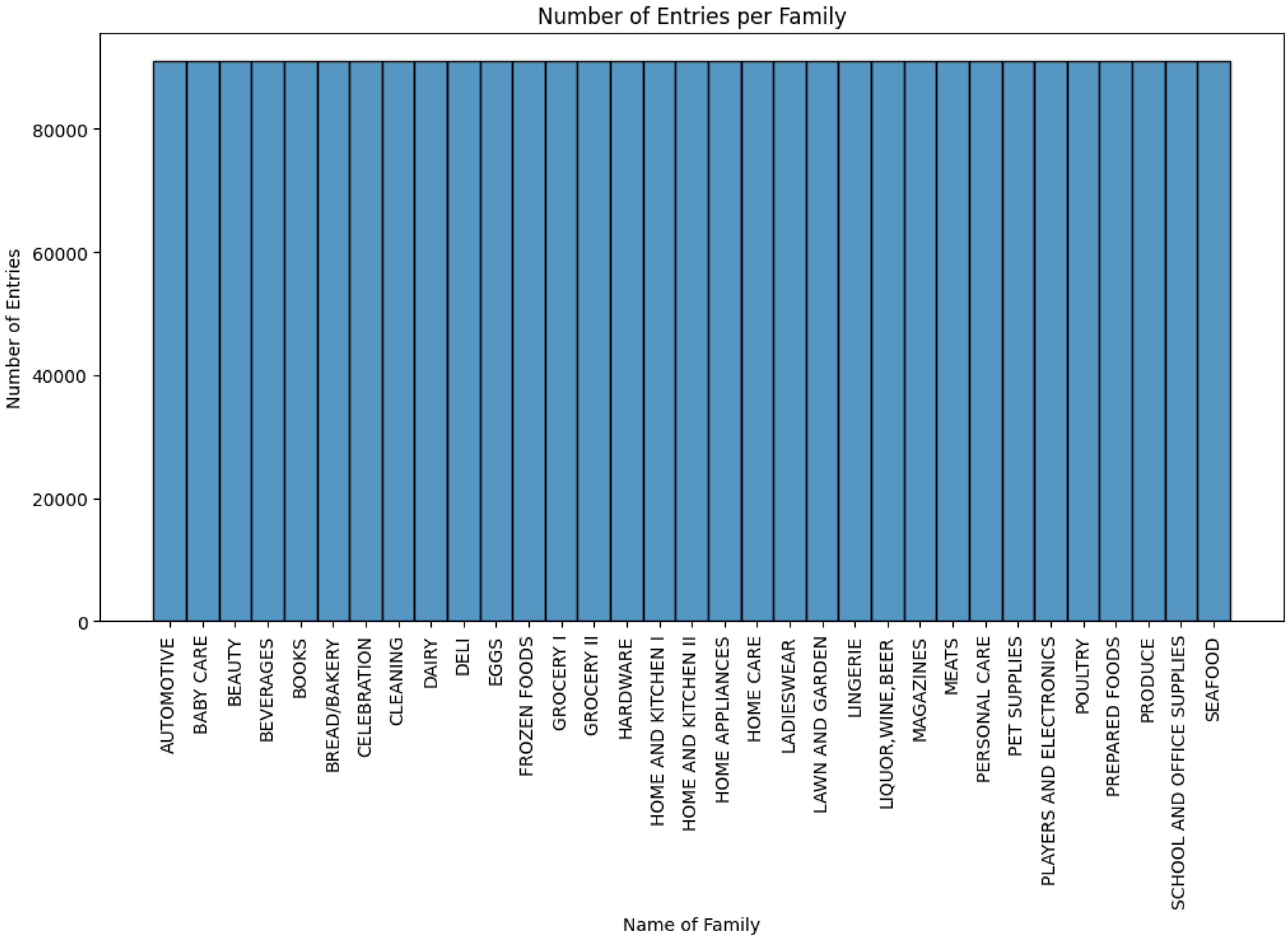
In [14]:
```
# Calculate total sales per store
total_sales_per_store = train_data.groupby("store_nbr")["sales"].sum()

# Plotting total sales per store
plt.figure(figsize=(12, 6))
sns.barplot(x=total_sales_per_store.index, y=total_sales_per_store.values)
plt.title("Total Sales per Store")
plt.xlabel("Store Number")
plt.ylabel("Total Sales")
plt.xticks(rotation=90)
plt.show()
del total_sales_per_store
```



There seems to be differences in revenue generated across different stores though Despite having the same number of data points, some stores generate much higher sales values than others. This variation could be due to several factors, such as store location, customer demographics, store size, product variety, pricing strategies, etc.

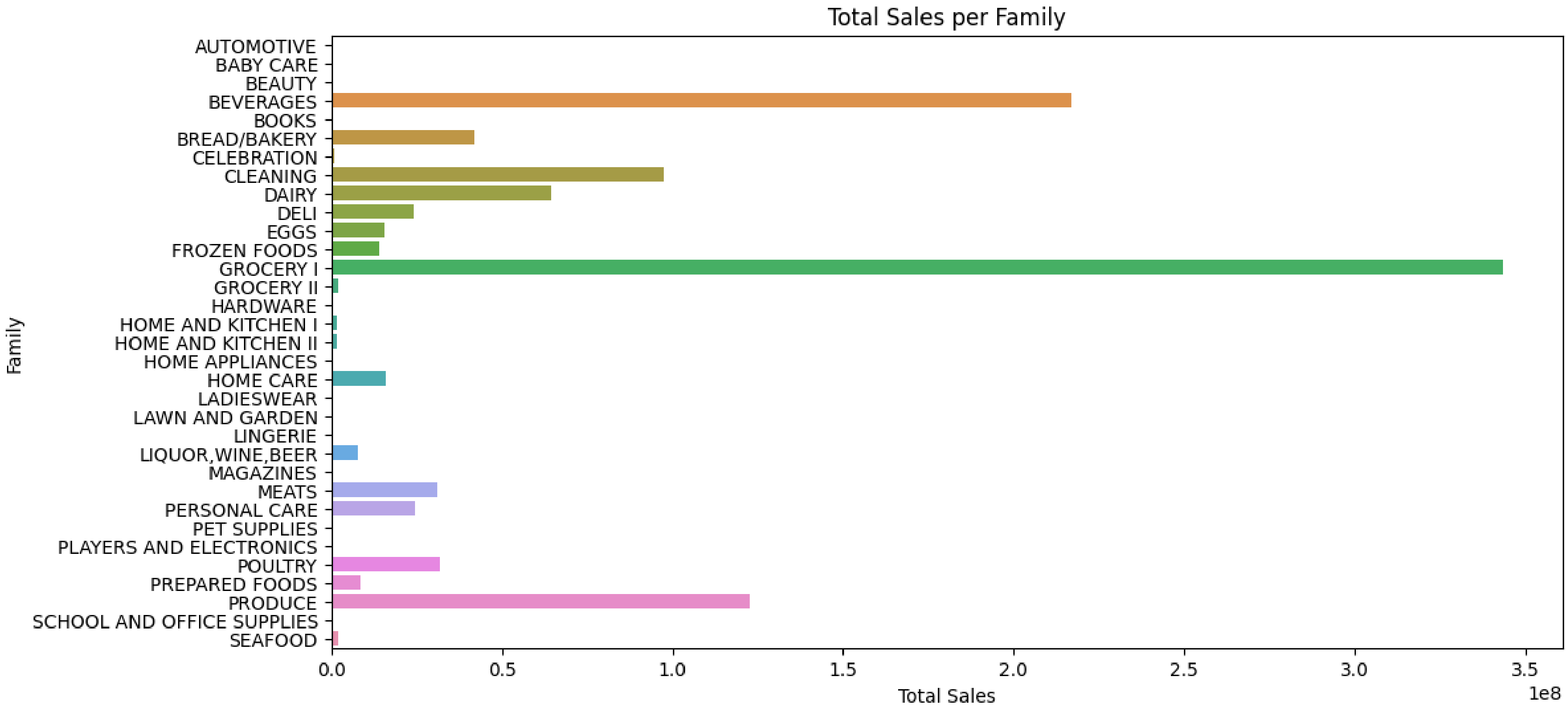
family

In [15]:
```
plt.figure(figsize=(12, 6))
sns.histplot(train_data["family"], bins=len(train_data["family"].unique()), kde=False)
plt.title("Number of Entries per Family")
plt.xlabel("Name of Family")
plt.ylabel("Number of Entries")
plt.xticks(rotation=90)
plt.show()
```

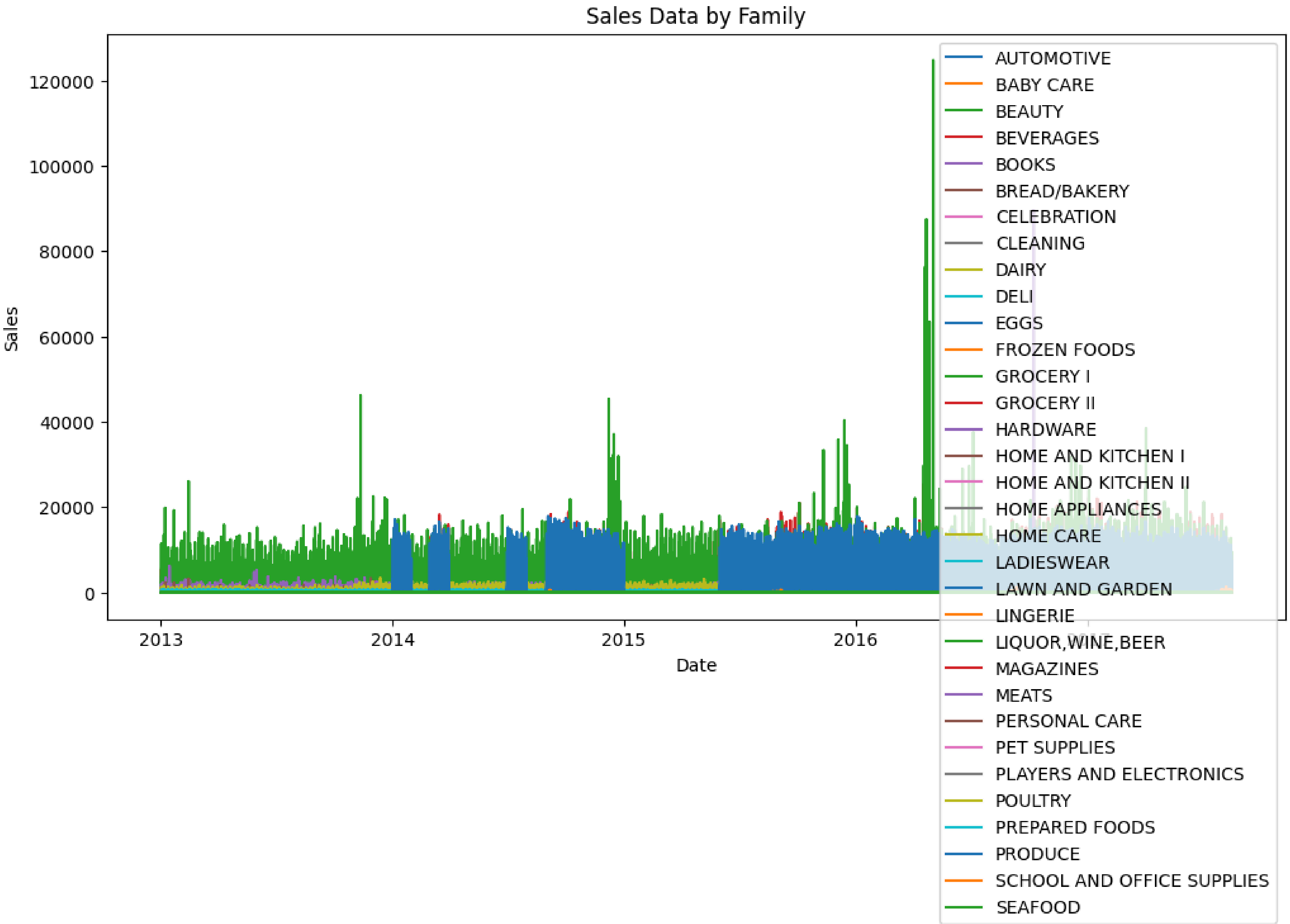

Number of Entries per Family
Number of Entries
0
20000
40000
60000
80000
AUTOMOTIVE
BABY CARE
BEAUTY
BEVERAGES
BOOKS
BREAD/BAKERY
CELEBRATION
CLEANING
DAIRY
DELI
EGGS
FROZEN FOODS
GROCERY I
GROCERY II
HARDWARE
HOME AND KITCHEN I
HOME AND KITCHEN II
HOME APPLIANCES
HOME CARE
LADIESWEAR
LAWN AND GARDEN
LINGERIE
LIQUOR,WINE,BEER
MAGAZINES
MEATS
PERSONAL CARE
PET SUPPLIES
PLAYERS AND ELECTRONICS
POULTRY
PREPARED FOODS
PRODUCE
SCHOOL AND OFFICE SUPPLIES
SEAFOOD
Name of Family

Similarly, data collection seems to be very evenly distributed for Family

In [16]:
```
# Calculate total sales per family
total_sales_per_family = train_data.groupby("family")["sales"].sum()

# Plotting total sales per family
plt.figure(figsize=(12, 6))
sns.barplot(x=total_sales_per_family.values, y=total_sales_per_family.index, orient='h')
plt.title("Total Sales per Family")
plt.xlabel("Total Sales")
plt.ylabel("Family")
plt.show()
del total_sales_per_family
```



In [17]:
```
# Trend in Sales per Family
plt.figure(figsize=(12, 6))
for family in train_data['family'].unique():
    family_data = train_data[train_data['family'] == family]
    family_data.set_index('date', inplace=True)
    plt.plot(family_data.index, family_data['sales'], label=family)
plt.legend()
plt.title('Sales Data by Family')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.show()
del family_data
```


Sales Data by Family
Sales
120000
100000
80000
60000
40000
20000
0
2013
2014
2015
2016
Date
AUTOMOTIVE
BABY CARE
BEAUTY
BEVERAGES
BOOKS
BREAD/BAKERY
CELEBRATION
CLEANING
DAIRY
DELI
EGGS
FROZEN FOODS
GROCERY I
GROCERY II
HARDWARE
HOME AND KITCHEN I
HOME AND KITCHEN II
HOME APPLIANCES
HOME CARE
LADIESWEAR
LAWN AND GARDEN
LINGERIE
LIQUOR,WINE,BEER
MAGAZINES
MEATS
PERSONAL CARE
PET SUPPLIES
PLAYERS AND ELECTRONICS
POULTRY
PREPARED FOODS
PRODUCE
SCHOOL AND OFFICE SUPPLIES
SEAFOOD

There seems to be a lot of changes throughout the course of the timeline. Seems like we need to cut and only train for the latter part of the data since many of the families weren't there to begin with in the first place

Additional datasets

In [18]:
```
oil.head()
```

Out[18]:

	date	dcoilwtico
0	2013-01-01	NaN
1	2013-01-02	93.14
2	2013-01-03	92.97
3	2013-01-04	93.12
4	2013-01-07	93.20

In [19]:
```
stores.head()
```

Out[19]:

	store_nbr	city	state	type	cluster
0	1	Quito	Pichincha	D	13
1	2	Quito	Pichincha	D	13
2	3	Quito	Pichincha	D	8
3	4	Quito	Pichincha	D	9
4	5	Santo Domingo	Santo Domingo de los Tsachilas	D	4

In [20]:
```
print(oil.isnull().sum())
```

```
date           0
dcoilwtico    43
dtype: int64
```

In [21]:
```
transactions.head()
```

Out[21]:

	date	store_nbr	transactions
0	2013-01-01	25	770
1	2013-01-02	1	2111
2	2013-01-02	2	2358
3	2013-01-02	3	3487
4	2013-01-02	4	1922

