In [1]:

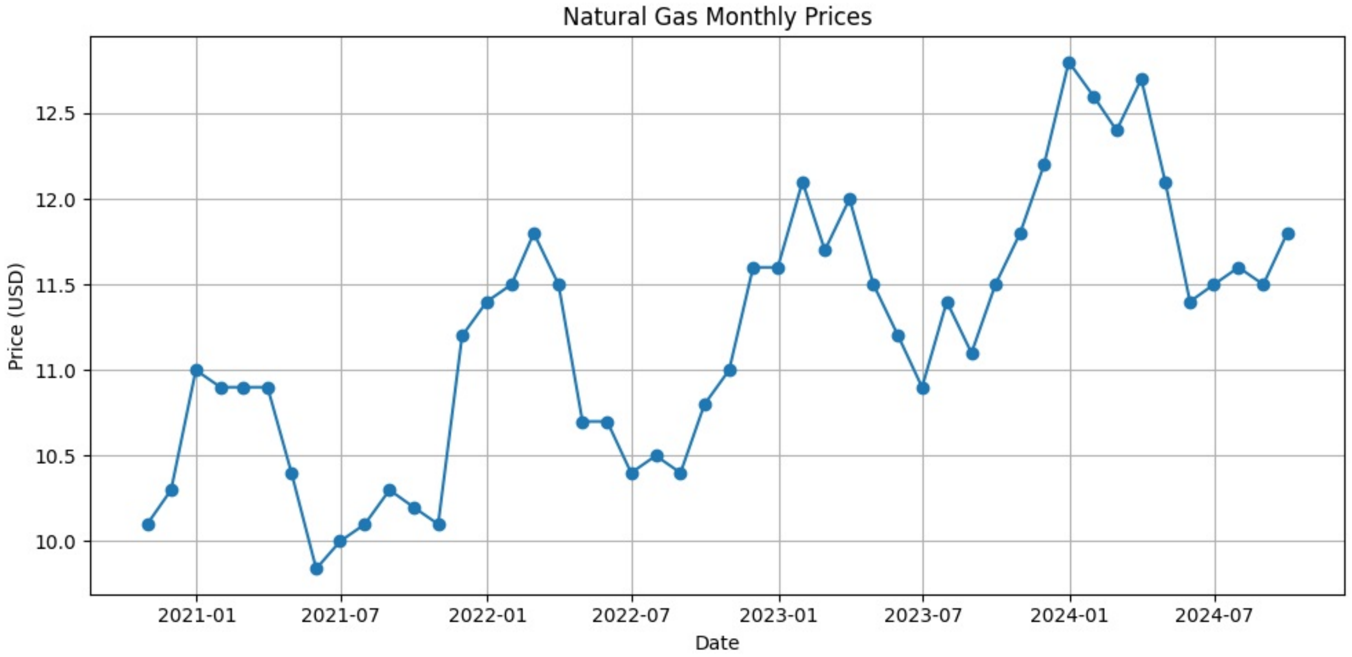
```
import pandas as pd
import matplotlib.pyplot as plt
from prophet import Prophet
from datetime import datetime

# STEP 1: Load the dataset
df = pd.read_csv("Nat_Gas.csv")  # Replace with your file path
df['Dates'] = pd.to_datetime(df['Dates'])
df = df.rename(columns={'Dates': 'ds', 'Prices': 'y'})  # Prophet requires 'ds' and 'y'
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_7500\3299703210.py:8: UserWarning: Could not infer format, so each
element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected
, please specify a format.
  df['Dates'] = pd.to_datetime(df['Dates'])
```

In [2]:

```
# Visualize the data
plt.figure(figsize=(10, 5))
plt.plot(df['ds'], df['y'], marker='o')
plt.title("Natural Gas Monthly Prices")
plt.xlabel("Date")
plt.ylabel("Price (USD)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [3]:

```
# Create and fit Prophet model
model = Prophet(yearly_seasonality=True, daily_seasonality=False)
model.fit(df)
```

```
17:38:34 - cmdstanpy - INFO - Chain [1] start processing
17:38:36 - cmdstanpy - INFO - Chain [1] done processing
```

Out[3]:

```
<prophet.forecaster.Prophet at 0x2c6c36e41a0>
```

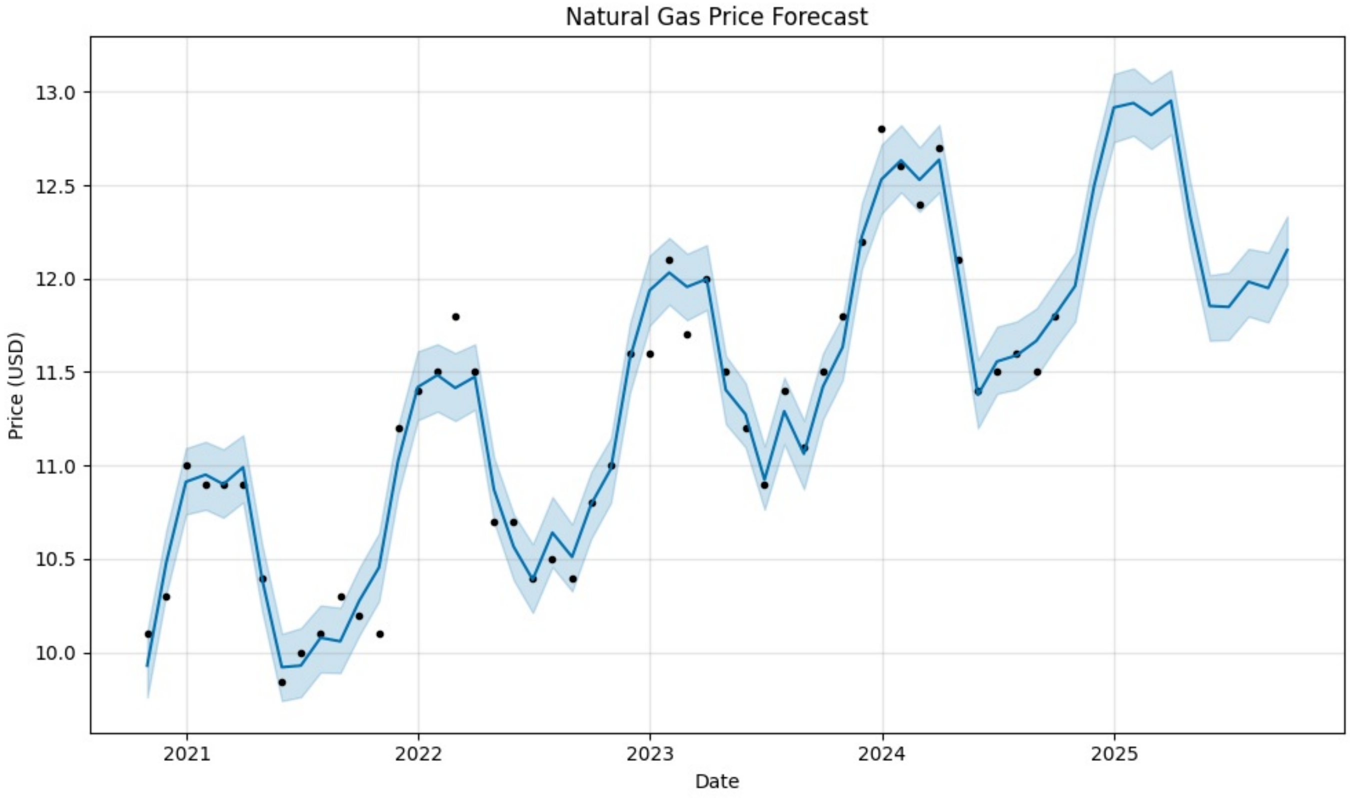
In [4]:

```
# Forecast 12 months into the future
future = model.make_future_dataframe(periods=12, freq='M')
forecast = model.predict(future)
```

```
C:\Users\LENOVO\AppData\Local\Programs\Python\Python313\Lib\site-packages\prophet\forecaster.py:1854: FutureWarn
ing: 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
  dates = pd.date_range(
```

In [5]:

```
# Plot forecast
model.plot(forecast)
plt.title("Natural Gas Price Forecast")
plt.xlabel("Date")
plt.ylabel("Price (USD)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [6]:

```python
# Function to get estimate for any date
def get_price_estimate(date_str):
    input_date = pd.to_datetime(date_str)
    closest = forecast.iloc[(forecast['ds'] - input_date).abs().argsort()[:1]]
    return round(float(closest['yhat'].values[0]), 2)

# Example usage
print("Estimated price on 2023-12-31:", get_price_estimate("2023-12-31"))
print("Estimated price on 2025-09-30:", get_price_estimate("2025-09-30"))
```

```
Estimated price on 2023-12-31: 12.53
Estimated price on 2025-09-30: 12.15
```

In [7]:

```python
def price_storage_contract(injection_date: str, withdrawal_date: str) -> float:
    """
    Calculate the estimated value of a gas storage contract based on forecasted prices.

    Args:
        injection_date (str): Date when gas is injected/stored (e.g. "2024-10-31").
        withdrawal_date (str): Date when gas is withdrawn/sold (e.g. "2025-03-31").

    Returns:
        float: Estimated value of the contract in USD (sell price - buy price)
    """
    buy_price = get_price_estimate(injection_date)
    sell_price = get_price_estimate(withdrawal_date)
    return round(sell_price - buy_price, 2)
```

In [8]:

```python
price_storage_contract("2024-10-31", "2025-03-31")
# Output: e.g., 1.75 (you gain $1.75 per unit stored over that period)
```

Out[8]:

```
0.99
```

In [9]:

```python
def price_storage_contract_full(
    injection_date: str,
    withdrawal_date: str,
    volume_mmbtu: float = 1_000_000,
    monthly_storage_fee: float = 100_000,
    injection_fee_per_mmbtu: float = 0.01,   # $10K per million MMBtu
    withdrawal_fee_per_mmbtu: float = 0.01,
    transport_fee_per_trip: float = 50_000
) -> float:
    """
    Calculate the full value of a natural gas storage contract.

    Args:
        injection_date (str): Purchase/injection date (format: 'YYYY-MM-DD')
        withdrawal_date (str): Sale/withdrawal date (format: 'YYYY-MM-DD')
        volume_mmbtu (float): Volume in MMBtu. Default = 1 million.
        monthly_storage_fee (float): Monthly storage rental fee.
        injection_fee_per_mmbtu (float): Fee per MMBtu to inject gas.
        withdrawal_fee_per_mmbtu (float): Fee per MMBtu to withdraw gas.
        transport_fee_per_trip (float): One-time transport fee per direction.

    Returns:
        float: Estimated contract value in USD.
    """
    # Estimate gas prices on input dates
    buy_price = get_price_estimate(injection_date)
    sell_price = get_price_estimate(withdrawal_date)

    # Gross profit
    gross_profit = (sell_price - buy_price) * volume_mmbtu

    # Number of months in storage
    months = pd.date_range(injection_date, withdrawal_date, freq='MS').nunique()

    # Total costs
    storage_cost = monthly_storage_fee * months
    injection_cost = injection_fee_per_mmbtu * volume_mmbtu
    withdrawal_cost = withdrawal_fee_per_mmbtu * volume_mmbtu
    transport_cost = 2 * transport_fee_per_trip  # To and from facility

    total_cost = storage_cost + injection_cost + withdrawal_cost + transport_cost

    # Final value
    contract_value = gross_profit - total_cost
    return round(contract_value, 2)
```

In [10]:

```
price_storage_contract_full(
    injection_date="2024-07-31",
    withdrawal_date="2025-01-31",
    volume_mmbtu=1_000_000,
    monthly_storage_fee=100_000,
    injection_fee_per_mmbtu=0.01,
    withdrawal_fee_per_mmbtu=0.01,
    transport_fee_per_trip=50_000
)
```

Out[10]:

```
630000.0
```

In [11]:

```
def price_storage_contract_general(
    injection_dates: list,
    withdrawal_dates: list,
    volume: float,
    injection_rate: float,
    withdrawal_rate: float,
    max_storage_volume: float,
    monthly_storage_fee: float
) -> float:
    """
    Prototype for pricing a multi-period natural gas storage contract.

    Parameters:
        injection_dates (list): Dates to inject gas (YYYY-MM-DD).
        withdrawal_dates (list): Dates to withdraw gas (YYYY-MM-DD).
        volume (float): Total volume to trade (MMBtu).
        injection_rate (float): Max daily injection rate (MMBtu).
        withdrawal_rate (float): Max daily withdrawal rate (MMBtu).
        max_storage_volume (float): Storage capacity (MMBtu).
        monthly_storage_fee (float): Monthly storage cost in USD.

    Returns:
        float: Estimated contract value.
    """
    injected = 0.0
    withdrawn = 0.0
    storage = 0.0
    buy_costs = 0.0
    sell_revenue = 0.0
    inventory_record = {}

    for date in sorted(injection_dates):
        if injected >= volume:
            break
        daily_injection = min(injection_rate, volume - injected, max_storage_volume - storage)
        price = get_price_estimate(date)
        buy_costs += daily_injection * price
        storage += daily_injection
        injected += daily_injection
        inventory_record[date] = storage

    for date in sorted(withdrawal_dates):
        if withdrawn >= volume:
            break
        daily_withdrawal = min(withdrawal_rate, volume - withdrawn, storage)
        price = get_price_estimate(date)
        sell_revenue += daily_withdrawal * price
        storage -= daily_withdrawal
        withdrawn += daily_withdrawal
        inventory_record[date] = storage

    # Storage duration (number of unique months between first and last activity)
    all_dates = injection_dates + withdrawal_dates
    start = pd.to_datetime(min(all_dates))
    end = pd.to_datetime(max(all_dates))
    storage_months = pd.date_range(start, end, freq='MS').nunique()
    total_storage_fees = storage_months * monthly_storage_fee

    value = sell_revenue - buy_costs - total_storage_fees
    return round(value, 2)
```

In [12]:

```
price_storage_contract_general(
    injection_dates=["2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04", "2024-06-05"],
    withdrawal_dates=["2024-12-01", "2024-12-02", "2024-12-03", "2024-12-04", "2024-12-05"],
    volume=500_000,  # 500K MMBtu
    injection_rate=100_000,  # per day
    withdrawal_rate=100_000,  # per day
    max_storage_volume=500_000,
    monthly_storage_fee=100_000
)
```

Out[12]:

```
-140000.0
```

In [14]:

```
def price_storage_contract_general_debug(
    injection_dates,
    withdrawal_dates,
    volume,
    injection_rate,
    withdrawal_rate,
    max_storage_volume,
    monthly_storage_fee
):
    import pandas as pd

    if len(injection_dates) != len(withdrawal_dates):
        raise ValueError("Injection and withdrawal dates must match in length.")

    total_value = 0
    total_storage_fees = 0
    total_injection_withdrawal_costs = 0

    for inject_date, withdraw_date in zip(injection_dates, withdrawal_dates):
        inject_date = pd.to_datetime(inject_date)
        withdraw_date = pd.to_datetime(withdraw_date)

        inject_price = get_price_estimate(inject_date)
        withdraw_price = get_price_estimate(withdraw_date)

        # Number of months the gas is stored (approx)
        storage_months = max((withdraw_date.year - inject_date.year) * 12 + (withdraw_date.month - inject_date.
        storage_fee = monthly_storage_fee * storage_months

        buy_cost = volume * inject_price
        sell_revenue = volume * withdraw_price

        value = sell_revenue - buy_cost - storage_fee
        total_value += value
        total_storage_fees += storage_fee

        print(f"\n--- Trade from {inject_date.date()} to {withdraw_date.date()} ---")
        print(f"Inject Price: ${inject_price:.2f}")
        print(f"Withdraw Price: ${withdraw_price:.2f}")
        print(f"Buy Cost: ${buy_cost:,.2f}")
        print(f"Sell Revenue: ${sell_revenue:,.2f}")
        print(f"Storage Months: {storage_months}")
        print(f"Storage Fee: ${storage_fee:,.2f}")
        print(f"Net Value for this trade: ${value:,.2f}")

    print(f"\n====== Summary ======")
    print(f"Total Value: ${total_value:,.2f}")
    print(f"Total Storage Fees: ${total_storage_fees:,.2f}")

    return total_value
```

In [15]:

```
price_storage_contract_general_debug(
    injection_dates=["2024-06-01"],
    withdrawal_dates=["2024-12-01"],
    volume=1_000_000,
    injection_rate=200_000,
    withdrawal_rate=200_000,
    max_storage_volume=1_000_000,
    monthly_storage_fee=100_000
)
```

```
--- Trade from 2024-06-01 to 2024-12-01 ---
Inject Price: $11.38
Withdraw Price: $12.50
Buy Cost: $11,380,000.00
Sell Revenue: $12,500,000.00
Storage Months: 6
Storage Fee: $600,000.00
Net Value for this trade: $520,000.00

====== Summary ======
Total Value: $520,000.00
Total Storage Fees: $600,000.00
```

```
Out[15]: 520000.0
```

```
In [ ]:
```