In [73]:
```
import pandas as pd
import locale
from dateutil import parser
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import numpy as np
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_
from googletrans import Translator
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.stattools import adfuller
```

In [37]:
```
# # Set the locale to French
# locale.setlocale(locale.LC_TIME, 'fr_FR.UTF-8')

# Load the data with ISO-8859-1 encoding and semicolon delimiter
file_path = 'Data/forecast_data_3years.csv'
data = pd.read_csv(file_path, encoding='ISO-8859-1', delimiter=';')

# Keep only the relevant columns
data = data[['mois', 'Spend', 'Conversions']]

# Convert 'Spend' to a numeric format by replacing commas with dots and
data['Spend'] = data['Spend'].str.replace(',', '.').astype(float)
```

In [38]:
```
data
```

Out[38]:

	mois	Spend	Conversions
0	juin-15	39456.23	1578
1	juil-15	57338.93	1864
2	août-15	31802.36	1138
3	sept-15	52195.88	2176
4	oct-15	27642.56	1185
5	nov-15	31215.60	1471
6	déc-15	64036.93	2942
7	janv-16	58342.15	2768
8	févr-16	18326.47	863
9	mars-16	90135.24	3127
10	avr-16	37002.10	1412
11	mai-16	59045.12	1819
12	juin-16	60456.89	2003
13	juil-16	55215.63	2423
14	août-16	49507.75	1645
15	sept-16	40022.35	1758
16	oct-16	65003.56	2015
17	nov-16	54787.22	2907
18	déc-16	81317.85	3503
19	janv-17	58010.37	3381
20	févr-17	27311.03	1300
21	mars-17	43592.32	1926
22	avr-17	69513.61	1601
23	mai-17	57322.57	1160
24	juin-17	77155.31	2373
25	juil-17	85107.24	2097
26	août-17	21384.64	653
27	sept-17	40919.94	888
28	oct-17	39589.99	1286
29	nov-17	107899.90	4071
30	déc-17	122175.16	1893
31	janv-18	62063.38	2041
32	févr-18	28985.01	1146
33	mars-18	52089.04	2127

	mois	Spend	Conversions
34	avr-18	16617.12	431
35	mai-18	93073.46	1792

In [39]:
```
# Dictionary to map French month names to English month abbreviations
month_map = {
    'janv': 'Jan',
    'févr': 'Feb',
    'mars': 'Mar',
    'avr': 'Apr',
    'mai': 'May',
    'juin': 'Jun',
    'juil': 'Jul',
    'août': 'Aug',
    'sept': 'Sep',
    'oct': 'Oct',
    'nov': 'Nov',
    'déc': 'Dec'
}

# Replace French month names with English abbreviations
for fr, en in month_map.items():
    data['mois'] = data['mois'].str.replace(fr, en)
```

In [40]:
```
# locale.setlocale(locale.LC_TIME, 'en_US.UTF-8')
```

In [41]:
```
data
```

Out[41]:

	mois	Spend	Conversions
0	Jun-15	39456.23	1578
1	Jul-15	57338.93	1864
2	Aug-15	31802.36	1138
3	Sep-15	52195.88	2176
4	Oct-15	27642.56	1185
5	Nov-15	31215.60	1471
6	Dec-15	64036.93	2942
7	Jan-16	58342.15	2768
8	Feb-16	18326.47	863
9	Mar-16	90135.24	3127
10	Apr-16	37002.10	1412
11	May-16	59045.12	1819
12	Jun-16	60456.89	2003
13	Jul-16	55215.63	2423
14	Aug-16	49507.75	1645
15	Sep-16	40022.35	1758
16	Oct-16	65003.56	2015
17	Nov-16	54787.22	2907
18	Dec-16	81317.85	3503
19	Jan-17	58010.37	3381
20	Feb-17	27311.03	1300
21	Mar-17	43592.32	1926
22	Apr-17	69513.61	1601
23	May-17	57322.57	1160
24	Jun-17	77155.31	2373
25	Jul-17	85107.24	2097
26	Aug-17	21384.64	653
27	Sep-17	40919.94	888
28	Oct-17	39589.99	1286
29	Nov-17	107899.90	4071
30	Dec-17	122175.16	1893
31	Jan-18	62063.38	2041
32	Feb-18	28985.01	1146
33	Mar-18	52089.04	2127

	mois	Spend	Conversions
34	Apr-18	16617.12	431
35	May-18	93073.46	1792

In [42]:
```
data['mois'] = pd.to_datetime(data['mois'], format='%b-%y')
```

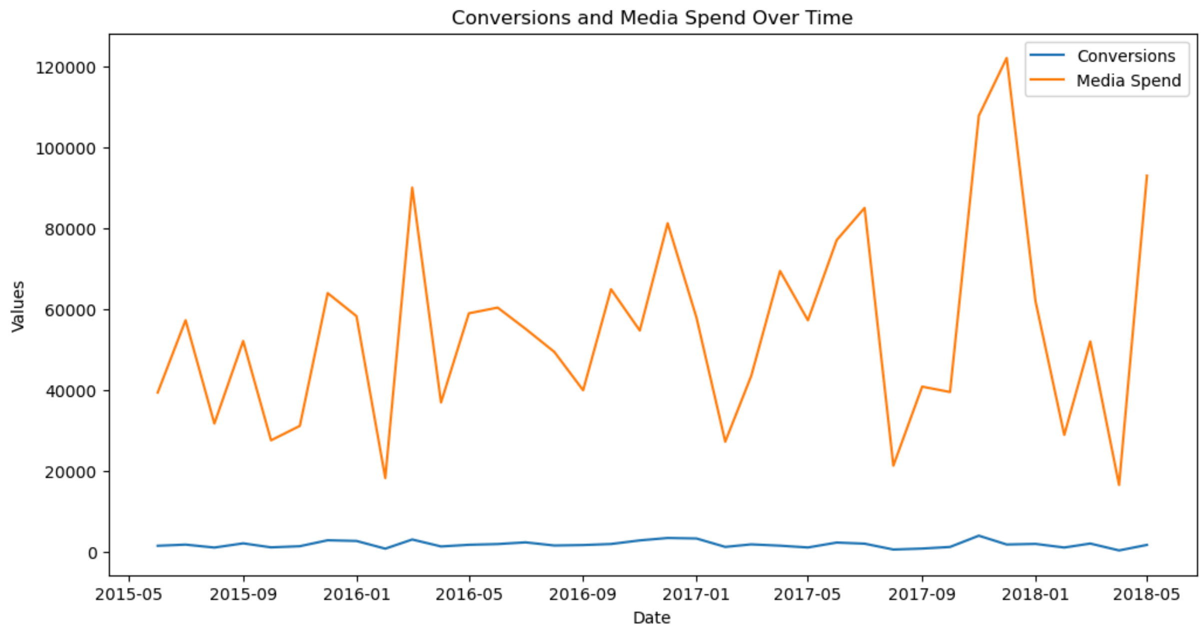
In [43]:
```
data
```

Out[43]:

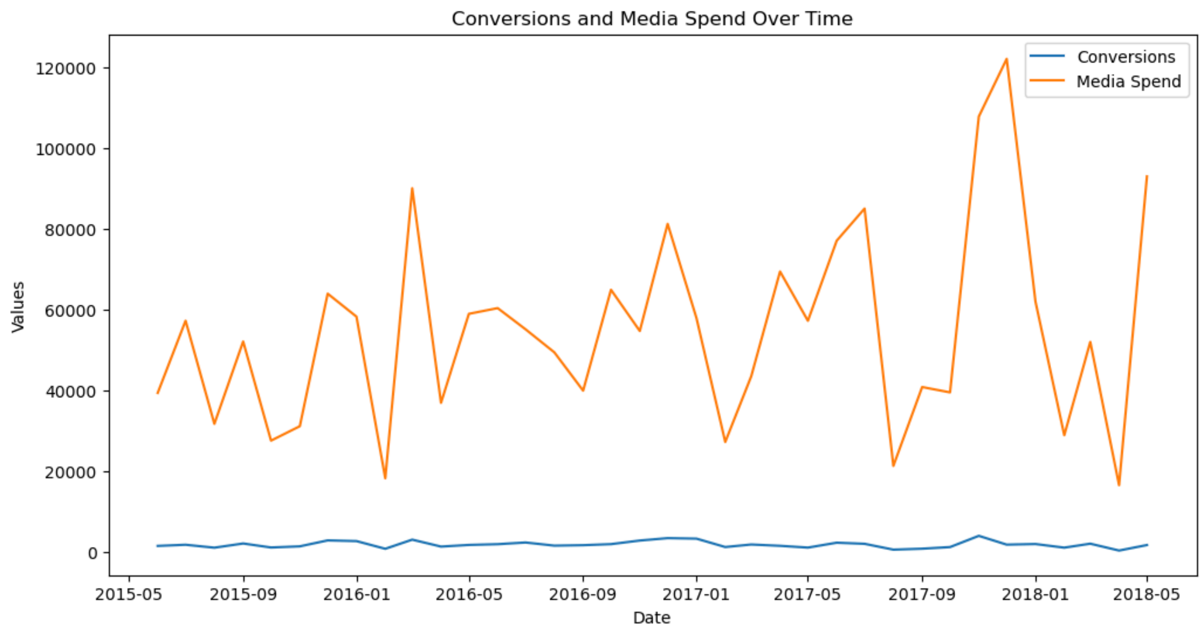
	mois	Spend	Conversions
0	2015-06-01	39456.23	1578
1	2015-07-01	57338.93	1864
2	2015-08-01	31802.36	1138
3	2015-09-01	52195.88	2176
4	2015-10-01	27642.56	1185
5	2015-11-01	31215.60	1471
6	2015-12-01	64036.93	2942
7	2016-01-01	58342.15	2768
8	2016-02-01	18326.47	863
9	2016-03-01	90135.24	3127
10	2016-04-01	37002.10	1412
11	2016-05-01	59045.12	1819
12	2016-06-01	60456.89	2003
13	2016-07-01	55215.63	2423
14	2016-08-01	49507.75	1645
15	2016-09-01	40022.35	1758
16	2016-10-01	65003.56	2015
17	2016-11-01	54787.22	2907
18	2016-12-01	81317.85	3503
19	2017-01-01	58010.37	3381
20	2017-02-01	27311.03	1300
21	2017-03-01	43592.32	1926
22	2017-04-01	69513.61	1601
23	2017-05-01	57322.57	1160
24	2017-06-01	77155.31	2373
25	2017-07-01	85107.24	2097
26	2017-08-01	21384.64	653
27	2017-09-01	40919.94	888
28	2017-10-01	39589.99	1286
29	2017-11-01	107899.90	4071
30	2017-12-01	122175.16	1893
31	2018-01-01	62063.38	2041
32	2018-02-01	28985.01	1146
33	2018-03-01	52089.04	2127

	mois	Spend	Conversions
34	2018-04-01	16617.12	431
35	2018-05-01	93073.46	1792

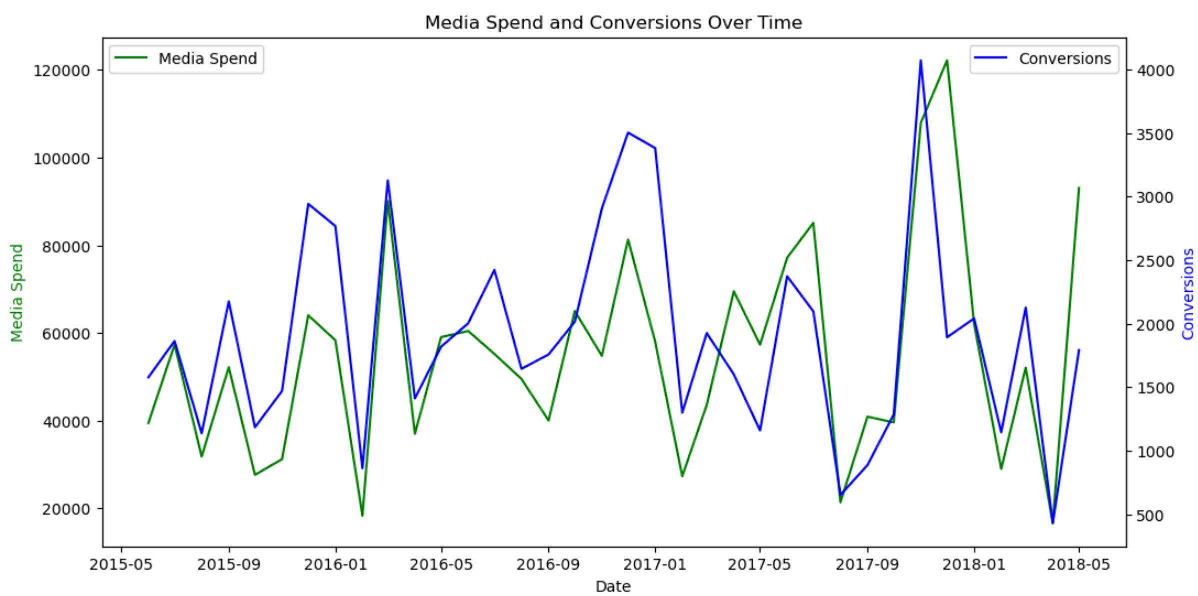
In [44]:
```
import matplotlib.pyplot as plt

# Plot the data
plt.figure(figsize=(12, 6))
plt.plot(data['mois'], data['Conversions'], label='Conversions')
plt.plot(data['mois'], data['Spend'], label='Media Spend')
plt.xlabel('Date')
plt.ylabel('Values')
plt.title('Conversions and Media Spend Over Time')
plt.legend()
plt.show()
```



In [45]:
```
import matplotlib.pyplot as plt

# Plot the data
plt.figure(figsize=(12, 6))
plt.plot(data['mois'], data['Conversions'], label='Conversions')
plt.plot(data['mois'], data['Spend'], label='Media Spend')
plt.xlabel('Date')
plt.ylabel('Values')
plt.title('Conversions and Media Spend Over Time')
plt.legend()
plt.show()
```

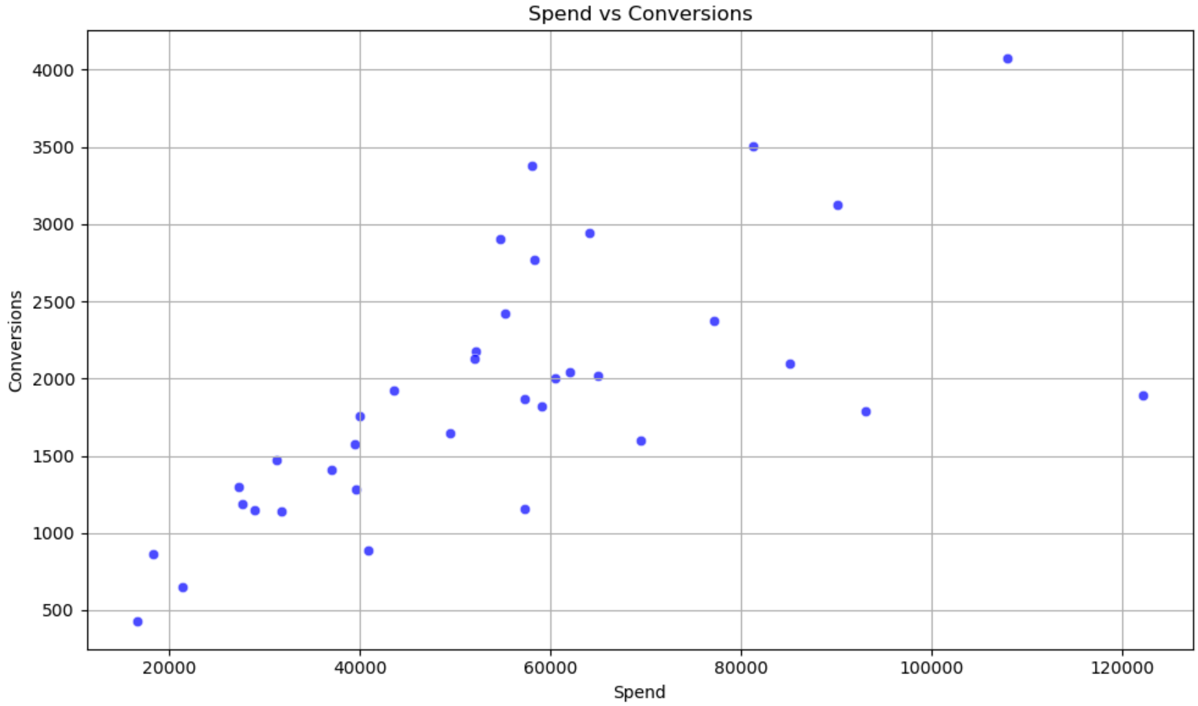


In [51]:
```
# Set up the figure and first axis
plt.figure(figsize=(12, 6))
ax1 = plt.gca()  # Get the current axis
sns.lineplot(data=data, x='mois', y='Spend', color="g", ax=ax1, label='Me

# Create a second y-axis for the Conversions
ax2 = ax1.twinx()
sns.lineplot(data=data, x='mois', y='Conversions', color="b", ax=ax2, lal

# Adding labels and title
ax1.set_xlabel('Date')
ax1.set_ylabel('Media Spend', color='g')
ax2.set_ylabel('Conversions', color='b')
plt.title('Media Spend and Conversions Over Time')

# Add legends for both axes
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Show the plot
plt.show()
```



In [77]:
```
#seasonality in data
```

In [76]:
```
spend = data['Spend']
conversions = data['Conversions']

# Plotting using matplotlib and seaborn
plt.figure(figsize=(10, 6))
sns.scatterplot(x=spend, y=conversions, color='blue', alpha=0.7)
plt.title('Spend vs Conversions')
plt.xlabel('Spend')
plt.ylabel('Conversions')
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [78]:
```
#linearity in data
```

In [ ]:

SARIMAX

In [54]:
```
from sklearn.model_selection import train_test_split

# Convert the index to a regular column if necessary
data.reset_index(drop=True, inplace=True)

# Split the data while keeping the time order
train, test = train_test_split(data, test_size=2, shuffle=False)

# Now, use the train and test sets as before in your ARIMAX setup
from statsmodels.tsa.statespace.sarimax import SARIMAX

arimax_model = SARIMAX(train['Conversions'], exog=train[['Spend']], orde
arimax_result = arimax_model.fit(disp=False)
arimax_forecast = arimax_result.forecast(steps=2, exog=test[['Spend']])
```

```
/Users/mohammedjawhar/anaconda3/envs/madc/lib/python3.11/site-packages/
statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible
starting MA parameters found. Using zeros as starting parameters.
  warn('Non-invertible starting MA parameters found.'
/Users/mohammedjawhar/anaconda3/envs/madc/lib/python3.11/site-packages/
statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few observa
tions to estimate starting parameters for seasonal ARMA. All parameters
except for variances will be set to zeros.
  warn('Too few observations to estimate starting parameters%s.'
/Users/mohammedjawhar/anaconda3/envs/madc/lib/python3.11/site-packages/
statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood o
ptimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

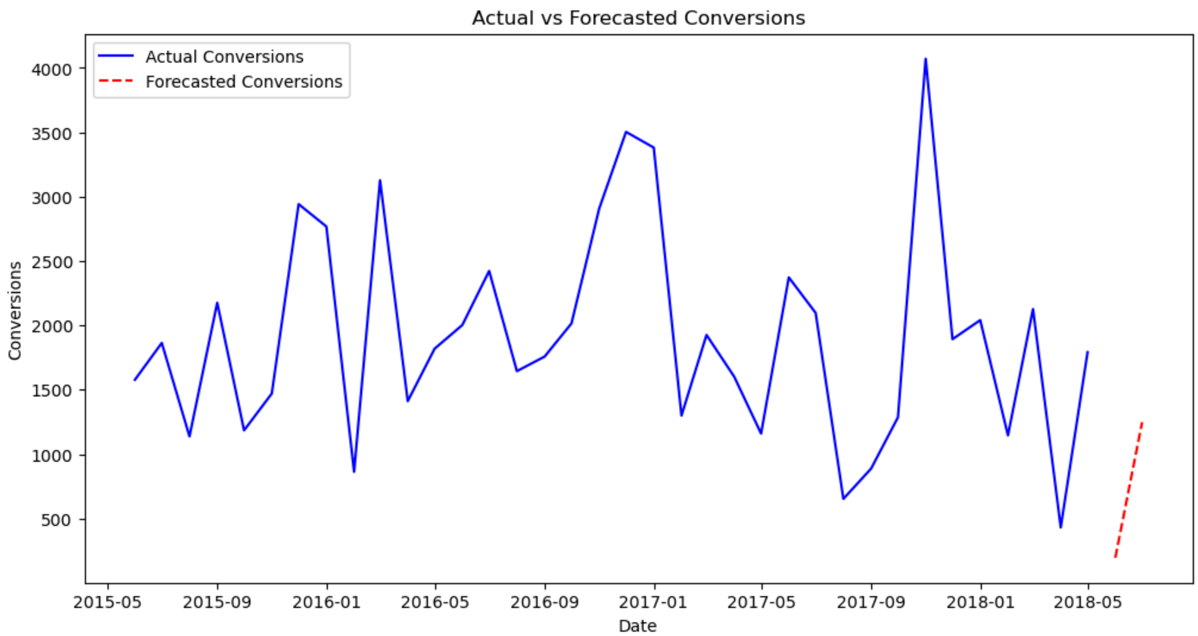
In [55]:
```
arimax_forecast
```

Out[55]:
```
34     195.628359
35    1248.805674
Name: predicted_mean, dtype: float64
```

In [62]:
```
# Assuming 'test' contains the actual conversions and 'arimax_forecast' c
actual = test['Conversions'].values
predicted = arimax_forecast.values

# Calculate MAE, MSE, and RMSE
r2 = r2_score(actual, predicted)
mae = mean_absolute_error(actual, predicted)
mse = mean_squared_error(actual, predicted)
rmse = np.sqrt(mse)

print("r2_score(r2):", r2)
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
r2_score(r2): 0.6215991879856424
Mean Absolute Error (MAE): 389.2829834610633
Mean Squared Error (MSE): 175229.94262781172
Root Mean Squared Error (RMSE): 418.6047570534904
```

In [56]:
```
# Plot actual conversions
plt.figure(figsize=(12, 6))
plt.plot(data['mois'], data['Conversions'], label='Actual Conversions', c

# Prepare to add ARIMAX forecasted values
# Create a new timestamp for the forecasted months and append to the exi
forecast_dates = pd.date_range(start=data['mois'].iloc[-1] + pd.DateOffs

# Plot ARIMAX forecasted conversions
plt.plot(forecast_dates, arimax_forecast, label='Forecasted Conversions'

# Adding labels and title
plt.xlabel('Date')
plt.ylabel('Conversions')
plt.title('Actual vs Forecasted Conversions')
plt.legend()

# Show the plot
plt.show()
```



Prophet

In [64]:
```
# !pip install prophet
```

In [66]:
```
data
```

Out[66]:

	mois	Spend	Conversions
0	2015-06-01	39456.23	1578
1	2015-07-01	57338.93	1864
2	2015-08-01	31802.36	1138
3	2015-09-01	52195.88	2176
4	2015-10-01	27642.56	1185
5	2015-11-01	31215.60	1471
6	2015-12-01	64036.93	2942
7	2016-01-01	58342.15	2768
8	2016-02-01	18326.47	863
9	2016-03-01	90135.24	3127
10	2016-04-01	37002.10	1412
11	2016-05-01	59045.12	1819
12	2016-06-01	60456.89	2003
13	2016-07-01	55215.63	2423
14	2016-08-01	49507.75	1645
15	2016-09-01	40022.35	1758
16	2016-10-01	65003.56	2015
17	2016-11-01	54787.22	2907
18	2016-12-01	81317.85	3503
19	2017-01-01	58010.37	3381
20	2017-02-01	27311.03	1300
21	2017-03-01	43592.32	1926
22	2017-04-01	69513.61	1601
23	2017-05-01	57322.57	1160
24	2017-06-01	77155.31	2373
25	2017-07-01	85107.24	2097
26	2017-08-01	21384.64	653
27	2017-09-01	40919.94	888
28	2017-10-01	39589.99	1286
29	2017-11-01	107899.90	4071
30	2017-12-01	122175.16	1893
31	2018-01-01	62063.38	2041
32	2018-02-01	28985.01	1146
33	2018-03-01	52089.04	2127

	mois	Spend	Conversions
34	2018-04-01	16617.12	431
35	2018-05-01	93073.46	1792

In [67]:
```
from prophet import Prophet
import pandas as pd

# Convert 'mois' to datetime if not already
data['mois'] = pd.to_datetime(data['mois'])

# Prepare the data for Prophet
prophet_df = data.rename(columns={'mois': 'ds', 'Conversions': 'y'})
prophet_df['Spend'] = data['Spend']  # Including Spend as a regressor

# Initialize and fit the Prophet model with Spend as an additional regres
prophet_model = Prophet()
prophet_model.add_regressor('Spend')
prophet_model.fit(prophet_df)

# Create a dataframe for future predictions
future_dates = prophet_model.make_future_dataframe(periods=2, freq='MS')

# Add future values of Spend assuming a continuation of the last observe
# This is just a placeholder; adjust according to your knowledge of futu
future_spend = [data['Spend'].iloc[-1], data['Spend'].iloc[-1]]
future_dates['Spend'] = [data['Spend'].iloc[-1]] * len(data['Spend']) + 

# Forecast
prophet_forecast = prophet_model.predict(future_dates)
prophet_forecast[['ds', 'yhat']].tail()
```

```
21:00:27 - cmdstanpy - INFO - Chain [1] start processing
21:00:27 - cmdstanpy - INFO - Chain [1] done processing
```

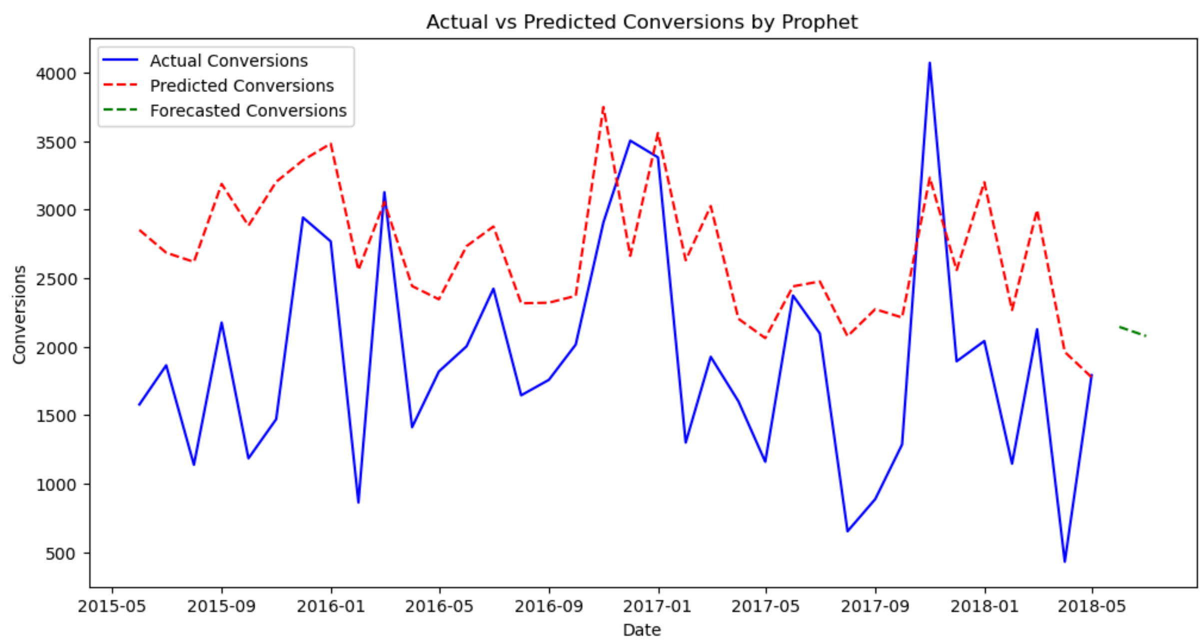
Out[67]:

	ds	yhat
33	2018-03-01	2997.043243
34	2018-04-01	1961.154118
35	2018-05-01	1776.652279
36	2018-06-01	2144.273308
37	2018-07-01	2077.143710

In [70]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Extract the actual and predicted values where predictions exist
actual = prophet_df['y']
predicted = prophet_forecast['yhat'].iloc[:len(prophet_df)]

# Calculate MAE, MSE, and RMSE
mae = mean_absolute_error(actual, predicted)
mse = mean_squared_error(actual, predicted)
rmse = np.sqrt(mse)
r2 = r2_score(actual, predicted)


print("r2_score (r2):", r2)
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
r2_score (r2): -0.47079220648883524
Mean Absolute Error (MAE): 872.7503083643517
Mean Squared Error (MSE): 981221.565115409
Root Mean Squared Error (RMSE): 990.5662850690048
```

In [69]:
```
import matplotlib.pyplot as plt

# Plot actual conversions
plt.figure(figsize=(12, 6))
plt.plot(prophet_df['ds'], prophet_df['y'], label='Actual Conversions', 

# Plot predicted conversions from Prophet
# Ensure to only plot the predictions for the existing data points
predicted = prophet_forecast.set_index('ds')['yhat'].iloc[:len(prophet_d
plt.plot(predicted.index, predicted, label='Predicted Conversions', colo

# Highlight the forecast period
future_predicted = prophet_forecast.set_index('ds')['yhat'].iloc[len(pro
plt.plot(future_predicted.index, future_predicted, label='Forecasted Con

# Adding labels and title
plt.xlabel('Date')
plt.ylabel('Conversions')
plt.title('Actual vs Predicted Conversions by Prophet')
plt.legend()

# Show the plot
plt.show()
```



Type *Markdown* and LaTeX: α^2

In [ ]:

Answers

i. Some of ideas are to use a regressor (Linear/Xgboost), but since our dataset is horizontally small, we do further research into the data. Upon further research, we figure

that the data is stationary, has somewhat of a linear relationship and had has seasonality, therefore we use SARIMAX. Upon doing a feasibility study based on Prophet, we understand that prophet is highly sensitive to outlier and has mediocre performance.

ii. A few disadvantage of using SARIMAX here is the complexity of deciding the parmeters (p,d,q) - model can be further improved by ploting ACF and PACF to get moving average(q) and p respectively. This method requires the data to be Stationary and assumes Linearity.

iii. Implementation above

In [ ]:

```

```