Predicting Ads Click-Through Rate (CTR) with Machine Learning in Python

Ads CTR Analysis, which stands for Click-Through Rate Analysis, involves assessing the efficiency of online advertisements by gauging how frequently users click on an ad's link to access the advertiser's website. This article offers a comprehensive guide on performing Ads CTR Analysis and Forecasting using Python, taking you through the process step by step.

Ads CTR Forecasting: Steps to Follow Ads CTR Analysis and Forecasting play a pivotal role in enabling businesses to evaluate the return on investment (ROI) of their advertising endeavors and make informed decisions to enhance ad effectiveness. Below outlines the sequential process we can adopt for Ads CTR Analysis and Forecasting:

1. Acquire ad data, encompassing metrics such as ad impressions (frequency of ad displays), clicks, and other pertinent factors.
2. Delve into the data to grasp its underlying characteristics and distribution. Compute fundamental statistics like the mean Click-Through Rate (CTR) and standard deviation.
3. Generate visual representations, such as line plots or bar graphs, to depict CTR patterns over time.
4. Undertake A/B tests where necessary to juxtapose the performance of distinct ad variants.
5. Scrutinize the CTR data to discern the variables influencing ad performance.
6. Construct a machine learning-based forecasting model to anticipate forthcoming CTR values. The journey initiates with data collection. I've identified an optimal dataset tailored for Ads CTR Analysis and Forecasting.

```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt
```

```
# Mounting to you own Google Colab drive
from google.colab import drive
try:
  drive.mount('/gdrive')
except:
  drive.mount('/content/gdrive', force_remount=True)
```

```
Drive already mounted at /gdrive; to attempt to forcibly remount, call drive.mount("/gdrive", force_remount=True).
```

```
%cd '/gdrive/MyDrive/CTR'
```

```
/gdrive/MyDrive/CTR
```

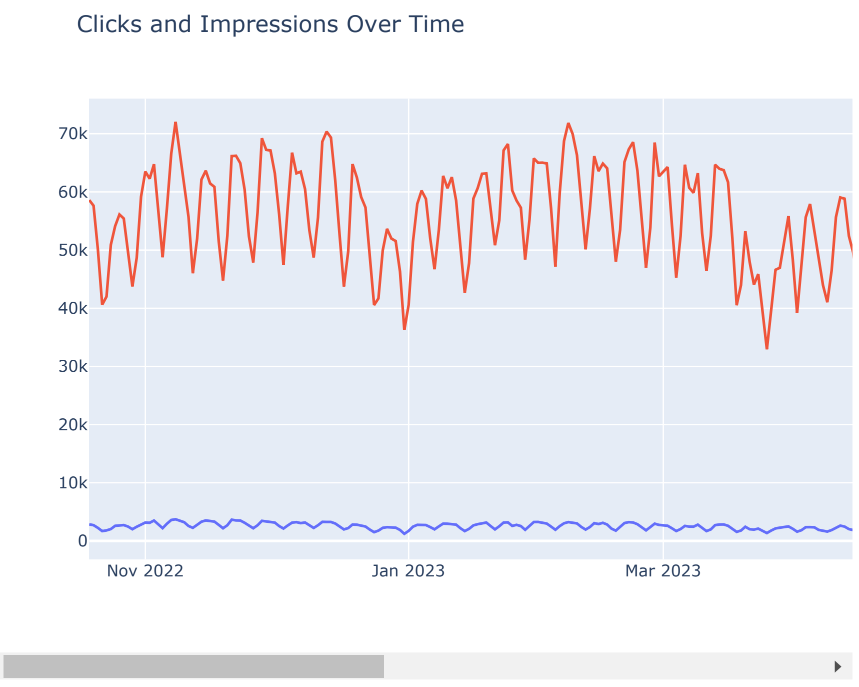
```
data = pd.read_csv('/gdrive/My Drive/CTR/ctr.csv')
print(data.head())
```

```
         Date  Clicks  Impressions
0  2022-10-19    2851        58598
1  2022-10-20    2707        57628
2  2022-10-21    2246        50135
3  2022-10-22    1686        40608
4  2022-10-23    1808        41999
```

```
# Data Preparation
data['Date'] = pd.to_datetime(data['Date'],
                                     format='%Y/%m/%d')
data.set_index('Date', inplace=True)
```

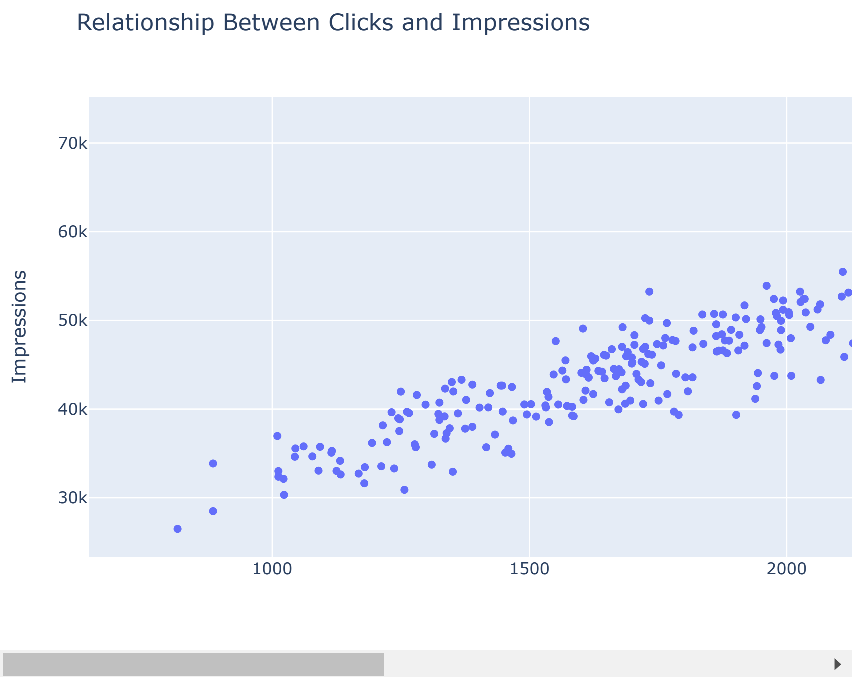
Visualizing clicks and impressions over time can provide valuable insights into the trends and patterns of ad performance.

```
# Visualize Clicks and Impressions
fig = go.Figure()
fig.add_trace(go.Scatter(x=data.index, y=data['Clicks'], mode='lines', name='Clicks'))
fig.add_trace(go.Scatter(x=data.index, y=data['Impressions'], mode='lines', name='Impressions'))
fig.update_layout(title='Clicks and Impressions Over Time')
fig.show()
```



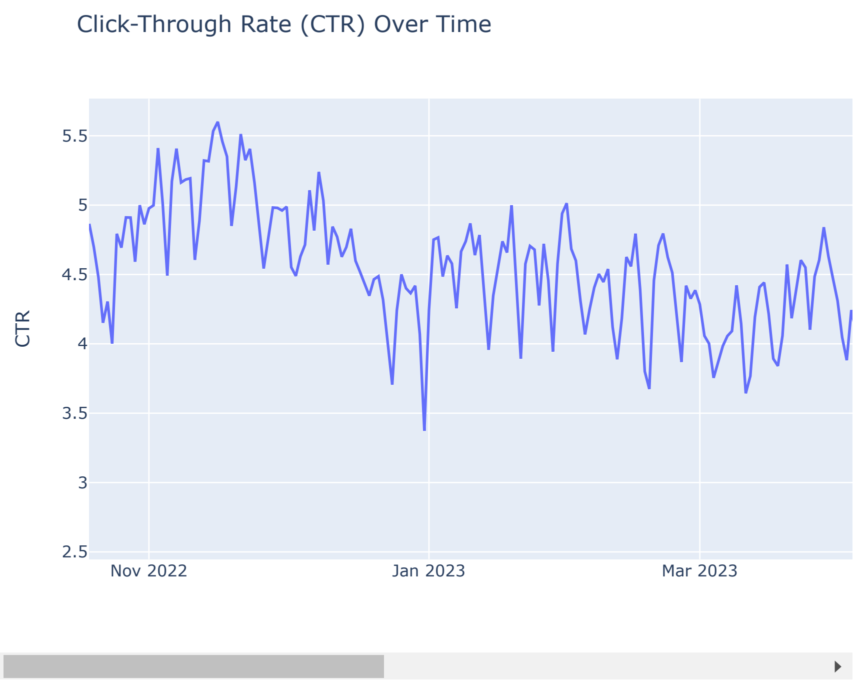
Visualizing the relationship between clicks and impressions can help understand how changes in impressions affect clicks and vice versa. One common way to visualize this relationship is by creating a scatter plot.

```
# Create a scatter plot to visualize the relationship between Clicks and Impressions
fig = px.scatter(data, x='Clicks', y='Impressions', title='Relationship Between Clicks and Impressions',
                 labels={'Clicks': 'Clicks', 'Impressions': 'Impressions'})

# Customize the layout
fig.update_layout(xaxis_title='Clicks', yaxis_title='Impressions')

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



The relationship between clicks and impressions is linear, it's insightful to calculate and visualize the Click-Through Rate (CTR) over time. CTR is typically calculated as the ratio of clicks to impressions.

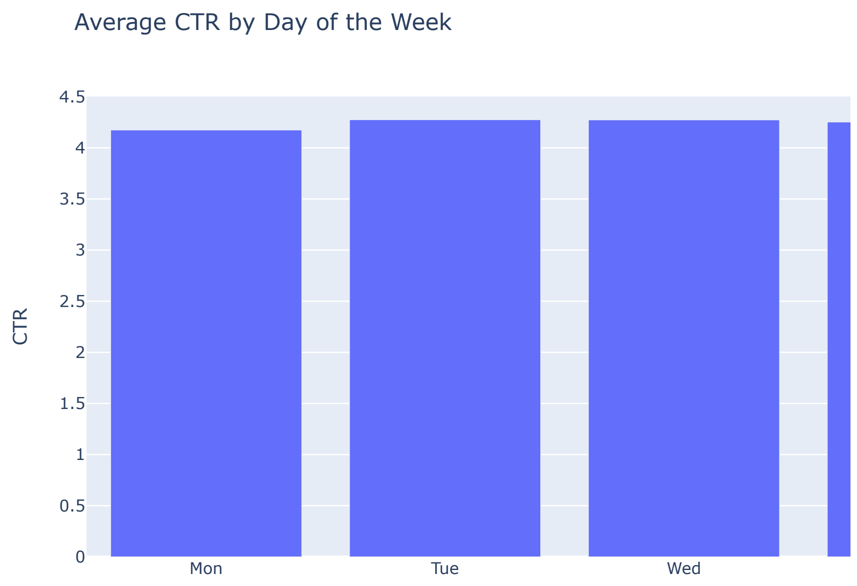
```
# Calculate and visualize CTR
data['CTR'] = (data['Clicks'] / data['Impressions']) * 100
fig = px.line(data, x=data.index, y='CTR', title='Click-Through Rate (CTR) Over Time')
fig.show()
```



Analyzing the average Click-Through Rate (CTR) by day of the week, we can calculate the mean CTR for each day and then visualize it.

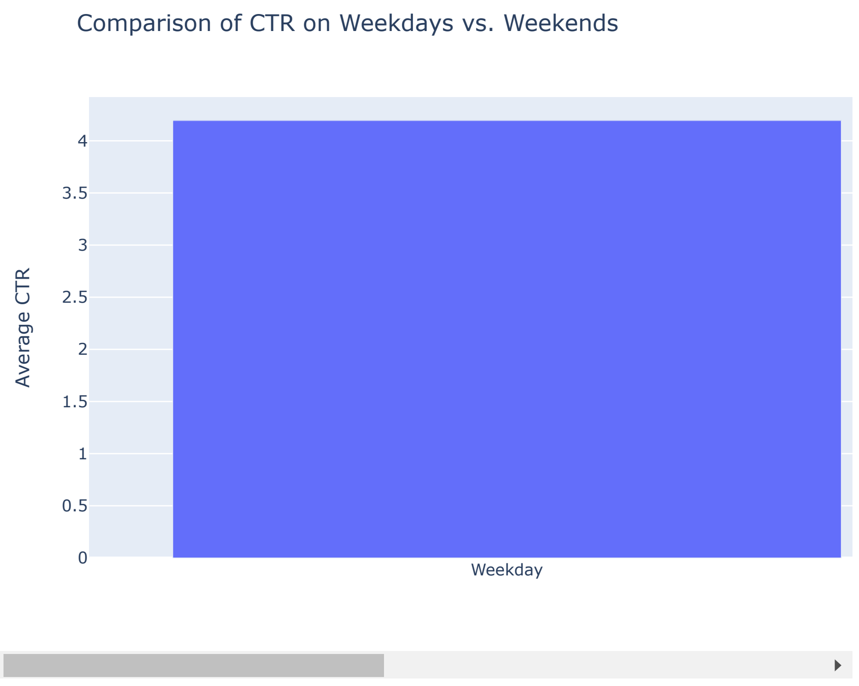
```
data['DayOfWeek'] = data.index.dayofweek
data['WeekOfMonth'] = data.index.week // 4

# EDA based on DayOfWeek
day_of_week_ctr = data.groupby('DayOfWeek')['CTR'].mean().reset_index()
day_of_week_ctr['DayOfWeek'] = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

fig = px.bar(day_of_week_ctr, x='DayOfWeek', y='CTR', title='Average CTR by Day of the Week')
fig.show()
```

```
    <ipython-input-45-de6b9b92e2b6>:2: FutureWarning:

    weekofyear and week have been deprecated, please use DatetimeIndex.isocalendar().week
```



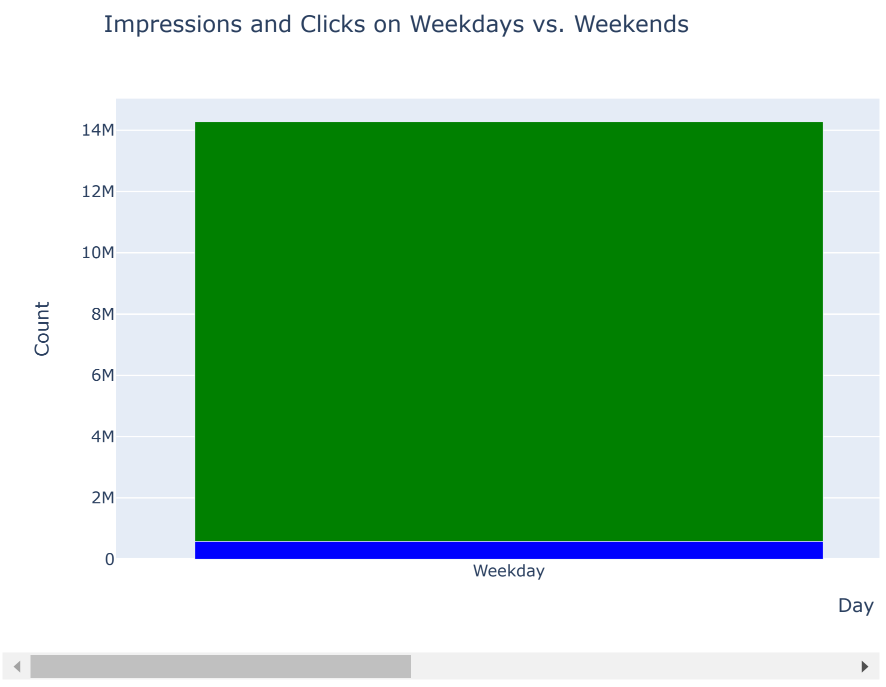
To compare the Click-Through Rate (CTR) between weekdays and weekends, you can calculate the average CTR separately for weekdays and weekends and then visualize the comparison.

```
# Create a new column 'DayCategory' to categorize weekdays and weekends
data['DayCategory'] = data['DayOfWeek'].apply(lambda x: 'Weekend' if x >= 5 else 'Weekday')

# Calculate average CTR for weekdays and weekends
ctr_by_day_category = data.groupby('DayCategory')['CTR'].mean().reset_index()

# Create a bar plot to compare CTR on weekdays vs. weekends
fig = px.bar(ctr_by_day_category, x='DayCategory', y='CTR', title='Comparison of CTR on Weekdays vs. Weekends',
             labels={'CTR': 'Average CTR'})

# Customize the layout
fig.update_layout(yaxis_title='Average CTR')

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



To compare impressions and clicks between weekdays and weekends, you can calculate the total number of impressions and clicks separately for weekdays and weekends and then visualize the comparison.

```
# Group the data by 'DayCategory' and calculate the sum of Clicks and Impressions for each category
grouped_data = data.groupby('DayCategory')[['Clicks', 'Impressions']].sum().reset_index()

# Create a grouped bar chart to visualize Clicks and Impressions on weekdays vs. weekends
fig = px.bar(grouped_data, x='DayCategory', y=['Clicks', 'Impressions'],
             title='Impressions and Clicks on Weekdays vs. Weekends',
             labels={'value': 'Count', 'variable': 'Metric'},
             color_discrete_sequence=['blue', 'green'])

# Customize the layout
fig.update_layout(yaxis_title='Count')
fig.update_xaxes(title_text='Day Category')

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

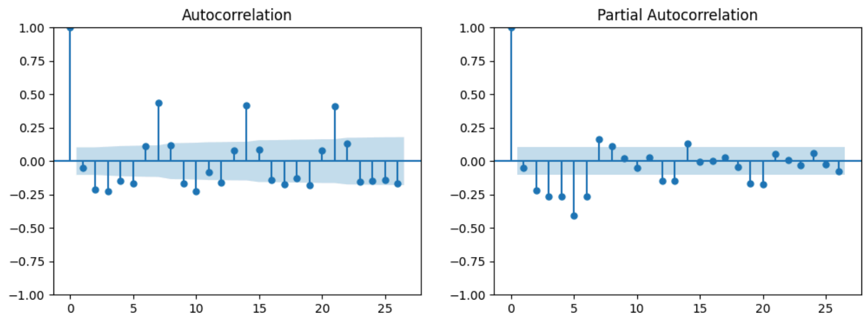


Ads CTR Forecasting

To forecast Ads Click-Through Rate (CTR) using Time Series forecasting techniques like SARIMA (Seasonal Autoregressive Integrated Moving Average), you'll need to determine the parameters p, d, and q for the SARIMA model. Here's how you can calculate these parameters:

1. **p (Autoregressive order):** This parameter represents the number of lag observations included in the model. It can be determined by examining the Autocorrelation Function (ACF) plot, where significant spikes beyond the confidence interval indicate the lag value for p.
2. **d (Integrated order):** This parameter represents the number of differences needed to make the series stationary. You can determine d by examining the order of differencing required to stabilize the series, typically done by differencing the series until it becomes stationary.
3. **q (Moving Average order):** This parameter represents the size of the moving average window. It can be determined by examining the Partial Autocorrelation Function (PACF) plot, where significant spikes beyond the confidence interval indicate the lag value for q.

```
data.reset_index(inplace=True)

from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf


# resetting index
time_series = data.set_index('Date')['CTR']

# Differencing
differenced_series = time_series.diff().dropna()

# Plot ACF and PACF of differenced time series
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
plot_acf(differenced_series, ax=axes[0])
plot_pacf(differenced_series, ax=axes[1])
plt.show()
```



We set up a SARIMAX model with order (1,1,1) and seasonal_order (1,1,1,12) based on the provided assumptions. Adjust the DataFrame and column names as per your data. Following model fitting, you can proceed with forecasting CTR values using this SARIMA model.

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

p, d, q, s = 1, 1, 1, 12

model = SARIMAX(time_series, order=(p, d, q), seasonal_order=(p, d, q, s))
results = model.fit()
print(results.summary())
```

```
    /usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning:

    No frequency information was provided, so inferred frequency D will be used.

    /usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning:

    No frequency information was provided, so inferred frequency D will be used.

                                         SARIMAX Results
    ==========================================================================================
    Dep. Variable:                                CTR   No. Observations:                  365
    Model:             SARIMAX(1, 1, 1)x(1, 1, 1, 12)   Log Likelihood                 -71.365
    Date:                            Wed, 20 Mar 2024   AIC                            152.730
    Time:                                    13:51:19   BIC                            172.048
    Sample:                                10-19-2022   HQIC                           160.418
                                         - 10-18-2023
    Covariance Type:                              opg
    ==============================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
    ------------------------------------------------------------------------------
    ar.L1          0.5266      0.070      7.513      0.000       0.389       0.664
    ma.L1         -0.9049      0.036    -25.361      0.000      -0.975      -0.835
    ar.S.L12      -0.1573      0.071     -2.225      0.026      -0.296      -0.019
    ma.S.L12      -0.9974      1.099     -0.908      0.364      -3.151       1.156
    sigma2         0.0772      0.084      0.917      0.359      -0.088       0.242
    ===================================================================================
    Ljung-Box (L1) (Q):                   5.64   Jarque-Bera (JB):                 1.20
    Prob(Q):                              0.02   Prob(JB):                         0.55
    Heteroskedasticity (H):               1.14   Skew:                            -0.01
    Prob(H) (two-sided):                  0.48   Kurtosis:                         3.28
    ===================================================================================

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

```
# Predict future values
future_steps = 100
predictions = results.predict(len(time_series), len(time_series) + future_steps - 1)
print(predictions)
```

```
    2023-10-19    3.852350
    2023-10-20    3.889426
    2023-10-21    3.820260
    2023-10-22    3.727494
    2023-10-23    3.710360
                    ...
    2024-01-22    3.545574
    2024-01-23    3.466648
    2024-01-24    3.561193
    2024-01-25    3.546697
    2024-01-26    3.580132
    Freq: D, Name: predicted_mean, Length: 100, dtype: float64
```

Visualizing the forecasted trend of Click-Through Rate (CTR) helps in understanding the predicted behavior of CTR over time. Typically, this involves plotting the observed CTR values along with the forecasted values generated by the SARIMA model.

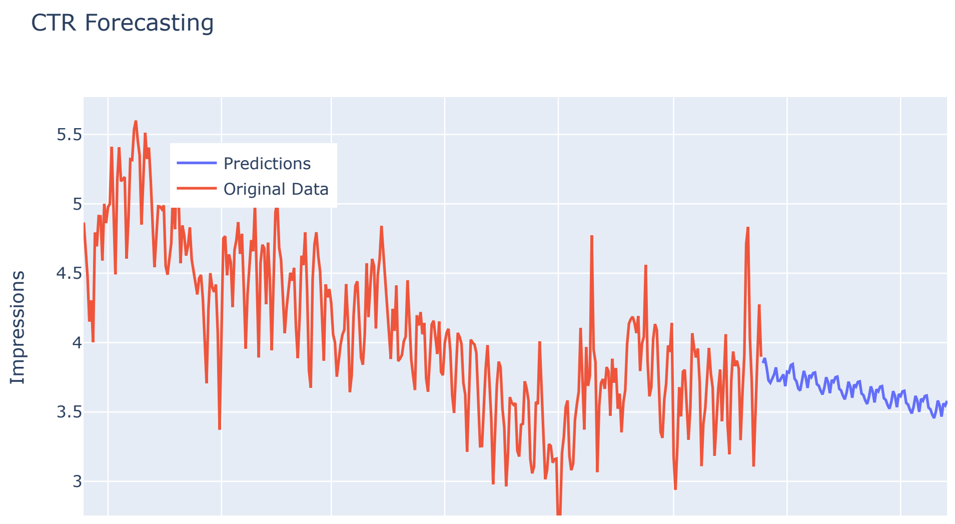
In the visualization:

1. **Observed CTR Trend:** Plot the historical CTR values over time. This provides context by showing the actual behavior of CTR in the past.
2. **Forecasted CTR Trend:** Overlay the forecasted CTR values on the same plot. These forecasted values are generated by the SARIMA model and represent the predicted future behavior of CTR.
3. **Confidence Intervals:** Optionally, include confidence intervals around the forecasted values to indicate the uncertainty of the predictions. These intervals give a range within which the actual CTR values are likely to fall.

By visualizing the forecasted trend of CTR, stakeholders can gain insights into potential future performance, aiding in decision-making processes related to advertising strategies and resource allocation.

Additionally, observing how the forecasted trend aligns with historical data can provide validation for the SARIMA model's effectiveness in capturing the underlying patterns and seasonality in the CTR data.

```
# Create a DataFrame with the original data and predictions
forecast = pd.DataFrame({'Original': time_series, 'Predictions': predictions})

# Plot the original data and predictions
fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast.index, y=forecast['Predictions'],
                         mode='lines', name='Predictions'))

fig.add_trace(go.Scatter(x=forecast.index, y=forecast['Original'],
                         mode='lines', name='Original Data'))

fig.update_layout(title='CTR Forecasting',
                  xaxis_title='Time Period',
                  yaxis_title='Impressions',
                  legend=dict(x=0.1, y=0.9),
                  showlegend=True)

fig.show()
```


CTR Forecasting
Predictions
Original Data
Impressions