In [1]:
```
import pandas as pd
```

In [2]:
```
# Load the dataset
file_path = r'C:\Users\zahus\Desktop\DATA science\Module 11-Dissertaion\Dataset\3-Inventory & Supply Chain-Warehouse and Retail

df = pd.read_csv(file_path)
```

In [3]:
```
# Display basic information about the dataset
df_info = df.info()
df_head = df.head()

df_info, df_head
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307645 entries, 0 to 307644
Data columns (total 9 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   YEAR              307645 non-null  int64  
 1   MONTH             307645 non-null  int64  
 2   SUPPLIER          307478 non-null  object 
 3   ITEM CODE         307645 non-null  object 
 4   ITEM DESCRIPTION  307645 non-null  object 
 5   ITEM TYPE         307644 non-null  object 
 6   RETAIL SALES      307642 non-null  float64
 7   RETAIL TRANSFERS  307645 non-null  float64
 8   WAREHOUSE SALES   307645 non-null  float64
dtypes: float64(3), int64(2), object(4)
memory usage: 21.1+ MB
```

Out[3]: (None,

	YEAR	MONTH	SUPPLIER	ITEM CODE	ITEM DESCRIPTION	ITEM TYPE	RETAIL SALES	RETAIL TRANSFERS	WAREHOUSE SALES
0	2020	1	REPUBLIC NATIONAL DISTRIBUTING CO	100009	BOOTLEG RED - 750ML	WINE	0.00	0.0	2.0
1	2020	1	PWSWN INC	100024	MOMENT DE PLAISIR - 750ML	WINE	0.00	1.0	4.0
2	2020	1	RELIABLE CHURCHILL LLLP	1001	S SMITH ORGANIC PEAR CIDER - 18.7OZ	BEER	0.00	0.0	1.0
3	2020	1	LANTERNA DISTRIBUTORS INC	100145	SCHLINK HAUS KABINETT - 750ML	WINE	0.00	0.0	1.0
4	2020	1	DIONYSOS IMPORTS INC	100293	SANTORINI GAVALA WHITE - 750ML	WINE	0.82	0.0	0.0

)

Data cleaning and preprocessing

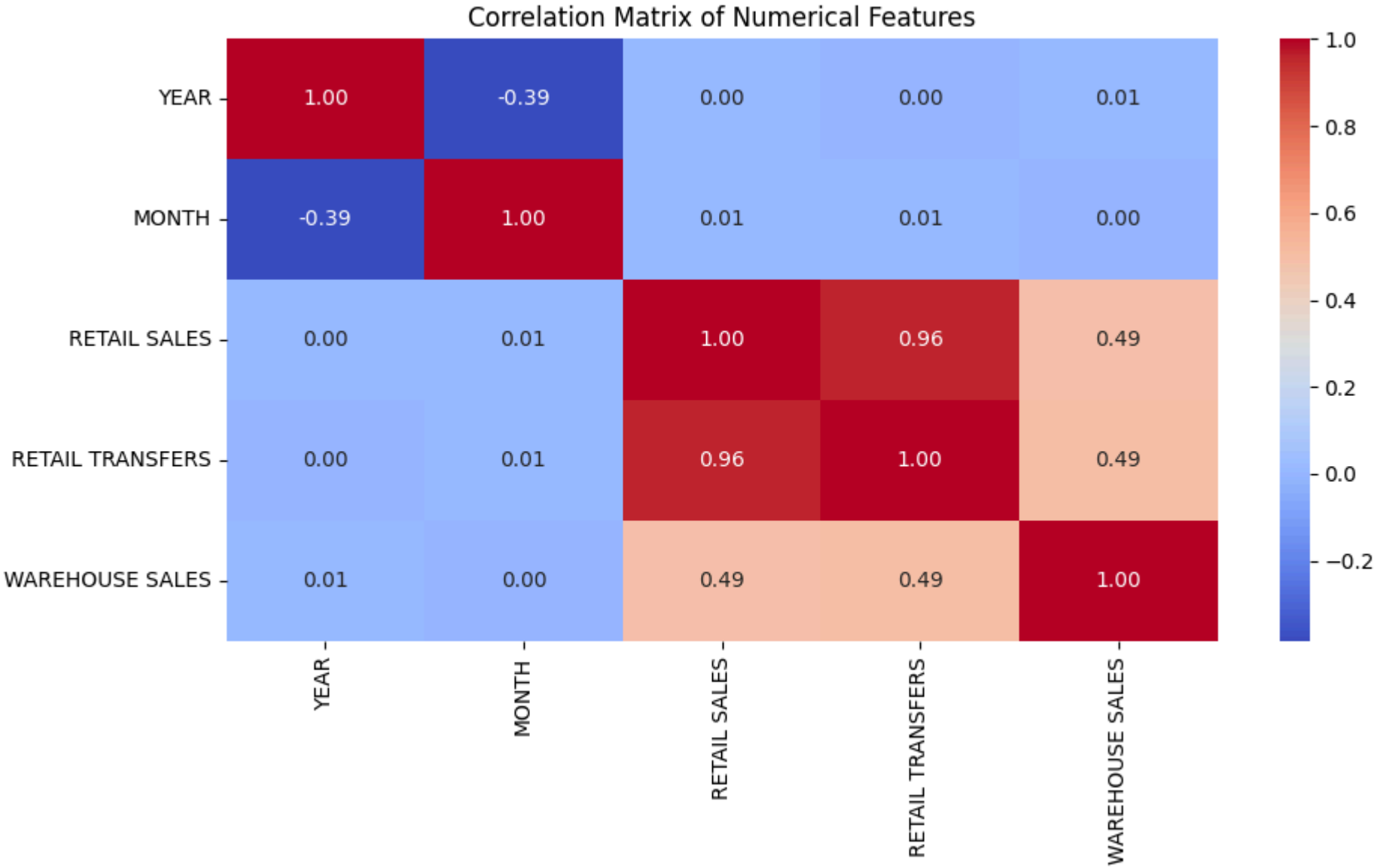
In [4]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
# Handle missing values
df_clean = df.copy()
df_clean['SUPPLIER'].fillna('UNKNOWN', inplace=True)
df_clean['ITEM TYPE'].fillna('UNKNOWN', inplace=True)
df_clean['RETAIL SALES'].fillna(0, inplace=True)
```

In [6]:
```
# Convert month and year into a datetime index for time series
df_clean['DATE'] = pd.to_datetime(df_clean[['YEAR', 'MONTH']].assign(DAY=1))
```

In [7]:
```
# Drop unnecessary columns for numerical analysis
df_numeric = df_clean.drop(columns=['SUPPLIER', 'ITEM CODE', 'ITEM DESCRIPTION', 'ITEM TYPE'])
```

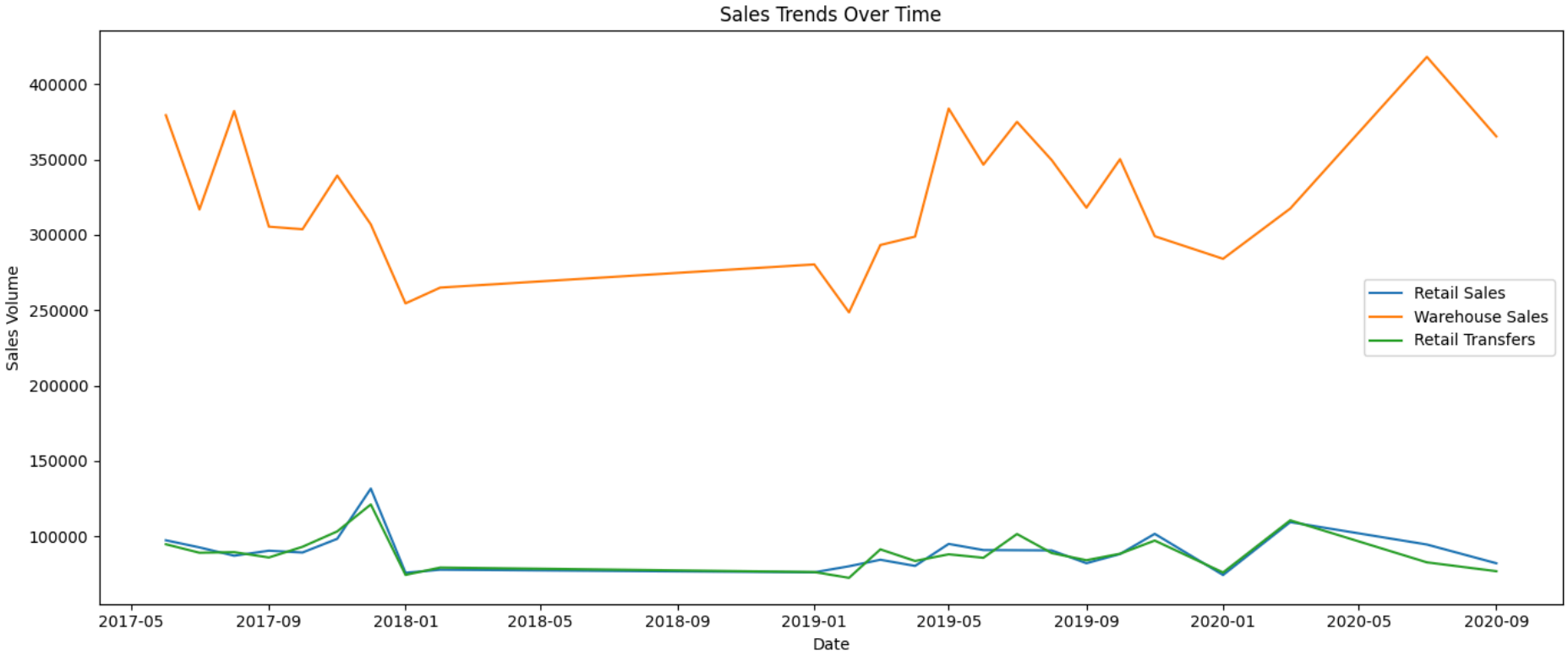
In [8]:
```
# Correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df_numeric.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix of Numerical Features")
plt.tight_layout()
plt.show()
```



Exploratory Data Analysis (EDA) for sales trends over time & Encoding categorical features (for models like XGBoost)

In [9]:
```
# 1. Exploratory Data Analysis (EDA) - Aggregate sales over time
sales_over_time = df_clean.groupby('DATE')[['RETAIL SALES', 'WAREHOUSE SALES', 'RETAIL TRANSFERS']].sum().reset_index()
```

In [10]:
```
# Plotting sales trends
plt.figure(figsize=(14, 6))
plt.plot(sales_over_time['DATE'], sales_over_time['RETAIL SALES'], label='Retail Sales')
plt.plot(sales_over_time['DATE'], sales_over_time['WAREHOUSE SALES'], label='Warehouse Sales')
plt.plot(sales_over_time['DATE'], sales_over_time['RETAIL TRANSFERS'], label='Retail Transfers')
plt.title("Sales Trends Over Time")
plt.xlabel("Date")
plt.ylabel("Sales Volume")
plt.legend()
plt.tight_layout()
plt.show()
```



In [11]:
```
# 2. Encoding categorical features
df_encoded = df_clean.copy()
df_encoded['SUPPLIER'] = df_encoded['SUPPLIER'].astype('category').cat.codes
df_encoded['ITEM CODE'] = df_encoded['ITEM CODE'].astype('category').cat.codes
df_encoded['ITEM TYPE'] = df_encoded['ITEM TYPE'].astype('category').cat.codes
df_encoded['ITEM DESCRIPTION'] = df_encoded['ITEM DESCRIPTION'].astype('category').cat.codes
```

In [12]:
```
# Select features and target variable
features = ['YEAR', 'MONTH', 'SUPPLIER', 'ITEM CODE', 'ITEM TYPE', 'RETAIL TRANSFERS', 'WAREHOUSE SALES']
target = 'RETAIL SALES'
```

In [13]:
```
X = df_encoded[features]
y = df_encoded[target]

X.head(), y.head()
```

Out[13]:
```
(   YEAR  MONTH  SUPPLIER  ITEM CODE  ITEM TYPE  RETAIL TRANSFERS  \
 0  2020      1       273          3          8               0.0
 1  2020      1       264          8          8               1.0
 2  2020      1       271         11          0               0.0
 3  2020      1       186         13          8               0.0
 4  2020      1        91         20          8               0.0

    WAREHOUSE SALES
 0              2.0
 1              4.0
 2              1.0
 3              1.0
 4              0.0  ,
 0    0.00
 1    0.00
 2    0.00
 3    0.00
 4    0.82
 Name: RETAIL SALES, dtype: float64)
```

Model Building and Comparison.

Linear Regression

Decision Tree Regressor

XGBoost Regressor

Random Forest Regressor

Train each model and evaluate using:

RMSE (Root Mean Squared Error)

MAE (Mean Absolute Error)

R^2 Score (coefficient of determination)

In [14]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
import numpy as np
```

In [15]:
```
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [16]:
```
# Dictionary to store model performance
model_performance = {}
```

In [17]:
```
# Helper function to evaluate models
def evaluate_model(name, model, X_test, y_test):
    y_pred = model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    model_performance[name] = {'RMSE': rmse, 'MAE': mae, 'R2': r2}
    return f"{name} -> RMSE: {rmse:.2f}, MAE: {mae:.2f}, R2: {r2:.4f}"
```

In [18]:
```
# 1. Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
lr_result = evaluate_model("Linear Regression", lr, X_test, y_test)
```

In [19]:
```
# 2. Decision Tree Regressor
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)
dt_result = evaluate_model("Decision Tree", dt, X_test, y_test)
```

In [20]:
```
# 3. Random Forest Regressor
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
rf_result = evaluate_model("Random Forest", rf, X_test, y_test)
```

In [21]:
```
# 4. XGBoost Regressor
xgbr = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, seed=42)
xgbr.fit(X_train, y_train)
xgb_result = evaluate_model("XGBoost", xgbr, X_test, y_test)
```

In [22]:
```
# Display results
lr_result, dt_result, rf_result, xgb_result, model_performance
```

Out[22]:
```
('Linear Regression -> RMSE: 6.94, MAE: 1.71, R2: 0.9528',
 'Decision Tree -> RMSE: 10.00, MAE: 2.18, R2: 0.9018',
 'Random Forest -> RMSE: 7.93, MAE: 1.62, R2: 0.9383',
 'XGBoost -> RMSE: 5.79, MAE: 1.59, R2: 0.9670',
 {'Linear Regression': {'RMSE': 6.935680377156881,
   'MAE': 1.7068267197474474,
   'R2': 0.9527537225506649},
  'Decision Tree': {'RMSE': 9.999537181752054,
   'MAE': 2.1780193079686003,
   'R2': 0.9017914603385995},
  'Random Forest': {'RMSE': 7.927966534340296,
   'MAE': 1.6164434738090983,
   'R2': 0.9382676129444419},
  'XGBoost': {'RMSE': 5.792527377013711,
   'MAE': 1.5928745563842177,
   'R2': 0.9670446715686284}})
```

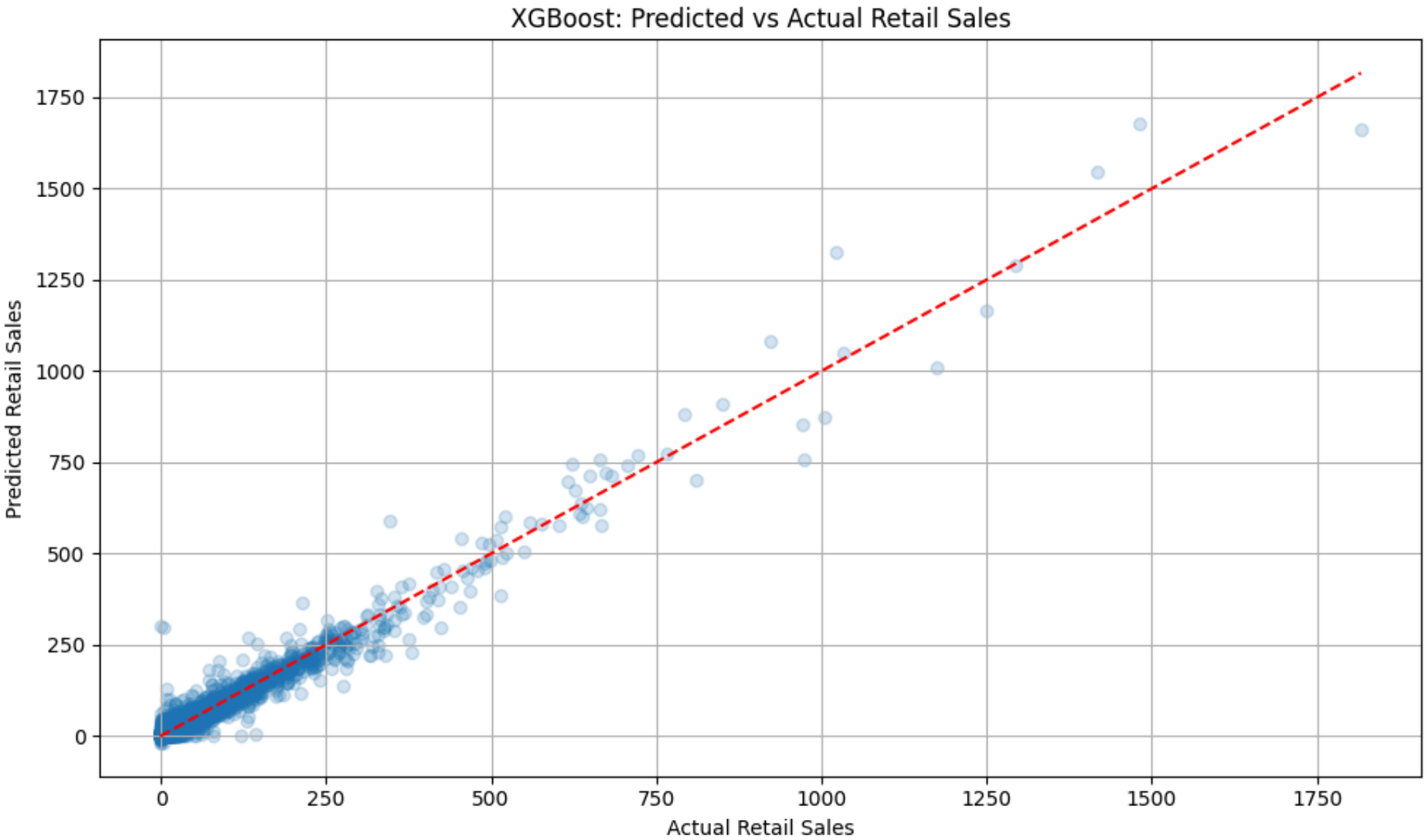
Generate plots comparing predictions vs actuals?

Proceed to LSTM or Transformer-based models?

Export the code for your dissertation appendix?

In [23]:
```
# Generate predictions with best model (XGBoost)
y_pred_xgb = xgbr.predict(X_test)
```

In [24]:
```
# Plot predicted vs actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_xgb, alpha=0.2)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--')
plt.title("XGBoost: Predicted vs Actual Retail Sales")
plt.xlabel("Actual Retail Sales")
plt.ylabel("Predicted Retail Sales")
plt.grid(True)
plt.tight_layout()
plt.show()
```



Proceed to LSTM or Transformer-based models

In [27]:
```
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
```

In [28]:
```
# Data cleaning
df['RETAIL SALES'].fillna(0, inplace=True)
df['RETAIL TRANSFERS'].fillna(0, inplace=True)
df['WAREHOUSE SALES'].fillna(0, inplace=True)
df['DATE'] = pd.to_datetime(df[['YEAR', 'MONTH']].assign(DAY=1))
```

In [29]:
```
# Step 1: Aggregate data monthly for time series
monthly_sales = df.groupby('DATE')[['RETAIL SALES', 'WAREHOUSE SALES', 'RETAIL TRANSFERS']].sum()
```

In [30]:
```
# Step 2: Normalize data
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(monthly_sales)
```

In [31]:
```
# Step 3: Prepare sequences for LSTM
def create_sequences(data, sequence_length=12):
    X, y = [], []
    for i in range(len(data) - sequence_length):
        X.append(data[i:i + sequence_length])
        y.append(data[i + sequence_length, 0])  # Predicting 'RETAIL SALES'
    return np.array(X), np.array(y)

sequence_length = 12
X_lstm, y_lstm = create_sequences(scaled_data, sequence_length)
```

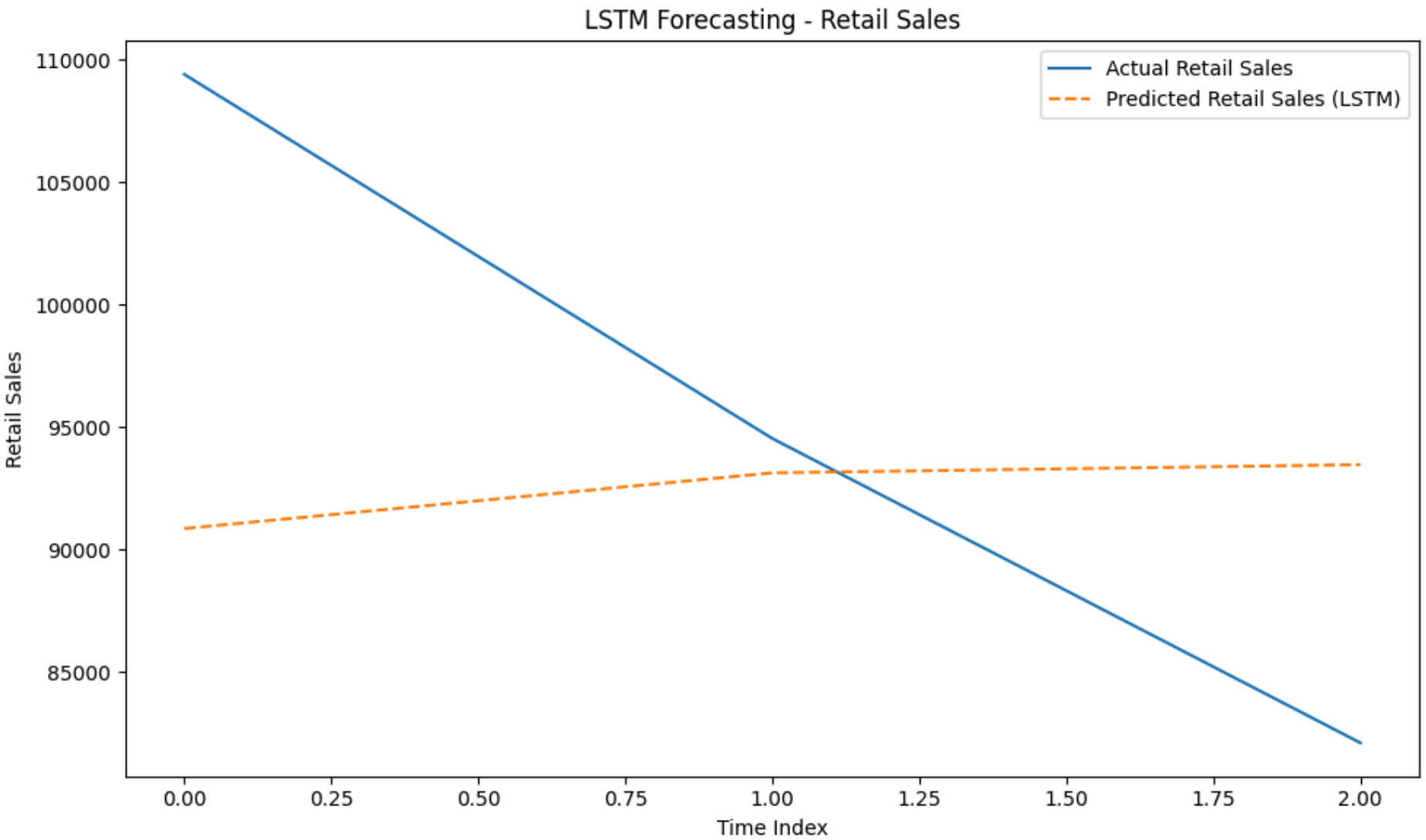
In [32]:
```
# Step 4: Train/test split for LSTM
split = int(0.8 * len(X_lstm))
X_train_lstm, X_test_lstm = X_lstm[:split], X_lstm[split:]
y_train_lstm, y_test_lstm = y_lstm[:split], y_lstm[split:]
```

In [33]:
```
# Step 5: Build LSTM model
model = Sequential([
    LSTM(50, activation='relu', input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])),
    Dense(1)
])
```

```
model.compile(optimizer='adam', loss='mse')
history = model.fit(X_train_lstm, y_train_lstm, epochs=20, validation_data=(X_test_lstm, y_test_lstm), verbose=0)
```

In [34]:
```
# Step 6: Predictions and inverse transform
y_pred_lstm = model.predict(X_test_lstm)
y_test_actual = scaler.inverse_transform(np.concatenate([y_test_lstm.reshape(-1, 1),
                                                         X_test_lstm[:, -1, 1:]], axis=1))[:, 0]
y_pred_actual = scaler.inverse_transform(np.concatenate([y_pred_lstm,
                                                         X_test_lstm[:, -1, 1:]], axis=1))[:, 0]
```

In [35]:
```
# Step 7: Plot predicted vs actual (LSTM)
plt.figure(figsize=(10, 6))
plt.plot(y_test_actual, label='Actual Retail Sales')
plt.plot(y_pred_actual, label='Predicted Retail Sales (LSTM)', linestyle='--')
plt.title("LSTM Forecasting - Retail Sales")
plt.xlabel("Time Index")
plt.ylabel("Retail Sales")
plt.legend()
plt.tight_layout()
plt.show()
```



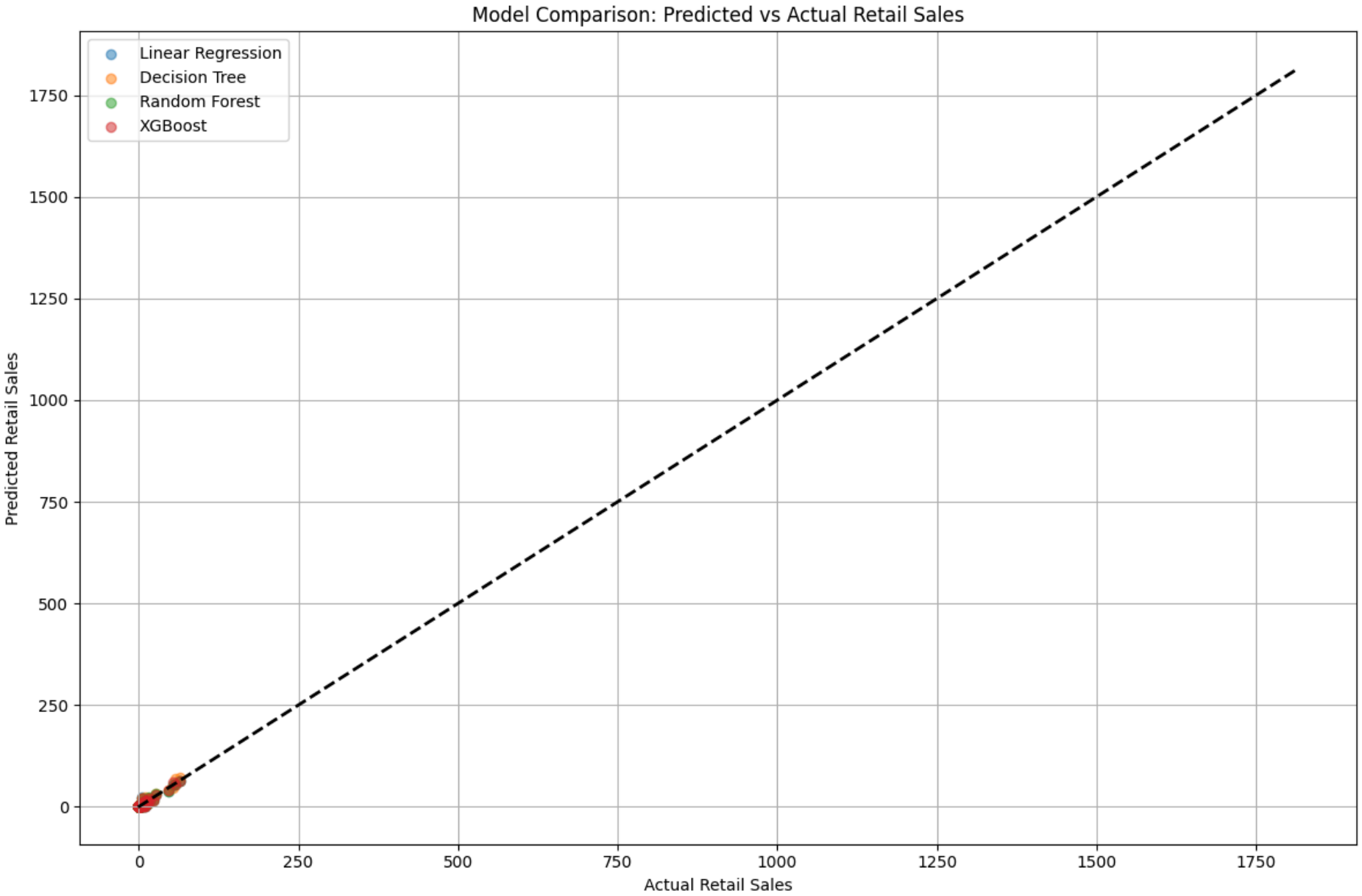
All models predictions vs actual retail sales

In [42]:
```
# Step 1: Train models and store predictions
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
```

```
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb

models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(random_state=42),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'XGBoost': xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, seed=42)
}

predictions = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    predictions[name] = y_pred
```

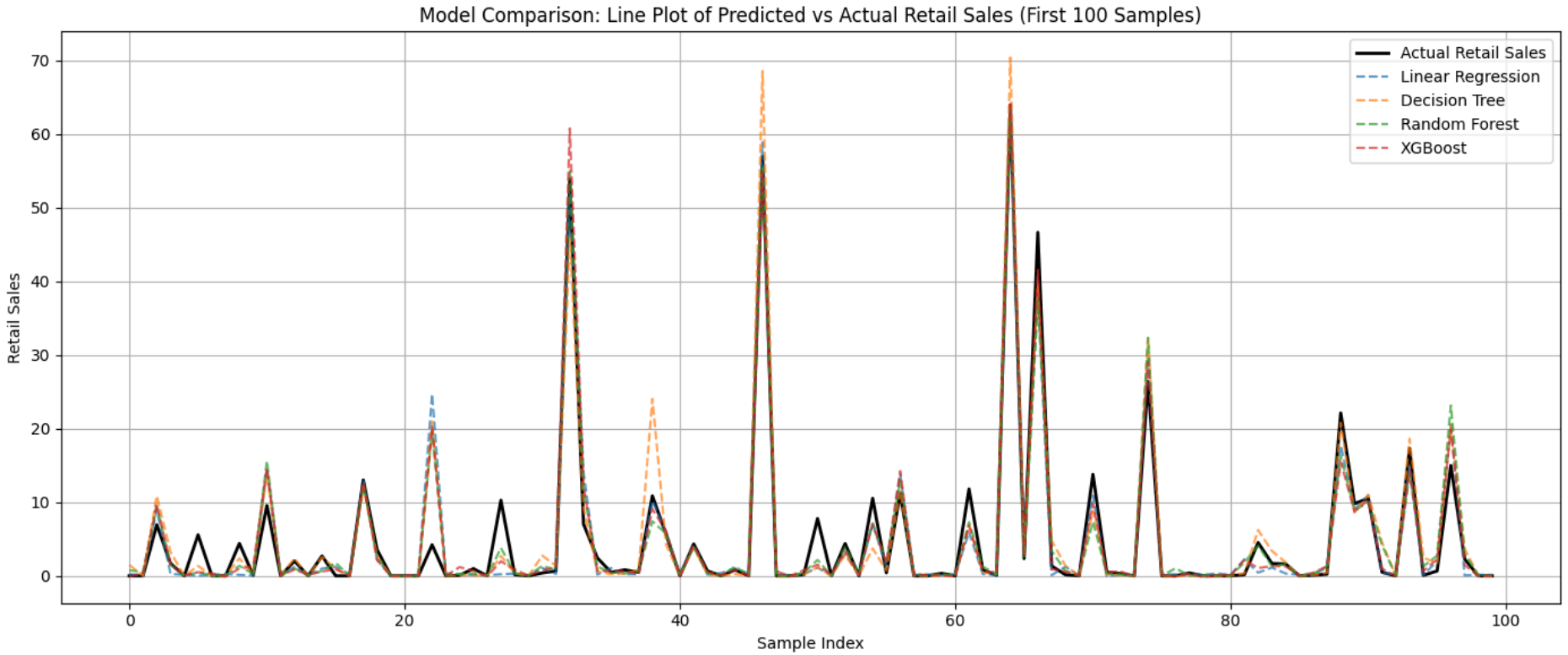
In [43]:

```
# Step 2: Visual comparison plot
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
for name, y_pred in predictions.items():
    plt.scatter(y_test[:100], y_pred[:100], label=name, alpha=0.5)

plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.xlabel("Actual Retail Sales")
plt.ylabel("Predicted Retail Sales")
plt.title("Model Comparison: Predicted vs Actual Retail Sales")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

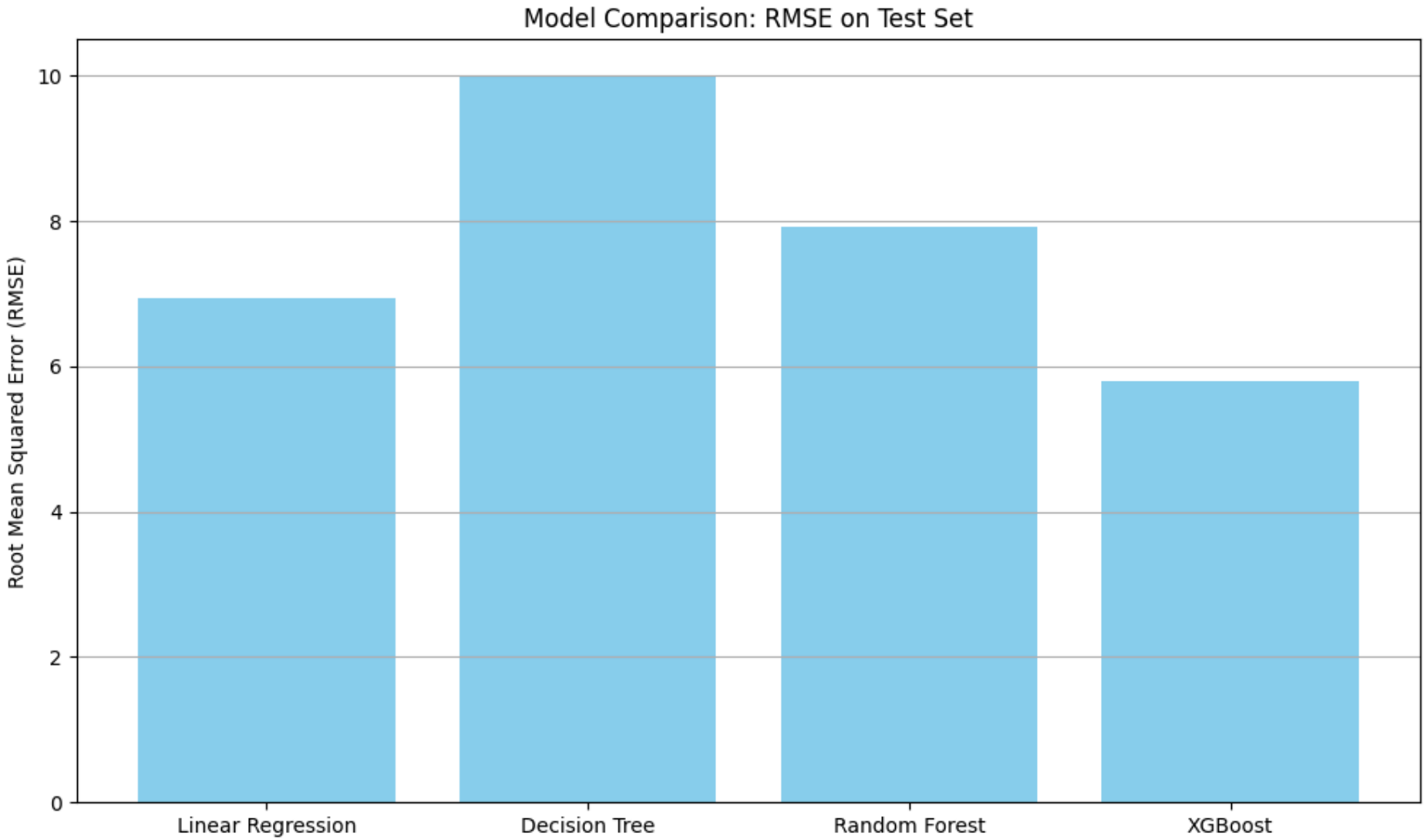


In [44]:
```
import matplotlib.pyplot as plt

# Take first 100 actual values
```

```
actual = y_test[:100].values

plt.figure(figsize=(14, 6))
plt.plot(actual, label='Actual Retail Sales', color='black', linewidth=2)

for name, y_pred in predictions.items():
    plt.plot(y_pred[:100], label=name, linestyle='--', alpha=0.7)

plt.title("Model Comparison: Line Plot of Predicted vs Actual Retail Sales (First 100 Samples)")
plt.xlabel("Sample Index")
plt.ylabel("Retail Sales")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

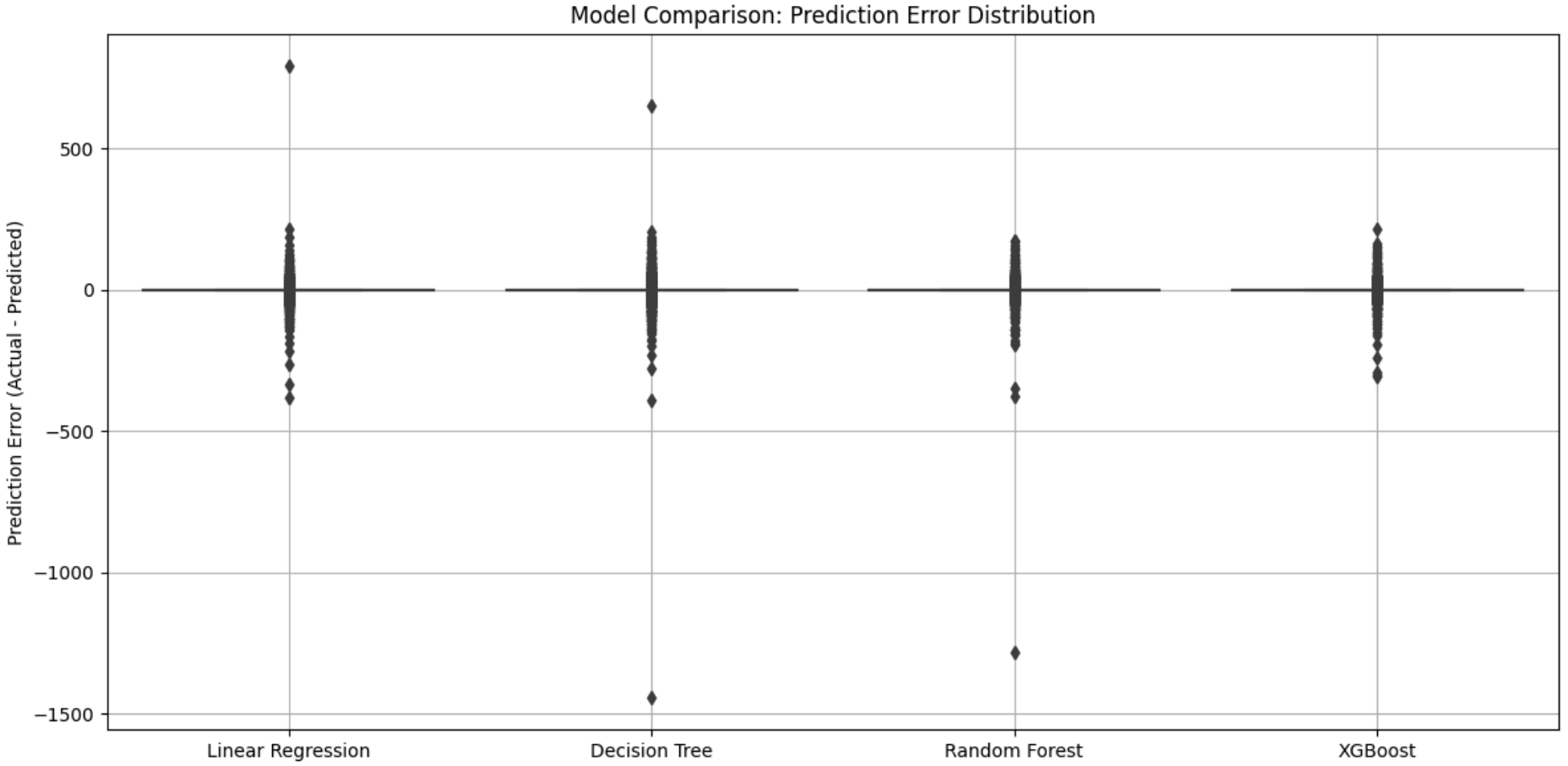


In [45]:
```
from sklearn.metrics import mean_squared_error
import numpy as np

rmse_scores = {name: np.sqrt(mean_squared_error(y_test, pred)) for name, pred in predictions.items()}

plt.figure(figsize=(10, 6))
plt.bar(rmse_scores.keys(), rmse_scores.values(), color='skyblue')
plt.title("Model Comparison: RMSE on Test Set")
plt.ylabel("Root Mean Squared Error (RMSE)")
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



In [46]:
```
import seaborn as sns

# Calculate errors
errors_df = pd.DataFrame({name: y_test.values - pred for name, pred in predictions.items()})

plt.figure(figsize=(12, 6))
sns.boxplot(data=errors_df)
```

```
plt.title("Model Comparison: Prediction Error Distribution")
plt.ylabel("Prediction Error (Actual - Predicted)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [ ]: