Instagram Reach Forecasting using Python

Let’s start this task by importing the necessary Python libraries and the dataset:

```
In [3]: import pandas as pd
        import plotly.graph_objs as go
        import plotly.express as px
        import plotly.io as pio
        pio.templates.default = "plotly_white"
```

```
In [4]: data = pd.read_csv("Instagram-Reach.csv", encoding = 'latin-1')
        print(data.head())
```

```
                  Date  Instagram reach
0  2022-04-01T00:00:00             7620
1  2022-04-02T00:00:00            12859
2  2022-04-03T00:00:00            16008
3  2022-04-04T00:00:00            24349
4  2022-04-05T00:00:00            20532
```

```
In [5]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 365 entries, 0 to 364
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Date             365 non-null    object
 1   Instagram reach  365 non-null    int64
dtypes: int64(1), object(1)
memory usage: 5.8+ KB
```

I’ll convert the Date column into datetime datatype to move forward:

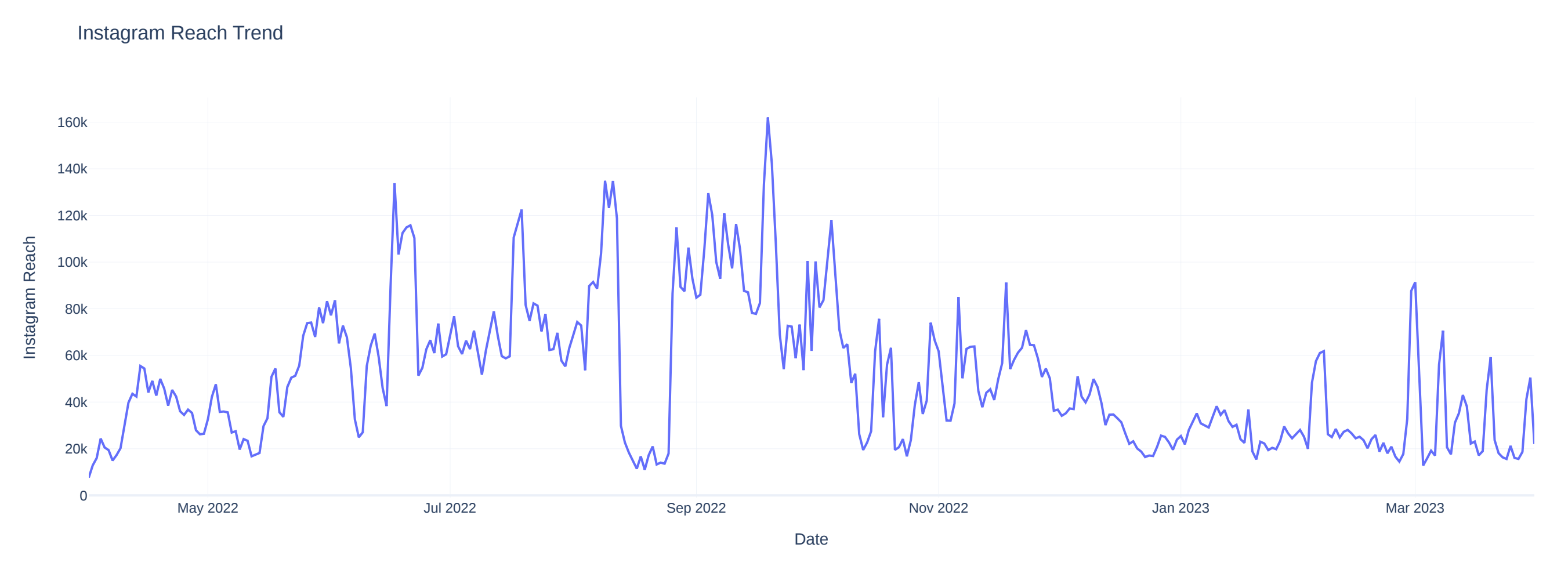
```
In [7]: data['Date'] = pd.to_datetime(data['Date'])
        print(data.head())
```

```
        Date  Instagram reach
0 2022-04-01             7620
1 2022-04-02            12859
2 2022-04-03            16008
3 2022-04-04            24349
4 2022-04-05            20532
```

Analyzing Reach

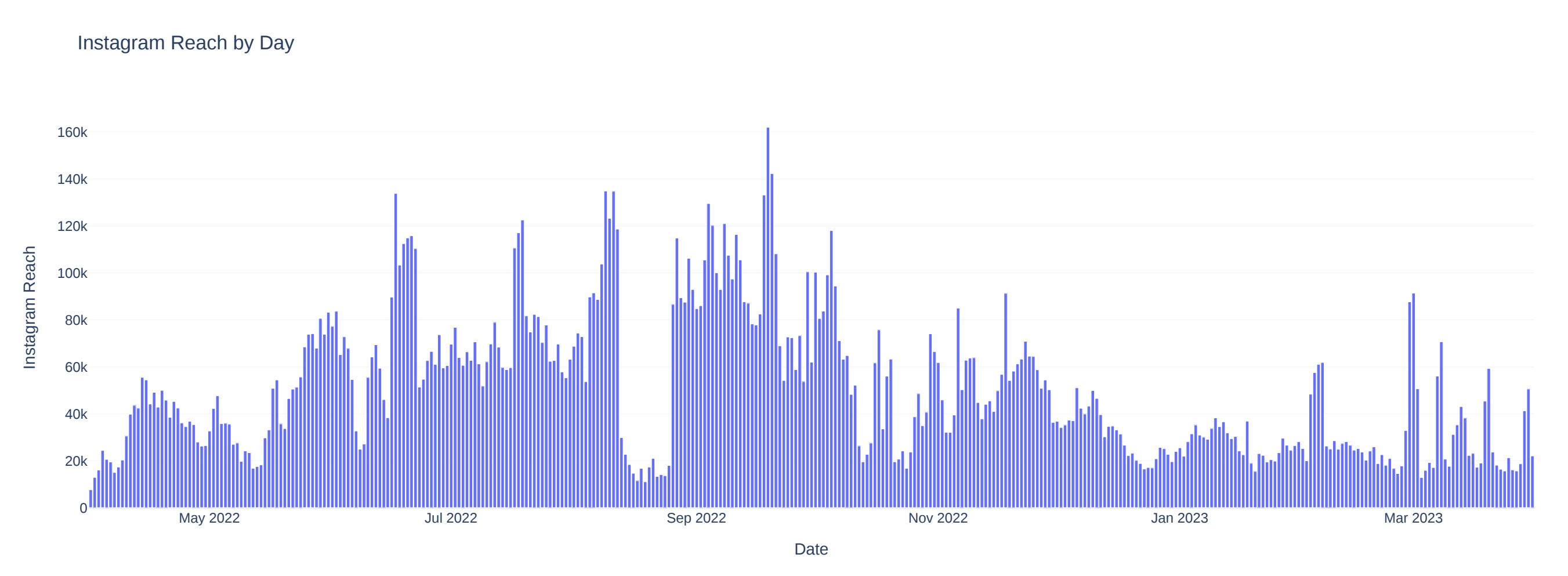
Let’s analyze the trend of Instagram reach over time using a line chart:

```
In [10]: fig = go.Figure()
         fig.add_trace(go.Scatter(x=data['Date'],
                                  y=data['Instagram reach'],
                                  mode='lines', name='Instagram reach'))
         fig.update_layout(title='Instagram Reach Trend', xaxis_title='Date',
                           yaxis_title='Instagram Reach')
         fig.show()
```



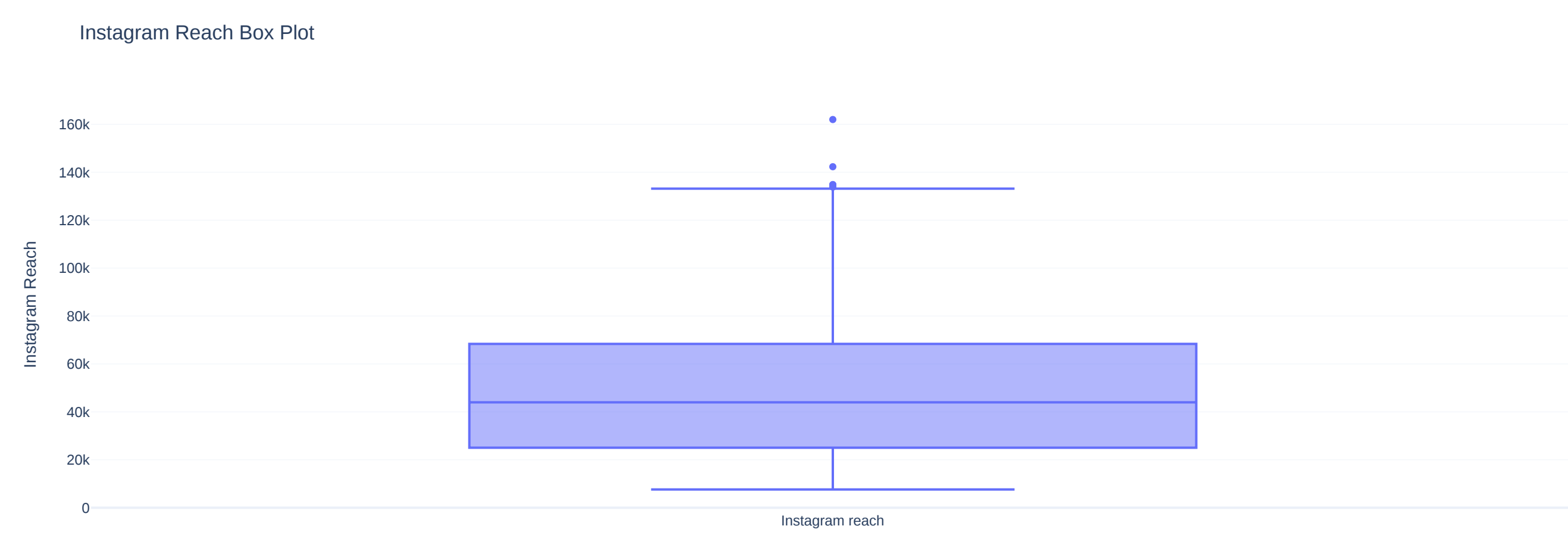
Now let’s analyze Instagram reach for each day using a bar chart:

```
In [12]: fig = go.Figure()
         fig.add_trace(go.Bar(x=data['Date'],
                              y=data['Instagram reach'],
                              name='Instagram reach'))
         fig.update_layout(title='Instagram Reach by Day',
                           xaxis_title='Date',
                           yaxis_title='Instagram Reach')
         fig.show()
```



Now let’s analyze the distribution of Instagram reach using a box plot:

```
In [14]: fig = go.Figure()
         fig.add_trace(go.Box(y=data['Instagram reach'],
                              name='Instagram reach'))
         fig.update_layout(title='Instagram Reach Box Plot',
                           yaxis_title='Instagram Reach')
         fig.show()
```



Now let’s create a day column and analyze reach based on the days of the week. To create a day column, we can use the dt.day_name() method to extract the day of the week from the Date column:

```
In [16]: data['Day'] = data['Date'].dt.day_name()
         print(data.head())
```

```
        Date  Instagram reach       Day
0 2022-04-01             7620    Friday
1 2022-04-02            12859  Saturday
2 2022-04-03            16008    Sunday
3 2022-04-04            24349    Monday
4 2022-04-05            20532   Tuesday
```

Now let’s analyze the reach based on the days of the week. For this, we can group the DataFrame by the Day column and calculate the mean, median, and standard deviation of the Instagram reach column for each day:

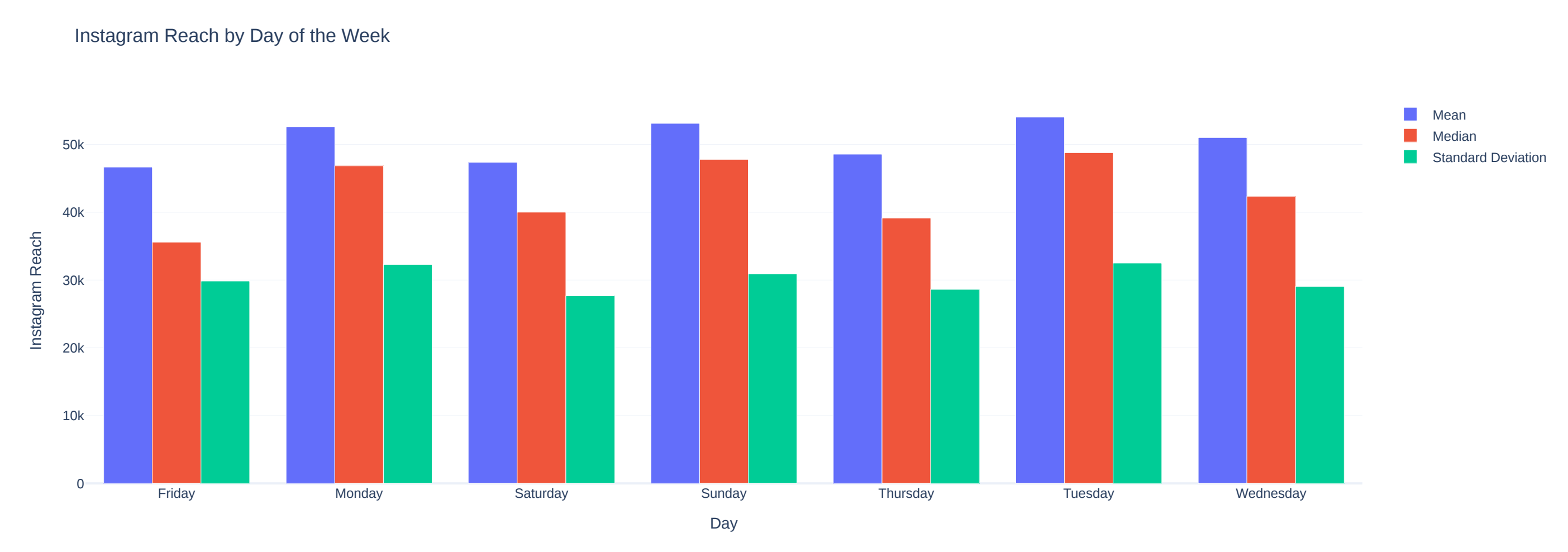
```
In [18]: import numpy as np

         day_stats = data.groupby('Day')['Instagram reach'].agg(['mean', 'median', 'std']).reset_index()
         print(day_stats)
```

```
         Day          mean   median           std
0     Friday  46666.849057  35574.0  29856.943036
1     Monday  52621.692308  46853.0  32296.071347
2   Saturday  47374.750000  40012.0  27667.043634
3     Sunday  53114.173077  47797.0  30906.162384
4   Thursday  48570.923077  39150.0  28623.220625
5    Tuesday  54030.557692  48786.0  32503.726482
6  Wednesday  51017.269231  42320.5  29047.869685
```

Now, let’s create a bar chart to visualize the reach for each day of the week:

```
In [20]: fig = go.Figure()
         fig.add_trace(go.Bar(x=day_stats['Day'],
                              y=day_stats['mean'],
                              name='Mean'))
         fig.add_trace(go.Bar(x=day_stats['Day'],
                              y=day_stats['median'],
                              name='Median'))
         fig.add_trace(go.Bar(x=day_stats['Day'],
                              y=day_stats['std'],
                              name='Standard Deviation'))
         fig.update_layout(title='Instagram Reach by Day of the Week',
                           xaxis_title='Day',
                           yaxis_title='Instagram Reach')
         fig.show()
```



Instagram Reach Forecasting using Time Series Forecasting

To forecast reach, we can use Time Series Forecasting. Let’s see how to use Time Series Forecasting to forecast the reach of my Instagram account step-by-step.

Let’s look at the Trends and Seasonal patterns of Instagram reach:

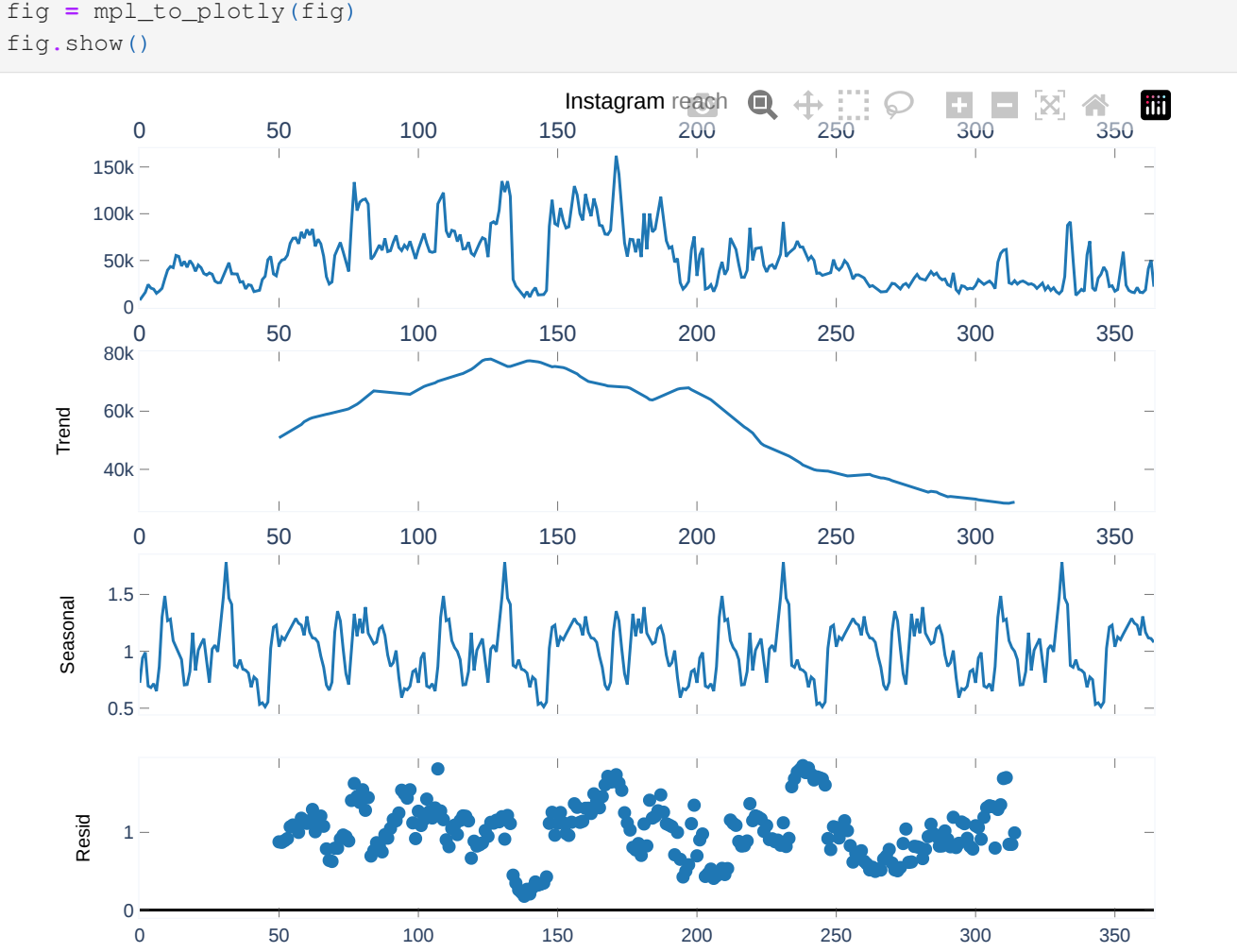
```
In [23]: from plotly.tools import mpl_to_plotly
         import matplotlib.pyplot as plt
         from statsmodels.tsa.seasonal import seasonal_decompose

         data = data[["Date", "Instagram reach"]]

         result = seasonal_decompose(data['Instagram reach'],
                                     model='multiplicative',
                                     period=100)

         fig = plt.figure()
         fig = result.plot()

         fig = mpl_to_plotly(fig)
         fig.show()
```



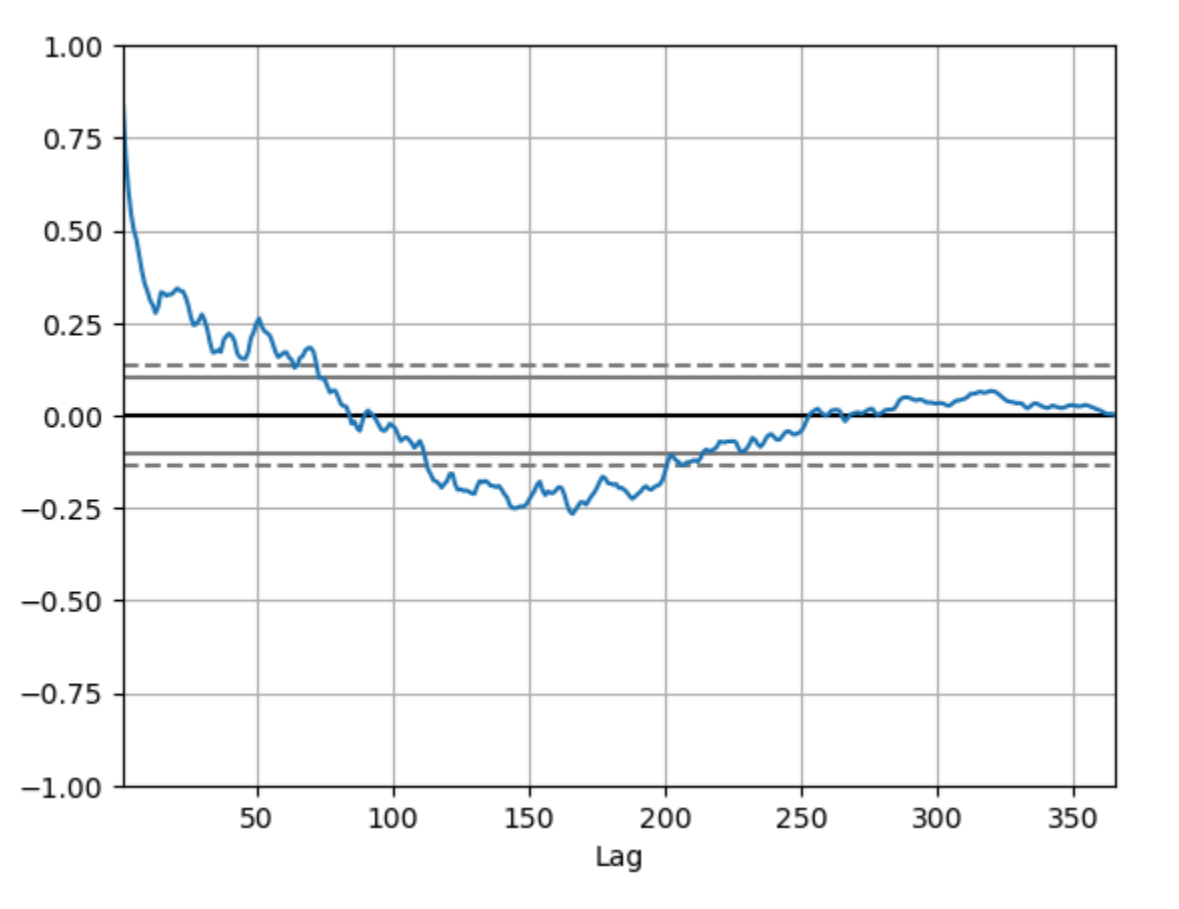

```
<Figure size 640x480 with 0 Axes>
```

The reach is affected by seasonality, so we can use the SARIMA model to forecast the reach of the Instagram account.

Now here’s how to visualize an autocorrelation plot to find the value of p:

```
In [24]: pd.plotting.autocorrelation_plot(data["Instagram reach"])
```

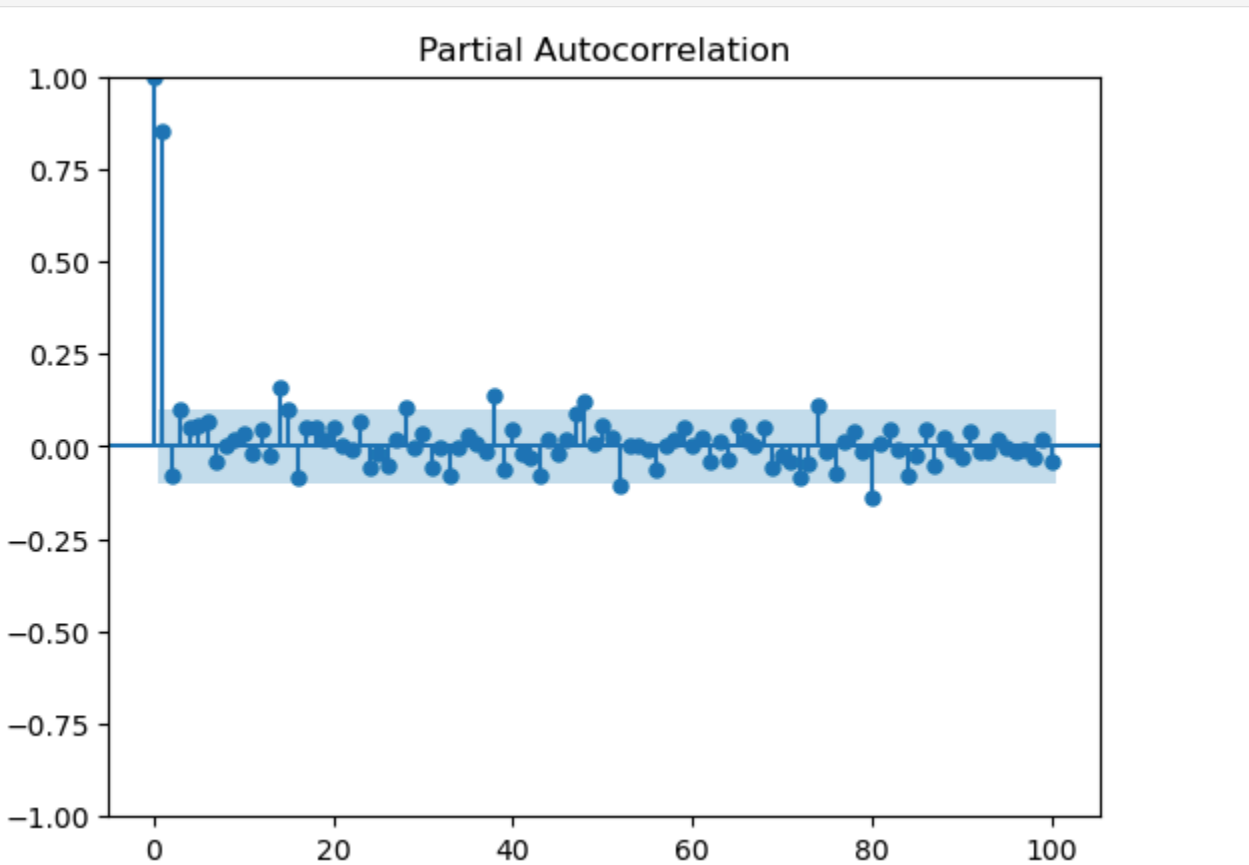
```
Out[24]: <Axes: xlabel='Lag', ylabel='Autocorrelation'>
```

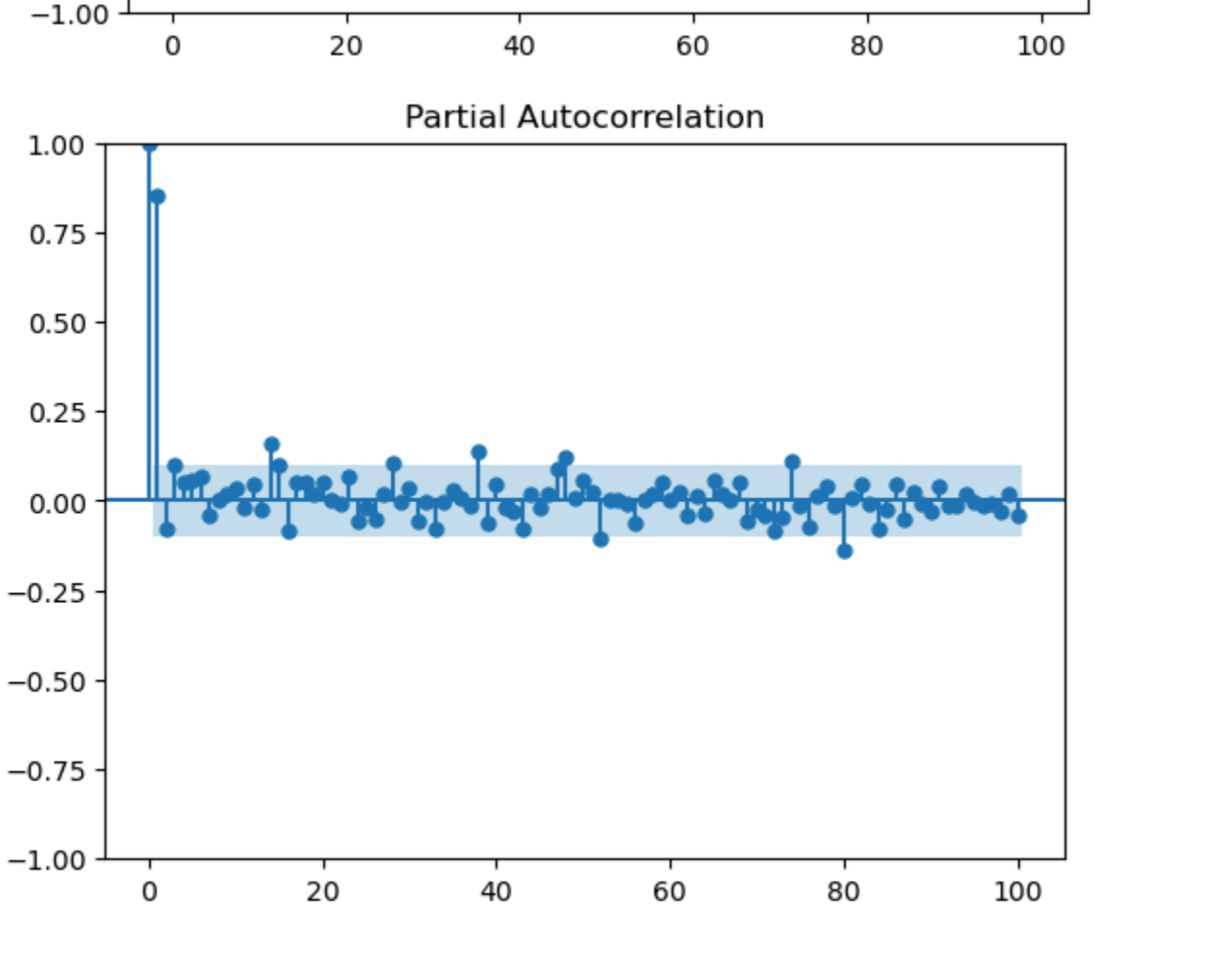


And now here’s how to visualize a partial autocorrelation plot to find the value of q:

```
In [25]: from statsmodels.graphics.tsaplots import plot_pacf
         plot_pacf(data["Instagram reach"], lags = 100)
```

Out[25]:





Now here’s how to train a model using SARIMA:

```
In [26]: p, d, q = 8, 1, 2
```

```
In [28]: import statsmodels.api as sm
         import warnings
         model=sm.tsa.statespace.SARIMAX(data['Instagram reach'],
                                         order=(p, d, q),
                                         seasonal_order=(p, d, q, 12))
         model=model.fit()
         print(model.summary())
```

```
C:\Users\prajw\anaconda3\Lib\site-packages\statsmodels\base\model.py:607: ConvergenceWarning:

Maximum Likelihood optimization failed to converge. Check mle_retvals
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                    Instagram reach   No. Observations:                  365
Model:             SARIMAX(8, 1, 2)x(8, 1, 2, 12)   Log Likelihood               -3938.513
Date:                            Tue, 20 May 2025   AIC                           7919.027
Time:                                    15:01:20   BIC                           8000.163
Sample:                                         0   HQIC                          7951.315
                                            - 365
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1897      6.612      0.029      0.977     -12.769      13.149
ar.L2          0.4757      6.156      0.077      0.938     -11.590      12.542
ar.L3         -0.1184      1.411     -0.084      0.933      -2.884       2.647
ar.L4          0.0424      0.266      0.159      0.874      -0.480       0.565
ar.L5         -0.0206      0.186     -0.111      0.912      -0.386       0.345
ar.L6          0.0314      0.265      0.118      0.906      -0.489       0.552
ar.L7          0.0092      0.421      0.022      0.982      -0.815       0.834
ar.L8         -0.0131      0.235     -0.056      0.955      -0.474       0.448
ma.L1         -0.2231      6.608     -0.034      0.973     -13.175      12.729
ma.L2         -0.7125      6.355     -0.112      0.911     -13.168      11.743
ar.S.L12      -1.0859      1.514     -0.717      0.473      -4.053       1.881
ar.S.L24      -1.7440      2.191     -0.796      0.426      -6.037       2.549
ar.S.L36      -1.4291      1.881     -0.760      0.447      -5.116       2.257
ar.S.L48      -1.0823      1.532     -0.707      0.480      -4.084       1.920
ar.S.L60      -0.7809      1.091     -0.716      0.474      -2.919       1.357
ar.S.L72      -0.4479      0.773     -0.579      0.563      -1.964       1.068
ar.S.L84      -0.2220      0.494     -0.449      0.653      -1.191       0.747
ar.S.L96      -0.0531      0.243     -0.218      0.827      -0.530       0.424
ma.S.L12       0.2238      1.515      0.148      0.883      -2.745       3.193
ma.S.L24       0.8232      1.243      0.662      0.508      -1.612       3.259
sigma2      4.863e+08   1.42e-07   3.43e+15      0.000    4.86e+08    4.86e+08
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):               215.14
Prob(Q):                              0.92   Prob(JB):                         0.00
Heteroskedasticity (H):               0.71   Skew:                             0.29
Prob(H) (two-sided):                  0.07   Kurtosis:                         6.79
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 3.05e+33. Standard errors may be unstable.
```

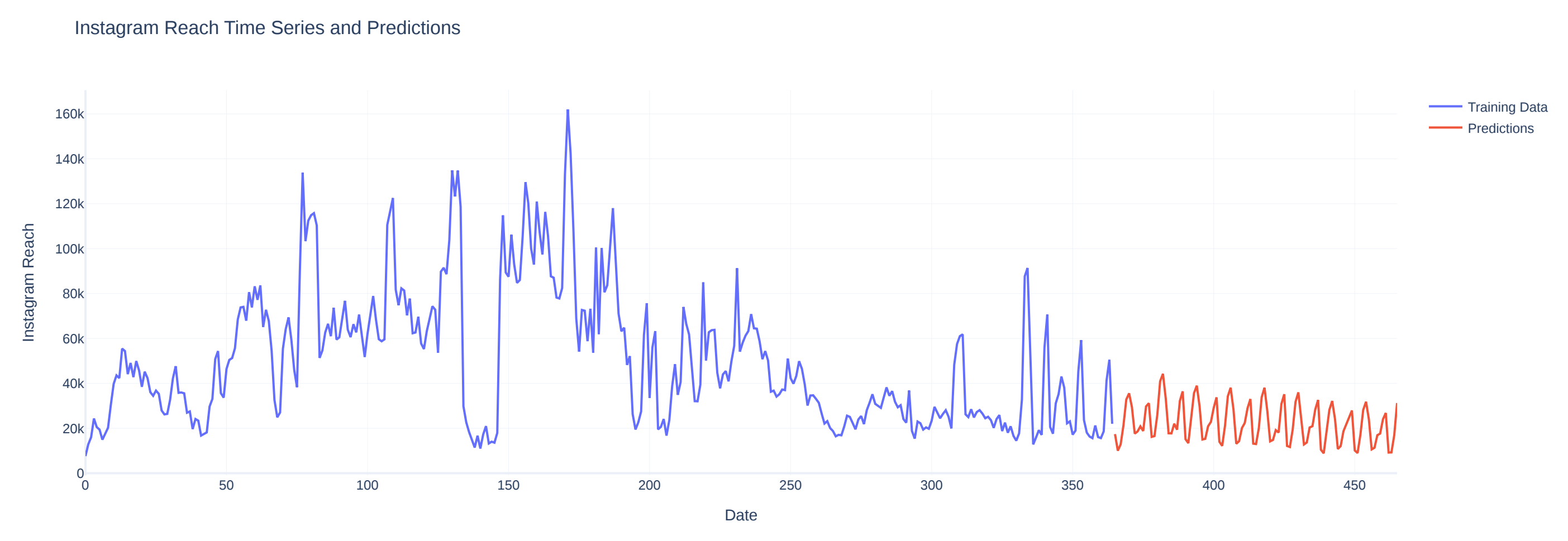
Now let’s make predictions using the model and have a look at the forecasted reach:

```
In [32]: predictions = model.predict(len(data), len(data)+100)

         trace_train = go.Scatter(x=data.index,
                                  y=data["Instagram reach"],
                                  mode="lines",
                                  name="Training Data")
         trace_pred = go.Scatter(x=predictions.index,
                                 y=predictions,
                                 mode="lines",
                                 name="Predictions")

         layout = go.Layout(title="Instagram Reach Time Series and Predictions",
                            xaxis_title="Date",
                            yaxis_title="Instagram Reach")

         fig = go.Figure(data=[trace_train, trace_pred], layout=layout)
         fig.show()
```



So this is how we can forecast the reach of an Instagram account using Time Series Forecasting.

Summary

Instagram reach prediction is the process of predicting the number of people that an Instagram post, story, or other content will be reached, based on historical data and various other factors.