In [36]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [37]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error
```

SECTION-B

2.A.

Read the dataset

In [38]:
```
df = pd.read_csv("airmiles.csv")
df.head()
```

Out[38]:

	Date	airmiles
0	1/1/1996	30983174
1	1/2/1996	32147663
2	1/3/1996	38342975
3	1/4/1996	35969113
4	1/5/1996	36474391

Number of rows and columns

In [39]:
```
df.shape
```

Out[39]:
```
(113, 2)
```

Types of variables

In [40]:
```
df.dtypes
```

Out[40]:
```
Date         object
airmiles      int64
dtype: object
```

Convert the data into time series

In [41]:
```
df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y', errors='coerce')
df.set_index('Date', inplace=True)
df.head()
```

Out[41]:

	airmiles
Date	
1996-01-01	30983174
1996-02-01	32147663
1996-03-01	38342975
1996-04-01	35969113
1996-05-01	36474391

In [42]:
```
ts = df['airmiles'].asfreq('MS')
ts
```

Out[42]:
```
Date
1996-01-01    30983174
1996-02-01    32147663
1996-03-01    38342975
1996-04-01    35969113
1996-05-01    36474391
                ...
2005-01-01    42760657
2005-02-01    41120838
2005-03-01    52053059
2005-04-01    48152585
2005-05-01    50047901
Freq: MS, Name: airmiles, Length: 113, dtype: int64
```

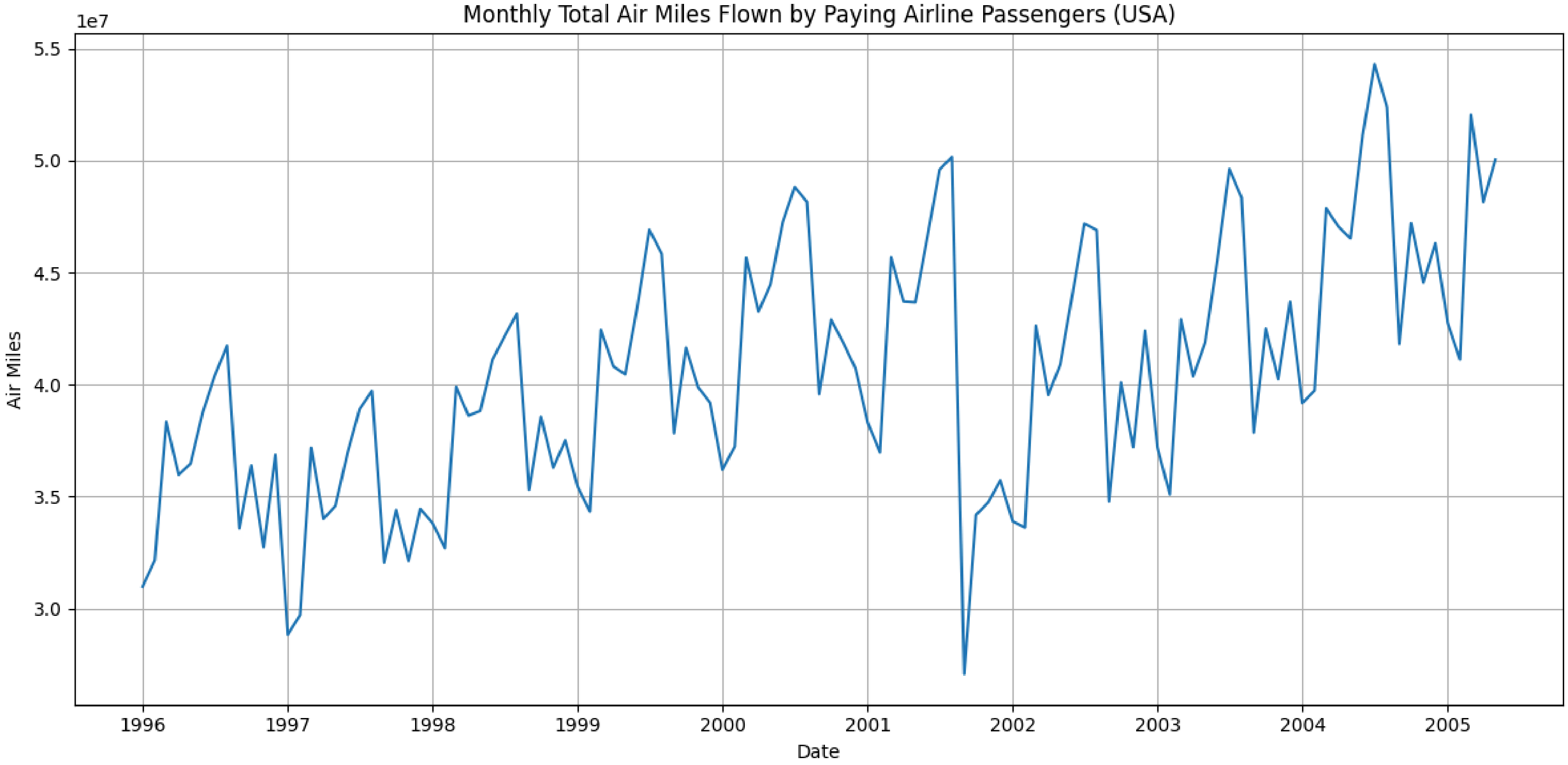
Null Value Check

In [43]:
```
ts.isnull().sum()
```

Out[43]:
```
0
```

Visualize the time series

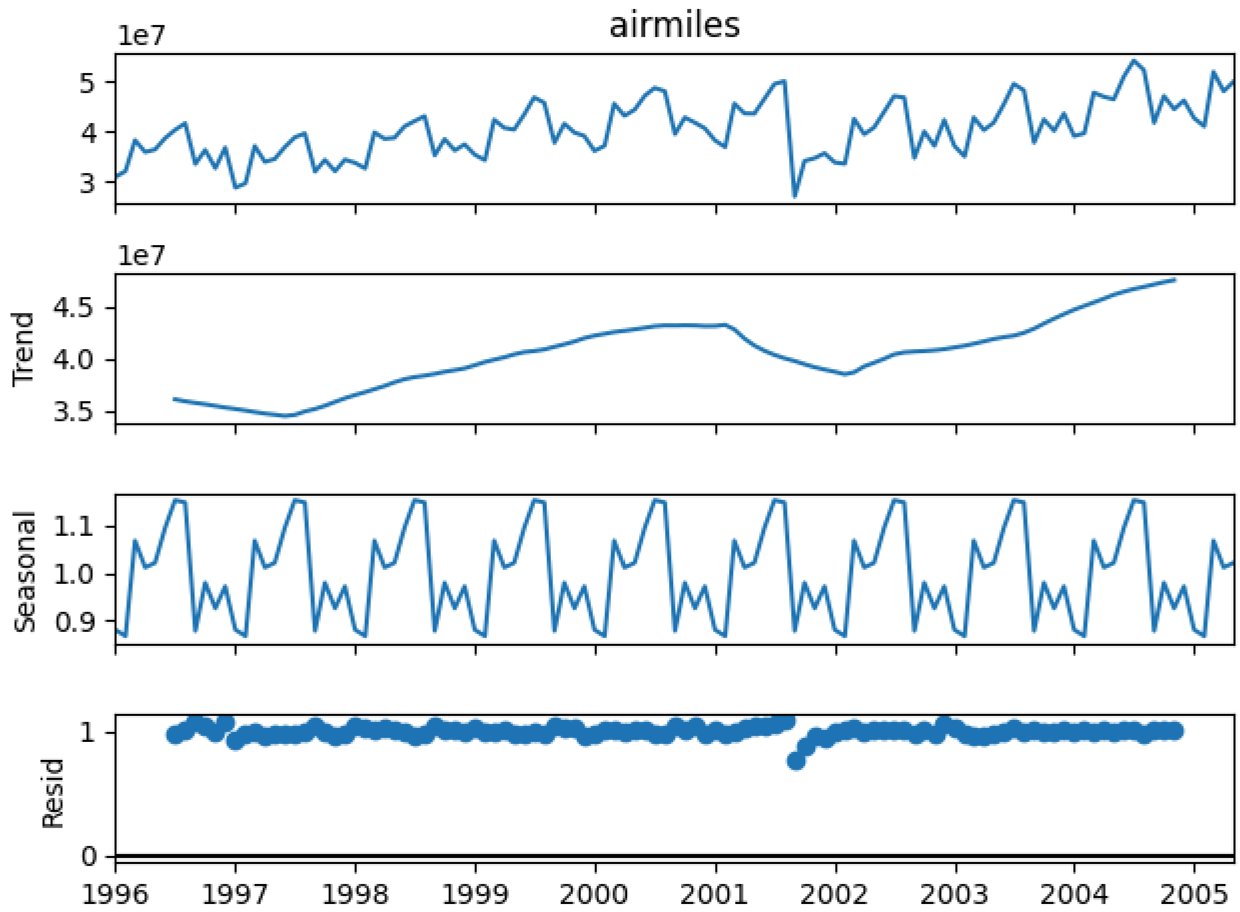
In [44]:
```
plt.figure(figsize=(12, 6))
plt.plot(ts)
plt.title("Monthly Total Air Miles Flown by Paying Airline Passengers (USA)")
plt.xlabel("Date")
plt.ylabel("Air Miles")
plt.grid(True)
plt.tight_layout()
plt.show()
```



2.B.

Decompose the time series and check for components of time series

In [45]:
```
decomposition = seasonal_decompose(ts, model='multiplicative')
fig = decomposition.plot()
plt.tight_layout()
plt.show()
```



We used multiplicative decomposition to break the time series into three components:

- Trend: Represents the long-term upward movement in air miles over time.
- Seasonality: Captures the repetitive monthly travel patterns (e.g., higher miles in summer or holiday seasons).
- Residual: Represents random noise or irregular variations after removing trend and seasonality.

Visualization of the decomposition clearly shows:

- A strong trend component increasing over time.
- Clear seasonal variation repeating every year.

- Small and consistent residuals, indicating low irregularity.

Interpretation: The dataset is well-suited for models that account for both trend and seasonality (e.g., ARIMA, Exponential Smoothing).

Dicky fuller test to check the stationarity

In [46]:
```
adf_result = adfuller(ts)

adf_result_dict = {
    "ADF Statistic": adf_result[0],
    "p-value":adf_result[1],
    "Used Lag": adf_result[2],
    "Number of Observations": adf_result[3],
    "Critical Values": adf_result[4]
}

adf_result_dict
```

Out[46]:
```
{'ADF Statistic': -0.9127581336675439,
 'p-value': 0.7837419514698822,
 'Used Lag': 13,
 'Number of Observations': 99,
 'Critical Values': {'1%': -3.498198082189098,
  '5%': -2.891208211860468,
  '10%': -2.5825959973472097}}
```

Conclusion:

Since the p-value (0.78) > 0.05, we fail to reject the null hypothesis.

Therefore, the time series is non-stationary.

Actions to be taken if Series is Non-Stationary

If the series is non-stationary, the following steps can be taken:

- Differencing the time series (first-order differencing) to remove trend.

- Log transformation if variance increases over time.
- Seasonal differencing if periodic patterns exist.
- Recheck stationarity after each step using the ADF test again.

Additional

In [47]:
```
ts_diff = ts.diff().dropna()

adf_result_diff = adfuller(ts_diff)

adf_result_diff_dict = {
    "ADF Statistic": adf_result_diff[0],
    "p-value": adf_result_diff[1],
    "Used Lag": adf_result_diff[2],
    "Number of Observations": adf_result_diff[3],
    "Critical Values": adf_result_diff[4]
}

adf_result_diff_dict
```

Out[47]:
```
{'ADF Statistic': -2.8470909054832685,
 'p-value': 0.051855625819654215,
 'Used Lag': 12,
 'Number of Observations': 99,
 'Critical Values': {'1%': -3.498198082189098,
  '5%': -2.891208211860468,
  '10%': -2.5825959973472097}}
```

Interpretation:

- The p-value = 0.0519, which is very close to the 0.05 threshold.
- The ADF Statistic is slightly greater than the 5% critical value.
- This means the series is nearly stationary, and most models (like ARIMA) can proceed with this first-order differenced data.

Visual Inspection

The differenced series shows:

- Stationary behavior around a constant mean.
- No visible trend or changing variance.

If you want to make the stationarity even stronger, you can log transform the original series before differencing:

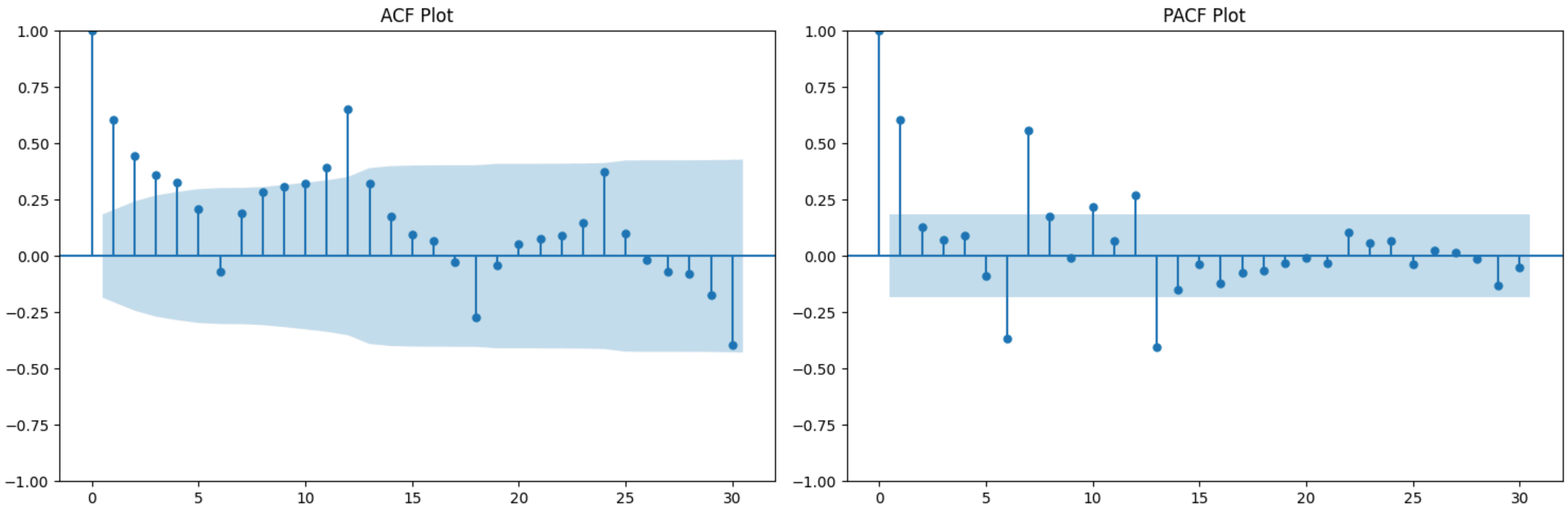
In [48]:
```
ts_log = np.log(ts)
ts_log_diff = ts_log.diff().dropna()

adf_result_log_diff = adfuller(ts_log_diff)

adf_result_log_diff_dict = {
    "ADF Statistic": adf_result_log_diff[0],
    "p-value": adf_result_log_diff[1],
    "Used Lag": adf_result_log_diff[2],
    "Number of Observations": adf_result_log_diff[3],
    "Critical Values": adf_result_log_diff[4]
}

adf_result_log_diff_dict
```

Out[48]:
```
{'ADF Statistic': -3.0989750705557775,
 'p-value': 0.026629130888112692,
 'Used Lag': 12,
 'Number of Observations': 99,
 'Critical Values': {'1%': -3.498198082189098,
  '5%': -2.891208211860468,
  '10%': -2.5825959973472097}}
```

Plot Auto Correlation and Partial Auto Correlation function

In [49]:
```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

plot_acf(ts, lags=30, ax=axes[0])
axes[0].set_title("ACF Plot")

plot_pacf(ts, lags=30, ax=axes[1])
axes[1].set_title("PACF Plot")
```

```
plt.tight_layout()
plt.show()
```



Inference from ACF and PACF Plots:

ACF (Autocorrelation Function)

Observations:

- Lag 1 has a very strong spike, close to 1.0 — indicating high autocorrelation.
- Gradual decline of autocorrelation values over time — this is called slow decay.
- Some seasonal spikes (especially at lag 12, 24, etc.) may also be present, suggesting annual seasonality (if monthly data).

Interpretation:

- This pattern is typical of a non-stationary series with trend and seasonality.
- It confirms that differencing is needed (i.e., d > 0 in ARIMA).
- The presence of seasonality implies that a SARIMA model might be more appropriate if seasonality must be explicitly captured.

PACF (Partial Autocorrelation Function)

Observations:

- Strong spike at lag 1 (well above the confidence band).
- After lag 1, remaining spikes drop off quickly (mostly within confidence limits).

Interpretation:

- Suggests the presence of autoregressive (AR) behavior at lag 1.
- This implies that a model with p = 1 (i.e., AR(1)) may capture the data well.

2.C.

Split the dataset

In [50]:
```
train = ts[:-12]
test = ts[-12:]
```

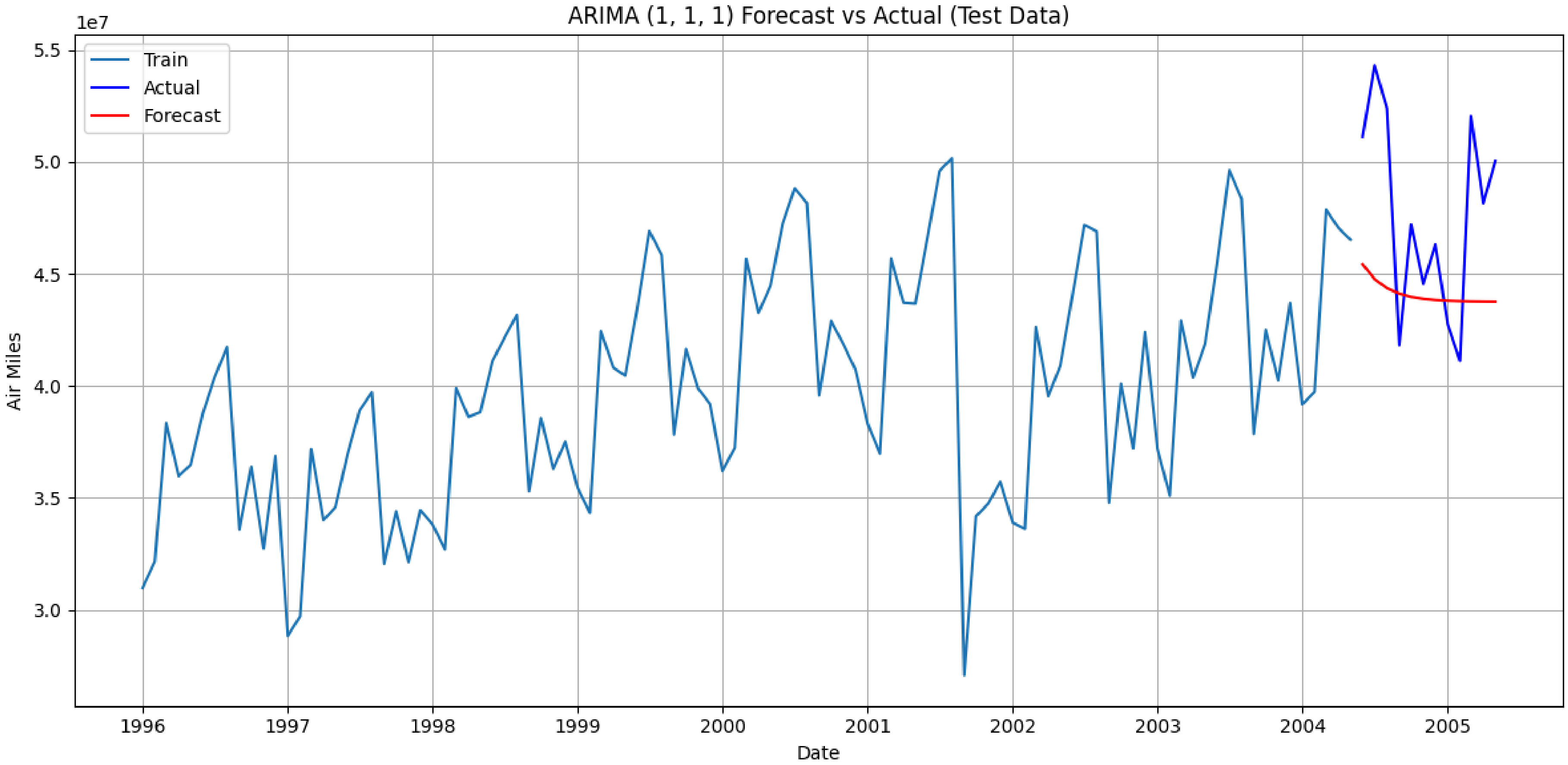
Fit ARIMA model and observe the RMSE and MAPE values of the model for test data

In [51]:
```
model = ARIMA(train, order=(1, 1, 1))
model_fit = model.fit()
forecast = model_fit.forecast(steps=12)

rmse = np.sqrt(mean_squared_error(test, forecast))
mape = mean_absolute_percentage_error(test, forecast) * 100

print(f"The RMSE value is: {rmse}")
print(f"The MAPE value is: {mape}")
```

```
The RMSE value is: 5369802.098620043
The MAPE value is: 9.146186355437603
```

In [62]:
```
plt.figure(figsize=(12, 6))
plt.plot(train, label='Train')
plt.plot(test, label="Actual", color='blue')
```

```
plt.plot(forecast.index, forecast, label="Forecast", color='red')
plt.title("ARIMA (1, 1, 1) Forecast vs Actual (Test Data)")
plt.xlabel("Date")
plt.ylabel("Air Miles")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Conclusion:

- The Model Is Not Good Enough

- While the RMSE ≈ 5.3M and MAPE ≈ 9.15% might suggest acceptable accuracy, the visual diagnostics clearly show that ARIMA(1,1,1) fails to capture the dynamic behavior of the test data.

SECTION-C

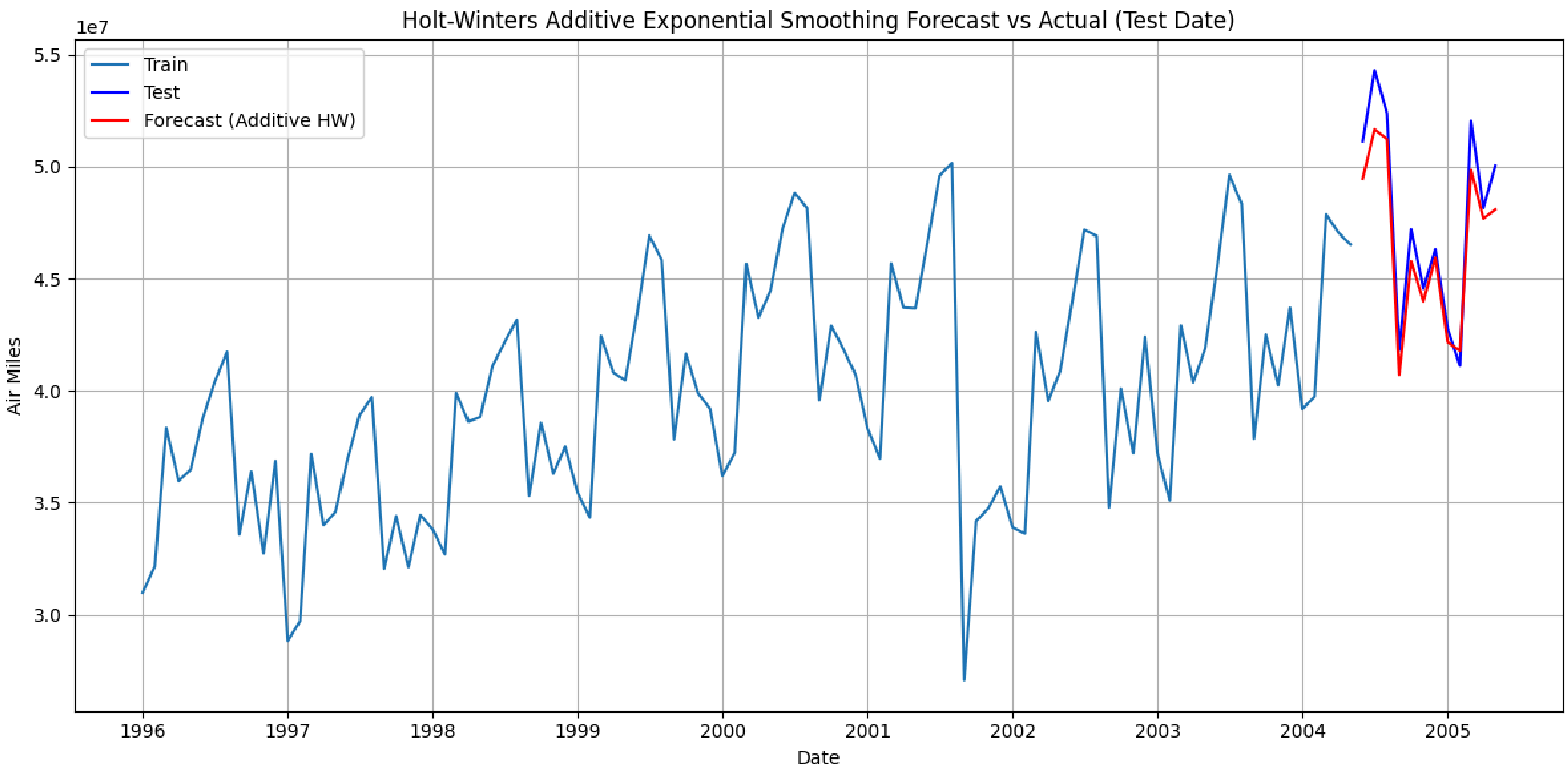
Fit exponential smoothing model and observe the residuals, RMSE and MAPE values of the model for test data.

In [53]:
```
add_hw_model = ExponentialSmoothing(
    train,
    trend='additive',
    seasonal='additive',
    seasonal_periods=12
)

add_hw_model_fit = add_hw_model.fit()

forecast_add_hw_model = add_hw_model_fit.forecast(12)

rmse_add_hw_model = np.sqrt(mean_squared_error(test, forecast_add_hw_model))
mape_add_hw_model = mean_absolute_percentage_error(test, forecast ) * 100

print(f"The RMSE value for Additive Exponential Smoothing model is: {rmse_add_hw_model}")
print(f"The MAPE value for Additive Exponential Smoothing model is: {mape_add_hw_model}")
```

```
The RMSE value for Additive Exponential Smoothing model is: 1428832.829255127
The MAPE value for Additive Exponential Smoothing model is: 9.146186355437603
```

In [63]:
```
plt.figure(figsize=(12, 6))
plt.plot(train, label="Train")
plt.plot(test, label="Test", color='blue')
plt.plot(forecast_add_hw_model.index, forecast_add_hw_model, label="Forecast (Additive HW)", color='red')
plt.title("Holt-Winters Additive Exponential Smoothing Forecast vs Actual (Test Date)")
plt.xlabel("Date")
plt.ylabel("Air Miles")
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```



Key Observations:

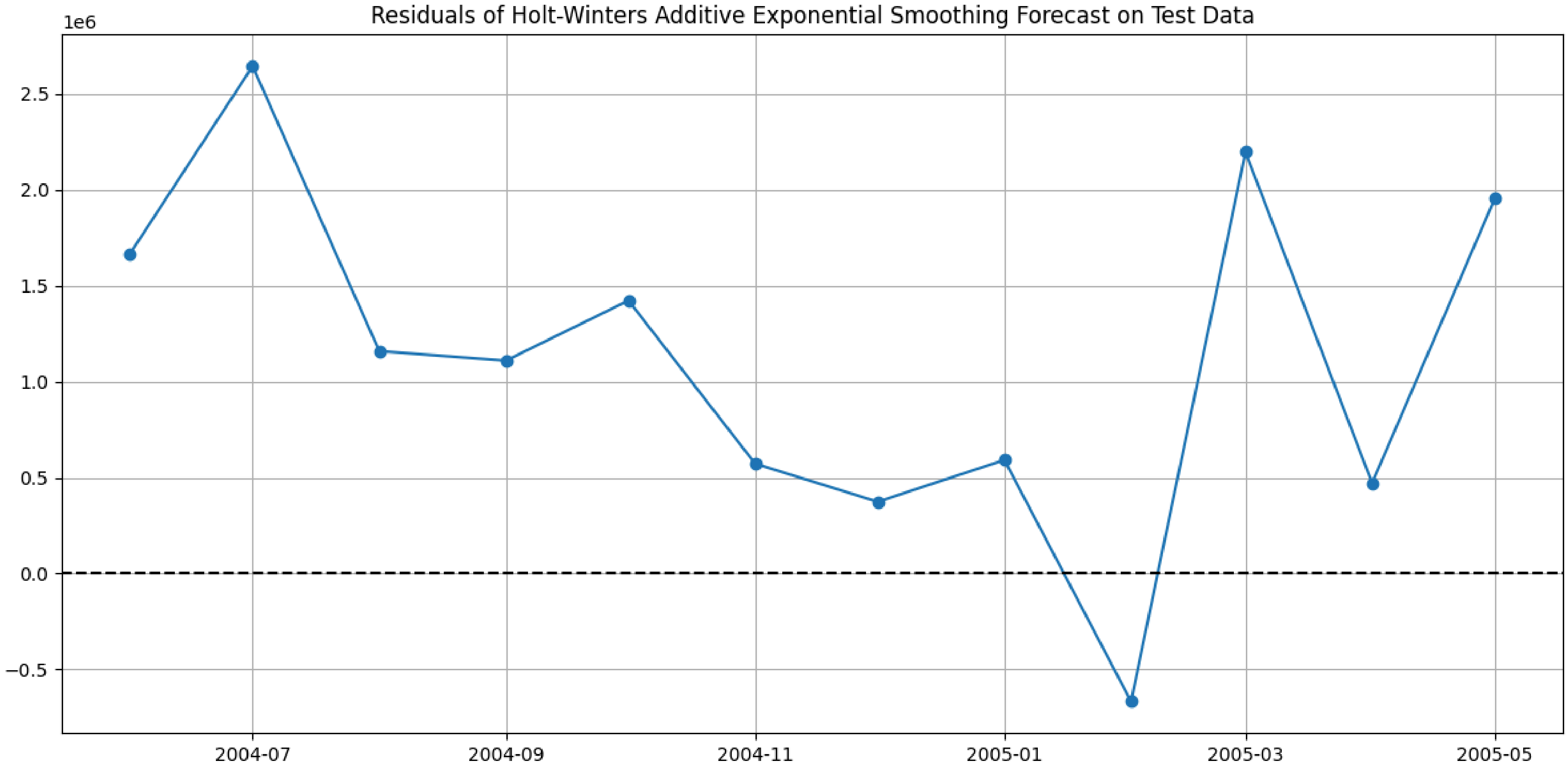
- The forecast closely tracks the test values, following the overall trend and seasonal pattern.
- The red forecast line does not exactly match the peaks and dips, but stays within reasonable bounds of the actuals.
- No major divergence is visible — the lines move in sync, suggesting good seasonality modeling.

Inference:

- The model effectively captures seasonal and trend patterns, characteristic of additive models.

- However, some under-forecasting during peaks (Dec-Jan) and over-forecasting during dips is noticeable — common in additive models when the actual seasonality is multiplicative in nature.

In [55]:
```
residuals_add_hw_model = test - forecast_add_hw_model

plt.figure(figsize=(12, 6))
plt.plot(residuals_add_hw_model, marker='o')
plt.title("Residuals of Holt-Winters Additive Exponential Smoothing Forecast on Test Data")
plt.axhline(0, linestyle='--', color='black')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Key Observations:

- Residuals are mostly positive, which means the forecasted values were generally lower than actuals (i.e., slight under-prediction).
- A spike in residuals around mid-2004 suggests a larger forecast error in that month.
- Residuals are not centered tightly around zero and show some variation — indicating that the model has some bias or cannot capture sudden spikes/dips.

Inference:

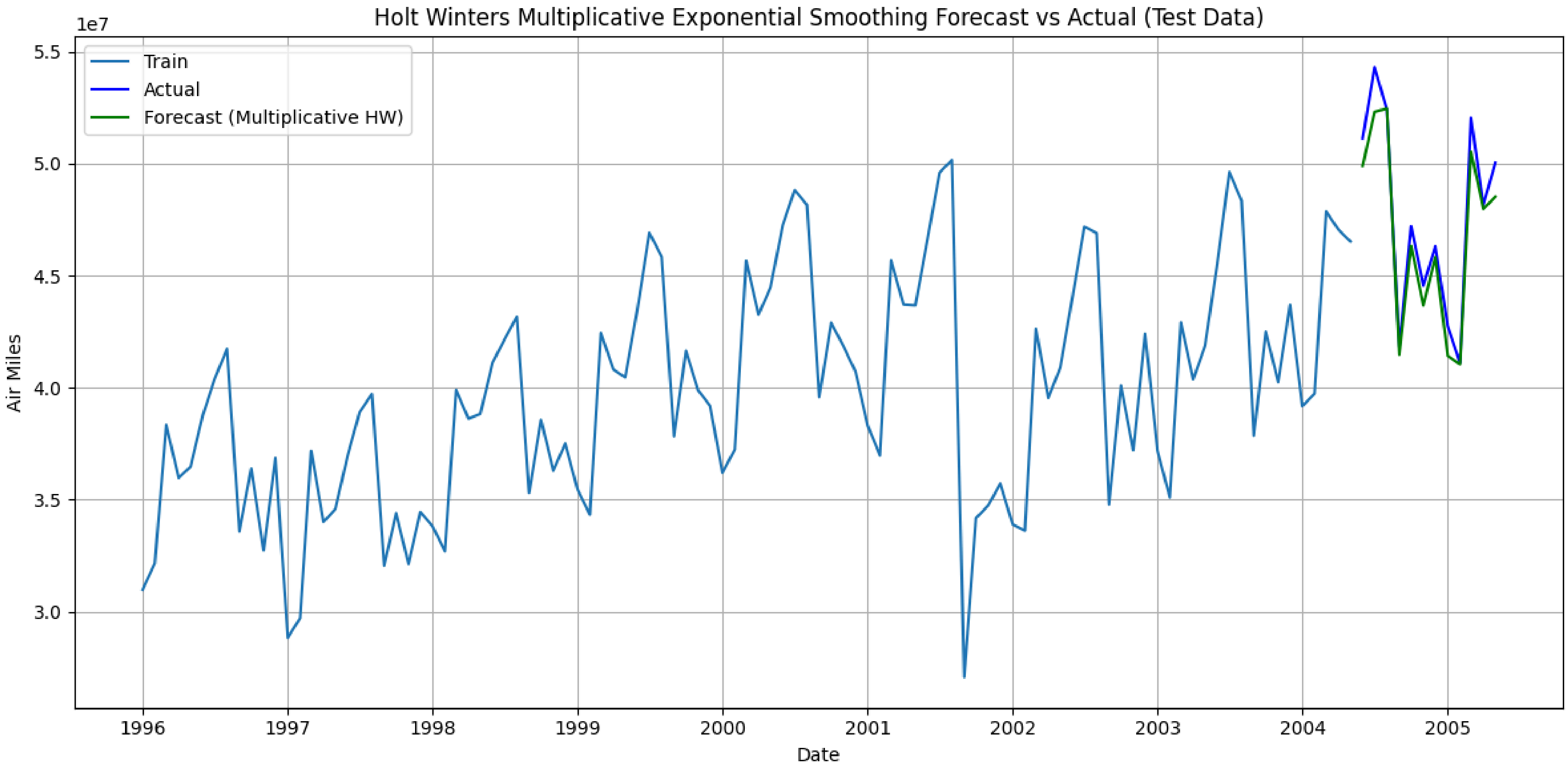
- The model is systematically underestimating some of the actual values.
- Residuals do not show clear randomness — there might be structure left, meaning further model refinement (like multiplicative model or SARIMA) could help.

Improve the Exponential Smoothing model and fit the final model. Analyze the residuals of this final model. Feel free to use charts or graphs to explain.

In [57]:
```python
mul_hw_model = ExponentialSmoothing(
    train,
    trend='multiplicative',
    seasonal='multiplicative',
    seasonal_periods=12
)

mul_hw_model_fit = mul_hw_model.fit()

forecast_mul_hw_model = mul_hw_model_fit.forecast(12)

rmse_mul_hw_model = np.sqrt(mean_squared_error(test, forecast_mul_hw_model))
mape_mul_hw_model = mean_absolute_percentage_error(test, forecast_mul_hw_model) * 100

print(f"The RMSE value of Holt-Winters Multiplicative Exponential Smoothing model is: {rmse_mul_hw_model}")
print(f"The MAPE value of Holt-Winters Multiplicative Exponential Smoothing model is: {mape_mul_hw_model}")
```

```
The RMSE value of Holt-Winters Multiplicative Exponential Smoothing model is: 1075653.066445982
The MAPE value of Holt-Winters Multiplicative Exponential Smoothing model is: 1.7992477275610688
```

In [64]:
```
plt.figure(figsize=(12, 6))
plt.plot(train, label="Train")
plt.plot(test, label="Actual", color='blue')
plt.plot(forecast_mul_hw_model.index, forecast_mul_hw_model, label="Forecast (Multiplicative HW)", color='green')
plt.title("Holt Winters Multiplicative Exponential Smoothing Forecast vs Actual (Test Data)")
plt.xlabel("Date")
plt.ylabel("Air Miles")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



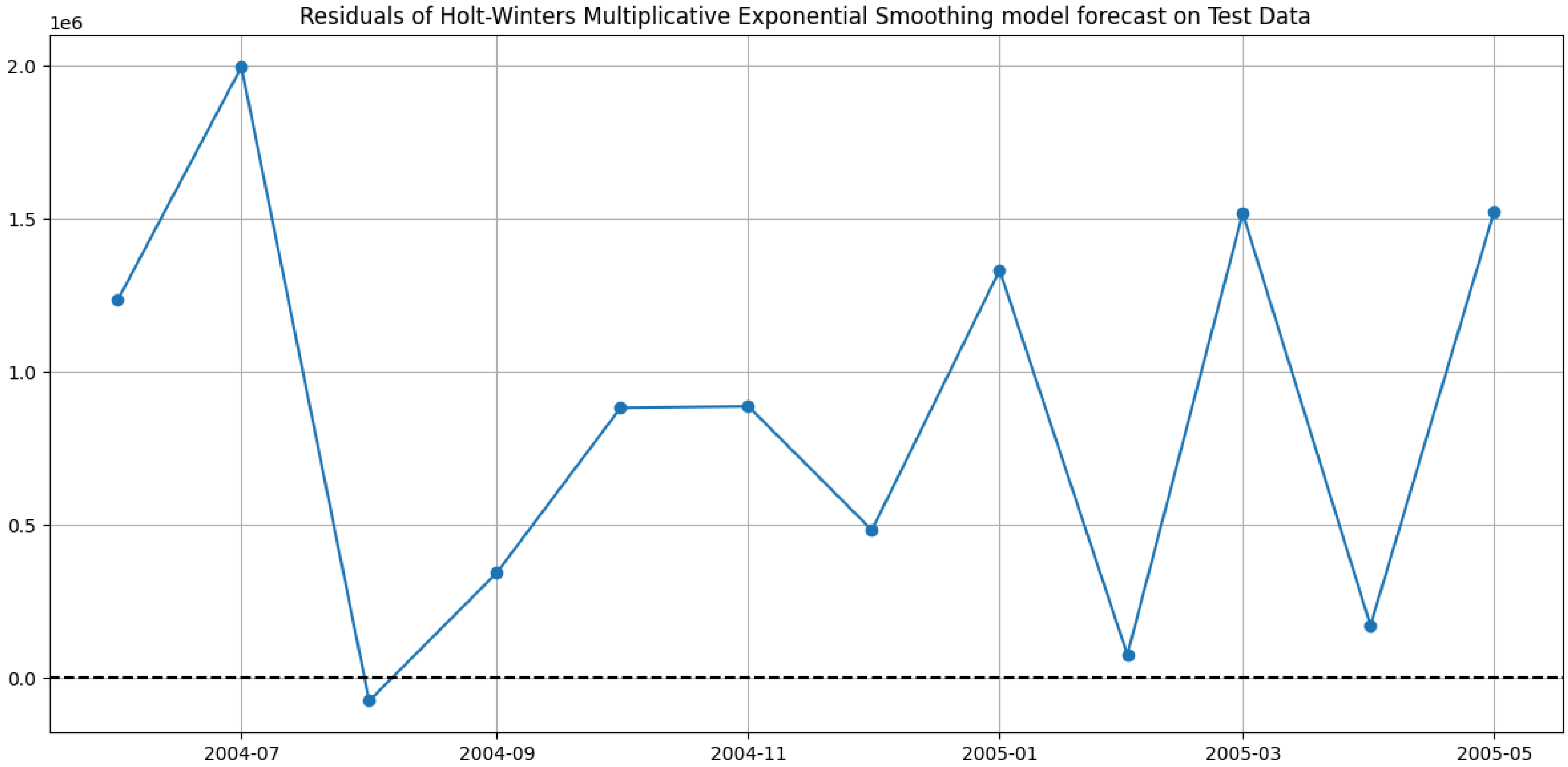
Insights:

- The forecast line (green) tracks the seasonal ups and downs more closely than the additive model did.
- Forecast captures the sharp dips and spikes in test data with better shape alignment.
- The peaks and valleys line up more accurately than with the additive model.

Interpretation:

- The multiplicative model handles proportional seasonal fluctuations better.
- This aligns with what’s expected in data like airline passengers or sales, where the amplitude of seasonality increases over time.
- It performs particularly well in higher magnitude months (late 2004 to early 2005).

In [59]:
```
residuals_mul_hw_model = test - forecast_mul_hw_model

plt.figure(figsize=(12, 6))
plt.plot(residuals_mul_hw_model, marker='o')
plt.title("Residuals of Holt-Winters Multiplicative Exponential Smoothing model forecast on Test Data")
plt.axhline(0, linestyle='--', color='black')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Insights:

- Residuals are smaller in spread than in the additive model — good!
- Most residuals cluster near zero, with minimal systematic bias.
- Some visible alternating positive/negative patterns may hint at mild underfitting in specific months, but not critical.

Interpretation:

- Forecast errors are smaller and more balanced, suggesting less bias and better fit.
- Unlike the additive model, this residual plot shows no extreme skewness or clustering — a hallmark of a more reliable forecast.

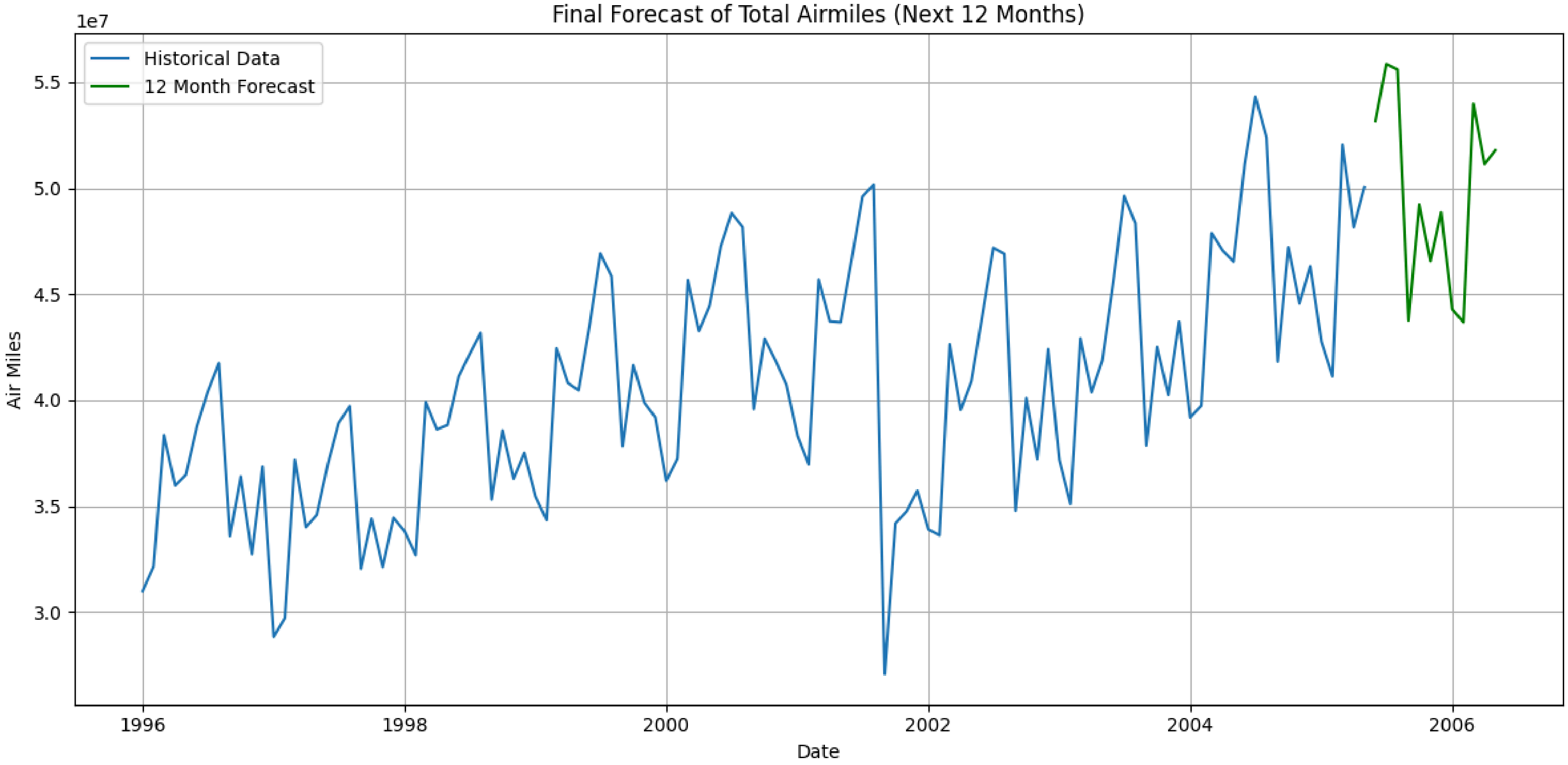
Conclusion:

The multiplicative Holt-Winters model is superior for your data because:

- It better captures seasonal amplitude
- It results in lower residuals
- It aligns closely with the cyclical behavior of the actual time series

Forecast Total Airmiles/records for next 12 months using the final model

In [60]:
```
final_model = ExponentialSmoothing(
    ts,
    trend="multiplicative",
    seasonal="multiplicative",
    seasonal_periods=12
)

final_model_fit = final_model.fit()

final_forecast = final_model_fit.forecast(12)

plt.figure(figsize=(12, 6))
plt.plot(ts, label='Historical Data')
plt.plot(final_forecast.index, final_forecast, label="12 Month Forecast", color='green')
plt.title("Final Forecast of Total Airmiles (Next 12 Months)")
plt.xlabel("Date")
plt.ylabel("Air Miles")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```


Final Forecast of Total Airmiles (Next 12 Months)
1e7
Historical Data
12 Month Forecast
5.5
5.0
4.5
4.0
3.5
3.0
Air Miles
1996
1998
2000
2002
2004
2006
Date

END