In [8]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

from textblob import TextBlob
```

In [9]:

```
nltk.download("stopwords")
nltk.download("wordnet")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\DELL\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\DELL\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[9]:

```
True
```

In [10]:

```
np.random.seed(42)

dates = pd.date_range(
    start="2021-01-01",
    periods=36,
    freq="ME"   # Month End (correct & future-proof)
)

sales = 200 + np.random.randn(36).cumsum() * 5
promo = np.random.randint(0, 2, size=36)

sales_df = pd.DataFrame({
    "date": dates,
    "sales": sales,
    "promo": promo
})

sales_df.set_index("date", inplace=True)

# Explicitly set frequency (removes warnings)
sales_df = sales_df.asfreq("ME")

sales_df.head()
```

Out[10]:

date	sales	promo
2021-01-31	202.483571	1
2021-02-28	201.792249	1
2021-03-31	205.030692	1
2021-04-30	212.645841	1
2021-05-31	211.475074	1

In [11]:

```
arima_model = ARIMA(
    sales_df["sales"],
    order=(1, 1, 0)
)

arima_fit = arima_model.fit()
sales_df["ARIMA_Forecast"] = arima_fit.fittedvalues
```

In [12]:

```
sarima_model = SARIMAX(
    sales_df["sales"],
    order=(1, 1, 0),
    seasonal_order=(1, 1, 0, 12),
    enforce_stationarity=True,
    enforce_invertibility=True
)

sarima_fit = sarima_model.fit(disp=False)
sales_df["SARIMA_Forecast"] = sarima_fit.fittedvalues
```

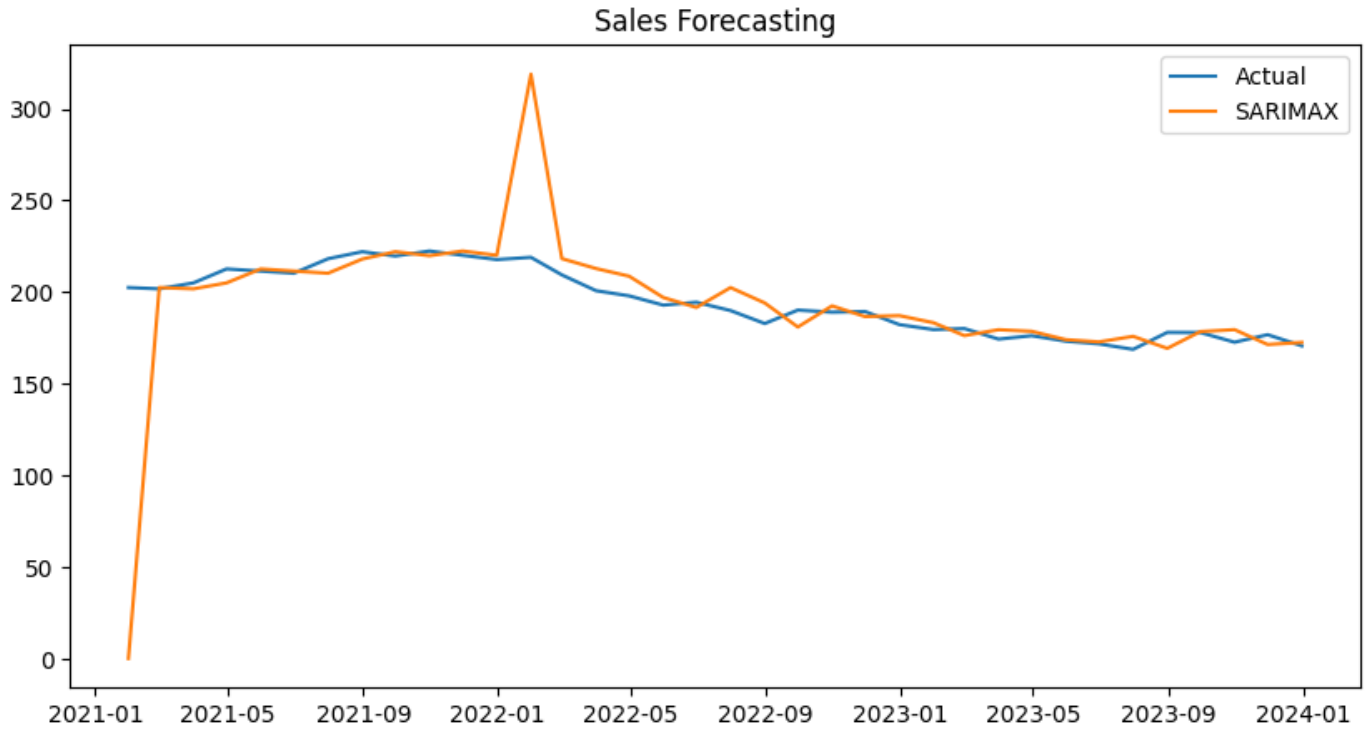
In [13]:

```
sarimax_model = SARIMAX(
    sales_df["sales"],
    exog=sales_df[["promo"]],
    order=(1, 1, 0),
    seasonal_order=(1, 1, 0, 12),
    enforce_stationarity=True,
    enforce_invertibility=True
)

sarimax_fit = sarimax_model.fit(disp=False)
sales_df["SARIMAX_Forecast"] = sarimax_fit.fittedvalues
```

In [14]:

```
plt.figure(figsize=(10, 5))
plt.plot(sales_df["sales"], label="Actual")
plt.plot(sales_df["SARIMAX_Forecast"], label="SARIMAX")
plt.title("Sales Forecasting")
plt.legend()
plt.show()
```



In [15]:

```
feedback_df = pd.DataFrame({
    "text": [
        "Delivery was very late and frustrating",
        "Great quality and fast delivery",
        "Price is too high for this product",
        "Excellent service and good discounts",
        "Product was out of stock again",
        "Very happy with the purchase"
    ],
    "label": [
        "delivery_issue",
        "positive",
        "price_issue",
        "positive",
        "availability_issue",
        "positive"
    ]
})

feedback_df
```

Out[15]:

	text	label
0	Delivery was very late and frustrating	delivery_issue
1	Great quality and fast delivery	positive
2	Price is too high for this product	price_issue
3	Excellent service and good discounts	positive
4	Product was out of stock again	availability_issue
5	Very happy with the purchase	positive

In [17]:

```
stop_words = set(stopwords.words("english"))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    text = text.lower()
    tokens = text.split()
    tokens = [
        lemmatizer.lemmatize(token)
        for token in tokens
        if token not in stop_words
    ]
    return " ".join(tokens)

feedback_df["clean_text"] = feedback_df["text"].apply(preprocess_text)
feedback_df
```

Out[17]:

	text	label	clean_text
0	Delivery was very late and frustrating	delivery_issue	delivery late frustrating
1	Great quality and fast delivery	positive	great quality fast delivery
2	Price is too high for this product	price_issue	price high product
3	Excellent service and good discounts	positive	excellent service good discount
4	Product was out of stock again	availability_issue	product stock
5	Very happy with the purchase	positive	happy purchase

In [18]:

```
def sentiment_score(text):
    return TextBlob(text).sentiment.polarity

feedback_df["sentiment"] = feedback_df["clean_text"].apply(sentiment_score)
feedback_df
```

Out[18]:

	text	label	clean_text	sentiment
0	Delivery was very late and frustrating	delivery_issue	delivery late frustrating	-0.35
1	Great quality and fast delivery	positive	great quality fast delivery	0.50
2	Price is too high for this product	price_issue	price high product	0.16
3	Excellent service and good discounts	positive	excellent service good discount	0.85
4	Product was out of stock again	availability_issue	product stock	0.00
5	Very happy with the purchase	positive	happy purchase	0.80

In [19]:

```
X = feedback_df["clean_text"]
y = feedback_df["label"]

vectorizer = TfidfVectorizer()
X_vec = vectorizer.fit_transform(X)
```

```
clf = LogisticRegression(max_iter=200)
clf.fit(X_vec, y)

preds = clf.predict(X_vec)

print(classification_report(y, preds))
```

```
                   precision    recall  f1-score   support

availability_issue       0.00      0.00      0.00         1
    delivery_issue       0.00      0.00      0.00         1
          positive       0.50      1.00      0.67         3
       price_issue       0.00      0.00      0.00         1

          accuracy                           0.50         6
         macro avg       0.12      0.25      0.17         6
      weighted avg       0.25      0.50      0.33         6
```

```
C:\Users\DELL\AppData\Roaming\Python\Python310\site-packages\sklearn\metrics\_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\DELL\AppData\Roaming\Python\Python310\site-packages\sklearn\metrics\_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\DELL\AppData\Roaming\Python\Python310\site-packages\sklearn\metrics\_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```

In [20]:

```
# Aggregate sentiment (mock monthly average)
avg_sentiment = feedback_df["sentiment"].mean()

sales_df["avg_sentiment"] = avg_sentiment
sales_df.head()
```

Out[20]:

date	sales	promo	ARIMA_Forecast	SARIMA_Forecast	SARIMAX_Forecast	avg_sentiment
2021-01-31	202.483571	1	0.000000	0.000000	0.178248	0.326667
2021-02-28	201.792249	1	202.483644	202.483411	202.483432	0.326667
2021-03-31	205.030692	1	201.781063	201.792255	201.792254	0.326667
2021-04-30	212.645841	1	205.083094	205.030687	205.030687	0.326667
2021-05-31	211.475074	1	212.769063	212.645829	212.645831	0.326667

In [21]:

```
sarimax_sentiment_model = SARIMAX(
    sales_df["sales"],
```

```
    exog=sales_df[["promo", "avg_sentiment"]],
    order=(1, 1, 0),
    seasonal_order=(1, 1, 0, 12),
    enforce_stationarity=True,
    enforce_invertibility=True
)

sarimax_sentiment_fit = sarimax_sentiment_model.fit(disp=False)
sales_df["Forecast_With_Sentiment"] = sarimax_sentiment_fit.fittedvalues
```

In [22]:

```
def generate_ai_insight(row):
    if row["avg_sentiment"] < 0:
        return "Negative customer sentiment is suppressing demand."
    elif row["promo"] == 1:
        return "Promotional activity is positively influencing sales."
    else:
        return "Sales are driven by stable baseline demand."

sales_df["AI_Insight"] = sales_df.apply(generate_ai_insight, axis=1)
sales_df.tail()
```

Out[22]:

date	sales	promo	ARIMA_Forecast	SARIMA_Forecast	SARIMAX_Forecast	avg_sentiment	Fore
2023-08-31	178.030823	0	168.720751	169.367358	169.324803	0.326667	
2023-09-30	177.963337	0	178.180683	178.483156	178.476019	0.326667	
2023-10-31	172.674783	1	177.962245	179.486175	179.532139	0.326667	
2023-11-30	176.787507	1	172.589208	171.388923	171.426125	0.326667	
2023-12-31	170.683289	0	176.854056	172.786522	172.565628	0.326667	

In [23]:

```
summary = {
    "Forecast_Model": "SARIMAX (Promo + Sentiment)",
    "Average_Sentiment": round(avg_sentiment, 3),
    "Top_Customer_Issues": feedback_df["label"].value_counts().to_dict(),
    "AI_Recommendation": "Improve delivery reliability to enhance demand and sentiment."
}
```

```
summary
```

Out[23]:

```
{'Forecast_Model': 'SARIMAX (Promo + Sentiment)',
 'Average_Sentiment': np.float64(0.327),
 'Top_Customer_Issues': {'positive': 3,
  'delivery_issue': 1,
  'price_issue': 1,
  'availability_issue': 1},
 'AI_Recommendation': 'Improve delivery reliability to enhance demand and sentiment.'}
```

In [ ]:

```

```