In [22]:
```
print(stores.isnull().sum())
```

```
store_nbr    0
city         0
state        0
type         0
cluster      0
dtype: int64
```

In [23]:
```
holidays.head()
```

Out[23]:

	date	type	locale	locale_name	description	transferred
0	2012-03-02	Holiday	Local	Manta	Fundacion de Manta	False
1	2012-04-01	Holiday	Regional	Cotopaxi	Provincializacion de Cotopaxi	False
2	2012-04-12	Holiday	Local	Cuenca	Fundacion de Cuenca	False
3	2012-04-14	Holiday	Local	Libertad	Cantonizacion de Libertad	False
4	2012-04-21	Holiday	Local	Riobamba	Cantonizacion de Riobamba	False

In [24]:
```
print(transactions.isnull().sum())
```

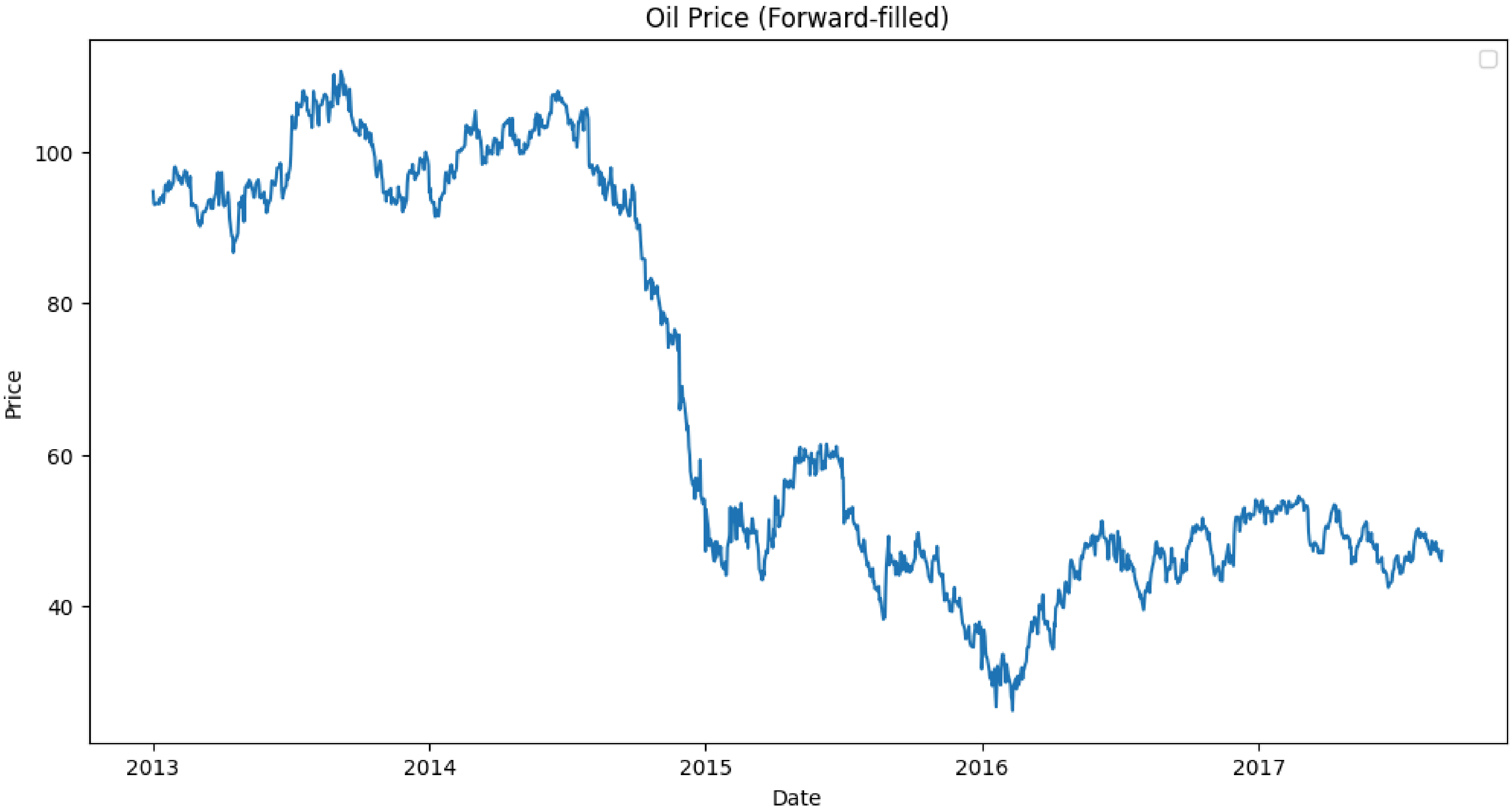
```
date            0
store_nbr       0
transactions    0
dtype: int64
```

Data Preprocessing

Seems like we need to deal with the oil price data

In [25]:
```
def preprocess_oil(oil):
    oil['month'] = oil['date'].dt.to_period('M')
    oil['month_avg'] = oil.groupby('month')['dcoilwtico'].transform('mean')
    oil['tmp'] = oil['dcoilwtico'].isna()
    oil['month_avg'] = oil['tmp'] * oil['month_avg']
    oil['dcoilwtico'].fillna(0, inplace=True)
    oil['dcoilwtico'] = oil['dcoilwtico'] + oil['month_avg']
    oil = oil.drop(['month', 'month_avg', 'tmp'], axis=1)
    return oil

oil = preprocess_oil(oil)
```

In [26]:
```
plt.figure(figsize=(12, 6))
plt.plot(oil["date"], oil["dcoilwtico"])
plt.legend()
plt.title('Oil Price (Forward-filled)')
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()
```



Similarly, I think I need to train the model on only the latter part of the data

Data Transforming

Join the different datasets into one single dataset for extra features

In [27]:
```
# Merge datasets
train_data = train_data.merge(stores, how='left', on='store_nbr')
train_data = train_data.merge(oil, how='left', on='date')
train_data = train_data.merge(holidays, how='left', on='date')
# Apply the same feature engineering steps to the test data
test_data = test_data.merge(stores, how='left', on='store_nbr')
test_data = test_data.merge(oil, how='left', on='date')
test_data = test_data.merge(holidays, how='left', on='date')
```

In [28]:
```
train_data.isnull().sum()
```

Out[28]:
```
date                 0
store_nbr            0
family               0
sales                0
onpromotion          0
city                 0
state                0
type_x               0
cluster              0
dcoilwtico      878526
type_y         2551824
locale         2551824
locale_name    2551824
description    2551824
transferred    2551824
dtype: int64
```

Drop uneeded columns

In [29]:
```
columns_to_drop = ['type_x', 'locale', 'locale_name', 'description', 'transferred']
train_data = train_data.drop(columns=columns_to_drop)
test_data = test_data.drop(columns=columns_to_drop)
```

Feature engineering for date - since XGBoost cannot directly capture this data type, we have to create these features

In [30]:
```
# Add custom features for train data
def add_custom_features(df):
    df['date'] = pd.to_datetime(df['date'])  # Ensure 'date' column is datetime type
    df['weekday'] = df['date'].dt.weekday
    df['year'] = df['date'].dt.year
    df['month'] = df['date'].dt.month
    df['day'] = df['date'].dt.day
    df['eomd'] = df['date'].apply(lambda x: calendar.monthrange(x.year, x.month)[1])
    df['payday'] = ((df['day'] == 15) | (df['day'] == df['eomd'])).astype(int)
    df['is_weekend'] = df['weekday'].isin([5, 6]).astype(int)
    df.drop(['eomd'], axis=1, inplace=True)  # Drop unnecessary columns
    return df

# Apply custom features to both train and test data
train_data = add_custom_features(train_data)
test_data = add_custom_features(test_data)
```

In [31]:
```
# # Add lag features for lag 7 and lag 30
# train_data['sales_lag_7'] = train_data.groupby(['store_nbr', 'family'])['sales'].shift(7)
# train_data['sales_lag_30'] = train_data.groupby(['store_nbr', 'family'])['sales'].shift(30)
# test_data['sales_lag_7'] = test_data.groupby(['store_nbr', 'family'])['onpromotion'].shift(7)
# test_data['sales_lag_30'] = test_data.groupby(['store_nbr', 'family'])['onpromotion'].shift(30)

# # Fill missing values after shifting
# train_data.fillna(0, inplace=True)
# test_data.fillna(0, inplace=True)
```

In [32]:
```
def add_lag_features(df, lags):
    for lag in lags:
        df[f'sales_lag_{lag}'] = df.groupby(['store_nbr', 'family'])['sales'].transform(lambda x: x.shift(lag))
    return df
```

In [33]:
```
def add_rolling_mean_features(df, windows):
    for window in windows:
        df[f'sales_roll_mean_{window}'] = df.groupby(['store_nbr', 'family'])['sales'].transform(
            lambda x: x.shift(1).rolling(window=window, min_periods=1).mean()) + add_random_noise(df)
    return df
```

In [34]:
```
def add_ewm_features(df, alphas, lags):
    for alpha in alphas:
        for lag in lags:
            df[f'sales_ewm_alpha_{str(alpha).replace(".", "")}_lag_{lag}'] = df.groupby(['store_nbr', 'family'])['sale
s'].transform(
                lambda x: x.shift(lag).ewm(alpha=alpha).mean())
    return df
```

In [35]:
```
def add_random_noise(df):
    return np.random.normal(scale=2.0, size=(len(df),))
```

In [36]:
```
train_data.tail()
```

Out[36]:

	date	store_nbr	family	sales	onpromotion	city	state	cluster	dcoilwtico	type_y	weekday	year	month	day	payday	i
3054343	2017-08-15	54	POULTRY	59.619	0	El Carmen	Manabi	3	47.57	Holiday	1	2017	8	15	1	
3054344	2017-08-15	54	PREPARED FOODS	94.000	0	El Carmen	Manabi	3	47.57	Holiday	1	2017	8	15	1	
3054345	2017-08-15	54	PRODUCE	915.371	76	El Carmen	Manabi	3	47.57	Holiday	1	2017	8	15	1	
3054346	2017-08-15	54	SCHOOL AND OFFICE SUPPLIES	0.000	0	El Carmen	Manabi	3	47.57	Holiday	1	2017	8	15	1	
3054347	2017-08-15	54	SEAFOOD	3.000	0	El Carmen	Manabi	3	47.57	Holiday	1	2017	8	15	1	

As we can see, the categorical columns that need encoding are: family, city, state, type_y, cluster, store_nbr

In [37]:
```
test_data.head()
```

Out[37]:

	date	store_nbr	family	onpromotion	city	state	cluster	dcoilwtico	type_y	weekday	year	month	day	payday	is_weekend
0	2017-08-16	1	AUTOMOTIVE	0	Quito	Pichincha	13	46.8	NaN	2	2017	8	16	0	0
1	2017-08-16	1	BABY CARE	0	Quito	Pichincha	13	46.8	NaN	2	2017	8	16	0	0
2	2017-08-16	1	BEAUTY	2	Quito	Pichincha	13	46.8	NaN	2	2017	8	16	0	0
3	2017-08-16	1	BEVERAGES	20	Quito	Pichincha	13	46.8	NaN	2	2017	8	16	0	0
4	2017-08-16	1	BOOKS	0	Quito	Pichincha	13	46.8	NaN	2	2017	8	16	0	0

In [38]:
```
del oil
del stores
del transactions
del holidays
```

In [39]:
```
# Concatenate train and test datasets
data = pd.concat([train_data, test_data], axis=0, ignore_index=True)
```

In [40]:
```
# Specify the lags, rolling windows, and EWM parameters
lags = [7, 14, 30]
windows = [7, 30]
ewm_alphas = [0.95, 0.9, 0.8]
ewm_lags = [7, 30]

# Apply lag features
data = add_lag_features(data, lags)

# Apply rolling mean features
data = add_rolling_mean_features(data, windows)

# Apply EWM features
data = add_ewm_features(data, ewm_alphas, ewm_lags)
```

In [41]:
```
data.fillna(0, inplace=True)
```

In [42]:
```
#Split back into train and test data
train_data = data[data['date'] <= '2017-08-15'].copy()
test_data = data[data['date'] > '2017-08-15'].copy()
```

In [43]:
```
# List of categorical features to encode
cat_features = ['family', 'city', 'state', 'type_y', 'cluster', 'store_nbr']

train_data_encoded = train_data
test_data_encoded = test_data

# Ensure all values in categorical columns are of string type
train_data_encoded[cat_features] = train_data_encoded[cat_features].astype(str)
test_data_encoded[cat_features] = test_data_encoded[cat_features].astype(str)

# Apply Label Encoding
label_encoders = {}
for col in cat_features:
    le = LabelEncoder()
    train_data_encoded[col] = le.fit_transform(train_data[col])
    test_data_encoded[col] = le.transform(test_data[col])
    label_encoders[col] = le
```

In [44]:
```
del train_data
```

In [45]:
```
train_data_encoded.head()
```

Out[45]:

	date	store_nbr	family	sales	onpromotion	city	state	cluster	dcoilwtico	type_y	...	sales_lag_14	sales_lag_30	sales_roll_mean_7	sales_rc
0	2013-01-01	0	0	0.0	0	18	12	4	94.756667	4	...	0.0	0.0	0.0	
1	2013-01-01	0	1	0.0	0	18	12	4	94.756667	4	...	0.0	0.0	0.0	
2	2013-01-01	0	2	0.0	0	18	12	4	94.756667	4	...	0.0	0.0	0.0	
3	2013-01-01	0	3	0.0	0	18	12	4	94.756667	4	...	0.0	0.0	0.0	
4	2013-01-01	0	4	0.0	0	18	12	4	94.756667	4	...	0.0	0.0	0.0	

5 rows × 27 columns

In [46]:
```
test_data_encoded.nunique()
```

Out[46]:
```
date                         16
store_nbr                    54
family                       33
sales                         1
onpromotion                 212
city                         22
state                        16
cluster                      17
dcoilwtico                   12
type_y                        2
weekday                       7
year                          1
month                         1
day                          16
payday                        2
is_weekend                    2
sales_lag_7                3838
sales_lag_14               6981
sales_lag_30               7877
sales_roll_mean_7         12475
sales_roll_mean_30        28512
sales_ewm_alpha_095_lag_7     11908
sales_ewm_alpha_095_lag_30    26903
sales_ewm_alpha_09_lag_7      11956
sales_ewm_alpha_09_lag_30     27139
sales_ewm_alpha_08_lag_7      12010
sales_ewm_alpha_08_lag_30     27348
dtype: int64
```

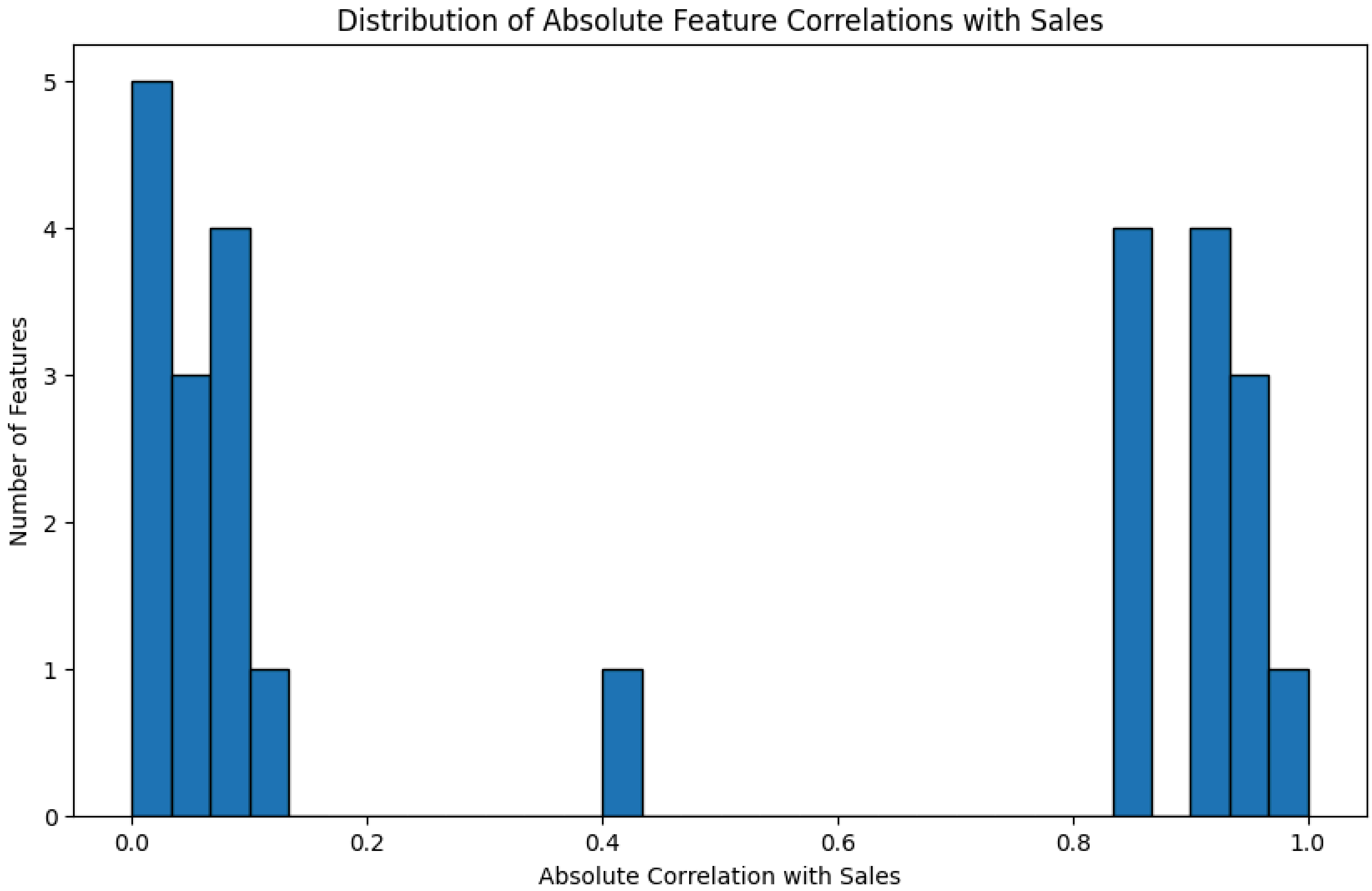
In [47]:
```
# Drop the original categorical columns since they are now encoded
train_data_encoded.drop(['date'], axis=1, inplace=True)
test_data_encoded.drop(['date'], axis=1, inplace=True)
```

In [48]:
```
# Calculate the correlation matrix
correlation_matrix = train_data_encoded.corr()

# Extract the correlation with the 'sales' column
correlation_with_sales = correlation_matrix['sales']
```

In [49]:
```
# Use the absolute values of the correlations
abs_correlation_with_sales = correlation_with_sales.abs()

# Get summary statistics for the absolute correlation with 'sales'
correlation_summary = abs_correlation_with_sales.describe()

# Print the summary statistics
print(correlation_summary)
```

```
count    26.000000
mean      0.460926
std       0.434118
min       0.000826
25%       0.050164
50%       0.271114
75%       0.926502
max       1.000000
Name: sales, dtype: float64
```

In [50]:
```
plt.figure(figsize=(10, 6))
plt.hist(abs_correlation_with_sales, bins=30, edgecolor='black')
plt.title('Distribution of Absolute Feature Correlations with Sales')
plt.xlabel('Absolute Correlation with Sales')
plt.ylabel('Number of Features')
plt.show()
```



In [51]:
```
# low_correlation_threshold = 0.01

# # Identify features with correlation below
# low_correlation_features = abs_correlation_with_sales[abs_correlation_with_sales <= low_correlation_threshold]

# # Print the features with the low correlation
# print("Features with low correlation to 'sales':")
# print(low_correlation_features)
```

In [52]:
```
# # Drop low-correlation features from the dataset
# train_data_encoded = train_data_encoded.drop(columns=low_correlation_features.index)
```

In [53]:
```
# # Drop only the features that exist in the test dataset
# common_features = low_correlation_features.index.intersection(test_data_encoded.columns)
# test_data_encoded = test_data_encoded.drop(columns=common_features)
```

Split

In [54]:
```
# Define the target and features
X = train_data_encoded.drop(columns=['sales'])
y = train_data_encoded['sales']
```

Model Building

Bayesyan Optuna Test

Perform a bayesyan optuna test to find the best hyper param combo. This is performed on the last 20% of the data

In [55]:
```
# Assuming train_data_encoded is your preprocessed DataFrame sorted by time
subset_frac = 0.20
subset_size = int(len(train_data_encoded) * subset_frac)
subsample_train_data = train_data_encoded.iloc[-subset_size:]
```

In [56]:
```
# Split features and target
X_subsample = subsample_train_data.drop(columns=["sales"])
y_subsample = subsample_train_data["sales"]
```

In [57]:
```
# Time-based train-validation split within the subset
split_index = int(0.8 * len(X_subsample))
X_sub_train, X_sub_val = X_subsample.iloc[:split_index], X_subsample.iloc[split_index:]
y_sub_train, y_sub_val = y_subsample.iloc[:split_index], y_subsample.iloc[split_index:]
```

In [58]:
```
# Define the Optuna objective function
def objective(trial):
    params = {
        # 'tree_method': 'gpu_hist',
        'tree_method': 'hist',
        'n_jobs': -1,
        'objective': 'reg:squarederror',
        'n_estimators': trial.suggest_int('n_estimators', 100, 300),
        'verbosity': 2,
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.1, log=True),
        'max_depth': trial.suggest_int('max_depth', 6, 14),
        'subsample': trial.suggest_float('subsample', 0.6, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.3, 1.0),
        'min_child_weight': trial.suggest_int('min_child_weight', 10, 24),
        'reg_lambda': trial.suggest_float('reg_lambda', 0.001, 1, log=True),
        'colsample_bynode': trial.suggest_float('colsample_bynode', 0.3, 0.9)
    }

    model = xgb.XGBRegressor(**params)

    model.fit(X_sub_train, y_sub_train, eval_set=[(X_sub_val, y_sub_val)], early_stopping_rounds=10, verbose=False)
    y_pred = model.predict(X_sub_val)
    rmse = mean_squared_error(y_sub_val, y_pred, squared=False)
    return rmse
```

In [59]:
```
# Create and optimize the study
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=50)
```

```
[I 2024-08-30 13:53:20,154] A new study created in memory with name: no-name-8e0c0e74-7a65-475c-8cf3-1827a32ff793
[I 2024-08-30 13:53:28,247] Trial 0 finished with value: 265.9229525405118 and parameters: {'n_estimators': 204, 'lea
rning_rate': 0.058163819376272116, 'max_depth': 7, 'subsample': 0.9105166559272806, 'colsample_bytree': 0.91913255165
39315, 'min_child_weight': 12, 'reg_lambda': 0.38887326331194283, 'colsample_bynode': 0.7314944208054527}. Best is tr
ial 0 with value: 265.9229525405118.
[I 2024-08-30 13:53:44,167] Trial 1 finished with value: 260.3169542060036 and parameters: {'n_estimators': 178, 'lea
rning_rate': 0.0450763260295260З, 'max_depth': 14, 'subsample': 0.6573685480734993, 'colsample_bytree': 0.30623676025
19679, 'min_child_weight': 11, 'reg_lambda': 0.0010313814457363153, 'colsample_bynode': 0.43752946711258744}. Best is
trial 1 with value: 260.3169542060036.
[I 2024-08-30 13:54:05,400] Trial 2 finished with value: 277.25941607517797 and parameters: {'n_estimators': 105, 'le
arning_rate': 0.02209144192063536, 'max_depth': 13, 'subsample': 0.6928319664213527, 'colsample_bytree': 0.3916320176
6422645, 'min_child_weight': 16, 'reg_lambda': 0.0021578589632463433, 'colsample_bynode': 0.38652045442737326}. Best
is trial 1 with value: 260.3169542060036.
[I 2024-08-30 13:54:32,601] Trial 3 finished with value: 264.23161106371197 and parameters: {'n_estimators': 174, 'le
arning_rate': 0.01475414783052355, 'max_depth': 12, 'subsample': 0.9895067547367239, 'colsample_bytree': 0.9422735462
029441, 'min_child_weight': 13, 'reg_lambda': 0.1631378502382755, 'colsample_bynode': 0.4546045697953453б}. Best is t
rial 1 with value: 260.3169542060036.
[I 2024-08-30 13:55:00,539] Trial 4 finished with value: 260.78426666435985 and parameters: {'n_estimators': 296, 'le
arning_rate': 0.012017354185562452, 'max_depth': 9, 'subsample': 0.8239697069333082, 'colsample_bytree': 0.5997798239
221913, 'min_child_weight': 16, 'reg_lambda': 0.0011843312066901989, 'colsample_bynode': 0.8718322294997018}. Best is
trial 1 with value: 260.3169542060036.
[I 2024-08-30 13:55:09,324] Trial 5 finished with value: 264.14792864962294 and parameters: {'n_estimators': 203, 'le
arning_rate': 0.07406942090493425, 'max_depth': 11, 'subsample': 0.6716735460937768, 'colsample_bytree': 0.5687614385
489372, 'min_child_weight': 20, 'reg_lambda': 0.07632470076991064, 'colsample_bynode': 0.35632238842423364}. Best is
trial 1 with value: 260.3169542060036.
[I 2024-08-30 13:55:26,964] Trial 6 finished with value: 262.87609553630983 and parameters: {'n_estimators': 257, 'le
arning_rate': 0.018664211948677838, 'max_depth': 8, 'subsample': 0.6850252271730313, 'colsample_bytree': 0.6058056329
655389, 'min_child_weight': 16, 'reg_lambda': 0.5768907837596097, 'colsample_bynode': 0.8096872953834899}. Best is tr
ial 1 with value: 260.3169542060036.
[I 2024-08-30 13:55:51,317] Trial 7 finished with value: 257.10252103255976 and parameters: {'n_estimators': 182, 'le
arning_rate': 0.019586340199163842, 'max_depth': 11, 'subsample': 0.9966431328269392, 'colsample_bytree': 0.757288186
1148778, 'min_child_weight': 18, 'reg_lambda': 0.02276784948820459, 'colsample_bynode': 0.6031373019695994}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 13:56:13,827] Trial 8 finished with value: 257.9796041200003 and parameters: {'n_estimators': 150, 'lea
rning_rate': 0.023927205127721975, 'max_depth': 13, 'subsample': 0.6407432972042266, 'colsample_bytree': 0.9949090713
560538, 'min_child_weight': 24, 'reg_lambda': 0.4287664450180171, 'colsample_bynode': 0.33392006992265655}. Best is t
rial 7 with value: 257.10252103255976.
[I 2024-08-30 13:56:34,509] Trial 9 finished with value: 260.03352226920305 and parameters: {'n_estimators': 213, 'le
arning_rate': 0.026614925597443707, 'max_depth': 13, 'subsample': 0.7092442127481942, 'colsample_bytree': 0.965728901
5795003, 'min_child_weight': 14, 'reg_lambda': 0.2845677911947328, 'colsample_bynode': 0.8132639053789592}. Best is t
rial 7 with value: 257.10252103255976.
```

```
[I 2024-08-30 13:56:46,715] Trial 10 finished with value: 259.8540609481273 and parameters: {'n_estimators': 110, 'le
arning_rate': 0.03992849022223801, 'max_depth': 10, 'subsample': 0.9984516845139227, 'colsample_bytree': 0.7778694971
463379, 'min_child_weight': 20, 'reg_lambda': 0.010684198484822476, 'colsample_bynode': 0.6178669838365132}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 13:57:01,321] Trial 11 finished with value: 258.23051617034304 and parameters: {'n_estimators': 141, 'l
earning_rate': 0.03115953078854308, 'max_depth': 11, 'subsample': 0.7938094965830738, 'colsample_bytree': 0.770685089
3629417, 'min_child_weight': 24, 'reg_lambda': 0.021070577730109955, 'colsample_bynode': 0.5646422963728073}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 13:57:31,208] Trial 12 finished with value: 355.03530482574297 and parameters: {'n_estimators': 151, 'l
earning_rate': 0.01016275755984057, 'max_depth': 14, 'subsample': 0.6094529153529586, 'colsample_bytree': 0.807682993
0113675, 'min_child_weight': 24, 'reg_lambda': 0.006961549793987708, 'colsample_bynode': 0.5784605009372661}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 13:57:54,170] Trial 13 finished with value: 258.9570092794584 and parameters: {'n_estimators': 149, 'le
arning_rate': 0.018898091358540547, 'max_depth': 12, 'subsample': 0.7843098673913101, 'colsample_bytree': 0.710067942
4711032, 'min_child_weight': 20, 'reg_lambda': 0.05304141847996039, 'colsample_bynode': 0.664793688903849}. Best is t
rial 7 with value: 257.10252103255976.
[I 2024-08-30 13:58:07,360] Trial 14 finished with value: 267.31751493136125 and parameters: {'n_estimators': 233, 'l
earning_rate': 0.02527382089042943, 'max_depth': 6, 'subsample': 0.8925357793081442, 'colsample_bytree': 0.8708134913
720035, 'min_child_weight': 22, 'reg_lambda': 0.09811282975702969, 'colsample_bynode': 0.483619377541625}. Best is tr
ial 7 with value: 257.10252103255976.
[I 2024-08-30 13:58:28,799] Trial 15 finished with value: 264.3311380778622 and parameters: {'n_estimators': 172, 'le
arning_rate': 0.015003093519896172, 'max_depth': 10, 'subsample': 0.751631178955576, 'colsample_bytree': 0.9981466054
689623, 'min_child_weight': 18, 'reg_lambda': 0.02014755674549245, 'colsample_bynode': 0.30172643997931586}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 13:58:34,712] Trial 16 finished with value: 267.9507958817273 and parameters: {'n_estimators': 132, 'le
arning_rate': 0.09965208880701126, 'max_depth': 12, 'subsample': 0.8695558839634198, 'colsample_bytree': 0.4852996191
8532866, 'min_child_weight': 21, 'reg_lambda': 0.9931386446133654, 'colsample_bynode': 0.5165704281271425}. Best is t
rial 7 with value: 257.10252103255976.
[I 2024-08-30 13:58:45,627] Trial 17 finished with value: 261.7851752174769 and parameters: {'n_estimators': 237, 'le
arning_rate': 0.035177679712493616, 'max_depth': 9, 'subsample': 0.9441810937802714, 'colsample_bytree': 0.8637828912
833903, 'min_child_weight': 10, 'reg_lambda': 0.0047255213177065936, 'colsample_bynode': 0.6866099534740684}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 13:59:10,304] Trial 18 finished with value: 259.3926732338537 and parameters: {'n_estimators': 183, 'le
arning_rate': 0.01587803750174985, 'max_depth': 11, 'subsample': 0.6131483477572174, 'colsample_bytree': 0.6878210118
15821, 'min_child_weight': 18, 'reg_lambda': 0.03299998741046945, 'colsample_bynode': 0.5332994143259323}. Best is tr
ial 7 with value: 257.10252103255976.
[I 2024-08-30 13:59:30,925] Trial 19 finished with value: 259.4969724146305 and parameters: {'n_estimators': 129, 'le
arning_rate': 0.028382221098801022, 'max_depth': 13, 'subsample': 0.8509912946165588, 'colsample_bytree': 0.525373654
1242603, 'min_child_weight': 22, 'reg_lambda': 0.14462821393829767, 'colsample_bynode': 0.6173478408209312}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 13:59:44,425] Trial 20 finished with value: 260.2722152880349 and parameters: {'n_estimators': 161, 'le
arning_rate': 0.049067860749627296, 'max_depth': 14, 'subsample': 0.7483483267773378, 'colsample_bytree': 0.878379770
```

```
9009018, 'min_child_weight': 14, 'reg_lambda': 0.041026379474511473, 'colsample_bynode': 0.7274745986272155}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:00:00,038] Trial 21 finished with value: 258.44258547402643 and parameters: {'n_estimators': 131, 'l
earning_rate': 0.03188562174537994, 'max_depth': 11, 'subsample': 0.8071468185698811, 'colsample_bytree': 0.780128389
537737, 'min_child_weight': 24, 'reg_lambda': 0.015906950676432326, 'colsample_bynode': 0.5720128733975881}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:00:18,894] Trial 22 finished with value: 257.9006358599823 and parameters: {'n_estimators': 149, 'le
arning_rate': 0.022966670447607692, 'max_depth': 11, 'subsample': 0.9437865811411186, 'colsample_bytree': 0.716218486
9113446, 'min_child_weight': 23, 'reg_lambda': 0.020563727263021896, 'colsample_bynode': 0.4026236829930311}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:00:35,969] Trial 23 finished with value: 262.5958918346757 and parameters: {'n_estimators': 184, 'le
arning_rate': 0.021954368907387858, 'max_depth': 9, 'subsample': 0.9587615995904213, 'colsample_bytree': 0.6812803592
689815, 'min_child_weight': 22, 'reg_lambda': 0.004336140203836729, 'colsample_bynode': 0.3195399111326539}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:00:53,675] Trial 24 finished with value: 263.3602039984663 and parameters: {'n_estimators': 118, 'le
arning_rate': 0.021017741482421647, 'max_depth': 12, 'subsample': 0.9279035604637605, 'colsample_bytree': 0.724111663
9747245, 'min_child_weight': 23, 'reg_lambda': 0.011286279549976864, 'colsample_bynode': 0.40447557421758684}. Best i
s trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:01:13,716] Trial 25 finished with value: 262.447038267628 and parameters: {'n_estimators': 160, 'lea
rning_rate': 0.016735943900409576, 'max_depth': 10, 'subsample': 0.9731393768812457, 'colsample_bytree': 0.6448183946
435682, 'min_child_weight': 18, 'reg_lambda': 0.06483569271962626, 'colsample_bynode': 0.35789930530767733}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:01:46,264] Trial 26 finished with value: 268.71849612800355 and parameters: {'n_estimators': 192, 'l
earning_rate': 0.012200259056554178, 'max_depth': 13, 'subsample': 0.873897366493825, 'colsample_bytree': 0.838433898
9981747, 'min_child_weight': 23, 'reg_lambda': 0.028814197501542856, 'colsample_bynode': 0.4268622216328567}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:02:02,974] Trial 27 finished with value: 262.19759331895006 and parameters: {'n_estimators': 221, 'l
earning_rate': 0.025049834721375328, 'max_depth': 10, 'subsample': 0.9225775762738286, 'colsample_bytree': 0.45285376
925405424, 'min_child_weight': 19, 'reg_lambda': 0.1737040745194476, 'colsample_bynode': 0.4778388565192029}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:02:17,513] Trial 28 finished with value: 259.54014229698276 and parameters: {'n_estimators': 163, 'l
earning_rate': 0.034853692647200445, 'max_depth': 12, 'subsample': 0.9573108845680187, 'colsample_bytree': 0.73379307
31758869, 'min_child_weight': 21, 'reg_lambda': 0.010058300025154286, 'colsample_bynode': 0.35222295596598213}. Best
is trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:02:35,828] Trial 29 finished with value: 263.2254547891936 and parameters: {'n_estimators': 198, 'le
arning_rate': 0.019000861912065854, 'max_depth': 8, 'subsample': 0.894512359060449, 'colsample_bytree': 0.89310276481
31166, 'min_child_weight': 23, 'reg_lambda': 0.34865763746149714, 'colsample_bynode': 0.6562526993268581}. Best is tr
ial 7 with value: 257.10252103255976.
[I 2024-08-30 14:02:54,829] Trial 30 finished with value: 257.879618216199 and parameters: {'n_estimators': 148, 'lea
rning_rate': 0.02374575560633277, 'max_depth': 11, 'subsample': 0.8363803998042586, 'colsample_bytree': 0.82299925530
46809, 'min_child_weight': 21, 'reg_lambda': 0.004703144043211581, 'colsample_bynode': 0.5265429287746056}. Best is t
rial 7 with value: 257.10252103255976.
```

```
[I 2024-08-30 14:03:14,928] Trial 31 finished with value: 258.2397829447349 and parameters: {'n_estimators': 145, 'le
arning_rate': 0.023401718727857172, 'max_depth': 11, 'subsample': 0.8290306661562629, 'colsample_bytree': 0.928368643
6062958, 'min_child_weight': 21, 'reg_lambda': 0.00422884688627248, 'colsample_bynode': 0.5314602621749719}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:03:30,053] Trial 32 finished with value: 257.1337653400259 and parameters: {'n_estimators': 164, 'le
arning_rate': 0.028597439106108166, 'max_depth': 11, 'subsample': 0.7475127906705218, 'colsample_bytree': 0.827575932
8462933, 'min_child_weight': 23, 'reg_lambda': 0.0018756148467665222, 'colsample_bynode': 0.4047624130362673 7}. Best
is trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:03:41,831] Trial 33 finished with value: 260.0516220773883 and parameters: {'n_estimators': 169, 'le
arning_rate': 0.04341799264286844, 'max_depth': 10, 'subsample': 0.7335218693949557, 'colsample_bytree': 0.8245611878
078166, 'min_child_weight': 19, 'reg_lambda': 0.0019786850858088878, 'colsample_bynode': 0.4016374032223777}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:03:56,618] Trial 34 finished with value: 260.3599350387516 and parameters: {'n_estimators': 187, 'le
arning_rate': 0.02909982671626887, 'max_depth': 11, 'subsample': 0.7694202960541275, 'colsample_bytree': 0.7527004484
679835, 'min_child_weight': 22, 'reg_lambda': 0.002780708603164761, 'colsample_bynode': 0.4621958683134279}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:04:15,641] Trial 35 finished with value: 265.4323899361993 and parameters: {'n_estimators': 117, 'le
arning_rate': 0.0211922745465558, 'max_depth': 12, 'subsample': 0.9784416293678443, 'colsample_bytree': 0.63953606861
9978, 'min_child_weight': 17, 'reg_lambda': 0.0013255292928782611, 'colsample_bynode': 0.4940988085393391}. Best is t
rial 7 with value: 257.10252103255976.
[I 2024-08-30 14:04:27,957] Trial 36 finished with value: 261.0397695119091 and parameters: {'n_estimators': 175, 'le
arning_rate': 0.036875722552369934, 'max_depth': 11, 'subsample': 0.7217326387165802, 'colsample_bytree': 0.787151338
1651188, 'min_child_weight': 23, 'reg_lambda': 0.006202929709363897, 'colsample_bynode': 0.4341636028224741}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:04:56,415] Trial 37 finished with value: 260.08870311000265 and parameters: {'n_estimators': 286, 'l
earning_rate': 0.01322686918984565, 'max_depth': 9, 'subsample': 0.8484385234800969, 'colsample_bytree': 0.8281713781
634452, 'min_child_weight': 15, 'reg_lambda': 0.0025221196103585664, 'colsample_bynode': 0.3830182663126108}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:05:05,131] Trial 38 finished with value: 259.56996037337234 and parameters: {'n_estimators': 137, 'l
earning_rate': 0.051750719140523295, 'max_depth': 10, 'subsample': 0.935092696404644, 'colsample_bytree': 0.674271172
7752986, 'min_child_weight': 19, 'reg_lambda': 0.0014099744057476465, 'colsample_bynode': 0.44832077063804043}. Best
is trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:05:27,061] Trial 39 finished with value: 260.18341385714706 and parameters: {'n_estimators': 158, 'l
earning_rate': 0.0175057357496054, 'max_depth': 11, 'subsample': 0.903861523550917, 'colsample_bytree': 0.89552213264
51777, 'min_child_weight': 21, 'reg_lambda': 0.0032288544748396287, 'colsample_bynode': 0.5026163157403041}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:05:35,307] Trial 40 finished with value: 269.67932885133683 and parameters: {'n_estimators': 100, 'l
earning_rate': 0.06845156750393985, 'max_depth': 12, 'subsample': 0.7022328419112465, 'colsample_bytree': 0.342869552
19772436, 'min_child_weight': 20, 'reg_lambda': 0.00656804571797458, 'colsample_bynode': 0.550455754807108}. Best is
trial 7 with value: 257.10252103255976.
[I 2024-08-30 14:05:58,761] Trial 41 finished with value: 256.99783251618436 and parameters: {'n_estimators': 153, 'l
earning_rate': 0.023651230305093564, 'max_depth': 13, 'subsample': 0.6486315747658347, 'colsample_bytree': 0.97138317
```

```
02706509, 'min_child_weight': 24, 'reg_lambda': 0.0019081767751734566, 'colsample_bynode': 0.3389273169234641}. Best
is trial 41 with value: 256.99783251618436.
[I 2024-08-30 14:06:18,387] Trial 42 finished with value: 257.55563498001874 and parameters: {'n_estimators': 119, 'l
earning_rate': 0.027119689165893884, 'max_depth': 13, 'subsample': 0.6512653291650081, 'colsample_bytree': 0.94799812
99551251, 'min_child_weight': 23, 'reg_lambda': 0.0010011566463668802, 'colsample_bynode': 0.3777205709313037}. Best
is trial 41 with value: 256.99783251618436.
[I 2024-08-30 14:06:37,465] Trial 43 finished with value: 258.2356661230258 and parameters: {'n_estimators': 124, 'le
arning_rate': 0.027069385462021263, 'max_depth': 13, 'subsample': 0.6621691613361437, 'colsample_bytree': 0.953383147
2002281, 'min_child_weight': 24, 'reg_lambda': 0.001588988298965067, 'colsample_bynode': 0.3516238146329671}. Best is
trial 41 with value: 256.99783251618436.
[I 2024-08-30 14:06:57,854] Trial 44 finished with value: 257.68582783916656 and parameters: {'n_estimators': 113, 'l
earning_rate': 0.0312778883524418, 'max_depth': 14, 'subsample': 0.6354735848360525, 'colsample_bytree': 0.9290677546
843662, 'min_child_weight': 22, 'reg_lambda': 0.0021274784652843184, 'colsample_bynode': 0.3781980381334649}. Best is
trial 41 with value: 256.99783251618436.
[I 2024-08-30 14:07:17,686] Trial 45 finished with value: 258.96011324447016 and parameters: {'n_estimators': 112, 'l
earning_rate': 0.031054378256669117, 'max_depth': 14, 'subsample': 0.6348664386238736, 'colsample_bytree': 0.92400837
88741854, 'min_child_weight': 12, 'reg_lambda': 0.001883196513828617, 'colsample_bynode': 0.3001889632025653}. Best i
s trial 41 with value: 256.99783251618436.
[I 2024-08-30 14:07:35,036] Trial 46 finished with value: 257.8714166548091 and parameters: {'n_estimators': 209, 'le
arning_rate': 0.04026973792155218, 'max_depth': 14, 'subsample': 0.6803859143914904, 'colsample_bytree': 0.9676932915
26055, 'min_child_weight': 22, 'reg_lambda': 0.0030423452437288996, 'colsample_bynode': 0.374883847992543}. Best is t
rial 41 with value: 256.99783251618436.
[I 2024-08-30 14:07:54,892] Trial 47 finished with value: 287.6746404592324 and parameters: {'n_estimators': 108, 'le
arning_rate': 0.019006771088360058, 'max_depth': 13, 'subsample': 0.641385059953241, 'colsample_bytree': 0.9816744974
588253, 'min_child_weight': 24, 'reg_lambda': 0.0012264273623111479, 'colsample_bynode': 0.42494333832565}. Best is t
rial 41 with value: 256.99783251618436.
[I 2024-08-30 14:08:13,098] Trial 48 finished with value: 259.50913700331625 and parameters: {'n_estimators': 124, 'l
earning_rate': 0.0327119801939395, 'max_depth': 14, 'subsample': 0.6018828156614213, 'colsample_bytree': 0.9180245994
972568, 'min_child_weight': 23, 'reg_lambda': 0.0011337988825510681, 'colsample_bynode': 0.33130535951287354}. Best i
s trial 41 with value: 256.99783251618436.
[I 2024-08-30 14:08:34,504] Trial 49 finished with value: 257.3774131478516 and parameters: {'n_estimators': 135, 'le
arning_rate': 0.02810720724056847, 'max_depth': 13, 'subsample': 0.6263101521099023, 'colsample_bytree': 0.9369785438
984802, 'min_child_weight': 17, 'reg_lambda': 0.00202801719797907, 'colsample_bynode': 0.601998429067923}. Best is tr
ial 41 with value: 256.99783251618436.
```

In [60]:
```
# Print the best parameters
best_params_optuna = study.best_params
print(f"Best parameters found with Optuna: {best_params_optuna}")
```

```
Best parameters found with Optuna: {'n_estimators': 153, 'learning_rate': 0.023651230305093564, 'max_depth': 13, 'sub
sample': 0.6486315747658347, 'colsample_bytree': 0.9713831702706509, 'min_child_weight': 24, 'reg_lambda': 0.00190817
67751734566, 'colsample_bynode': 0.3389273169234641}
```

Model retrain with the best parameters (Still only using the last 20% of the dataset)

In [61]:
```
# Retrain the final model on the full dataset with the best parameters
final_model = xgb.XGBRegressor(**best_params_optuna)
```

Fit the model

Submission

Model re-run on full dataset

In [62]:
```
# Train on the full dataset
# final_model = xgb.XGBRegressor(**best_params_optuna)
final_model.fit(X_subsample, y_subsample, verbose=True)
```

Out[62]:

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=0.3389273169234641,
             colsample_bytree=0.9713831702706509, device=None,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, feature_types=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.023651230305093564, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=13, max_leaves=None,
             min_child_weight=24, missing=nan, monotone_constraints=None,
```

In [75]:
```
test_data_encoded.head()
```

Out[75]:

	store_nbr	family	onpromotion	city	state	cluster	dcoilwtico	type_y	weekday	year	...	sales_lag_14	sales_lag_30	sales_roll_mean_7
3054348	0	0	0	18	12	4	46.8	0	2	2017	...	4.0	2.0	0.205373
3054349	0	1	0	18	12	4	46.8	0	2	2017	...	0.0	0.0	-4.801548
3054350	0	2	2	18	12	4	46.8	0	2	2017	...	2.0	5.0	6.604357
3054351	0	3	20	18	12	4	46.8	0	2	2017	...	2645.0	2381.0	1757.715942
3054352	0	4	0	18	12	4	46.8	0	2	2017	...	0.0	1.0	-3.156069

5 rows × 25 columns

Make predictions on the test dataset

In [64]:
```
print(set(final_model.get_booster().feature_names) - set(X.columns))
```

```
set()
```

In [65]:
```
test_data_encoded = test_data_encoded[X.columns]
```

In [66]:
```
# Predict on the test data
y_test_pred = final_model.predict(test_data_encoded)
y_test_pred = np.clip(y_test_pred, 0, None)
```

In [85]:
```
plt.plot(test_data['id'],y_test_pred)
plt.xlabel("Data Point ID")
plt.ylabel("Predicted future Sales")
plt.title("Future Predictions")
```

Out[85]:
```
Text(0.5, 1.0, 'Future Predictions')
```