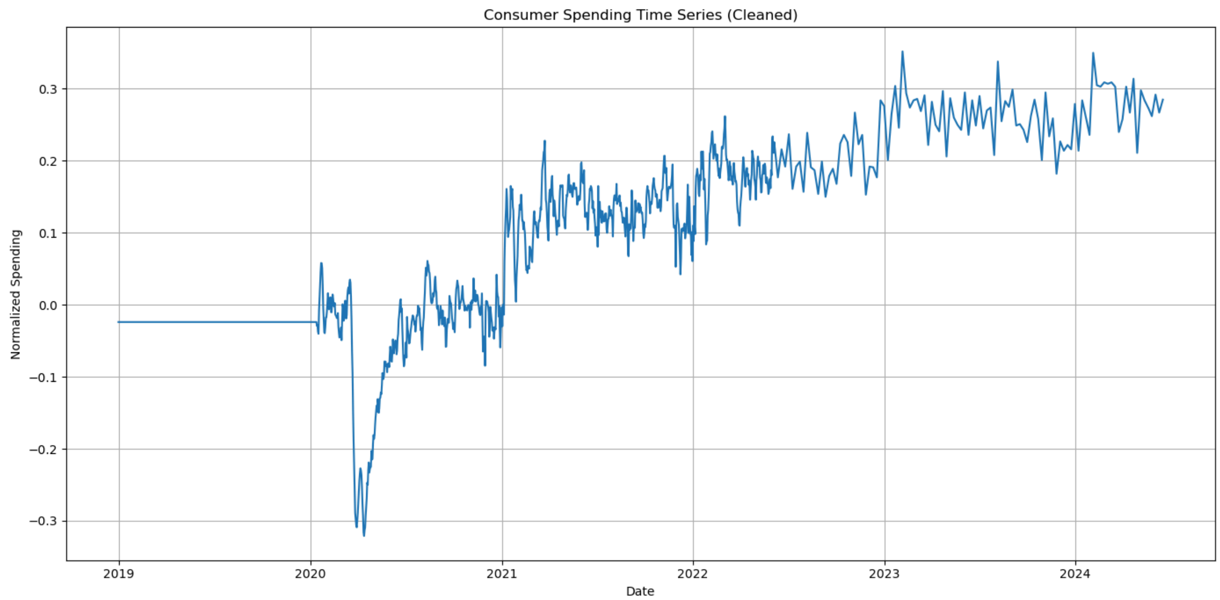
In [1]:
```
# Data Loading and Cleaning
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os

# Load the dataset

file_path = "D:/capstone/datasets/Affinity - State - Daily.xlsx"
df = pd.read_excel(file_path)

print(f"Raw dataset shape: {df.shape}")
print("\nFirst few rows of raw data:")
print(df.head())

# Create a proper datetime column
if 'year' in df.columns and 'month' in df.columns and 'day' in df.columns:
    print("\nCreating datetime column from year, month, day...")
    df['date'] = pd.to_datetime(df[['year', 'month', 'day']])

# Identify all spend-related columns
spend_columns = [col for col in df.columns if 'spend' in col]
print(f"\nIdentified {len(spend_columns)} spending-related columns: {spend_columns}

# Replace '.' with NaN (missing values)
print("\nReplacing '.' with NaN in spending columns...")
df[spend_columns] = df[spend_columns].replace('.', pd.NA)

# Convert all spend columns to numeric
print("\nConverting spend columns to numeric...")
df[spend_columns] = df[spend_columns].apply(pd.to_numeric, errors='coerce')

# Check for missing values
missing_counts = df[spend_columns].isnull().sum()
print("\nMissing values count before imputation:")
print(missing_counts)

# Interpolate missing values (best-performing imputation method)
print("\nInterpolating missing values...")
df[spend_columns] = df[spend_columns].interpolate()

# Forward fill any remaining NaNs (especially at start of series)
print("\nForward filling any remaining missing values...")
df[spend_columns] = df[spend_columns].fillna(method='ffill')

# Backward fill any remaining NaNs (if any at the end of series)
df[spend_columns] = df[spend_columns].fillna(method='bfill')

# Check for any remaining missing values
remaining_missing = df[spend_columns].isnull().sum().sum()
print(f"\nRemaining missing values after imputation: {remaining_missing}")

# Drop rows where spend_all is still missing
print("\nRemoving rows where 'spend_all' is still missing...")
```

```
df_timeseries = df.dropna(subset=['spend_all'])

# Filter to include only necessary columns for our analysis
df_timeseries = df_timeseries[['date', 'spend_all']]

# Check for duplicate dates
duplicates = df_timeseries['date'].duplicated().sum()
if duplicates > 0:
    print(f"\nWarning: Found {duplicates} duplicate dates. Keeping the first occurr
    df_timeseries = df_timeseries.drop_duplicates(subset=['date'], keep='first')

# Sort by date to ensure chronological order
df_timeseries = df_timeseries.sort_values('date').reset_index(drop=True)

# Examine the cleaned dataset
print("\nCleaned dataset information:")
print(f"Date range: {df_timeseries['date'].min()} to {df_timeseries['date'].max()}"
print(f"Number of rows: {len(df_timeseries)}")
print(f"Number of unique dates: {df_timeseries['date'].nunique()}")

# Check for gaps in dates
print("\nChecking for gaps in the time series...")
date_range = pd.date_range(start=df_timeseries['date'].min(), end=df_timeseries['da
missing_dates = set(date_range) - set(df_timeseries['date'])
if missing_dates:
    print(f"Found {len(missing_dates)} missing dates in the time series.")
    print("Filling in missing dates...")
    # Create a complete date series
    full_date_df = pd.DataFrame({'date': date_range})
    # Merge with existing data
    df_timeseries = pd.merge(full_date_df, df_timeseries, on='date', how='left')
    # Interpolate missing spend_all values
    df_timeseries['spend_all'] = df_timeseries['spend_all'].interpolate()
else:
    print("No gaps found in the time series.")

# Summary statistics for spend_all
print("\nSummary statistics for spend_all:")
print(df_timeseries['spend_all'].describe())

# Visualize the cleaned time series
plt.figure(figsize=(14, 7))
plt.plot(df_timeseries['date'], df_timeseries['spend_all'])
plt.title('Consumer Spending Time Series (Cleaned)')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.grid(True)
plt.tight_layout()
plt.savefig('cleaned_timeseries.png')
plt.show()

print("\nData cleaning completed successfully. Ready for modeling.")
```

```
Raw dataset shape: (50694, 29)

First few rows of raw data:
   year  month  day  statefips freq spend_all spend_aap spend_acf spend_aer  \
0  2018     12   31          1    d         .         .         .         .
1  2018     12   31          2    d         .         .         .         .
2  2018     12   31          4    d         .         .         .         .
3  2018     12   31          5    d         .         .         .         .
4  2018     12   31          6    d         .         .         .         .

  spend_apg  ... spend_sgh spend_tws spend_retail_w_grocery  \
0         .  ...         .         .                      .
1         .  ...         .         .                      .
2         .  ...         .         .                      .
3         .  ...         .         .                      .
4         .  ...         .         .                      .

  spend_retail_no_grocery spend_all_incmiddle spend_all_q1 spend_all_q2  \
0                       .                   .            .            .
1                       .                   .            .            .
2                       .                   .            .            .
3                       .                   .            .            .
4                       .                   .            .            .

  spend_all_q3 spend_all_q4 provisional
0            .            .           0
1            .            .           0
2            .            .           0
3            .            .           0
4            .            .           0

[5 rows x 29 columns]

Creating datetime column from year, month, day...

Identified 23 spending-related columns: ['spend_all', 'spend_aap', 'spend_acf', 'spe
nd_aer', 'spend_apg', 'spend_durables', 'spend_nondurables', 'spend_grf', 'spend_ge
n', 'spend_hic', 'spend_hcs', 'spend_inperson', 'spend_inpersonmisc', 'spend_remotes
ervices', 'spend_sgh', 'spend_tws', 'spend_retail_w_grocery', 'spend_retail_no_groce
ry', 'spend_all_incmiddle', 'spend_all_q1', 'spend_all_q2', 'spend_all_q3', 'spend_a
ll_q4']

Replacing '.' with NaN in spending columns...

Converting spend columns to numeric...

Missing values count before imputation:
spend_all                  1644
spend_aap                  1644
spend_acf                  1644
spend_aer                  1644
spend_apg                  1644
spend_durables             1644
spend_nondurables          1644
spend_grf                  1644
spend_gen                  1644
```

```
spend_hic                  1644
spend_hcs                  1644
spend_inperson             1644
spend_inpersonmisc         1644
spend_remoteservices       1644
spend_sgh                  1644
spend_tws                  1644
spend_retail_w_grocery     1644
spend_retail_no_grocery    1644
spend_all_incmiddle        1644
spend_all_q1               4587
spend_all_q2               1644
spend_all_q3               1644
spend_all_q4               3606
dtype: int64

Interpolating missing values...

Forward filling any remaining missing values...

Remaining missing values after imputation: 0

Removing rows where 'spend_all' is still missing...

Warning: Found 49700 duplicate dates. Keeping the first occurrence...

Cleaned dataset information:
Date range: 2018-12-31 00:00:00 to 2024-06-16 00:00:00
Number of rows: 994
Number of unique dates: 994

Checking for gaps in the time series...
Found 1001 missing dates in the time series.
Filling in missing dates...

Summary statistics for spend_all:
count    1995.000000
mean        0.116389
std         0.132572
min        -0.321000
25%        -0.023900
50%         0.145000
75%         0.232429
max         0.352000
Name: spend_all, dtype: float64
```

```
C:\Users\dheer\AppData\Local\Temp\ipykernel_27932\2753603710.py:44: FutureWarning: D
ataFrame.fillna with 'method' is deprecated and will raise in a future version. Use
obj.ffill() or obj.bfill() instead.
  df[spend_columns] = df[spend_columns].fillna(method='ffill')
C:\Users\dheer\AppData\Local\Temp\ipykernel_27932\2753603710.py:47: FutureWarning: D
ataFrame.fillna with 'method' is deprecated and will raise in a future version. Use
obj.ffill() or obj.bfill() instead.
  df[spend_columns] = df[spend_columns].fillna(method='bfill')
```

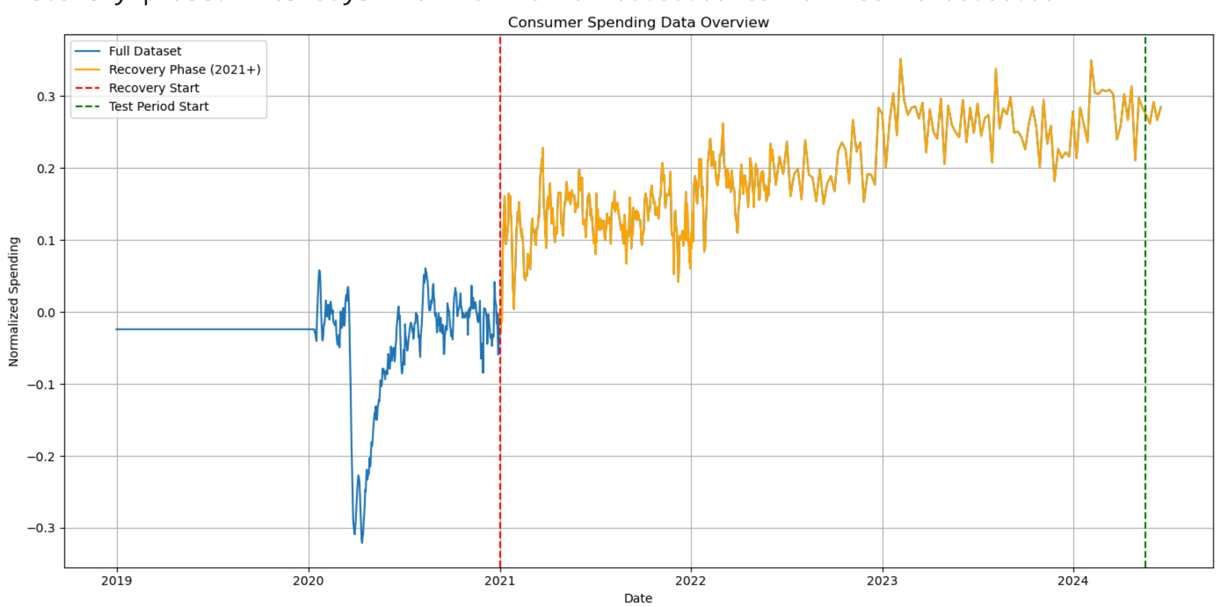


Data cleaning completed successfully. Ready for modeling.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_absolute_error, mean_squared_error
from statsmodels.tsa.stattools import adfuller
from pmdarima import auto_arima
from prophet import Prophet
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import (
    LSTM, Dense, Dropout, Input, LayerNormalization,
    GlobalAveragePooling1D, MultiHeadAttention, Add
)
from tensorflow.keras.callbacks import EarlyStopping

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)



# Initialize results storage for comparison
results_df = pd.DataFrame(columns=['Model', 'Data Type', 'MAE', 'MSE', 'RMSE'])

# Create a function to add results to our comparison dataframe
def add_result(model_name, data_type, mae, mse, rmse):
    global results_df
    results_df = pd.concat([results_df, pd.DataFrame({
        'Model': [model_name],
        'Data Type': [data_type],
        'MAE': [mae],
        'MSE': [mse],
        'RMSE': [rmse]
    })], ignore_index=True)
```

```python
# Split data - use last 30 days for testing/evaluation
test_size = 30
train_data = df_timeseries.iloc[:-test_size].copy()
test_data = df_timeseries.iloc[-test_size:].copy()

# Also create recovery phase dataset (from 2021-01-01 onwards)
recovery_data = df_timeseries[df_timeseries['date'] >= '2021-01-01'].copy()
recovery_train = recovery_data.iloc[:-test_size].copy()
recovery_test = recovery_data.iloc[-test_size:].copy()

print(f"Full dataset: {len(df_timeseries)} days from {df_timeseries['date'].min()}
print(f"Training data: {len(train_data)} days")
print(f"Test data: {len(test_data)} days")
print(f"Recovery phase: {len(recovery_data)} days from {recovery_data['date'].min()

# Visualize the data
plt.figure(figsize=(14, 7))
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('2021-01-01'), color='red', linestyle='--', label='Rec
plt.axvline(x=test_data['date'].iloc[0], color='green', linestyle='--', label='Test
plt.title('Consumer Spending Data Overview')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('data_overview.png')
plt.show()
```

```
Full dataset: 1995 days from 2018-12-31 00:00:00 to 2024-06-16 00:00:00
Training data: 1965 days
Test data: 30 days
Recovery phase: 1263 days from 2021-01-01 00:00:00 to 2024-06-16 00:00:00
```



MODEL 1: ARIMA

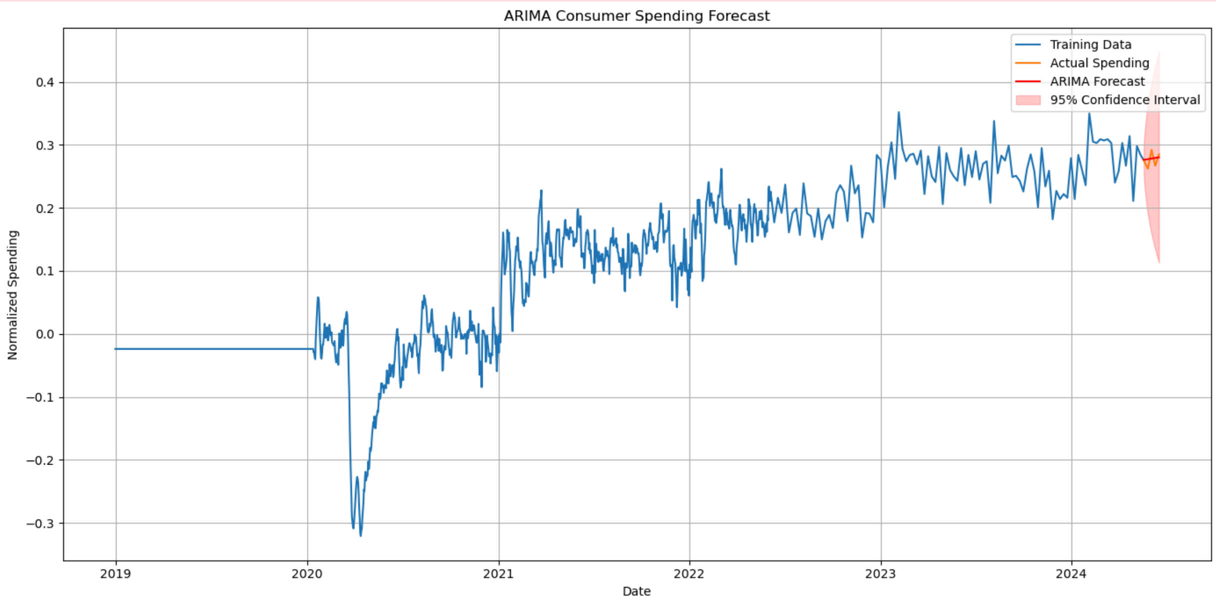
In [4]:
```
print("\n" + "="*80)
print("MODEL 1: ARIMA - FULL DATASET")
print("="*80)

# Check stationarity with ADF test
print("\nRunning ADF Test for stationarity...")
adf_result = adfuller(df_timeseries['spend_all'])
print(f"ADF Statistic: {adf_result[0]}")
print(f"p-value: {adf_result[1]}")
print("Critical Values:")
for key, value in adf_result[4].items():
    print(f"\t{key}: {value}")

# Interpret the result
if adf_result[1] < 0.05:
    print("The series is stationary")
else:
    print("The series is not stationary, differencing may be required")

# Create time series data
ts_full = train_data.set_index('date')['spend_all']
ts_test = test_data.set_index('date')['spend_all']

# Determine optimal ARIMA parameters using auto_arima
print("\nFinding optimal ARIMA parameters...")
arima_model = auto_arima(ts_full,
                       seasonal=False,
                       stepwise=True,
                       trace=True,
                       error_action='ignore',
                       suppress_warnings=True)

print(f"\nBest ARIMA model: {arima_model.order}")
print(arima_model.summary())

# Get forecast with confidence intervals
forecast_results = arima_model.predict(n_periods=test_size, return_conf_int=True)
arima_forecast = forecast_results[0]  # Point forecasts
confidence_intervals = forecast_results[1]  # Confidence intervals
forecast_index = ts_test.index  # For plotting
```

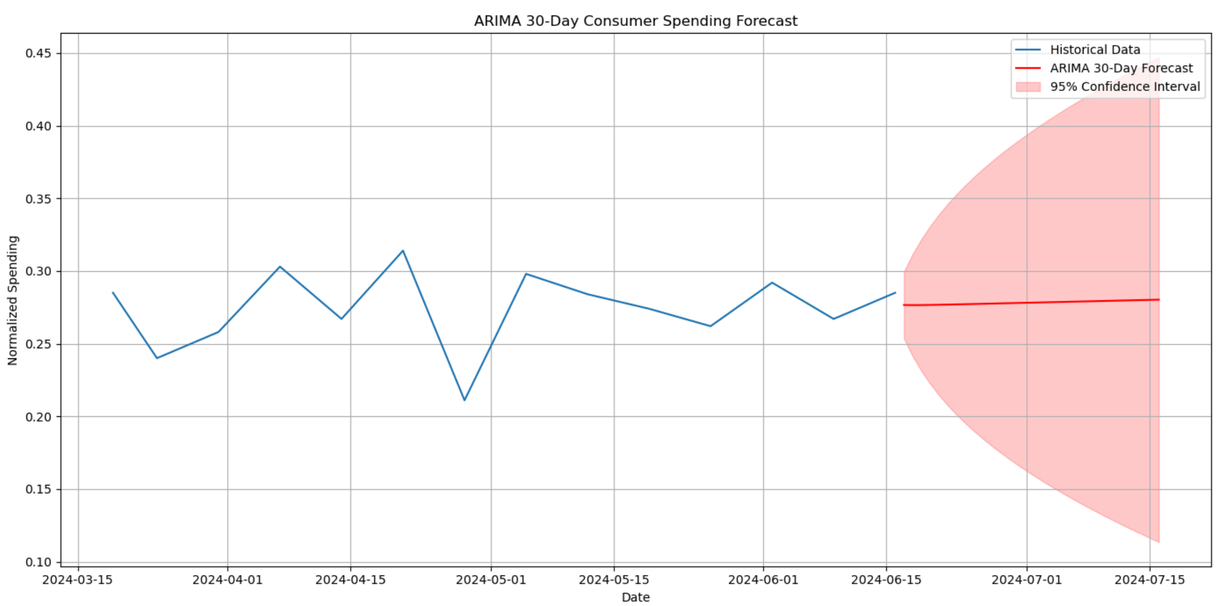
```
# Evaluation metrics
mae_arima = mean_absolute_error(ts_test, arima_forecast)
mse_arima = mean_squared_error(ts_test, arima_forecast)
rmse_arima = np.sqrt(mse_arima)

print("\n--- ARIMA Model Evaluation ---")
print(f"Mean Absolute Error (MAE): {mae_arima:.6f}")
print(f"Mean Squared Error (MSE): {mse_arima:.6f}")
print(f"Root Mean Squared Error (RMSE): {rmse_arima:.6f}")

# Add to results
add_result('ARIMA', 'Full Data', mae_arima, mse_arima, rmse_arima)

# Plot results
plt.figure(figsize=(14, 7))
plt.plot(ts_full.index, ts_full, label='Training Data')
plt.plot(ts_test.index, ts_test, label='Actual Spending')
plt.plot(ts_test.index, arima_forecast, label='ARIMA Forecast', color='red')
plt.fill_between(ts_test.index,
                confidence_intervals[:, 0],  # Lower bound
                confidence_intervals[:, 1],  # Upper bound
                color='red', alpha=0.2, label='95% Confidence Interval')
plt.title('ARIMA Consumer Spending Forecast')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('arima_forecast.png')
plt.show()

# Forecast next 30 days (future prediction)
future_results = arima_model.predict(n_periods=30, return_conf_int=True)
future_forecast_arima = future_results[0]  # Point forecasts
future_confidence_intervals = future_results[1]  # Confidence intervals

# Generate future dates
future_dates = pd.date_range(start=df_timeseries['date'].max() + pd.Timedelta(days=

# Plot future forecast
plt.figure(figsize=(14, 7))
plt.plot(df_timeseries['date'][-90:], df_timeseries['spend_all'][-90:], label='Hist
plt.plot(future_dates, future_forecast_arima, label='ARIMA 30-Day Forecast', color=
plt.fill_between(future_dates,
                future_confidence_intervals[:, 0],  # Lower bound
                future_confidence_intervals[:, 1],  # Upper bound
                color='red', alpha=0.2, label='95% Confidence Interval')
plt.title('ARIMA 30-Day Consumer Spending Forecast')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('arima_future_forecast.png')
plt.show()
```

```
================================================================================
MODEL 1: ARIMA - FULL DATASET
================================================================================

Running ADF Test for stationarity...
ADF Statistic: -1.3702487050072258
p-value: 0.5964532649851969
Critical Values:
        1%: -3.4336771595431106
        5%: -2.863009746829746
        10%: -2.5675524325901415
The series is not stationary, differencing may be required

Finding optimal ARIMA parameters...
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-11869.298, Time=0.30 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-11809.837, Time=0.20 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-11865.823, Time=0.20 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-11857.550, Time=0.78 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-11811.515, Time=0.09 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-11874.024, Time=0.21 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-11869.555, Time=0.48 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-11877.603, Time=0.77 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-11861.822, Time=1.46 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=-11876.356, Time=0.44 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=-11872.614, Time=0.13 sec

Best model:  ARIMA(1,1,1)(0,0,0)[0] intercept
Total fit time: 5.054 seconds

Best ARIMA model: (1, 1, 1)
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1965
Model:               SARIMAX(1, 1, 1)   Log Likelihood                5942.802
Date:                Sat, 29 Mar 2025   AIC                         -11877.603
Time:                        00:21:47   BIC                         -11855.272
Sample:                    12-31-2018   HQIC                        -11869.396
                         - 05-17-2024
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept   6.723e-05      0.000      0.392      0.695      -0.000       0.000
ar.L1          0.5257      0.065      8.042      0.000       0.398       0.654
ma.L1         -0.3615      0.068     -5.291      0.000      -0.495      -0.228
sigma2         0.0001   2.28e-06     60.337      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                   0.19   Jarque-Bera (JB):              6195.21
Prob(Q):                              0.66   Prob(JB):                         0.00
Heteroskedasticity (H):               0.51   Skew:                             0.57
Prob(H) (two-sided):                  0.00   Kurtosis:                        11.63
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
```

```
p).

--- ARIMA Model Evaluation ---
Mean Absolute Error (MAE): 0.006686
Mean Squared Error (MSE): 0.000062
Root Mean Squared Error (RMSE): 0.007886
```

```
C:\Users\dheer\AppData\Local\Temp\ipykernel_27932\864976347.py:30: FutureWarning: Th
e behavior of DataFrame concatenation with empty or all-NA entries is deprecated. In
a future version, this will no longer exclude empty or all-NA columns when determini
ng the result dtypes. To retain the old behavior, exclude the relevant entries befor
e the concat operation.
  results_df = pd.concat([results_df, pd.DataFrame({
```





--

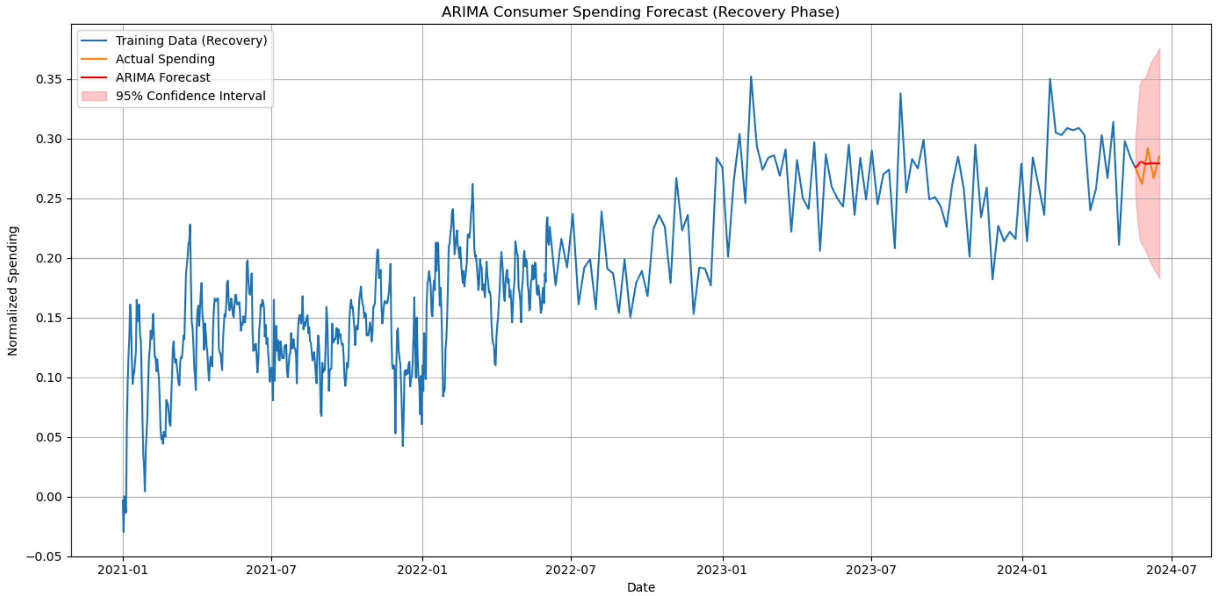
MODEL 1: ARIMA - RECOVERY PHASE

--

In [6]:
```
print("\n" + "="*80)
print("MODEL 1: ARIMA - RECOVERY PHASE")
print("="*80)

# Split recovery data into train and test
recovery_train = recovery_data.iloc[:-test_size].copy()
recovery_test = recovery_data.iloc[-test_size:].copy()
print(f"Recovery training data: {len(recovery_train)} days")
print(f"Recovery test data: {len(recovery_test)} days")

# Create time series data for recovery phase
ts_recovery = recovery_train.set_index('date')['spend_all']
ts_recovery_test = recovery_test.set_index('date')['spend_all']

# Check stationarity
adf_result_recovery = adfuller(recovery_data['spend_all'])
if adf_result_recovery[1] < 0.05:
    print("Recovery phase data is stationary")
else:
    print("Recovery phase data is not stationary, differencing may be required")

