In [11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error
from statsmodels.tsa.stattools import adfuller
from prophet import Prophet
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import LSTM, Dense, Input, MultiHeadAttention, LayerNo
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from pmdarima import auto_arima

# Step 1: Load the dataset (assuming the time series is already prepared from your
df_timeseries = pd.read_excel("D:/capstone/datasets/Affinity - State - Daily.xlsx")

# Create a proper datetime column if needed
if 'year' in df_timeseries.columns and 'month' in df_timeseries.columns and 'day' i
    df_timeseries['date'] = pd.to_datetime(df_timeseries[['year', 'month', 'day']])

# Process data similar to your previous code if needed
# (Assuming the data is already preprocessed with a 'date' and 'spend_all' column)
# Step 4: Replace '.' with NaN (missing values)
# Step 3: Identify all spend-related columns
spend_columns = [col for col in df_timeseries.columns if 'spend' in col]
df_timeseries[spend_columns] = df_timeseries[spend_columns].replace('.', pd.NA)
# Step 5: Convert all spend columns to numeric
df_timeseries[spend_columns] = df_timeseries[spend_columns].apply(pd.to_numeric, er
# Step 6: Interpolate missing values (best-performing imputation method)
# Interpolation first
df_timeseries[spend_columns] = df_timeseries[spend_columns].interpolate()
df_timeseries
# Then forward fill to handle start-of-series gaps
df_timeseries[spend_columns] = df_timeseries[spend_columns].fillna(method='ffill')
df_timeseries
# Check how many missing values are left in each column
missing_summary = df_timeseries[spend_columns].isnull().sum()

# Print only columns that still have missing values
print(missing_summary[missing_summary > 0])

# STEP 3: Drop rows where spend_all is missing (like 2018-12-31)
df_timeseries = df_timeseries.dropna(subset=['spend_all'])

# Step 2: Filter data for recovery phase (2022-01-01 onwards)
recovery_data = df_timeseries[df_timeseries['date'] >= '2022-01-01'].copy()
print(f"Full dataset: {len(df_timeseries)} days from {df_timeseries['date'].min()}
print(f"Recovery phase data: {len(recovery_data)} days from {recovery_data['date'].

# Step 3: Keep the last 30 days for testing all models
test_data = df_timeseries.iloc[-30:].copy()
actual_values = test_data['spend_all'].values
test_dates = test_data['date']
print(f"Test data for evaluation: {len(test_data)} days from {test_data['date'].min
```

```
# Plot full data vs recovery data
plt.figure(figsize=(14, 6))
plt.plot(df_timeseries['date'], df_timeseries['spend_all'], label='Full Dataset')
plt.plot(recovery_data['date'], recovery_data['spend_all'], label='Recovery Phase',
plt.axvline(x=pd.to_datetime('2022-01-01'), color='black', linestyle='--', label='R
plt.title('Consumer Spending: Full Data vs Recovery Phase')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('data_comparison.png')
plt.close()

# Initialize results dictionary
results = {
    'Model': [],
    'Data': [],
    'MAE': [],
    'MSE': [],
    'RMSE': []
}

# Create a forecast date range for all models
forecast_dates = pd.date_range(start=df_timeseries['date'].max() + pd.Timedelta(day

# Helper function to calculate and store metrics
def evaluate_model(model_name, data_type, predictions, actual):
    mae = mean_absolute_error(actual, predictions)
    mse = mean_squared_error(actual, predictions)
    rmse = np.sqrt(mse)

    results['Model'].append(model_name)
    results['Data'].append(data_type)
    results['MAE'].append(mae)
    results['MSE'].append(mse)
    results['RMSE'].append(rmse)

    print(f"{model_name} - {data_type} - MAE: {mae:.6f}, MSE: {mse:.6f}, RMSE: {rms
    return mae, mse, rmse

# Helper function to plot forecasts
def plot_forecast(model_name, data_type, dates, actual, forecast, test_dates, test_
    plt.figure(figsize=(14, 6))
    # Plot actual historical data
    if data_type == 'Full Data':
        plt.plot(df_timeseries['date'], df_timeseries['spend_all'], label='Historic
    else:
        plt.plot(recovery_data['date'], recovery_data['spend_all'], label='Historic

    # Plot test period (last 30 days) and forecast
    plt.plot(test_dates, test_values, label='Actual (Test Period)', color='green',
    plt.plot(dates, forecast, label=f'{model_name} Forecast', color='red', linestyl

    plt.title(f'{model_name} Forecast using {data_type}')
    plt.xlabel('Date')
```

```
    plt.ylabel('Normalized Spending')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(f'{model_name}_{data_type.replace(" ", "_")}_forecast.png')
    plt.close()

#------------------------------------------------------------------------------
# 1. PROPHET MODEL - FULL DATA
#------------------------------------------------------------------------------
print("\n--- Prophet Model - Full Data ---")
# Prepare data for Prophet (needs 'ds' and 'y' columns)
prophet_df = df_timeseries.rename(columns={'date': 'ds', 'spend_all': 'y'})

# Train Prophet model
prophet_model_full = Prophet()
prophet_model_full.fit(prophet_df)

# Create future dataframe for prediction
future_full = prophet_model_full.make_future_dataframe(periods=30)
forecast_prophet_full = prophet_model_full.predict(future_full)

# Extract the last 30 days of predictions for comparison with actual
prophet_full_predictions = forecast_prophet_full['yhat'].iloc[-30:].values

# Evaluate
evaluate_model('Prophet', 'Full Data', prophet_full_predictions, actual_values)

# Extract the forecast for plotting
prophet_full_forecast = forecast_prophet_full['yhat'].iloc[-30:].values

# Plot
plot_forecast('Prophet', 'Full Data', forecast_dates, actual_values,
              prophet_full_forecast, test_dates, actual_values)

#------------------------------------------------------------------------------
# 2. PROPHET MODEL - RECOVERY DATA
#------------------------------------------------------------------------------
print("\n--- Prophet Model - Recovery Data ---")
# Prepare recovery data for Prophet
prophet_df_recovery = recovery_data.rename(columns={'date': 'ds', 'spend_all': 'y'}

# Train Prophet model on recovery data
prophet_model_recovery = Prophet()
prophet_model_recovery.fit(prophet_df_recovery)

# Create future dataframe for prediction
future_recovery = prophet_model_recovery.make_future_dataframe(periods=30)
forecast_prophet_recovery = prophet_model_recovery.predict(future_recovery)

# Extract the last 30 predictions for comparison with actual
prophet_recovery_predictions = forecast_prophet_recovery['yhat'].iloc[-30:].values

# Evaluate
evaluate_model('Prophet', 'Recovery Data', prophet_recovery_predictions, actual_val
```

```
# Extract the forecast for plotting
prophet_recovery_forecast = forecast_prophet_recovery['yhat'].iloc[-30:].values

# Plot
plot_forecast('Prophet', 'Recovery Data', forecast_dates, actual_values,
              prophet_recovery_forecast, test_dates, actual_values)

#---------------------------------------------------------------------------------
# 3. ARIMA MODEL - FULL DATA
#---------------------------------------------------------------------------------
print("\n--- ARIMA Model - Full Data ---")
# Prepare data for ARIMA
ts_full = df_timeseries.set_index('date')['spend_all']

# Check stationarity
adf_result = adfuller(ts_full)
if adf_result[1] < 0.05:
    print("The time series is stationary")
else:
    print("The time series is not stationary, differencing may be required")

# Find optimal ARIMA parameters
arima_model_full = auto_arima(ts_full,
                             seasonal=False,
                             stepwise=True,
                             suppress_warnings=True,
                             error_action='ignore')

print(f"Best ARIMA model: {arima_model_full.order}")

# Forecast next 30 days
arima_forecast_full = arima_model_full.predict(n_periods=30)

# Evaluate
evaluate_model('ARIMA', 'Full Data', arima_forecast_full, actual_values)

# Plot
plot_forecast('ARIMA', 'Full Data', forecast_dates, actual_values,
              arima_forecast_full, test_dates, actual_values)

#---------------------------------------------------------------------------------
# 4. ARIMA MODEL - RECOVERY DATA
#---------------------------------------------------------------------------------
print("\n--- ARIMA Model - Recovery Data ---")
# Prepare recovery data for ARIMA
ts_recovery = recovery_data.set_index('date')['spend_all']

# Check stationarity
adf_result = adfuller(ts_recovery)
if adf_result[1] < 0.05:
    print("The recovery time series is stationary")
else:
    print("The recovery time series is not stationary, differencing may be required

# Find optimal ARIMA parameters
arima_model_recovery = auto_arima(ts_recovery,
```

```
                                 seasonal=False,
                                 stepwise=True,
                                 suppress_warnings=True,
                                 error_action='ignore')

print(f"Best ARIMA model for recovery data: {arima_model_recovery.order}")

# Forecast next 30 days
arima_forecast_recovery = arima_model_recovery.predict(n_periods=30)

# Evaluate
evaluate_model('ARIMA', 'Recovery Data', arima_forecast_recovery, actual_values)

# Plot
plot_forecast('ARIMA', 'Recovery Data', forecast_dates, actual_values,
              arima_forecast_recovery, test_dates, actual_values)

#---------------------------------------------------------------------------------
# 5. LSTM MODEL - FULL DATA
#---------------------------------------------------------------------------------
print("\n--- LSTM Model - Full Data ---")
# Prepare data for LSTM
data_full = df_timeseries['spend_all'].values.reshape(-1, 1)

# Normalize the data
scaler = MinMaxScaler()
scaled_data_full = scaler.fit_transform(data_full)

# Create sequences (30 days -> 1 day prediction)
def create_sequences(data, seq_length=30):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)

# Create sequences for full data
X_full, y_full = create_sequences(scaled_data_full)

# Use last 30 sequences for testing (these should match with your test data)
X_train_lstm_full = X_full[:-30]
y_train_lstm_full = y_full[:-30]
X_test_lstm_full = X_full[-30:]

# Define the LSTM model
lstm_model_full = Sequential()
lstm_model_full.add(LSTM(units=50, return_sequences=False, input_shape=(30, 1)))
lstm_model_full.add(Dense(units=1))

# Compile the model
lstm_model_full.compile(optimizer='adam', loss='mse')

# Train the model
lstm_model_full.fit(X_train_lstm_full, y_train_lstm_full, epochs=20, batch_size=16,

# Predict using LSTM
```

```
lstm_preds_scaled_full = lstm_model_full.predict(X_test_lstm_full)
lstm_preds_full = scaler.inverse_transform(lstm_preds_scaled_full).flatten()

# Evaluate
evaluate_model('LSTM', 'Full Data', lstm_preds_full, actual_values)

# Plot
plot_forecast('LSTM', 'Full Data', forecast_dates, actual_values,
              lstm_preds_full, test_dates, actual_values)

#---------------------------------------------------------------------------------
# 6. LSTM MODEL - RECOVERY DATA
#---------------------------------------------------------------------------------
print("\n--- LSTM Model - Recovery Data ---")
# Prepare recovery data for LSTM
data_recovery = recovery_data['spend_all'].values.reshape(-1, 1)

# Normalize the recovery data
scaler_recovery = MinMaxScaler()
scaled_data_recovery = scaler_recovery.fit_transform(data_recovery)

# Create sequences for recovery data
X_recovery, y_recovery = create_sequences(scaled_data_recovery)

# Use last 30 sequences for testing (or fewer if recovery period is shorter)
test_size = min(30, len(X_recovery) // 5)  # Use 20% of data or 30 days, whichever
X_train_lstm_recovery = X_recovery[:-test_size]
y_train_lstm_recovery = y_recovery[:-test_size]
X_test_lstm_recovery = X_recovery[-test_size:]

# Define the LSTM model for recovery data
lstm_model_recovery = Sequential()
lstm_model_recovery.add(LSTM(units=50, return_sequences=False, input_shape=(30, 1))
lstm_model_recovery.add(Dense(units=1))

# Compile the model
lstm_model_recovery.compile(optimizer='adam', loss='mse')

# Train the model on recovery data
lstm_model_recovery.fit(X_train_lstm_recovery, y_train_lstm_recovery, epochs=20, ba

# For prediction on the actual test set, we need the last 30 days of recovery data
last_sequence = scaled_data_recovery[-30:].reshape(1, 30, 1)

# Generate predictions one by one for next 30 days
lstm_recovery_forecast_scaled = []
current_sequence = last_sequence.copy()

for _ in range(30):
    # Predict the next value
    next_pred = lstm_model_recovery.predict(current_sequence)[0]
    # Add to forecast
    lstm_recovery_forecast_scaled.append(next_pred[0])
    # Update sequence (remove oldest, add newest prediction)
    current_sequence = np.append(current_sequence[:, 1:, :],
                                 [[next_pred]],
```

```
                                axis=1)

# Convert back to original scale
lstm_recovery_forecast = scaler_recovery.inverse_transform(
    np.array(lstm_recovery_forecast_scaled).reshape(-1, 1)).flatten()

# Evaluate
evaluate_model('LSTM', 'Recovery Data', lstm_recovery_forecast, actual_values)

# Plot
plot_forecast('LSTM', 'Recovery Data', forecast_dates, actual_values,
              lstm_recovery_forecast, test_dates, actual_values)

#------------------------------------------------------------------------------
# 7. TRANSFORMER MODEL - FULL DATA
#------------------------------------------------------------------------------
print("\n--- Transformer Model - Full Data ---")
# Reuse the scaled data from LSTM section

# Define Transformer block
def transformer_block(inputs, num_heads=4, ff_dim=64, dropout=0.1):
    attention_output = MultiHeadAttention(
        num_heads=num_heads, key_dim=inputs.shape[-1])(inputs, inputs)
    attention_output = Dropout(dropout)(attention_output)
    out1 = LayerNormalization(epsilon=1e-6)(Add()([inputs, attention_output]))

    ffn_output = Dense(ff_dim, activation='relu')(out1)
    ffn_output = Dense(inputs.shape[-1])(ffn_output)
    ffn_output = Dropout(dropout)(ffn_output)

    return LayerNormalization(epsilon=1e-6)(Add()([out1, ffn_output]))

# Define the full Transformer model
def build_transformer_model(seq_length=30, dim=1):
    inputs = Input(shape=(seq_length, dim))
    x = transformer_block(inputs)
    x = GlobalAveragePooling1D()(x)
    outputs = Dense(1)(x)
    model = Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer='adam', loss='mse')
    return model

# Build and train Transformer model on full data
transformer_model_full = build_transformer_model()
transformer_model_full.fit(X_train_lstm_full, y_train_lstm_full, epochs=20, batch_s

# Predict using Transformer
transformer_preds_scaled_full = transformer_model_full.predict(X_test_lstm_full)
transformer_preds_full = scaler.inverse_transform(transformer_preds_scaled_full).fl

# Evaluate
evaluate_model('Transformer', 'Full Data', transformer_preds_full, actual_values)

# Plot
plot_forecast('Transformer', 'Full Data', forecast_dates, actual_values,
              transformer_preds_full, test_dates, actual_values)
```

```
#--------------------------------------------------------------------------------
# 8. TRANSFORMER MODEL - RECOVERY DATA
#--------------------------------------------------------------------------------
print("\n--- Transformer Model - Recovery Data ---")
# Build and train Transformer model on recovery data
transformer_model_recovery = build_transformer_model()
transformer_model_recovery.fit(X_train_lstm_recovery, y_train_lstm_recovery, epochs

# For prediction on the actual test set, reuse the approach from LSTM
last_sequence = scaled_data_recovery[-30:].reshape(1, 30, 1)
transformer_recovery_forecast_scaled = []
current_sequence = last_sequence.copy()

for _ in range(30):
    # Predict the next value
    next_pred = transformer_model_recovery.predict(current_sequence)[0]
    # Add to forecast
    transformer_recovery_forecast_scaled.append(next_pred[0])
    # Update sequence (remove oldest, add newest prediction)
    current_sequence = np.append(current_sequence[:, 1:, :],
                                [[next_pred]],
                                axis=1)

# Convert back to original scale
transformer_recovery_forecast = scaler_recovery.inverse_transform(
    np.array(transformer_recovery_forecast_scaled).reshape(-1, 1)).flatten()

# Evaluate
evaluate_model('Transformer', 'Recovery Data', transformer_recovery_forecast, actua

# Plot
plot_forecast('Transformer', 'Recovery Data', forecast_dates, actual_values,
              transformer_recovery_forecast, test_dates, actual_values)

#--------------------------------------------------------------------------------
# RESULTS COMPARISON
#--------------------------------------------------------------------------------
# Create a DataFrame with all results
results_df = pd.DataFrame(results)
print("\nComparison of all models:")
print(results_df)

# Plot comparison of errors
plt.figure(figsize=(15, 10))

# RMSE comparison
plt.subplot(3, 1, 1)
for model in results_df['Model'].unique():
    model_data = results_df[results_df['Model'] == model]
    plt.bar(model_data['Data'], model_data['RMSE'], label=model)
plt.title('RMSE Comparison')
plt.ylabel('RMSE')
plt.legend()

# MA
```

```
C:\Users\dheer\AppData\Local\Temp\ipykernel_18136\1287248434.py:33: FutureWarning: D
ataFrame.fillna with 'method' is deprecated and will raise in a future version. Use
obj.ffill() or obj.bfill() instead.
  df_timeseries[spend_columns] = df_timeseries[spend_columns].fillna(method='ffill')
spend_all                 663
spend_aap                 663
spend_acf                 663
spend_aer                 663
spend_apg                 663
spend_durables            663
spend_nondurables         663
spend_grf                 663
spend_gen                 663
spend_hic                 663
spend_hcs                 663
spend_inperson            663
spend_inpersonmisc        663
spend_remoteservices      663
spend_sgh                 663
spend_tws                 663
spend_retail_w_grocery    663
spend_retail_no_grocery   663
spend_all_incmiddle       663
spend_all_q1              663
spend_all_q2              663
spend_all_q3              663
spend_all_q4              663
dtype: int64
Full dataset: 50031 days from 2020-01-13 00:00:00 to 2024-06-16 00:00:00
Recovery phase data: 13362 days from 2022-01-01 00:00:00 to 2024-06-16 00:00:00
Test data for evaluation: 30 days from 2024-06-16 00:00:00 to 2024-06-16 00:00:00

--- Prophet Model - Full Data ---
21:24:45 - cmdstanpy - INFO - Chain [1] start processing
21:25:18 - cmdstanpy - INFO - Chain [1] done processing
Prophet - Full Data - MAE: 0.104606, MSE: 0.015532, RMSE: 0.124626

--- Prophet Model - Recovery Data ---
21:25:19 - cmdstanpy - INFO - Chain [1] start processing
21:25:20 - cmdstanpy - INFO - Chain [1] done processing
Prophet - Recovery Data - MAE: 0.104448, MSE: 0.015407, RMSE: 0.124124

--- ARIMA Model - Full Data ---
The time series is not stationary, differencing may be required
Best ARIMA model: (2, 1, 4)
ARIMA - Full Data - MAE: 0.116270, MSE: 0.018623, RMSE: 0.136466
D:\anaconda\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:836: ValueWarning: N
o supported index is available. Prediction results will be given with an integer ind
ex beginning at `start`.
  return get_prediction_index(
D:\anaconda\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:836: FutureWarning:
No supported index is available. In the next version, calling this method in a model
without a supported index will result in an exception.
  return get_prediction_index(
```

```
--- ARIMA Model - Recovery Data ---
The recovery time series is stationary
Best ARIMA model for recovery data: (4, 1, 4)
ARIMA - Recovery Data - MAE: 0.113401, MSE: 0.017974, RMSE: 0.134068
```

```
D:\anaconda\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:836: ValueWarning: N
o supported index is available. Prediction results will be given with an integer ind
ex beginning at `start`.
  return get_prediction_index(
D:\anaconda\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:836: FutureWarning:
No supported index is available. In the next version, calling this method in a model
without a supported index will result in an exception.
  return get_prediction_index(
```

```
--- LSTM Model - Full Data ---
```

```
D:\anaconda\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not p
ass an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

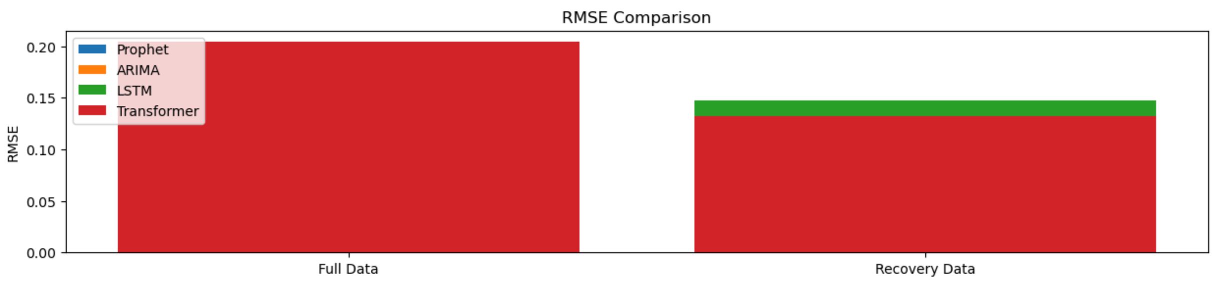
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step
LSTM - Full Data - MAE: 0.030141, MSE: 0.002546, RMSE: 0.050458

--- LSTM Model - Recovery Data ---
```

```
D:\anaconda\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not p
ass an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
LSTM - Recovery Data - MAE: 0.118560, MSE: 0.021663, RMSE: 0.147185

--- Transformer Model - Full Data ---
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 137ms/step
Transformer - Full Data - MAE: 0.173493, MSE: 0.041973, RMSE: 0.204873

--- Transformer Model - Recovery Data ---
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
Transformer - Recovery Data - MAE: 0.108028, MSE: 0.017574, RMSE: 0.132567

Comparison of all models:
         Model           Data       MAE       MSE      RMSE
0      Prophet      Full Data  0.104606  0.015532  0.124626
1      Prophet  Recovery Data  0.104448  0.015407  0.124124
2        ARIMA      Full Data  0.116270  0.018623  0.136466
3        ARIMA  Recovery Data  0.113401  0.017974  0.134068
4         LSTM      Full Data  0.030141  0.002546  0.050458
5         LSTM  Recovery Data  0.118560  0.021663  0.147185
6  Transformer      Full Data  0.173493  0.041973  0.204873
7  Transformer  Recovery Data  0.108028  0.017574  0.132567
```

Out[11]: `<matplotlib.legend.Legend at 0x2058b46bd10>`



In [ ]: