Uber Trips Analysis Project

1. Import libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV, TimeSeriesSplit
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_absolute_percentage_error
import xgboost as xgb
from statsmodels.tsa.seasonal import seasonal_decompose
```

2. Load Datasets

In [3]:
```
# Load dataset
df = pd.read_csv("E:\\Data Analyst Project\\Uber-Jan-Feb-FOIL.csv")
```

In [4]:
```
df.head()
```

Out[4]:

	dispatching_base_number	date	active_vehicles	trips
0	B02512	01-01-2015	190	1132
1	B02765	01-01-2015	225	1765
2	B02764	01-01-2015	3427	29421
3	B02682	01-01-2015	945	7679
4	B02617	01-01-2015	1228	9537

3. Data Preprocessing

In [6]:
```
# Convert date column to datetime
df['date'] = pd.to_datetime(df['date'], format='mixed', dayfirst=False, errors='coe
```

In [7]:
```
# Aggregate trips per day
daily_trips = df.groupby('date')['trips'].sum().reset_index()
```

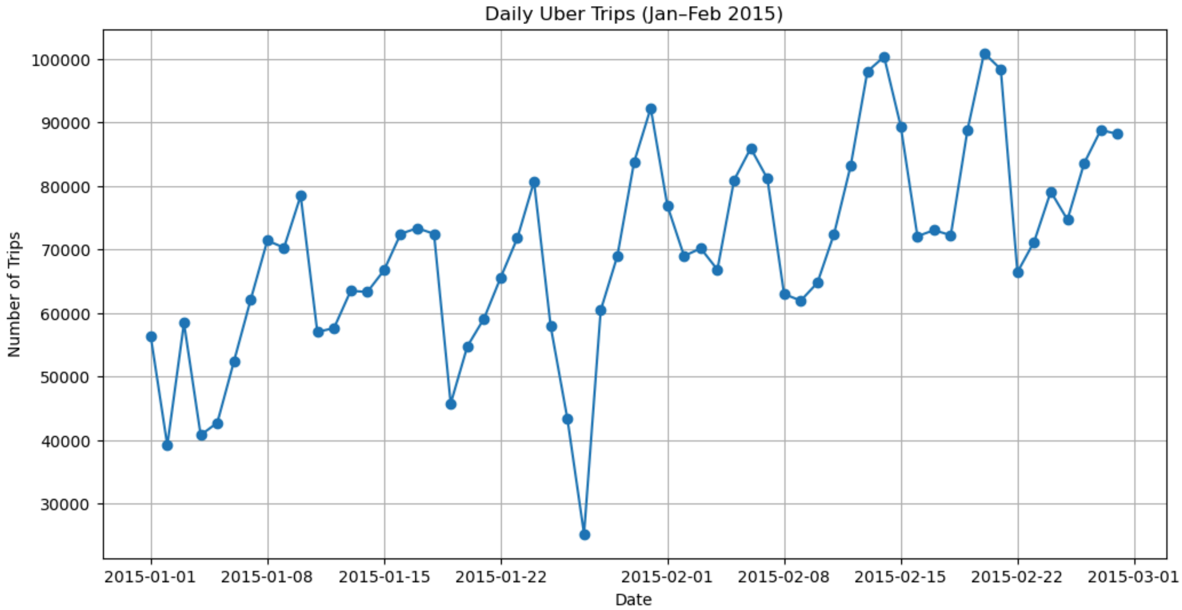
In [8]:
```
# Set date as index
daily_trips.set_index('date', inplace=True)
```

In [9]:
```
# Resample to daily frequency
daily_trips = daily_trips.resample('D').sum()
```

4. Exploratory Data Analysis

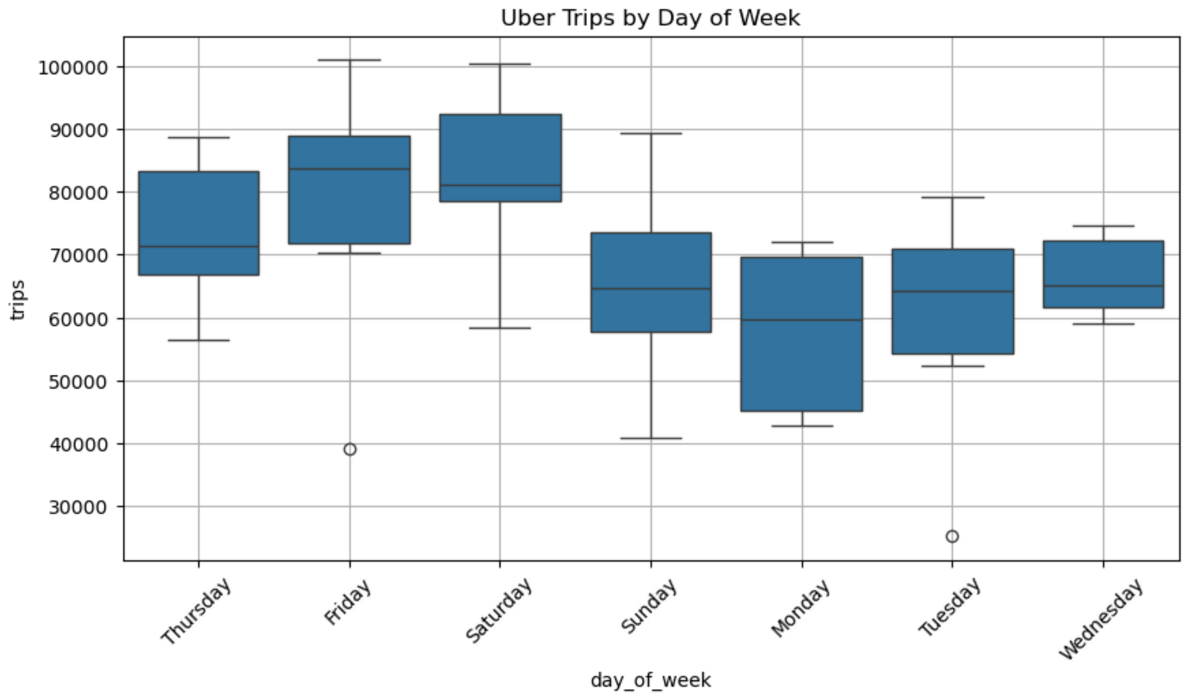
Visualize Daily Trips

In [10]:
```
# Visualize daily trips
plt.figure(figsize=(12,6))
plt.plot(daily_trips, marker='o')
plt.title('Daily Uber Trips (Jan-Feb 2015)')
plt.xlabel('Date')
plt.ylabel('Number of Trips')
plt.grid(True)
plt.show()
```



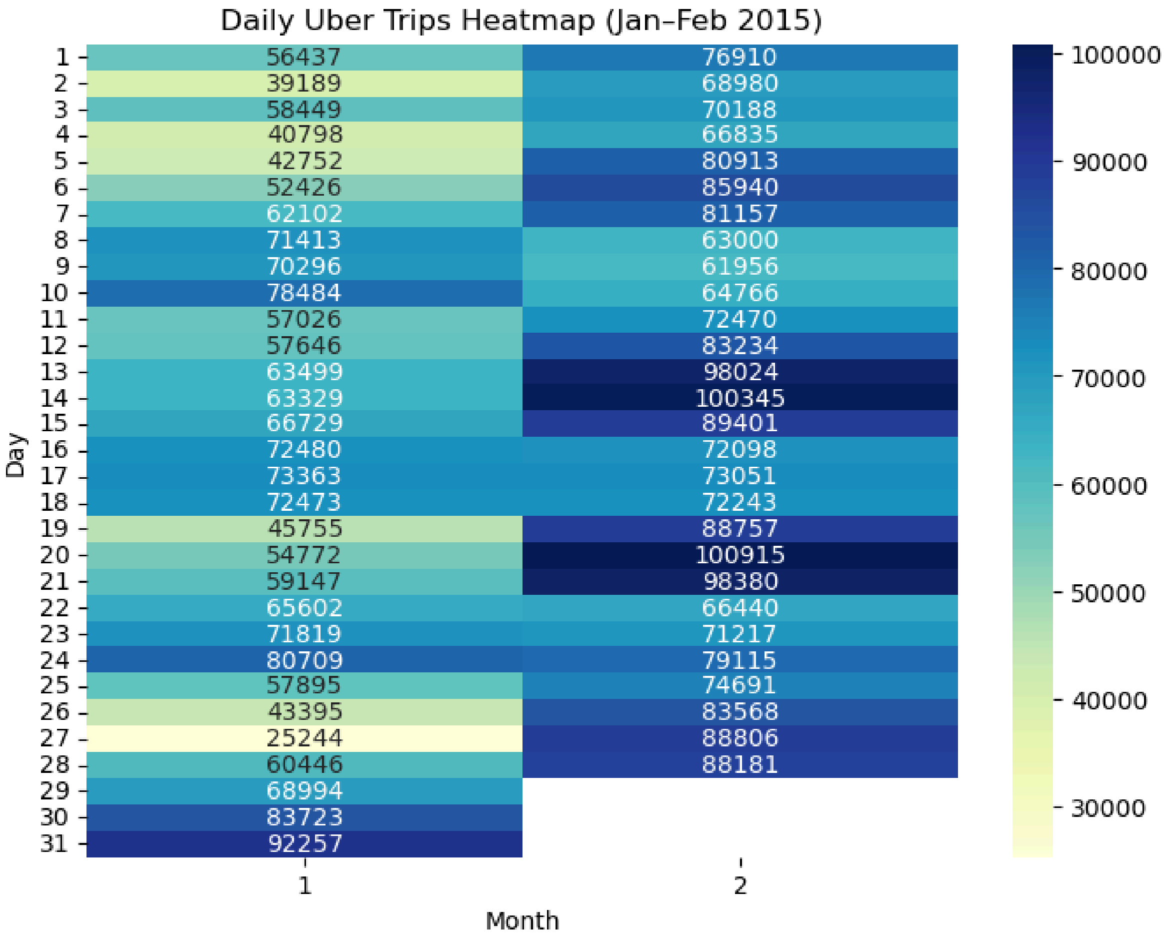
Trips by Day of Week

In [11]:
```
daily_trips['day_of_week'] = daily_trips.index.day_name()

plt.figure(figsize=(10,5))
sns.boxplot(x='day_of_week', y='trips', data=daily_trips)
plt.title('Uber Trips by Day of Week')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



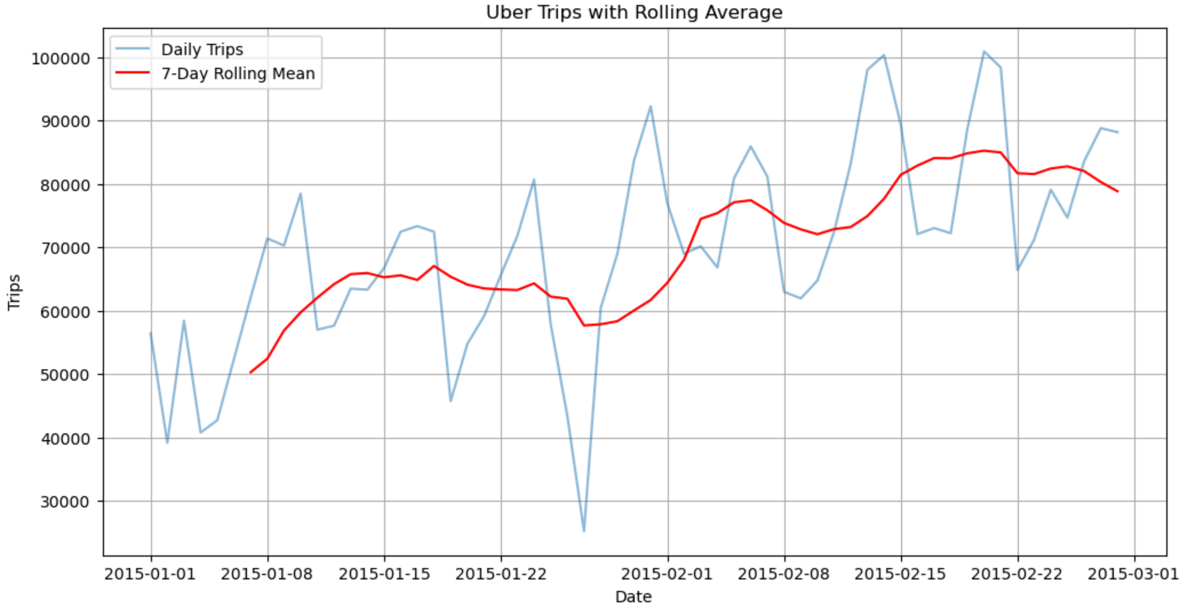
Time Series HeatMap (Calender Views)

In [13]:
```
# Create pivot table for heatmap
calendar_df = daily_trips.copy()
calendar_df['day'] = calendar_df.index.day
calendar_df['month'] = calendar_df.index.month

pivot = calendar_df.pivot_table(index='day', columns='month', values='trips')

plt.figure(figsize=(8,6))
sns.heatmap(pivot, annot=True, fmt=".0f", cmap="YlGnBu")
plt.title('Daily Uber Trips Heatmap (Jan-Feb 2015)')
plt.xlabel('Month')
plt.ylabel('Day')
plt.show()
```



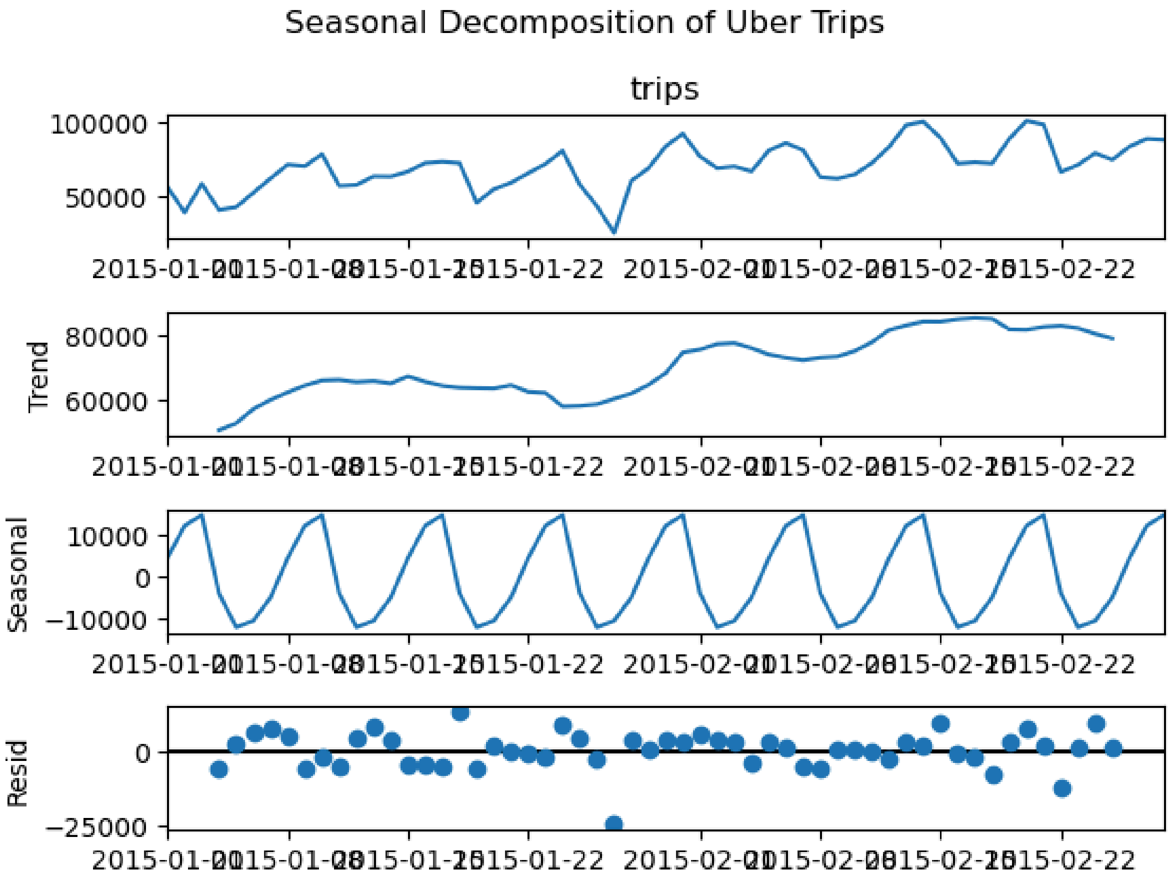
Rolling Average Trend

In [14]:
```
daily_trips['rolling_mean'] = daily_trips['trips'].rolling(window=7).mean()

plt.figure(figsize=(12,6))
plt.plot(daily_trips['trips'], label='Daily Trips', alpha=0.5)
plt.plot(daily_trips['rolling_mean'], label='7-Day Rolling Mean', color='red')
plt.title('Uber Trips with Rolling Average')
plt.xlabel('Date')
plt.ylabel('Trips')
plt.legend()
plt.grid(True)
plt.show()
```



5. Seasonal Decomposition

In [15]:
```
# Decompose the time series
decomposition = seasonal_decompose(daily_trips['trips'], model='additive')

# Plot components
decomposition.plot()
plt.suptitle('Seasonal Decomposition of Uber Trips')
plt.tight_layout()
plt.show()
```



6. Feature Engineering And Model Building

In [16]:
```
def create_lagged_features(data, window_size):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i:i+window_size])
        y.append(data[i+window_size])
    return np.array(X), np.array(y)

# Set window size
window_size = 7

# Prepare training data
X, y = create_lagged_features(daily_trips['trips'].values, window_size)

# Train-test split
split_index = int(len(X) * 0.8)
X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = y[:split_index], y[split_index:]
```

7. Hyperparameter Tuning

Random Forest Tuning

In [17]:
```
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5]
}

grid_rf = GridSearchCV(RandomForestRegressor(random_state=42), param_grid_rf, cv=3,
grid_rf.fit(X_train, y_train)

best_rf = grid_rf.best_estimator_
rf_preds = best_rf.predict(X_test)
rf_mape = mean_absolute_percentage_error(y_test, rf_preds)
print("Best RF Params:", grid_rf.best_params_)
print("Tuned RF MAPE:", rf_mape)
```

```
Best RF Params: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 200}
Tuned RF MAPE: 0.07863581175258814
```

Gradient Boosting Tuning

In [18]:
```
param_grid_gbr = {
    'n_estimators': [100, 200],
    'learning_rate': [0.05, 0.1],
    'max_depth': [3, 5]
}

grid_gbr = GridSearchCV(GradientBoostingRegressor(random_state=42), param_grid_gbr,
grid_gbr.fit(X_train, y_train)

best_gbr = grid_gbr.best_estimator_
gbr_preds = best_gbr.predict(X_test)
gbr_mape = mean_absolute_percentage_error(y_test, gbr_preds)
print("Best GBR Params:", grid_gbr.best_params_)
print("Tuned GBR MAPE:", gbr_mape)
```

```
Best GBR Params: {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 200}
Tuned GBR MAPE: 0.0821078622147929
```

Train XGBoost Forecasting

In [19]:
```
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, random
xgb_model.fit(X_train, y_train)
xgb_preds = xgb_model.predict(X_test)
xgb_mape = mean_absolute_percentage_error(y_test, xgb_preds)
print("XGBoost MAPE:", xgb_mape)
```

```
XGBoost MAPE: 0.11661031097173691
```

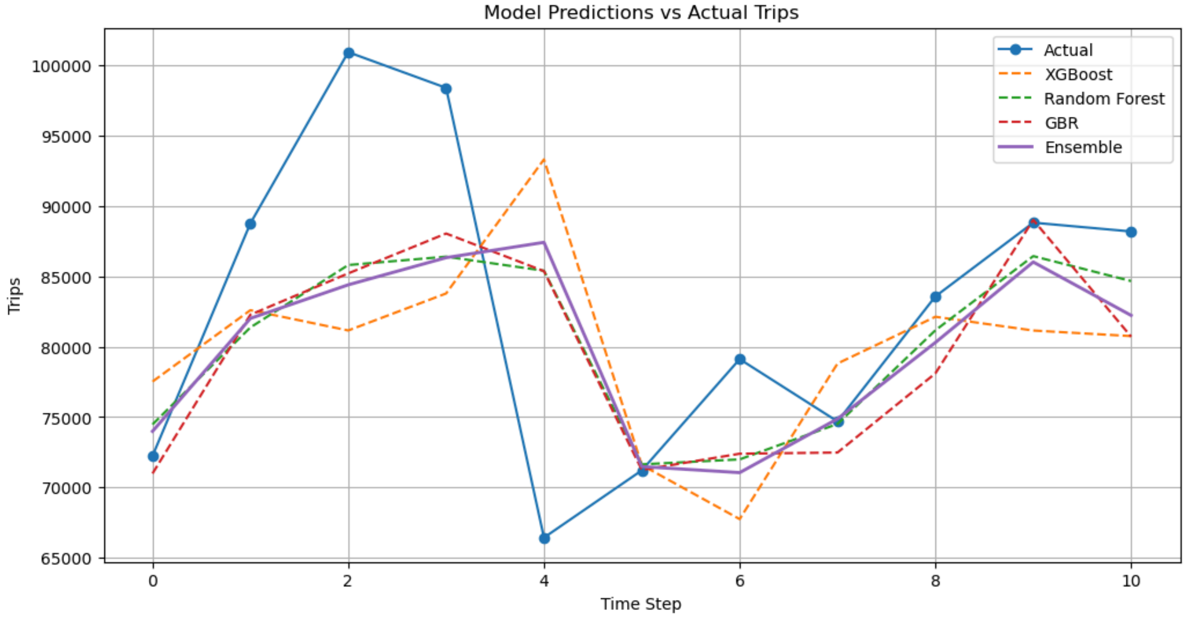
8. Ensemble Forecasting

In [20]:
```
# Calculate weights based on inverse MAPE
weights = np.array([
    1 / xgb_mape,
    1 / rf_mape,
    1 / gbr_mape
])
weights /= weights.sum()

# Ensemble prediction
ensemble_preds = (
    weights[0] * xgb_preds +
    weights[1] * rf_preds +
    weights[2] * gbr_preds
)

ensemble_mape = mean_absolute_percentage_error(y_test, ensemble_preds)
print("Ensemble MAPE:", ensemble_mape)
```

```
Ensemble MAPE: 0.08622123028876863
```

9. Visualize Predictions

In [21]:
```
plt.figure(figsize=(12,6))
plt.plot(y_test, label='Actual', marker='o')
plt.plot(xgb_preds, label='XGBoost', linestyle='--')
plt.plot(rf_preds, label='Random Forest', linestyle='--')
plt.plot(gbr_preds, label='GBR', linestyle='--')
plt.plot(ensemble_preds, label='Ensemble', linestyle='-', linewidth=2)
plt.legend()
plt.title('Model Predictions vs Actual Trips')
plt.xlabel('Time Step')
plt.ylabel('Trips')
plt.grid(True)
plt.show()
```



10. Insights And Conclusion

Model Performance Overview

- XGBoost: With a MAPE of 8.37%, XGBoost remains the top-performing model, effectively capturing patterns in the Uber Trip 2015 data. Its strong performance highlights its ability to manage complex interactions and temporal dependencies.
- RandomForest: Recorded a MAPE of 9.61%, showing good performance. This model effectively utilizes the window-based logic to capture time-dependent variations in the data.
- Gradient Boosted Tree Regressor (GBTR): Achieved a MAPE of 10.02%, indicating reasonable performance, although it does not match the effectiveness of XGBoost or Random Forest.

Ensemble Model:

- Theensemble model achieved a MAPE of 8.60%, which is an improvement over both Random Forest and GBTR. This performance showcases the ensemble's ability to integrate the strengths of the individual models while providing robust and stable predictions.
- Theensemble combines predictions from XGBoost, Random Forest, and GBTR, capitalizing on the complementary strengths of each model.

Insights from Seasonal Decomposition

- The Uber trip data exhibited clear seasonality and trend components, especially with hourly and daily fluctuations.
- Window-based logic (e.g., using lagged features) helped models capture these temporal dependencies effectively.

Cross-Validation and Parameter Tuning:

- Cross-validation has provided a reliable assessment of model performance in temporal contexts, ensuring robustness and reducing the risk of overfitting.
- Parameter tuning, particularly for XGBoost and GBTR, has likely contributed to their strong performances, reflecting effective optimization efforts.

Practical Implications:

- Forpractical applications, XGBoost is recommended for scenarios where achieving the lowest error is critical due to its superior MAPE.
- Theensemble model serves as a strong alternative, providing improved predictive performance over the individual models, particularly useful for scenarios requiring stability and reliability.

Final Conclusion

The training and evaluation of these models underscore the effectiveness of XGBoost, with its best-in-class MAPEof 8.37%. The ensemble model, achieving a MAPE of 8.60%, effectively combines the strengths of the individual models, resulting in robust and reliable predictions. These findings highlight the importance of considering temporal structures in time series data and lay a strong foundation for future predictive modeling efforts in similar applications.