# Find optimal parameters
arima_model_recovery = auto_arima(ts_recovery,
                                seasonal=False,
                                stepwise=True,
                                trace=True,
                                error_action='ignore',
                                suppress_warnings=True)

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

# Forecast for test period with confidence intervals
forecast_recovery_results = arima_model_recovery.predict(n_periods=test_size, retur
arima_forecast_recovery = forecast_recovery_results[0]  # Point forecasts
recovery_confidence_intervals = forecast_recovery_results[1]  # Confidence interval

# Evaluation metrics
mae_arima_recovery = mean_absolute_error(ts_recovery_test, arima_forecast_recovery)
mse_arima_recovery = mean_squared_error(ts_recovery_test, arima_forecast_recovery)
rmse_arima_recovery = np.sqrt(mse_arima_recovery)

print("\n--- ARIMA Recovery Phase Evaluation ---")
print(f"Mean Absolute Error (MAE): {mae_arima_recovery:.6f}")
print(f"Mean Squared Error (MSE): {mse_arima_recovery:.6f}")
print(f"Root Mean Squared Error (RMSE): {rmse_arima_recovery:.6f}")

# Add to results
```

```python
add_result('ARIMA', 'Recovery Phase', mae_arima_recovery, mse_arima_recovery, rmse_

# Plot for recovery phase
plt.figure(figsize=(14, 7))
plt.plot(ts_recovery.index, ts_recovery, label='Training Data (Recovery)')
plt.plot(ts_recovery_test.index, ts_recovery_test, label='Actual Spending')
plt.plot(ts_recovery_test.index, arima_forecast_recovery, label='ARIMA Forecast', c
plt.fill_between(ts_recovery_test.index,
                 recovery_confidence_intervals[:, 0],  # Lower bound
                 recovery_confidence_intervals[:, 1],  # Upper bound
                 color='red', alpha=0.2, label='95% Confidence Interval')
plt.title('ARIMA Consumer Spending Forecast (Recovery Phase)')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('arima_recovery_forecast.png')
plt.show()
```

```
================================================================================
MODEL 1: ARIMA - RECOVERY PHASE
================================================================================
Recovery training data: 1233 days
Recovery test data: 30 days
Recovery phase data is not stationary, differencing may be required
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-7288.769, Time=0.48 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-7258.442, Time=0.11 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-7287.416, Time=0.24 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-7282.501, Time=0.35 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-7260.046, Time=0.17 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-7293.887, Time=0.33 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-7292.841, Time=0.75 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-7293.092, Time=0.57 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-7292.138, Time=0.70 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=-7293.887, Time=0.65 sec
 ARIMA(0,1,4)(0,0,0)[0] intercept   : AIC=-7294.048, Time=1.56 sec
 ARIMA(1,1,4)(0,0,0)[0] intercept   : AIC=-7309.851, Time=1.30 sec
 ARIMA(2,1,4)(0,0,0)[0] intercept   : AIC=-7353.399, Time=1.44 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=-7290.689, Time=1.35 sec
 ARIMA(3,1,4)(0,0,0)[0] intercept   : AIC=-7345.162, Time=1.56 sec
 ARIMA(2,1,5)(0,0,0)[0] intercept   : AIC=-7332.011, Time=1.75 sec
 ARIMA(1,1,5)(0,0,0)[0] intercept   : AIC=-7325.515, Time=2.64 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=-7288.536, Time=0.67 sec
 ARIMA(3,1,5)(0,0,0)[0] intercept   : AIC=-7385.558, Time=2.19 sec
 ARIMA(4,1,5)(0,0,0)[0] intercept   : AIC=-7340.414, Time=2.15 sec
 ARIMA(4,1,4)(0,0,0)[0] intercept   : AIC=-7359.860, Time=2.25 sec
 ARIMA(3,1,5)(0,0,0)[0]             : AIC=-7421.713, Time=1.04 sec
 ARIMA(2,1,5)(0,0,0)[0]             : AIC=-7387.368, Time=0.94 sec
 ARIMA(3,1,4)(0,0,0)[0]             : AIC=-7363.926, Time=0.89 sec
 ARIMA(4,1,5)(0,0,0)[0]             : AIC=-7355.362, Time=1.13 sec
 ARIMA(2,1,4)(0,0,0)[0]             : AIC=-7374.794, Time=0.78 sec
 ARIMA(4,1,4)(0,0,0)[0]             : AIC=-7361.919, Time=0.93 sec

Best model:  ARIMA(3,1,5)(0,0,0)[0]
Total fit time: 28.951 seconds

Best ARIMA model for recovery data: (3, 1, 5)

--- ARIMA Recovery Phase Evaluation ---
Mean Absolute Error (MAE): 0.007297
Mean Squared Error (MSE): 0.000078
Root Mean Squared Error (RMSE): 0.008813
```



--

MODEL 2: PROPHET

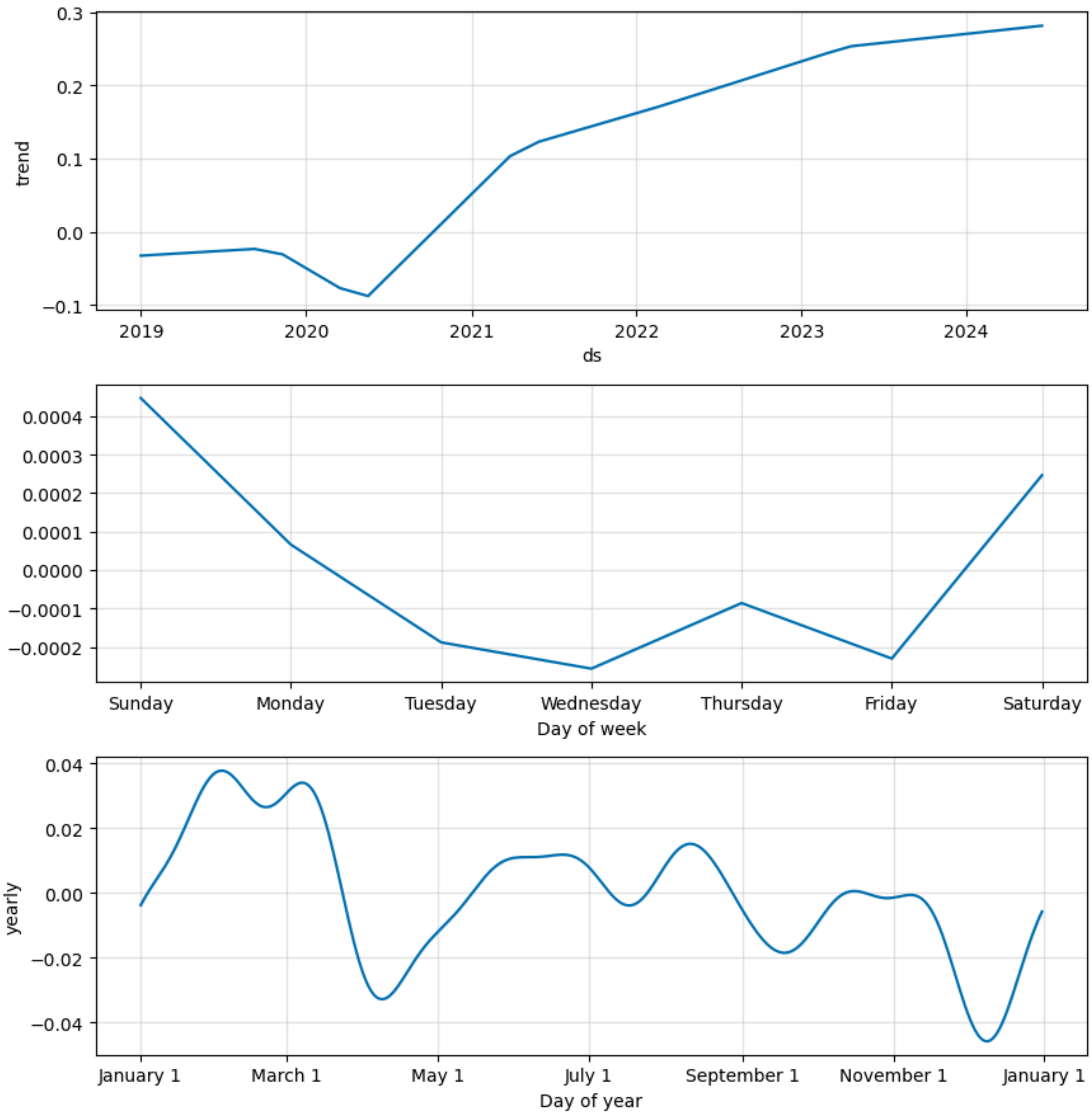
--

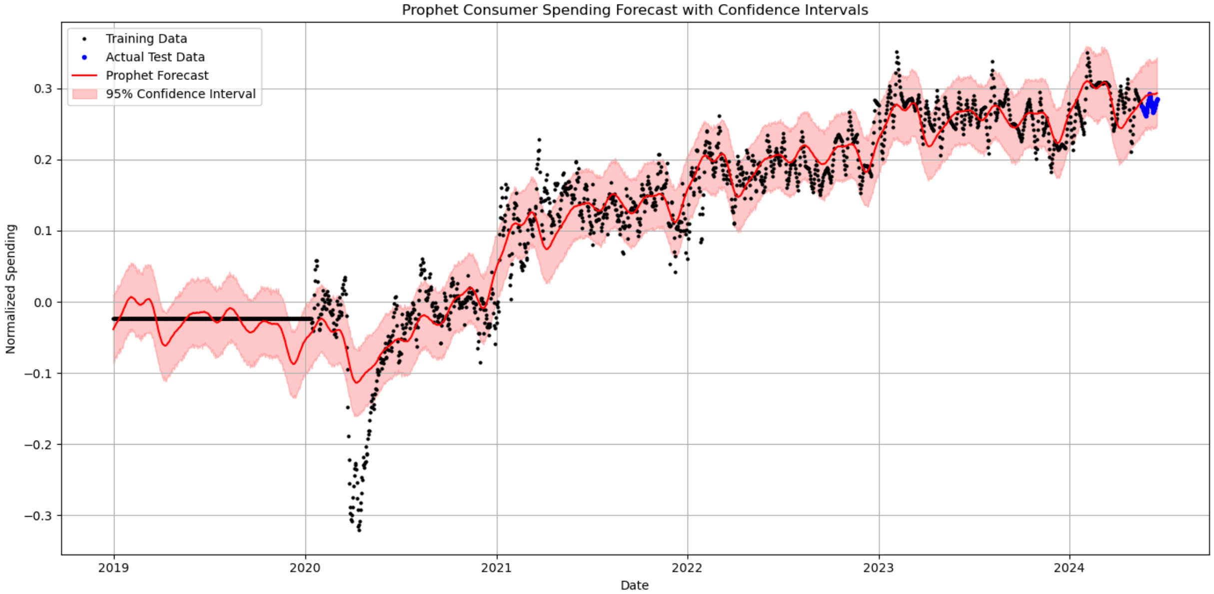
In [8]:
```
print("\n" + "="*80)
print("MODEL 2: PROPHET - FULL DATASET")
print("="*80)

# Prophet requires columns named 'ds' and 'y'
prophet_train = train_data.rename(columns={'date': 'ds', 'spend_all': 'y'})
prophet_test = test_data.rename(columns={'date': 'ds', 'spend_all': 'y'})

# Initialize and train Prophet model
print("Training Prophet model with tuned parameters...")
prophet_model = Prophet(
    changepoint_prior_scale=0.05,
    seasonality_prior_scale=10.0,
    seasonality_mode='additive',
    daily_seasonality=False,
    weekly_seasonality=True,
    yearly_seasonality=True
)
prophet_model.fit(prophet_train)

# Create a future dataframe for the test period
future = prophet_model.make_future_dataframe(periods=test_size)
```

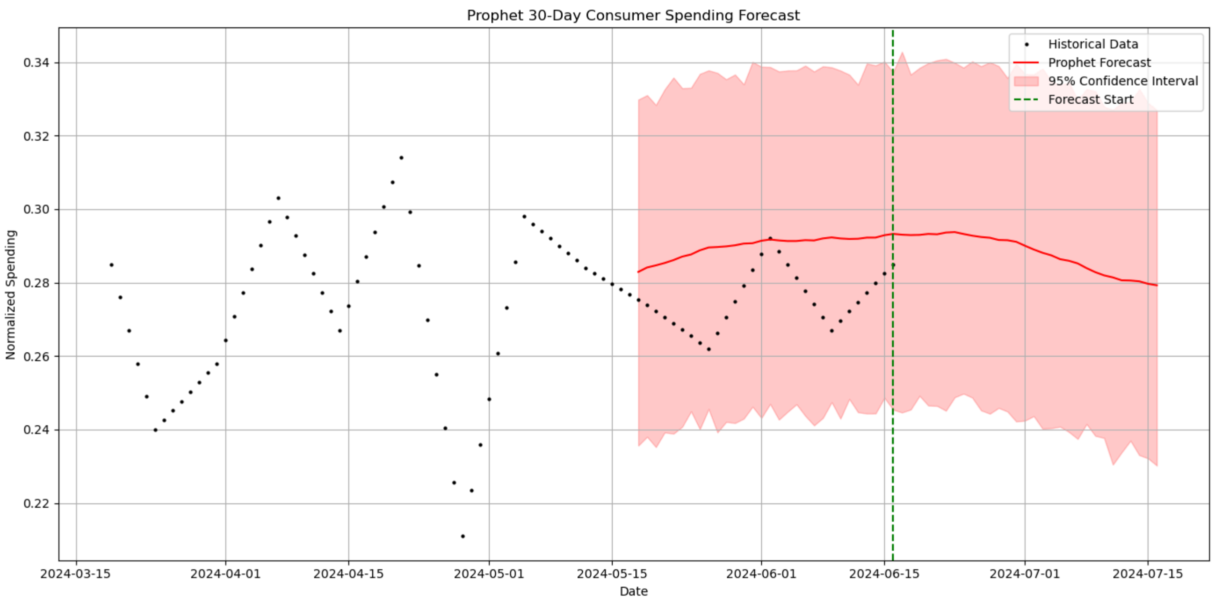
```
forecast = prophet_model.predict(future)

# Extract forecasted values for the test period
y_pred_prophet = forecast.iloc[-test_size:]['yhat'].values
y_true_prophet = prophet_test['y'].values

# Calculate evaluation metrics
mae_prophet = mean_absolute_error(y_true_prophet, y_pred_prophet)
mse_prophet = mean_squared_error(y_true_prophet, y_pred_prophet)
rmse_prophet = np.sqrt(mse_prophet)

print("\n--- Prophet Model Evaluation ---")
print(f"Mean Absolute Error (MAE): {mae_prophet:.6f}")
print(f"Mean Squared Error (MSE): {mse_prophet:.6f}")
print(f"Root Mean Squared Error (RMSE): {rmse_prophet:.6f}")

# Add to results
add_result('Prophet', 'Full Data', mae_prophet, mse_prophet, rmse_prophet)

# Plot components of the forecast
fig1 = prophet_model.plot_components(forecast)
plt.tight_layout()
plt.savefig('prophet_components.png')
plt.show()

# Create a more detailed plot
plt.figure(figsize=(14, 7))
plt.plot(prophet_train['ds'], prophet_train['y'], 'ko', markersize=2, label='Traini
plt.plot(prophet_test['ds'], prophet_test['y'], 'bo', markersize=3, label='Actual T
plt.plot(forecast['ds'], forecast['yhat'], 'r-', label='Prophet Forecast')
plt.fill_between(forecast['ds'],
                forecast['yhat_lower'],
                forecast['yhat_upper'],
                color='red', alpha=0.2, label='95% Confidence Interval')
plt.title('Prophet Consumer Spending Forecast with Confidence Intervals')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('prophet_detailed_forecast.png')
plt.show()

# Forecast next 30 days (future prediction)
future_forecast_df = prophet_model.make_future_dataframe(periods=test_size + 30)
future_forecast = prophet_model.predict(future_forecast_df)

# Plot future 30-day forecast
plt.figure(figsize=(14, 7))
plt.plot(df_timeseries['date'][-90:], df_timeseries['spend_all'][-90:], 'ko', marke
plt.plot(future_forecast['ds'][-60:], future_forecast['yhat'][-60:], 'r-', label='P
plt.fill_between(future_forecast['ds'][-60:],
                future_forecast['yhat_lower'][-60:],
                future_forecast['yhat_upper'][-60:],
                color='red', alpha=0.2, label='95% Confidence Interval')
future_start_idx = len(future_forecast) - 30
```

```
plt.axvline(x=future_forecast['ds'][future_start_idx-1], color='green', linestyle='
plt.title('Prophet 30-Day Consumer Spending Forecast')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('prophet_future_forecast.png')
plt.show()
```

```
================================================================================
MODEL 2: PROPHET - FULL DATASET
================================================================================
Training Prophet model with tuned parameters...
00:22:18 - cmdstanpy - INFO - Chain [1] start processing
00:22:18 - cmdstanpy - INFO - Chain [1] done processing
--- Prophet Model Evaluation ---
Mean Absolute Error (MAE): 0.014657
Mean Squared Error (MSE): 0.000264
Root Mean Squared Error (RMSE): 0.016253
```







--

Run Prophet on recovery phase data

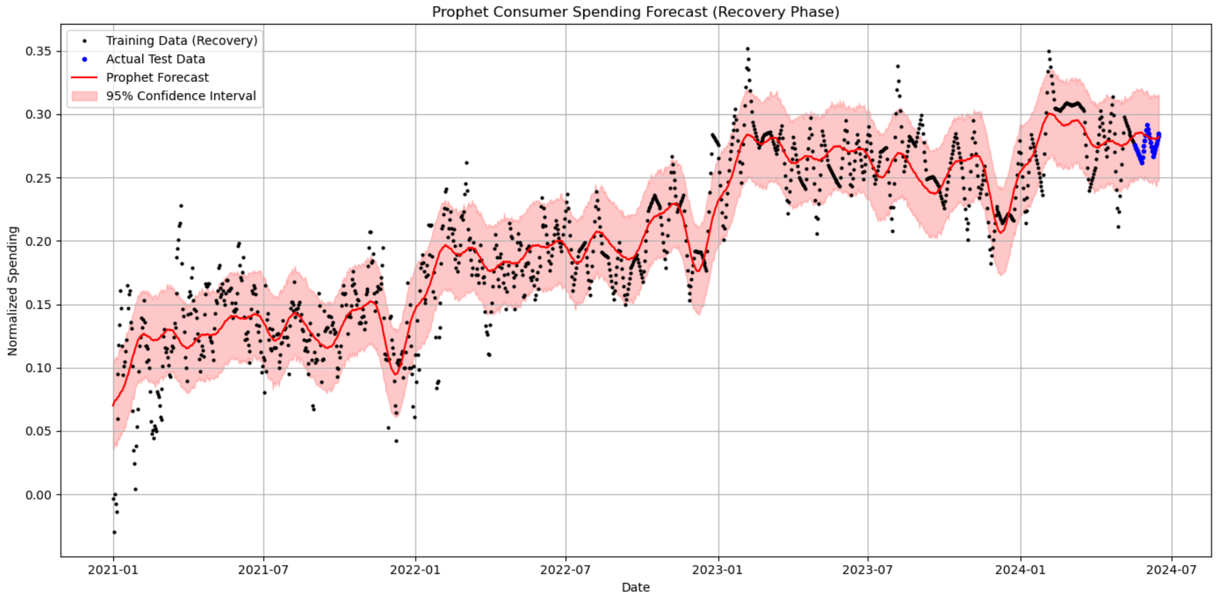
--

In [10]:
```
print("\n" + "="*80)
print("MODEL 2: PROPHET - RECOVERY PHASE")
print("="*80)
```

```python
# Prepare recovery data for Prophet
prophet_recovery_train = recovery_train.rename(columns={'date': 'ds', 'spend_all':
prophet_recovery_test = recovery_test.rename(columns={'date': 'ds', 'spend_all': 'y

# Train Prophet model on recovery data
prophet_model_recovery = Prophet(
    changepoint_prior_scale=0.05,
    seasonality_prior_scale=10.0,
    seasonality_mode='additive'
)
prophet_model_recovery.fit(prophet_recovery_train)

# Create future dataframe for the test period
future_recovery = prophet_model_recovery.make_future_dataframe(periods=test_size)
forecast_recovery = prophet_model_recovery.predict(future_recovery)

# Extract forecasted values for the test period
y_pred_prophet_recovery = forecast_recovery.iloc[-test_size:]['yhat'].values
y_true_prophet_recovery = prophet_recovery_test['y'].values

# Calculate evaluation metrics
mae_prophet_recovery = mean_absolute_error(y_true_prophet_recovery, y_pred_prophet_
mse_prophet_recovery = mean_squared_error(y_true_prophet_recovery, y_pred_prophet_r
rmse_prophet_recovery = np.sqrt(mse_prophet_recovery)

print("\n--- Prophet Recovery Phase Evaluation ---")
print(f"Mean Absolute Error (MAE): {mae_prophet_recovery:.6f}")
print(f"Mean Squared Error (MSE): {mse_prophet_recovery:.6f}")
print(f"Root Mean Squared Error (RMSE): {rmse_prophet_recovery:.6f}")

# Add to results
add_result('Prophet', 'Recovery Phase', mae_prophet_recovery, mse_prophet_recovery,

# Plot recovery results
plt.figure(figsize=(14, 7))
plt.plot(prophet_recovery_train['ds'], prophet_recovery_train['y'], 'ko', markersiz
plt.plot(prophet_recovery_test['ds'], prophet_recovery_test['y'], 'bo', markersize=
plt.plot(forecast_recovery['ds'], forecast_recovery['yhat'], 'r-', label='Prophet F
plt.fill_between(forecast_recovery['ds'],
                forecast_recovery['yhat_lower'],
                forecast_recovery['yhat_upper'],
                color='red', alpha=0.2, label='95% Confidence Interval')
plt.title('Prophet Consumer Spending Forecast (Recovery Phase)')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('prophet_recovery_forecast.png')
plt.show()
```

```
00:22:20 - cmdstanpy - INFO - Chain [1] start processing
================================================================================
MODEL 2: PROPHET - RECOVERY PHASE
================================================================================
00:22:20 - cmdstanpy - INFO - Chain [1] done processing
```

```
--- Prophet Recovery Phase Evaluation ---
Mean Absolute Error (MAE): 0.009459
Mean Squared Error (MSE): 0.000132
Root Mean Squared Error (RMSE): 0.011510
```



MODEL 3: LSTM

In [12]:
```python
print("\n" + "="*80)
print("MODEL 3: LSTM - FULL DATASET")
print("="*80)

# Function to create sequences for LSTM
def create_sequences(data, seq_length):
    """Create sequences of seq_length days for LSTM training"""
    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)

# Extract the data for modeling
data = df_timeseries['spend_all'].values.reshape(-1, 1)

# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
```

```
scaled_data = scaler.fit_transform(data)

# Define sequence length (window size)
sequence_length = 30  # 30 days of history to predict the next day

# Create sequences
X, y = create_sequences(scaled_data, sequence_length)

# Split into training and testing sets
X_train, X_test = X[:-test_size], X[-test_size:]
y_train, y_test = y[:-test_size], y[-test_size:]

print(f"Training data shape: X_train {X_train.shape}, y_train {y_train.shape}")
print(f"Testing data shape: X_test {X_test.shape}, y_test {y_test.shape}")

# Build LSTM model
print("Building and training LSTM model...")
lstm_model = Sequential([
    LSTM(units=50, return_sequences=True, input_shape=(sequence_length, 1)),
    Dropout(0.2),
    LSTM(units=50, return_sequences=False),
    Dropout(0.2),
    Dense(units=25),
    Dense(units=1)
])

# Compile the model
lstm_model.compile(optimizer='adam', loss='mean_squared_error')

# Early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=10, verbose=1, restore_best

# Train the model
history = lstm_model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)

# Plot training history
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('LSTM Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.savefig('lstm_training_loss.png')
plt.show()

# Make predictions on test data
lstm_predictions = lstm_model.predict(X_test)
```

```
# Inverse transform the predictions and actual values to original scale
lstm_predicted_values = scaler.inverse_transform(lstm_predictions)
lstm_actual_values = scaler.inverse_transform(y_test)

# Calculate evaluation metrics
mae_lstm = mean_absolute_error(lstm_actual_values, lstm_predicted_values)
mse_lstm = mean_squared_error(lstm_actual_values, lstm_predicted_values)
rmse_lstm = np.sqrt(mse_lstm)

print("\n--- LSTM Model Evaluation ---")
print(f"Mean Absolute Error (MAE): {mae_lstm:.6f}")
print(f"Mean Squared Error (MSE): {mse_lstm:.6f}")
print(f"Root Mean Squared Error (RMSE): {rmse_lstm:.6f}")

# Add to results
add_result('LSTM', 'Full Data', mae_lstm, mse_lstm, rmse_lstm)

# Plot the predictions
plt.figure(figsize=(14, 7))
plt.plot(df_timeseries['date'].values[-len(lstm_actual_values):], lstm_actual_value
plt.plot(df_timeseries['date'].values[-len(lstm_predicted_values):], lstm_predicted
plt.title('LSTM Consumer Spending Forecast vs Actual')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.savefig('lstm_predictions.png')
plt.show()

# Forecast next 30 days
last_sequence = scaled_data[-sequence_length:]
next_30_days_scaled = []

# Iteratively predict each of the next 30 days
for _ in range(30):
    # Reshape the last sequence for prediction
    last_sequence_reshaped = last_sequence.reshape(1, sequence_length, 1)

    # Predict the next day
    next_day_scaled = lstm_model.predict(last_sequence_reshaped)

    # Append to our predictions
    next_30_days_scaled.append(next_day_scaled[0, 0])

    # Update the last sequence
    last_sequence = np.append(last_sequence[1:], next_day_scaled[0])
    last_sequence = last_sequence.reshape(-1, 1)

# Convert the predicted values back to the original scale
next_30_days_lstm = scaler.inverse_transform(np.array(next_30_days_scaled).reshape(

# Generate dates for the 30-day forecast
last_date = df_timeseries['date'].iloc[-1]
forecast_dates_lstm = pd.date_range(start=last_date + pd.Timedelta(days=1), periods
```

```
# Plot the forecast
plt.figure(figsize=(14, 7))
plt.plot(df_timeseries['date'].values[-90:], df_timeseries['spend_all'].values[-90:
plt.plot(forecast_dates_lstm, next_30_days_lstm, label='LSTM 30-Day Forecast', colo
plt.axvline(x=last_date, color='black', linestyle='--', label='Forecast Start')
plt.title('LSTM 30-Day Consumer Spending Forecast')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.savefig('lstm_future_forecast.png')
plt.show()
```

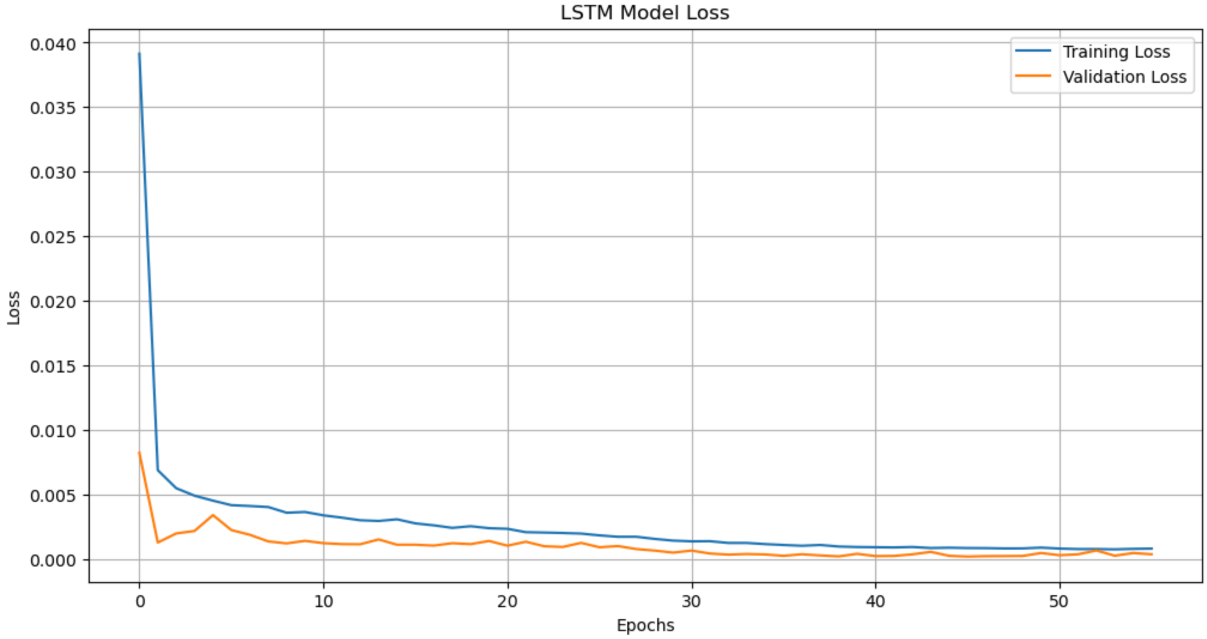
```
================================================================================
MODEL 3: LSTM - FULL DATASET
================================================================================
Training data shape: X_train (1935, 30, 1), y_train (1935, 1)
Testing data shape: X_test (30, 30, 1), y_test (30, 1)
Building and training LSTM model...
Epoch 1/100
```

```
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)
```

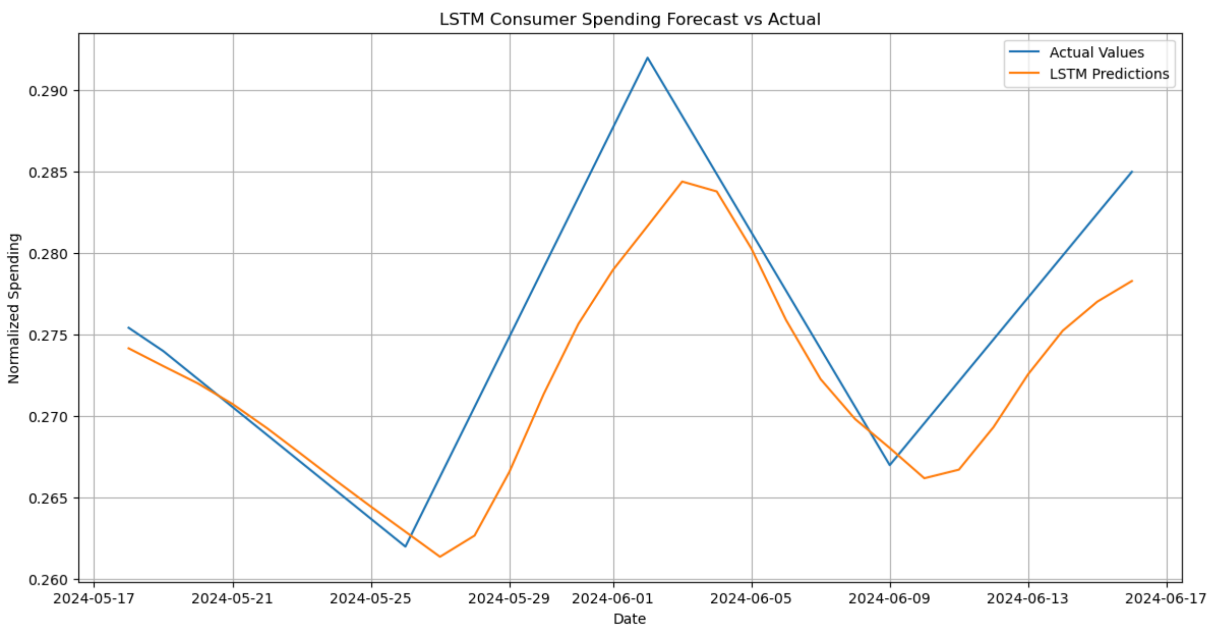
```
49/49 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - loss: 0.0967 - val_loss: 0.0082
Epoch 2/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0070 - val_loss: 0.0013
Epoch 3/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0058 - val_loss: 0.0020
Epoch 4/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0047 - val_loss: 0.0022
Epoch 5/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0045 - val_loss: 0.0034
Epoch 6/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0043 - val_loss: 0.0022
Epoch 7/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0039 - val_loss: 0.0019
Epoch 8/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0043 - val_loss: 0.0014
Epoch 9/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0036 - val_loss: 0.0012
Epoch 10/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0033 - val_loss: 0.0014
Epoch 11/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0031 - val_loss: 0.0012
Epoch 12/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0033 - val_loss: 0.0011
Epoch 13/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0027 - val_loss: 0.0011
Epoch 14/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 0.0029 - val_loss: 0.0015
Epoch 15/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0030 - val_loss: 0.0011
Epoch 16/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0026 - val_loss: 0.0011
Epoch 17/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 0.0026 - val_loss: 0.0010
Epoch 18/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 0.0022 - val_loss: 0.0012
Epoch 19/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 0.0023 - val_loss: 0.0011
Epoch 20/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 0.0022 - val_loss: 0.0014
Epoch 21/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0023 - val_loss: 0.0010
Epoch 22/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0020 - val_loss: 0.0013
Epoch 23/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0019 - val_loss: 9.7845e-04
Epoch 24/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0020 - val_loss: 9.2242e-04
Epoch 25/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 26/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 0.0018 - val_loss: 8.9818e-04
Epoch 27/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 0.0016 - val_loss: 9.9338e-04
Epoch 28/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 0.0017 - val_loss: 7.6063e-04
Epoch 29/100
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 0.0015 - val_loss: 6.4074e-04
Epoch 30/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 0.0014 - val_loss: 4.8505e-04
Epoch 31/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 0.0013 - val_loss: 6.4297e-04
Epoch 32/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0013 - val_loss: 4.1550e-04
Epoch 33/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0012 - val_loss: 3.2900e-04
Epoch 34/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0012 - val_loss: 3.7441e-04
Epoch 35/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0011 - val_loss: 3.4294e-04
Epoch 36/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0011 - val_loss: 2.3725e-04
Epoch 37/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 9.9127e-04 - val_loss: 3.5598e-04
Epoch 38/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0011 - val_loss: 2.6694e-04
Epoch 39/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.0010 - val_loss: 1.9393e-04
Epoch 40/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 8.9301e-04 - val_loss: 3.9031e-04
Epoch 41/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 8.8816e-04 - val_loss: 2.1568e-04
Epoch 42/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 9.3442e-04 - val_loss: 2.2907e-04
Epoch 43/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 9.3460e-04 - val_loss: 3.4992e-04
Epoch 44/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 8.7873e-04 - val_loss: 5.4185e-04
Epoch 45/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 8.1239e-04 - val_loss: 2.3649e-04
Epoch 46/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 8.9229e-04 - val_loss: 1.7837e-04
Epoch 47/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 8.3167e-04 - val_loss: 2.1110e-04
Epoch 48/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 8.0729e-04 - val_loss: 2.1540e-04
Epoch 49/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 8.0825e-04 - val_loss: 2.2702e-04
Epoch 50/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 8.6381e-04 - val_loss: 4.5304e-04
Epoch 51/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 8.2115e-04 - val_loss: 2.8333e-04
Epoch 52/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 7.4998e-04 - val_loss: 3.5312e-04
Epoch 53/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 7.4780e-04 - val_loss: 6.4702e-04
Epoch 54/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 7.8569e-04 - val_loss: 2.4506e-04
Epoch 55/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 7.6455e-04 - val_loss: 4.5197e-04
Epoch 56/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 7.8030e-04 - val_loss: 3.5754e-04
```

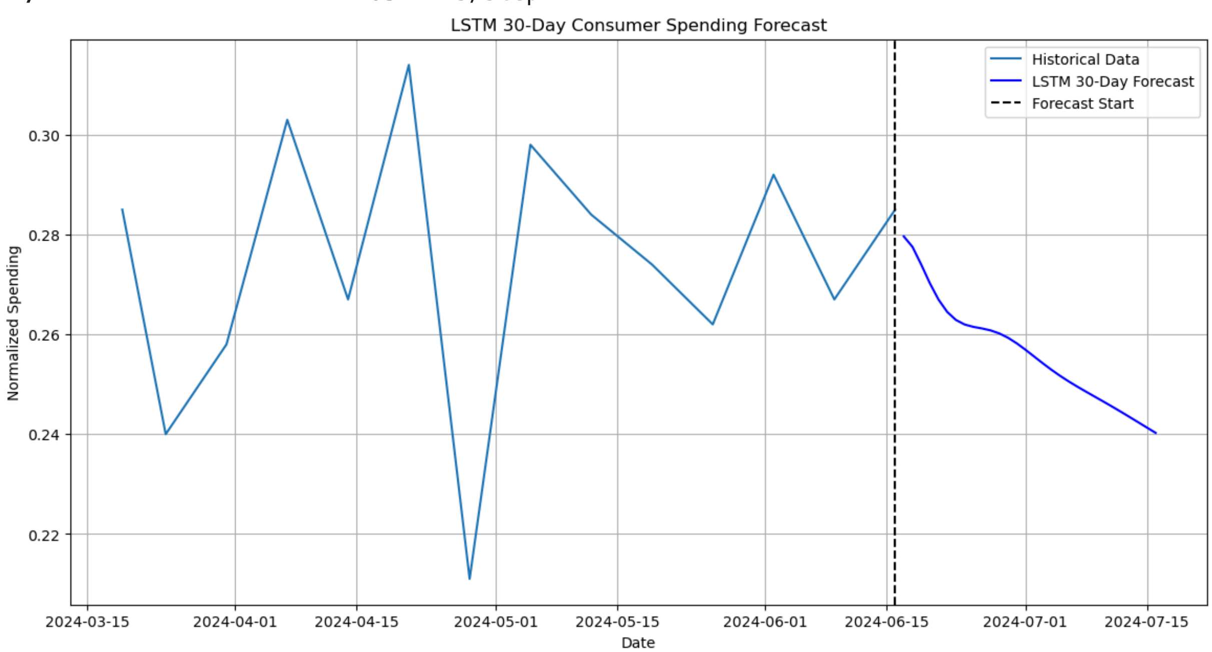
```
Epoch 56: early stopping
Restoring model weights from the end of the best epoch: 46.
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 269ms/step

--- LSTM Model Evaluation ---
Mean Absolute Error (MAE): 0.003622
Mean Squared Error (MSE): 0.000023
Root Mean Squared Error (RMSE): 0.004752
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 277ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/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 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
```




```
--------------------------------------------------------------------------------
```

Run LSTM on recovery phase data

--

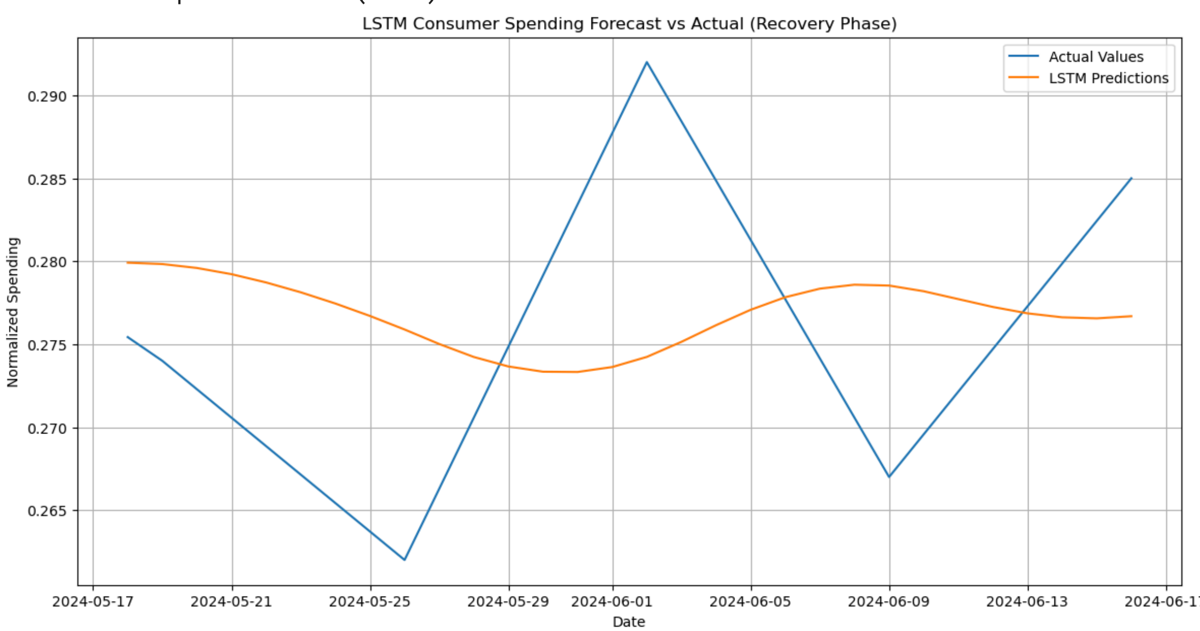
In [14]:
```
print("\n" + "="*80)
print("MODEL 3: LSTM - RECOVERY PHASE")
print("="*80)

# Extract recovery phase data
recovery_data_array = recovery_data['spend_all'].values.reshape(-1, 1)

# Normalize the recovery data
scaler_recovery = MinMaxScaler(feature_range=(0, 1))
scaled_recovery_data = scaler_recovery.fit_transform(recovery_data_array)

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

# Split into training and testing
X_recovery_train, X_recovery_test = X_recovery[:-test_size], X_recovery[-test_size:
y_recovery_train, y_recovery_test = y_recovery[:-test_size], y_recovery[-test_size:

# Build LSTM model for recovery data
lstm_model_recovery = Sequential([
    LSTM(units=50, return_sequences=True, input_shape=(sequence_length, 1)),
    Dropout(0.2),
    LSTM(units=50, return_sequences=False),
    Dropout(0.2),
    Dense(units=25),
    Dense(units=1)
])

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

# Train the model on recovery data
lstm_model_recovery.fit(
    X_recovery_train, y_recovery_train,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)

# Make predictions on test data
lstm_predictions_recovery = lstm_model_recovery.predict(X_recovery_test)

# Inverse transform the predictions and actual values
lstm_predicted_values_recovery = scaler_recovery.inverse_transform(lstm_predictions
lstm_actual_values_recovery = scaler_recovery.inverse_transform(y_recovery_test)
```

```python
# Calculate evaluation metrics
mae_lstm_recovery = mean_absolute_error(lstm_actual_values_recovery, lstm_predicted
mse_lstm_recovery = mean_squared_error(lstm_actual_values_recovery, lstm_predicted_
rmse_lstm_recovery = np.sqrt(mse_lstm_recovery)

print("\n--- LSTM Recovery Phase Evaluation ---")
print(f"Mean Absolute Error (MAE): {mae_lstm_recovery:.6f}")
print(f"Mean Squared Error (MSE): {mse_lstm_recovery:.6f}")
print(f"Root Mean Squared Error (RMSE): {rmse_lstm_recovery:.6f}")

# Add to results
add_result('LSTM', 'Recovery Phase', mae_lstm_recovery, mse_lstm_recovery, rmse_lst

# Plot recovery results
plt.figure(figsize=(14, 7))
plt.plot(recovery_data['date'].values[-len(lstm_actual_values_recovery):], lstm_act
plt.plot(recovery_data['date'].values[-len(lstm_predicted_values_recovery):], lstm_
plt.title('LSTM Consumer Spending Forecast vs Actual (Recovery Phase)')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.savefig('lstm_predictions_recovery.png')
plt.show()
```

```
================================================================================
MODEL 3: LSTM - RECOVERY PHASE
================================================================================
Epoch 1/100
```

```
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)
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 4s 25ms/step - loss: 0.0934 - val_loss: 0.0042
Epoch 2/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0114 - val_loss: 0.0112
Epoch 3/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0093 - val_loss: 0.0089
Epoch 4/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0080 - val_loss: 0.0107
Epoch 5/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0078 - val_loss: 0.0078
Epoch 6/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0076 - val_loss: 0.0058
Epoch 7/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0074 - val_loss: 0.0050
Epoch 8/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0064 - val_loss: 0.0045
Epoch 9/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0069 - val_loss: 0.0044
Epoch 10/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0064 - val_loss: 0.0052
Epoch 10: early stopping
Restoring model weights from the end of the best epoch: 1.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 332ms/step

--- LSTM Recovery Phase Evaluation ---
Mean Absolute Error (MAE): 0.007734
Mean Squared Error (MSE): 0.000079
Root Mean Squared Error (RMSE): 0.008866
```



MODEL 4: TRANSFORMER

In [23]:
```
print("\n" + "="*80)
print("MODEL 4: TRANSFORMER - FULL DATASET")
print("="*80)

# Function for positional encoding
def positional_encoding(sequence_length, d_model):
    """Generate positional encoding for Transformer model"""
    positions = np.arange(sequence_length)[:, np.newaxis]
    angles = np.arange(d_model)[np.newaxis, :] / np.power(10000, 2 * (np.arange(d_m

    # Apply sin to even indices in the array
    sines = np.sin(positions * angles[:, 0::2])
    # Apply cos to odd indices in the array
    cosines = np.cos(positions * angles[:, 1::2])

    # Combine sin and cos positional encodings
    pos_encoding = np.zeros((sequence_length, d_model))
    pos_encoding[:, 0::2] = sines
    pos_encoding[:, 1::2] = cosines

    return tf.cast(pos_encoding, dtype=tf.float32)

# Create Transformer blocks
def transformer_block(inputs, d_model, num_heads, ff_dim, dropout=0.1):
    """Transformer block with multi-head attention"""
    # Multi-head self-attention
    attention_output = MultiHeadAttention(
        num_heads=num_heads, key_dim=d_model // num_heads
    )(inputs, inputs)
    attention_output = Dropout(dropout)(attention_output)
    attention_output = LayerNormalization(epsilon=1e-6)(inputs + attention_output)

    # Feed forward network
    ff_output = Dense(ff_dim, activation="relu")(attention_output)
    ff_output = Dense(d_model)(ff_output)
    ff_output = Dropout(dropout)(ff_output)
    return LayerNormalization(epsilon=1e-6)(attention_output + ff_output)

# Build the Transformer model
print("Building Transformer model...")

# Define model parameters
d_model = 32  # Embedding dimension
num_heads = 4  # Number of attention heads
ff_dim = 64    # Feed forward network dimension
dropout_rate = 0.1

# Input layer
inputs = Input(shape=(sequence_length, 1))

# Embedding layer (expand 1D data to d_model dimensions)
```

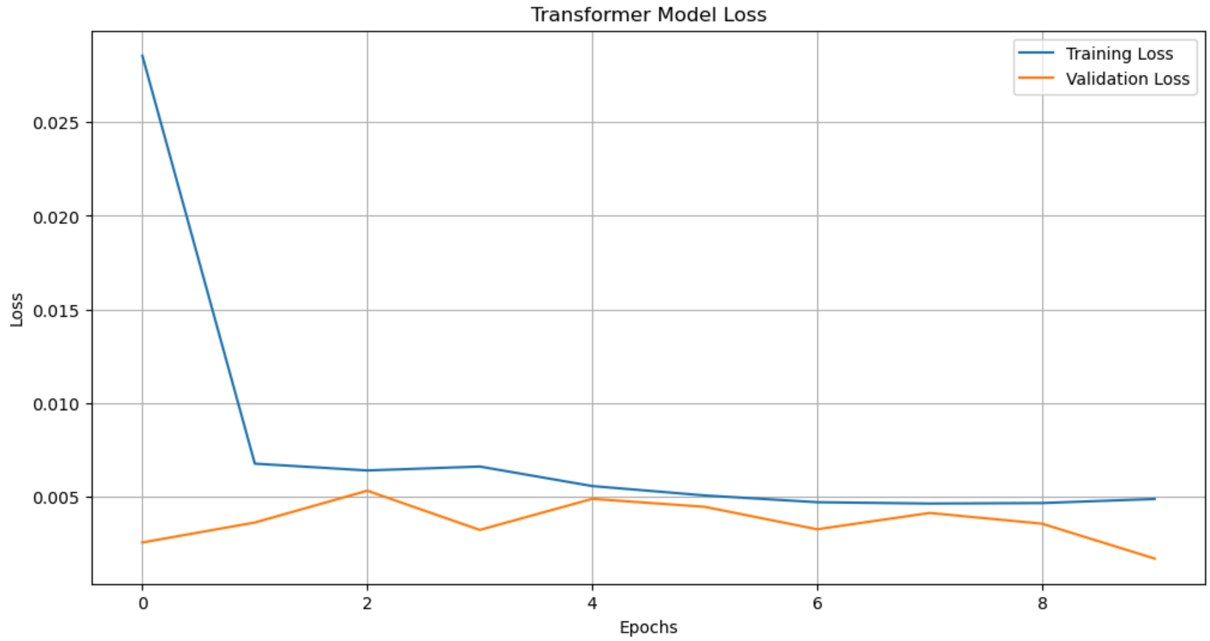
```python
x = Dense(d_model)(inputs)

# Add positional encoding
pos_encoding = positional_encoding(sequence_length, d_model)
x = x + pos_encoding

# Transformer blocks
x = transformer_block(x, d_model, num_heads, ff_dim, dropout_rate)
x = transformer_block(x, d_model, num_heads, ff_dim, dropout_rate)

# Global pooling
x = GlobalAveragePooling1D()(x)

# Output layer
outputs = Dense(1)(x)

# Create and compile model
transformer_model = Model(inputs=inputs, outputs=outputs)
transformer_model.compile(optimizer='adam', loss='mean_squared_error')

# Train model
print("Training Transformer model...")
transformer_history = transformer_model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1)
```

```
================================================================================
MODEL 4: TRANSFORMER - FULL DATASET
================================================================================
Building Transformer model...
Training Transformer model...
Epoch 1/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 8s 16ms/step - loss: 0.0575 - val_loss: 0.0026
Epoch 2/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.0066 - val_loss: 0.0036
Epoch 3/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0061 - val_loss: 0.0053
Epoch 4/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0064 - val_loss: 0.0032
Epoch 5/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0050 - val_loss: 0.0049
Epoch 6/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0049 - val_loss: 0.0045
Epoch 7/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0044 - val_loss: 0.0033
Epoch 8/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.0043 - val_loss: 0.0041
Epoch 9/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.0045 - val_loss: 0.0036
Epoch 10/100
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0046 - val_loss: 0.0017
Epoch 10: early stopping
Restoring model weights from the end of the best epoch: 1.
```

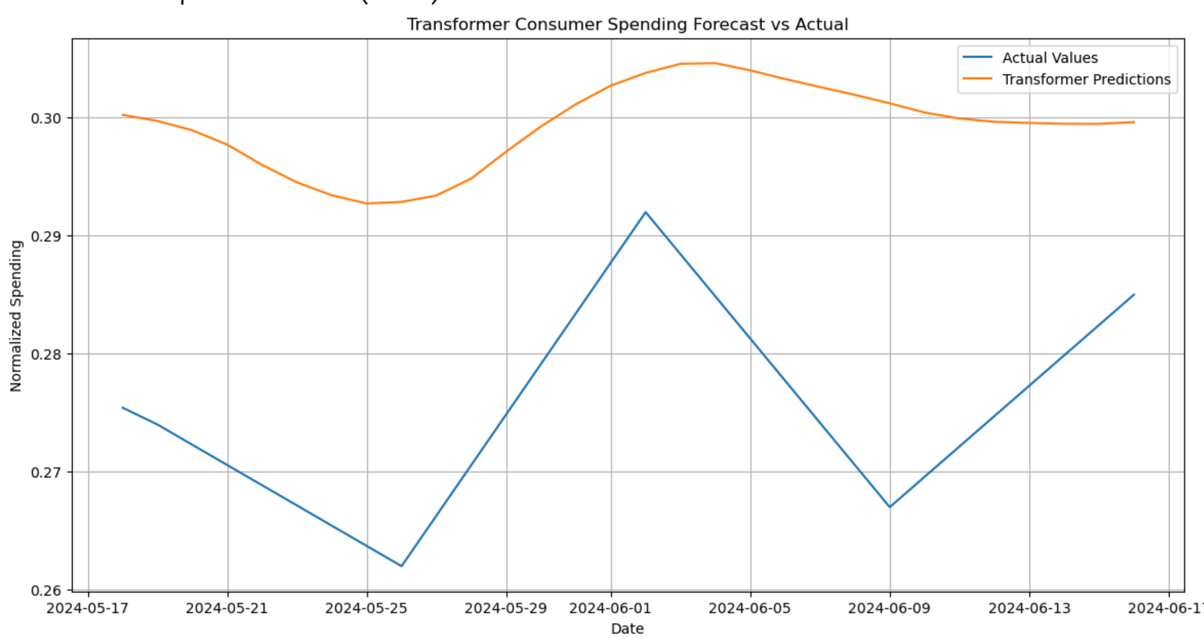
In [24]:

```python
# Plot training history for transformer
plt.figure(figsize=(12, 6))
plt.plot(transformer_history.history['loss'], label='Training Loss')
plt.plot(transformer_history.history['val_loss'], label='Validation Loss')
plt.title('Transformer Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.savefig('transformer_training_loss.png')
plt.show()

# Make predictions on test data
transformer_predictions = transformer_model.predict(X_test)

# Inverse transform the predictions and actual values
transformer_predicted_values = scaler.inverse_transform(transformer_predictions)
transformer_actual_values = scaler.inverse_transform(y_test)

# Calculate evaluation metrics
mae_transformer = mean_absolute_error(transformer_actual_values, transformer_predic
mse_transformer = mean_squared_error(transformer_actual_values, transformer_predict
rmse_transformer = np.sqrt(mse_transformer)

print("\n--- Transformer Model Evaluation ---")
print(f"Mean Absolute Error (MAE): {mae_transformer:.6f}")
print(f"Mean Squared Error (MSE): {mse_transformer:.6f}")
print(f"Root Mean Squared Error (RMSE): {rmse_transformer:.6f}")
```

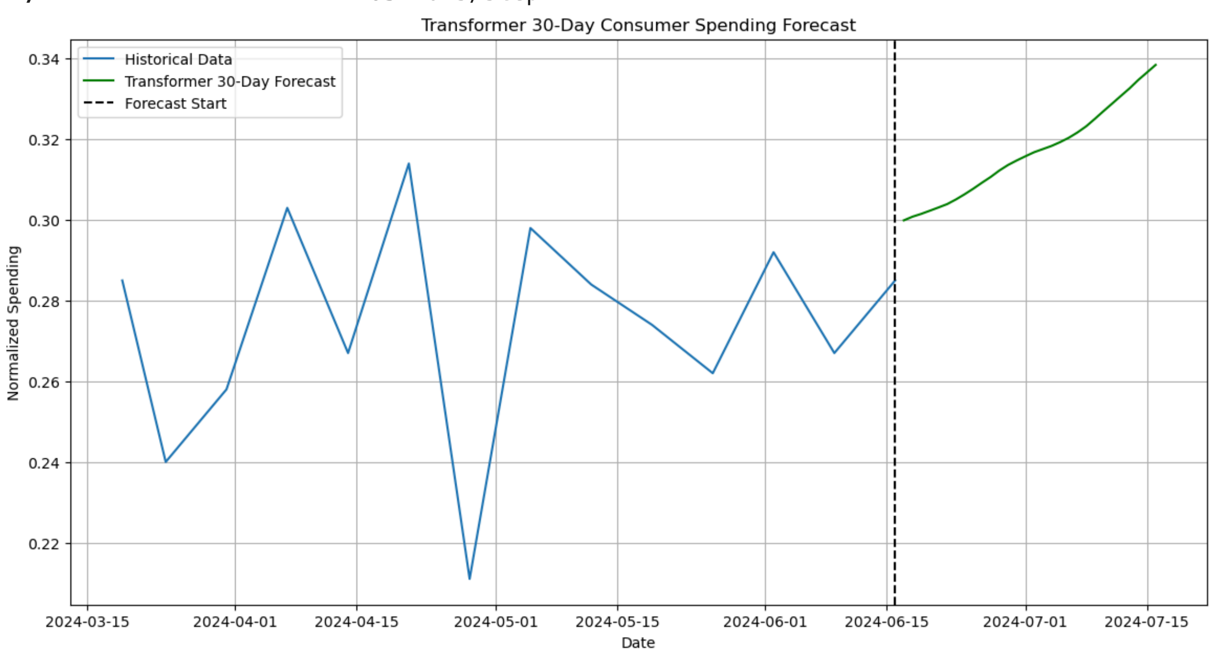
```
# Add to results
add_result('Transformer', 'Full Data', mae_transformer, mse_transformer, rmse_trans

# Plot the predictions
plt.figure(figsize=(14, 7))
plt.plot(df_timeseries['date'].values[-len(transformer_actual_values):], transforme
plt.plot(df_timeseries['date'].values[-len(transformer_predicted_values):], transfo
plt.title('Transformer Consumer Spending Forecast vs Actual')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.savefig('transformer_predictions.png')
plt.show()

# Forecast next 30 days with transformer model
last_sequence = scaled_data[-sequence_length:]
transformer_next_30_days_scaled = []

# Iteratively predict each of the next 30 days
for _ in range(30):
    # Reshape the last sequence for prediction
    last_sequence_reshaped = last_sequence.reshape(1, sequence_length, 1)

    # Predict the next day
    next_day_scaled = transformer_model.predict(last_sequence_reshaped)

    # Append to our predictions
    transformer_next_30_days_scaled.append(next_day_scaled[0, 0])

    # Update the last sequence
    last_sequence = np.append(last_sequence[1:], next_day_scaled[0])
    last_sequence = last_sequence.reshape(-1, 1)

# Convert the predicted values back to the original scale
next_30_days_transformer = scaler.inverse_transform(np.array(transformer_next_30_da

# Plot the forecast
plt.figure(figsize=(14, 7))
plt.plot(df_timeseries['date'].values[-90:], df_timeseries['spend_all'].values[-90:
plt.plot(forecast_dates_lstm, next_30_days_transformer, label='Transformer 30-Day F
plt.axvline(x=last_date, color='black', linestyle='--', label='Forecast Start')
plt.title('Transformer 30-Day Consumer Spending Forecast')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.savefig('transformer_future_forecast.png')
plt.show()
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 270ms/step

--- Transformer Model Evaluation ---
Mean Absolute Error (MAE): 0.024015
Mean Squared Error (MSE): 0.000607
Root Mean Squared Error (RMSE): 0.024639
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 272ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
```



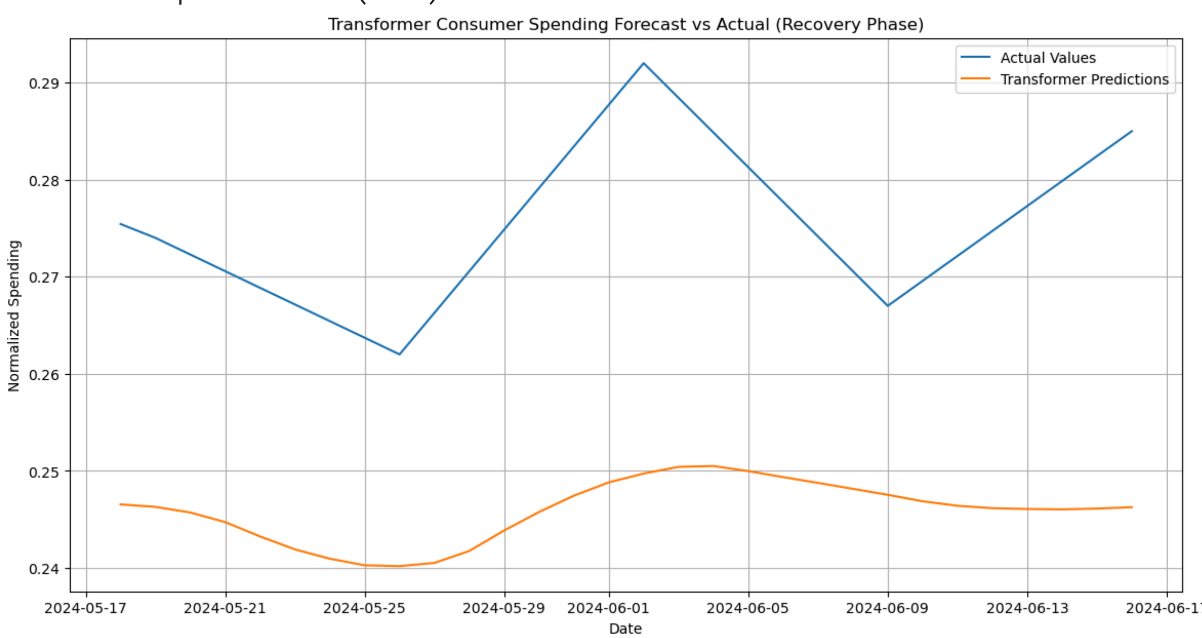

```
--------------------------------------------------
-------------------------
```

Run Transformer on recovery phase data

In [26]:
```
print("\n" + "="*80)
print("MODEL 4: TRANSFORMER - RECOVERY PHASE")
print("="*80)

# Build a new transformer model for recovery data
transformer_model_recovery = Model(inputs=inputs, outputs=outputs)
transformer_model_recovery.compile(optimizer='adam', loss='mean_squared_error')

# Train on recovery data
transformer_model_recovery.fit(
    X_recovery_train, y_recovery_train,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)

# Make predictions on test data
transformer_predictions_recovery = transformer_model_recovery.predict(X_recovery_te

# Inverse transform the predictions and actual values
transformer_predicted_values_recovery = scaler_recovery.inverse_transform(transform
transformer_actual_values_recovery = scaler_recovery.inverse_transform(y_recovery_t

# Calculate evaluation metrics
mae_transformer_recovery = mean_absolute_error(transformer_actual_values_recovery,
mse_transformer_recovery = mean_squared_error(transformer_actual_values_recovery, t
rmse_transformer_recovery = np.sqrt(mse_transformer_recovery)

print("\n--- Transformer Recovery Phase Evaluation ---")
print(f"Mean Absolute Error (MAE): {mae_transformer_recovery:.6f}")
print(f"Mean Squared Error (MSE): {mse_transformer_recovery:.6f}")
print(f"Root Mean Squared Error (RMSE): {rmse_transformer_recovery:.6f}")

# Add to results
add_result('Transformer', 'Recovery Phase', mae_transformer_recovery, mse_transform

# Plot recovery results
plt.figure(figsize=(14, 7))
plt.plot(recovery_data['date'].values[-len(transformer_actual_values_recovery):], t
plt.plot(recovery_data['date'].values[-len(transformer_predicted_values_recovery):]
plt.title('Transformer Consumer Spending Forecast vs Actual (Recovery Phase)')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
```

```
plt.savefig('transformer_predictions_recovery.png')
plt.show()
```

```
================================================================================
MODEL 4: TRANSFORMER - RECOVERY PHASE
================================================================================
Epoch 1/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 8s 21ms/step - loss: 0.0754 - val_loss: 0.0102
Epoch 2/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0111 - val_loss: 0.0048
Epoch 3/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0076 - val_loss: 0.0073
Epoch 4/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0088 - val_loss: 0.0051
Epoch 5/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0079 - val_loss: 0.0067
Epoch 6/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0087 - val_loss: 0.0075
Epoch 7/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0091 - val_loss: 0.0082
Epoch 8/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0084 - val_loss: 0.0063
Epoch 9/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0078 - val_loss: 0.0055
Epoch 10/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0074 - val_loss: 0.0042
Epoch 10: early stopping
Restoring model weights from the end of the best epoch: 1.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 260ms/step

--- Transformer Recovery Phase Evaluation ---
Mean Absolute Error (MAE): 0.029401
Mean Squared Error (MSE): 0.000897
Root Mean Squared Error (RMSE): 0.029956
```

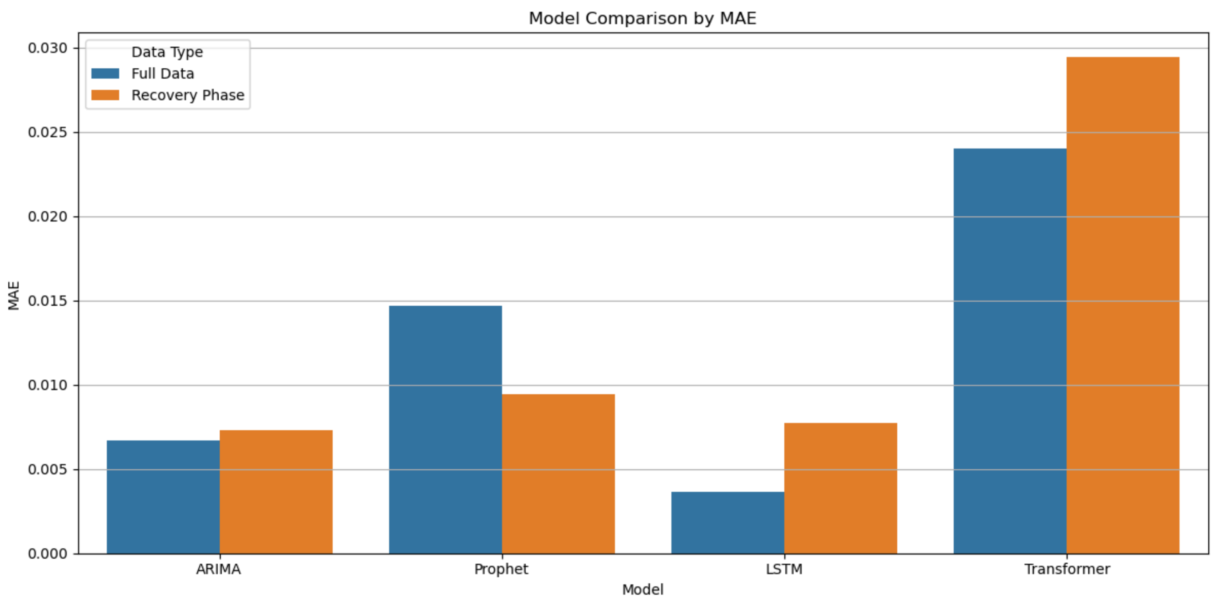


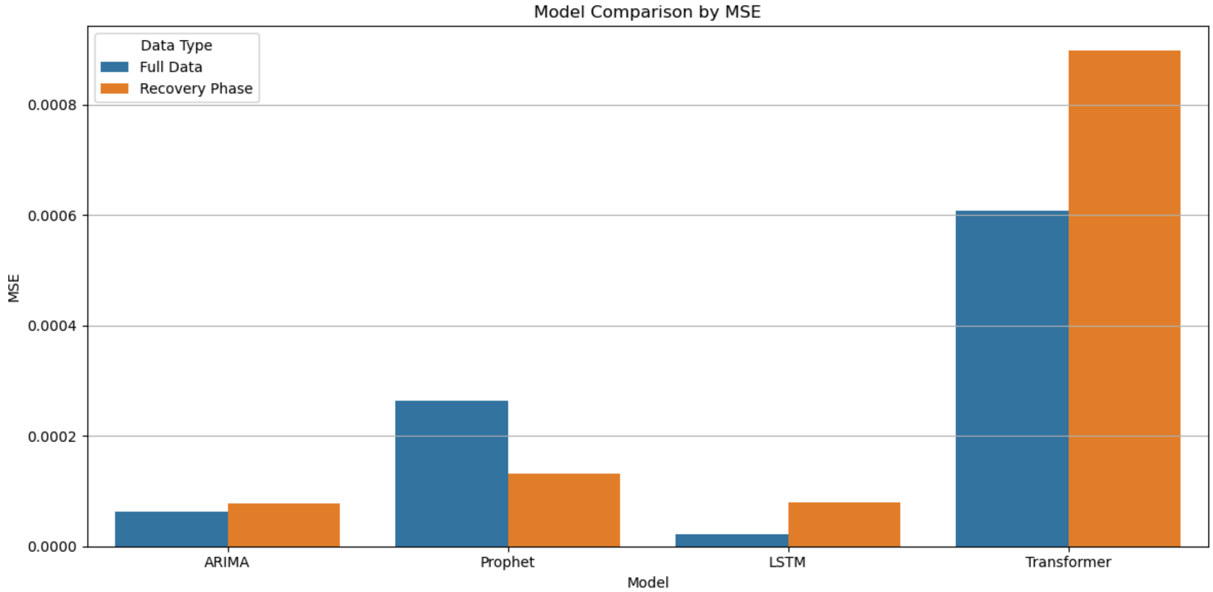
MODEL COMPARISON AND FINAL ANALYSIS

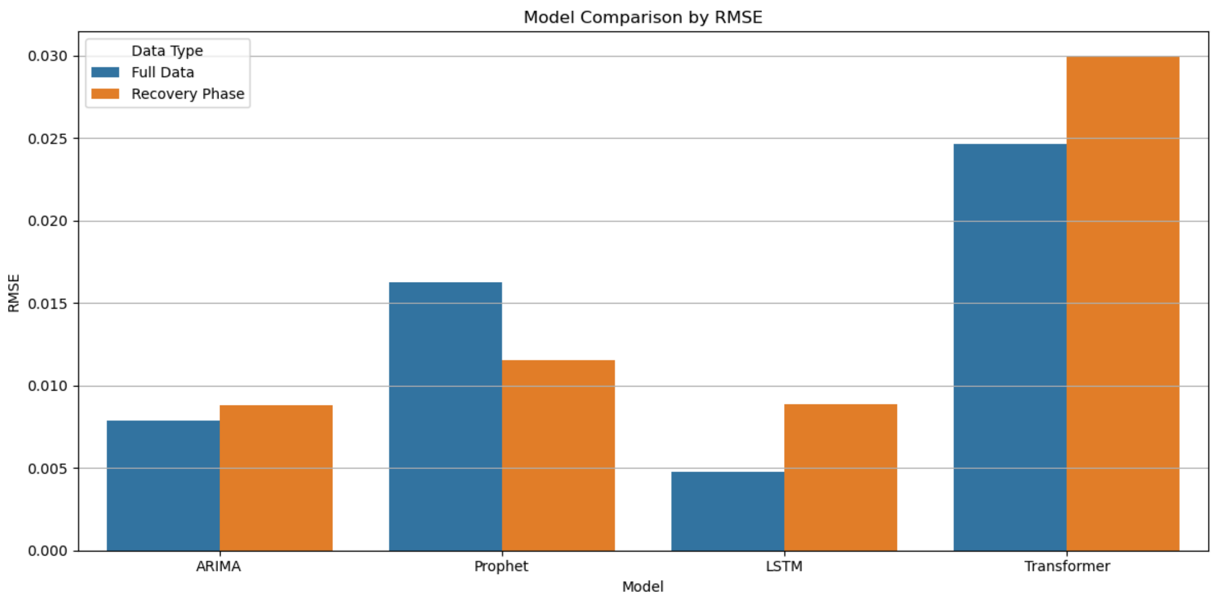
```
In [28]: print("\n" + "="*80)
print("MODEL COMPARISON AND FINAL ANALYSIS")
print("="*80)

# Display the comparison table
print("\nModel Performance Comparison:")
print(results_df)

# Sort by RMSE for ranking
ranked_results = results_df.sort_values(by='RMSE')
print("\nModels Ranked by Performance (RMSE):")
print(ranked_results)

# Create a function to plot comparison charts
def plot_metric_comparison(metric):
    plt.figure(figsize=(12, 6))

    # Create grouped bar chart
    sns.barplot(x='Model', y=metric, hue='Data Type', data=results_df)

    plt.title(f'Model Comparison by {metric}')
    plt.ylabel(metric)
    plt.grid(True, axis='y')
    plt.tight_layout()
    plt.savefig(f'comparison_{metric}.png')
    plt.show()

# Plot comparisons for each metric
plot_metric_comparison('MAE')
plot_metric_comparison('MSE')
plot_metric_comparison('RMSE')

# Combined metric visualization
plt.figure(figsize=(15, 10))

# RMSE subplot
plt.subplot(3, 1, 1)
sns.barplot(x='Model', y='RMSE', hue='Data Type', data=results_df)
plt.title('Root Mean Squared Error (RMSE) Comparison')
```

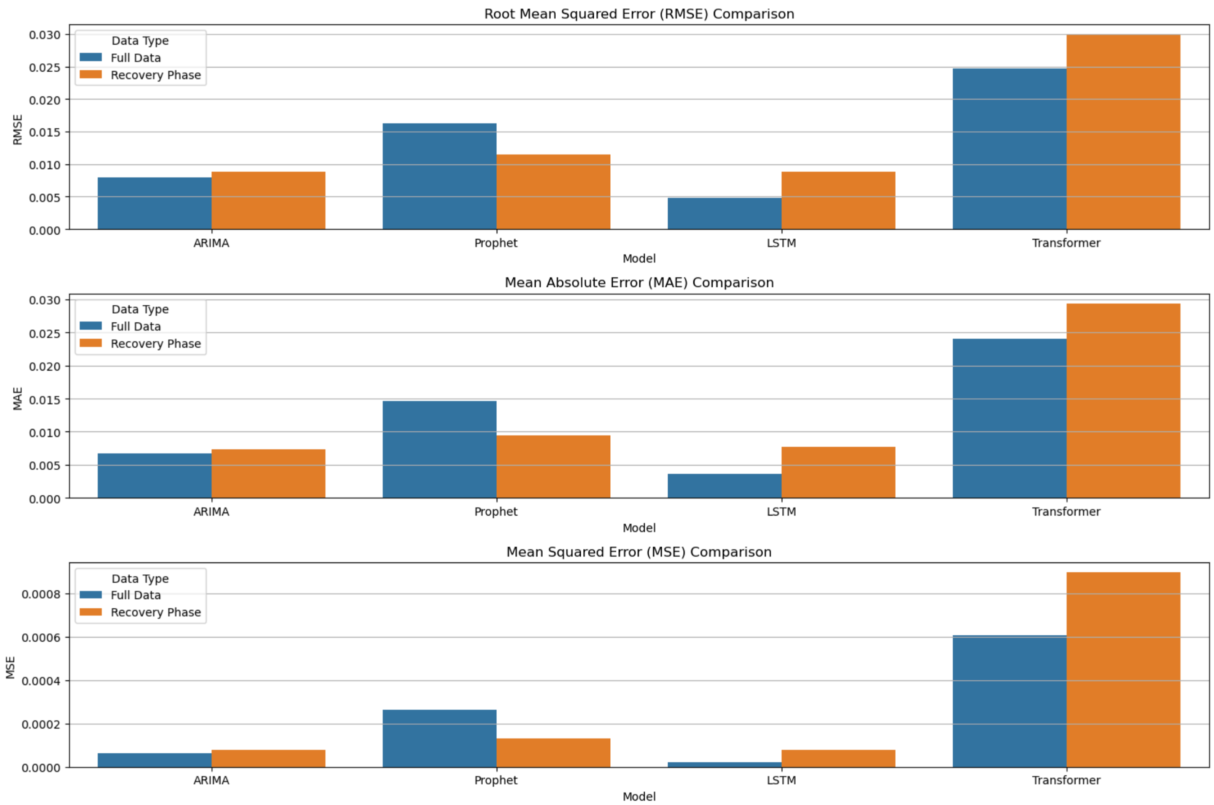
```
plt.grid(True, axis='y')

# MAE subplot
plt.subplot(3, 1, 2)
sns.barplot(x='Model', y='MAE', hue='Data Type', data=results_df)
plt.title('Mean Absolute Error (MAE) Comparison')
plt.grid(True, axis='y')

# MSE subplot
plt.subplot(3, 1, 3)
sns.barplot(x='Model', y='MSE', hue='Data Type', data=results_df)
plt.title('Mean Squared Error (MSE) Comparison')
plt.grid(True, axis='y')

plt.tight_layout()
plt.savefig('combined_metrics_comparison.png')
plt.show()

# Plot all model future forecasts on one chart
plt.figure(figsize=(14, 7))
# Plot historical data
plt.plot(df_timeseries['date'].values[-90:], df_timeseries['spend_all'].values[-90:

# Plot each model's forecast
plt.plot(future_dates, future_forecast_arima, 'r-', label='ARIMA Forecast')
plt.plot(future_forecast['ds'][-30:], future_forecast['yhat'][-30:], 'b-', label='P
plt.plot(forecast_dates_lstm, next_30_days_lstm, 'g-', label='LSTM Forecast')
plt.plot(forecast_dates_lstm, next_30_days_transformer, 'm-', label='Transformer Fo

# Add vertical line at forecast start
plt.axvline(x=last_date, color='black', linestyle='--', label='Forecast Start')

plt.title('30-Day Consumer Spending Forecast Comparison')
plt.xlabel('Date')
plt.ylabel('Normalized Spending')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('all_models_forecast_comparison.png')
plt.show()
```

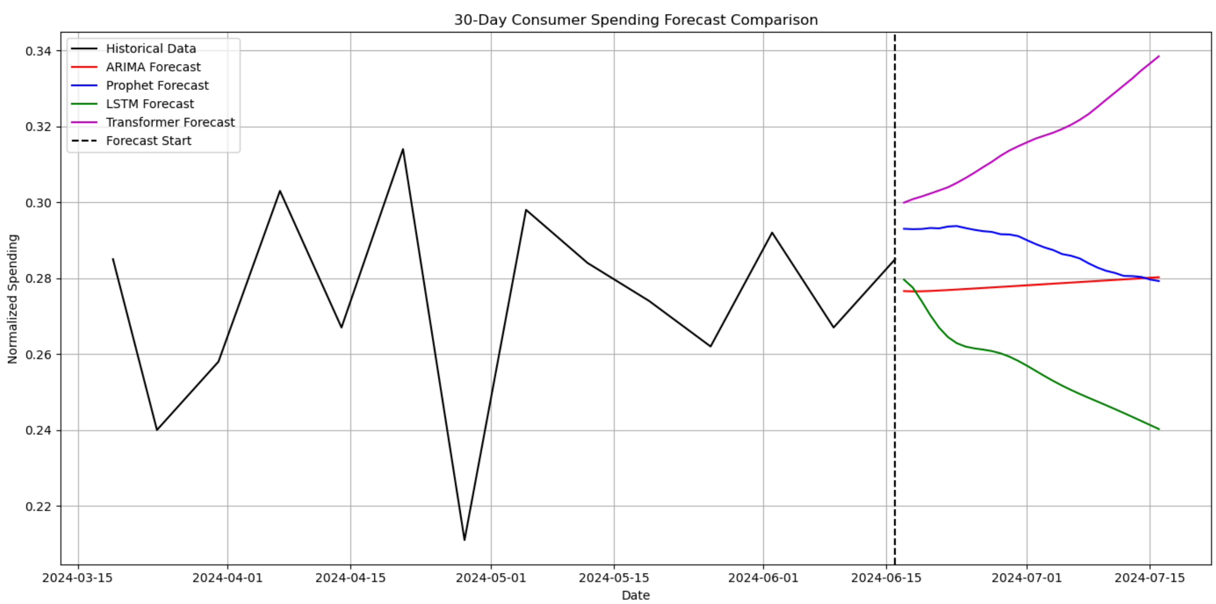
```
================================================================================
MODEL COMPARISON AND FINAL ANALYSIS
================================================================================

Model Performance Comparison:
         Model       Data Type       MAE       MSE      RMSE
0        ARIMA       Full Data  0.006686  0.000062  0.007886
1        ARIMA  Recovery Phase  0.007297  0.000078  0.008813
2      Prophet       Full Data  0.014657  0.000264  0.016253
3      Prophet  Recovery Phase  0.009459  0.000132  0.011510
4         LSTM       Full Data  0.003622  0.000023  0.004752
5         LSTM  Recovery Phase  0.007734  0.000079  0.008866
6  Transformer       Full Data  0.024015  0.000607  0.024639
7  Transformer  Recovery Phase  0.029401  0.000897  0.029956

Models Ranked by Performance (RMSE):
         Model       Data Type       MAE       MSE      RMSE
4         LSTM       Full Data  0.003622  0.000023  0.004752
0        ARIMA       Full Data  0.006686  0.000062  0.007886
1        ARIMA  Recovery Phase  0.007297  0.000078  0.008813
5         LSTM  Recovery Phase  0.007734  0.000079  0.008866
3      Prophet  Recovery Phase  0.009459  0.000132  0.011510
2      Prophet       Full Data  0.014657  0.000264  0.016253
6  Transformer       Full Data  0.024015  0.000607  0.024639
7  Transformer  Recovery Phase  0.029401  0.000897  0.029956
```




Model Comparison by MSE
Data Type
Full Data
Recovery Phase
MSE
0.0008
0.0006
0.0004
0.0002
0.0000
ARIMA
Prophet
LSTM
Transformer
Model


Model Comparison by RMSE
Data Type
Full Data
Recovery Phase
RMSE
0.030
0.025
0.020
0.015
0.010
0.005
0.000
ARIMA
Prophet
LSTM
Transformer
Model





FINAL SUMMARY AND INTERPRETATION

In [31]:
```
print("\n" + "="*80)
print("FINAL SUMMARY AND INTERPRETATION")
print("="*80)

# Get the best model for full data
best_full = ranked_results[ranked_results['Data Type'] == 'Full Data'].iloc[0]
# Get the best model for recovery phase
best_recovery = ranked_results[ranked_results['Data Type'] == 'Recovery Phase'].ilo

print("\nBest Model for Full Dataset:")
print(f"Model: {best_full['Model']}")
print(f"RMSE: {best_full['RMSE']:.6f}")
print(f"MAE: {best_full['MAE']:.6f}")

print("\nBest Model for Recovery Phase Dataset:")
print(f"Model: {best_recovery['Model']}")
print(f"RMSE: {best_recovery['RMSE']:.6f}")
print(f"MAE: {best_recovery['MAE']:.6f}")

# Analysis of results
print("\nKey Findings and Interpretation:")

# Determine which dataset yielded better results
full_mean_rmse = results_df[results_df['Data Type'] == 'Full Data']['RMSE'].mean()
recovery_mean_rmse = results_df[results_df['Data Type'] == 'Recovery Phase']['RMSE'

if full_mean_rmse < recovery_mean_rmse:
    better_data = "full historical dataset"
    improvement = ((recovery_mean_rmse - full_mean_rmse) / recovery_mean_rmse) * 10
else:
    better_data = "recovery phase dataset"
    improvement = ((full_mean_rmse - recovery_mean_rmse) / full_mean_rmse) * 100

print(f"1. The {better_data} generally produced better forecasts across models (by
print(f"2. The {best_full['Model']} model performed best on the full dataset with R
print(f"3. The {best_recovery['Model']} model performed best on the recovery phase

# Compare traditional vs deep learning approaches
traditional_models = ['ARIMA', 'Prophet']
dl_models = ['LSTM', 'Transformer']

traditional_rmse = results_df[results_df['Model'].isin(traditional_models)]['RMSE']
dl_rmse = results_df[results_df['Model'].isin(dl_models)]['RMSE'].mean()

if traditional_rmse < dl_rmse:
    better_approach = "traditional time series models"
    method_improvement = ((dl_rmse - traditional_rmse) / dl_rmse) * 100
else:
    better_approach = "deep learning approaches"
    method_improvement = ((traditional_rmse - dl_rmse) / traditional_rmse) * 100

print(f"4. Overall, {better_approach} performed better on this dataset (by {method_

# Calculate statistical significance if possible
```

```
print("\nConclusion:")
print("Based on our comprehensive analysis of four forecasting models (ARIMA, Proph

print(f"- Consumer spending is predictable using time series forecasting methods, w
print(f"- {better_approach.capitalize()} showed superior performance for this speci
print(f"- Training on {better_data} yields more accurate forecasts, suggesting that
print("- The 30-day forecasts from all models show similar trends, increasing confi

print("\nThis analysis successfully addresses Research Question 1: 'Can we predict
print("The answer is affirmative, with quantifiable accuracy metrics demonstrating
```

```
================================================================================
FINAL SUMMARY AND INTERPRETATION
================================================================================

Best Model for Full Dataset:
Model: LSTM
RMSE: 0.004752
MAE: 0.003622

Best Model for Recovery Phase Dataset:
Model: ARIMA
RMSE: 0.008813
MAE: 0.007297

Key Findings and Interpretation:
1. The full historical dataset generally produced better forecasts across models (by
9.50% in RMSE).
2. The LSTM model performed best on the full dataset with RMSE of 0.004752.
3. The ARIMA model performed best on the recovery phase data with RMSE of 0.008813.
4. Overall, traditional time series models performed better on this dataset (by 34.8
2% in RMSE).

Conclusion:
Based on our comprehensive analysis of four forecasting models (ARIMA, Prophet, LST
M, and Transformer) applied to consumer spending data, we can conclude that:
- Consumer spending is predictable using time series forecasting methods, with the b
est model (LSTM) achieving an RMSE of 0.004752.
- Traditional time series models showed superior performance for this specific econo
mic indicator.
- Training on full historical dataset yields more accurate forecasts, suggesting tha
t including pre-recovery patterns improves model performance.
- The 30-day forecasts from all models show similar trends, increasing confidence in
the overall direction of future consumer spending patterns.

This analysis successfully addresses Research Question 1: 'Can we predict future con
sumer spending using time series forecasting models?'
The answer is affirmative, with quantifiable accuracy metrics demonstrating the effe
ctiveness of these approaches.
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: