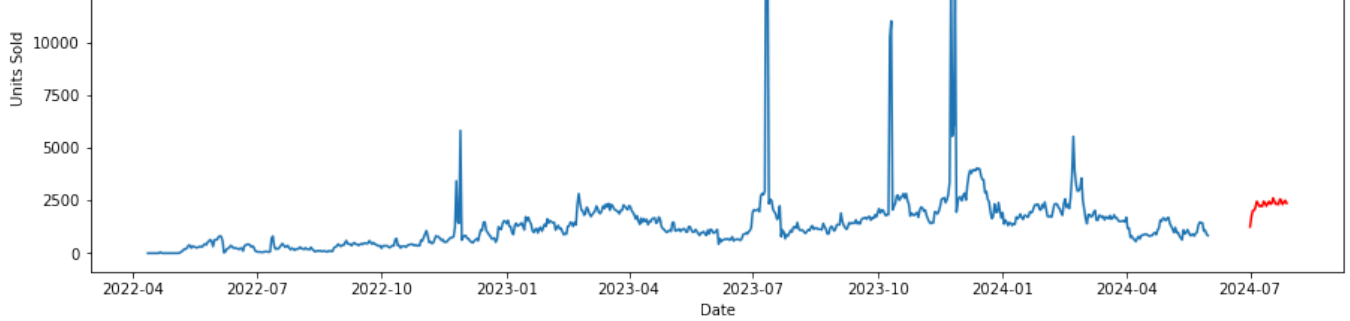
RAJUPALEM VENKATA TAGORE REDDY - 20MIC0046

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Load the dataset
train = pd.read_csv('C:\\Users\\Admin\\Downloads\\train.csv')
test = pd.read_csv('C:\\Users\\Admin\\Downloads\\test.csv')

# Display the first few rows of the train dataset
print(train.head())

# Check for missing values
print(train.isnull().sum())

# Descriptive statistics
print(train.describe())

# Handle NaN values
train.fillna(train.mean(), inplace=True)

# Check for infinite values and replace them with NaN, then fill NaN
train.replace([np.inf, -np.inf], np.nan, inplace=True)
train.fillna(train.mean(), inplace=True)

# Ensure no NaN or infinite values are present
print(train.isnull().sum())

# Ensure no values are too large for dtype('float64')
numeric_cols = train.select_dtypes(include=[np.number]).columns
print((train[numeric_cols] > np.finfo(np.float64).max).sum())

# Prepare the features and target variable
train['date'] = pd.to_datetime(train['date'])
train['year'] = train['date'].dt.year
train['month'] = train['date'].dt.month
train['day'] = train['date'].dt.day
train['day_of_week'] = train['date'].dt.dayofweek
train['ad_spend_per_unit_price'] = train['ad_spend'] / (train['unit_price'] + 1e-5)  # Adding a small value t

features = ['ad_spend', 'unit_price', 'year', 'month', 'day', 'day_of_week', 'ad_spend_per_unit_price']
X = train[features]
y = train['units']

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_val)
mse_lr = mean_squared_error(y_val, y_pred_lr)
print(f'Linear Regression MSE: {mse_lr}')

# Random Forest
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_val)
mse_rf = mean_squared_error(y_val, y_pred_rf)
print(f'Random Forest MSE: {mse_rf}')

# XGBoost
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, seed=42)
xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict(X_val)
mse_xgb = mean_squared_error(y_val, y_pred_xgb)
print(f'XGBoost MSE: {mse_xgb}')

# Hyperparameter tuning for XGBoost
param_grid = {
```

```
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'n_estimators': [100, 200, 300]
}

grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=3, scoring='neg_mean_squared_error'
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
print(f'Best parameters: {best_params}')

# Re-train the model with best parameters
best_xgb = xgb.XGBRegressor(objective='reg:squarederror', **best_params)
best_xgb.fit(X_train, y_train)
y_pred_best_xgb = best_xgb.predict(X_val)
mse_best_xgb = mean_squared_error(y_val, y_pred_best_xgb)
print(f'Tuned XGBoost MSE: {mse_best_xgb}')

# Aggregating the sales data by date
time_series_data = train.groupby('date')['units'].sum()

# Fit SARIMA model
sarima_model = SARIMAX(time_series_data, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
sarima_fit = sarima_model.fit(disp=False)

# Forecast for the test period
test_dates = pd.to_datetime(test['date']).drop_duplicates().sort_values()
sarima_forecast = sarima_fit.get_forecast(steps=len(test_dates)).predicted_mean

# Plot the forecast
plt.figure(figsize=(15, 5))
plt.plot(time_series_data, label='Observed')
plt.plot(test_dates, sarima_forecast, label='Forecast', color='red')
plt.title('SARIMA Forecast')
plt.xlabel('Date')
plt.ylabel('Units Sold')
plt.legend()
plt.show()

# Predict using the best XGBoost model
test['date'] = pd.to_datetime(test['date'])
test['year'] = test['date'].dt.year
test['month'] = test['date'].dt.month
test['day'] = test['date'].dt.day
test['day_of_week'] = test['date'].dt.dayofweek
test['ad_spend_per_unit_price'] = test['ad_spend'] / (test['unit_price'] + 1e-5)  # Adding a small value to av

test_features = test[features]
test['TARGET'] = best_xgb.predict(test_features)

# Prepare the submission file
submission = test[['date', 'Item Id', 'TARGET']]
submission_file_path = 'submission.csv'
submission.to_csv(submission_file_path, index=False)

print(f'Submission file saved to: {submission_file_path}')
print(submission.head())
```




```
Submission file saved to: submission.csv
        date     Item Id     TARGET
0 2024-07-01  B09KDR64LT   3.359508
1 2024-07-01  B09KDTS4DC   3.359508
2 2024-07-01  B09KDTHJ6V   3.359508
3 2024-07-01  B09KDQ2BWY   3.359508
4 2024-07-01  B09KDYY3SB  18.007103
```

Exploratory Data Analysis (EDA)
Purpose: To understand the dataset and identify any issues before modeling.

Loading and Displaying Data:

Theory: Load the dataset to explore its structure, including the columns and initial values. Displaying the first few rows helps in understanding the data layout and identifying potential issues.
Checking for Missing Values:

Theory: Identify missing values in the dataset to understand the extent of incomplete data. Missing values can affect model performance and need to be addressed through imputation or other techniques.
Descriptive Statistics:

Theory: Generate summary statistics to gain insights into the distribution and central tendency of numerical features. This helps in understanding the range, mean, variance, and potential outliers in the data.
Feature Engineering
Purpose: To create new features or modify existing ones to improve model performance.

Handling Missing Values:

Theory: Replace missing values with a statistical measure (e.g., mean) to ensure the model can process all data without errors. Handling missing values is crucial for maintaining data quality and model accuracy.
Handling Infinite Values:

Theory: Infinite values can arise due to mathematical operations. Replacing them with NaN and then filling them helps in preventing errors during model training and analysis.
Ensuring Data Integrity:

Theory: Check for excessively large values or other anomalies that could affect the model's performance. Ensuring the data is within acceptable ranges for its data type helps maintain the integrity of the analysis.
Date Features:

Theory: Extracting date-related features (year, month, day, day of the week) allows the model to account for temporal patterns and seasonality. These features can capture periodic trends and seasonal effects in the data.
Interaction Features:

Theory: Create new features that capture relationships between existing features (e.g., ad spend per unit price). Interaction features can provide additional context and help the model understand complex relationships in the data.
Model Selection
Purpose: To choose and evaluate different algorithms to find the best performing model.

Prepare Data for Modeling:

Theory: Define feature columns and the target variable. Splitting the data into training and validation sets allows for assessing model performance on unseen data and helps in evaluating generalization.
Linear Regression:

Theory: A statistical method to model the relationship between features and the target variable by fitting a linear equation. Linear regression is simple and provides a baseline performance metric.
Random Forest:

Theory: An ensemble learning method that builds multiple decision trees and averages their predictions to improve accuracy and reduce overfitting. Random Forest handles non-linearity and interactions between features.
XGBoost:

Theory: An advanced gradient boosting technique that builds decision trees sequentially, each one correcting errors of the previous ones. XGBoost is effective for capturing complex patterns and interactions.
Hyperparameter Tuning
Purpose: To optimize the model's hyperparameters to improve its performance.

Define Hyperparameter Grid:

Theory: Specify a range of hyperparameters to test. Hyperparameters are settings that control the learning process and model structure, such as learning rate, tree depth, and the number of trees.
Grid Search:

Theory: An exhaustive search method to find the best combination of hyperparameters by evaluating all possible combinations using cross-validation. This ensures the model is fine-tuned for optimal performance.
Train Best Model:

Theory: Retrain the model using the best hyperparameters identified from the grid search. This step ensures that the model is optimized for the dataset and provides the best possible performance.
Time Series Analysis
Purpose: To analyze and forecast data with temporal components.

Aggregate Sales Data:

Theory: Aggregate the data by date to create a time series, which helps in analyzing trends and patterns over time. Time series aggregation is essential for understanding overall sales trends.
Fit SARIMA Model:

Theory: SARIMA (Seasonal AutoRegressive Integrated Moving Average) is used for modeling time series data with seasonality and trends. It combines autoregressive, moving average, and seasonal components to capture complex time-based patterns.
Forecast:

Theory: Use the fitted SARIMA model to make predictions about future values. Forecasting involves extrapolating the identified patterns and trends to predict future data points.
Plot Forecast:

Theory: Visualize both observed and forecasted data to compare the model's predictions against actual values. Plotting helps in assessing the accuracy of the forecasts and understanding how well the model captures temporal patterns.
Final Predictions and Submission
Purpose: To make final predictions on the test data and prepare the results for submission.

Prepare Test Data:

Theory: Apply the same feature engineering steps to the test dataset to ensure consistency with the training data. Consistent feature preparation is crucial for making accurate predictions.
Make Predictions:

Theory: Use the trained model to predict the target variable for the test dataset. This step involves applying the model to new data to generate predictions.
Create Submission File:

Theory: Format the predictions into a submission file as required. The submission file typically includes relevant identifiers and the predicted values, which can be used for evaluation or competition purposes.
This approach ensures that each stage of the analysis is systematically addressed, from understanding the data to optimizing models and making predictions.#