Data Set: Daily Website Visitors

Objective: This project analyzes a dataset of daily website visitors to forecast future visits using various time series models, including Moving Average, Exponential Smoothing, and ARIMA. The primary objective is to create accurate models for predicting visitor patterns and to evaluate the best-performing model using statistical metrics.

1) Calling Libraries:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.tsa.stattools import adfuller
```

- **pandas** is used for handling and manipulating structured data, especially time series, in a **DataFrame** format.
- **matplotlib** is a plotting library. **pyplot** is used to create visualizations like time series plots, ACF, and PACF plots.
- **numpy** is a library for numerical computations, often used for mathematical operations on arrays or matrices.
- **ARIMA** is a time series model combining Autoregression (AR), Differencing (I), and Moving Average (MA) to forecast future values based on past data.
- **adfuller** is used to perform the **Augmented Dickey-Fuller (ADF) test** to check if a time series is stationary or needs differencing.
- Used to create **ACF** (Autocorrelation) and **PACF** (Partial Autocorrelation) plots, which help determine ARIMA parameters.
- Performs the **Ljung-Box test** to check if residuals are independently distributed (no autocorrelation).

2) Import and Load Data

```
# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/PA Internal
Test/daily_website_visitor_DS7_final.csv', index_col='Date', parse_dates=True)

# Display the first few rows
print(data.head())
```

Output

```
           Unique.Visits
Date
9/14/2014           1582
9/15/2014           2528
9/16/2014           2630
9/17/2014           2614
9/18/2014           2366
```

Observation:
The dataset contains **2167 records** spanning from **9/14/2014 to 8/19/2020**, and no missing values are present in the dataset.

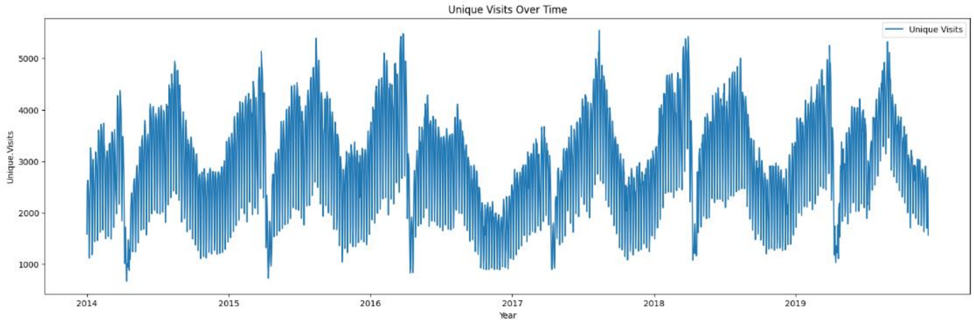
Inference:

- Since the data is clean and well-structured, there’s no need for further handling of missing values.
- The Unique.Visits column contains integer values representing the number of daily visits, making it ready for time series analysis.

3) Exploratory Data Analysis (EDA):

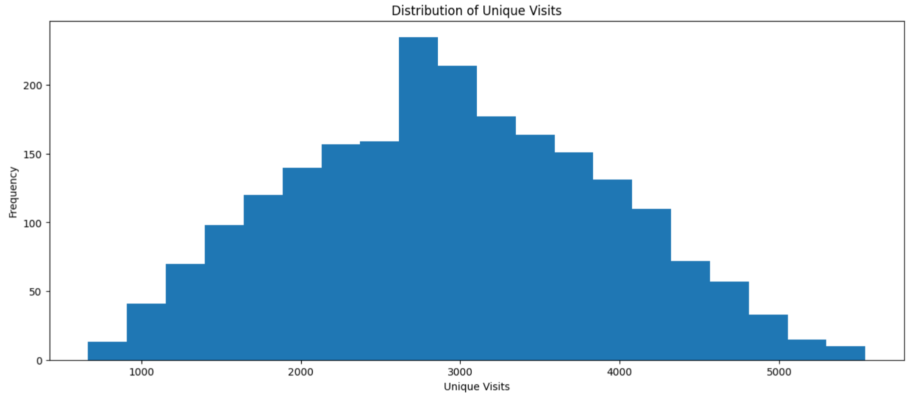
```
import matplotlib.pyplot as plt

# Plot the time series for 'Unique Visits' prices
plt.figure(figsize=(20, 6))
plt.plot(data['Unique.Visits'], label='Unique Visits')
plt.title('Unique Visits Over Time')
plt.xlabel('Year')
plt.ylabel('Unique.Visits')
# Convert the index to DatetimeIndex and extract year
plt.xticks(data.index[::365], pd.to_datetime(data.index[::365]).year)  # Show year labels every 365 days
plt.legend()
plt.show()
```

Output



The time series plot of **unique daily visits** shows clear **fluctuations** in website traffic, with recurring peaks and valleys. This could indicate a **seasonal trend** in the data, where visitor counts rise and fall periodically over time.

```
plt.figure(figsize=(15, 6))
plt.hist(data['Unique.Visits'], bins=20)
plt.title('Distribution of Unique Visits')
plt.xlabel('Unique Visits')
plt.ylabel('Frequency')
plt.show()
```



The histogram shows a **right-skewed distribution**, meaning most days have moderate visitor counts, but a few days have extremely high visitor counts. This might be driven by factors like promotions, special events, or holidays.

Technical Terms

Technical Terms:

- **Seasonality**: In time series data, seasonality refers to regular, repeating patterns that occur at regular intervals (e.g., daily, weekly, or annually). In this case, the website might receive more traffic during certain times of the year.
- **Right-skewed distribution**: This is a type of data distribution where the tail on the right side (higher values) is longer or fatter than the left. In other words, there are fewer large values, but they have a significant impact.

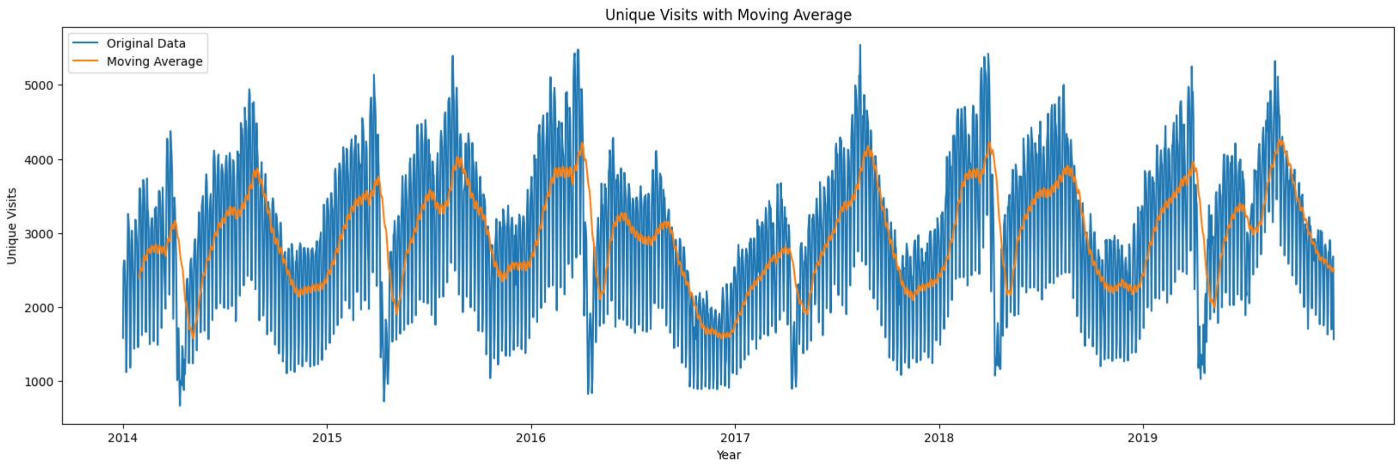
4) Moving Average Smoothing

Technical Terms:

- **Moving Average (MA)**: This is a technique used to smooth out short-term fluctuations and highlight longer-term trends in a time series. A moving average is computed by averaging a certain number of consecutive data points (here, 30 days).
- **Window size**: This refers to the number of data points used to compute each average in the moving average. For example, a window size of 30 means that the average is computed over every 30 consecutive days.
- The moving average reveals a clear **underlying trend** in website traffic. It reduces noise (daily variability) but also removes some important details, such as day-to-day fluctuations. While it highlights long-term patterns, it is not suitable for accurately predicting individual day values.

```
import pandas as pd
import matplotlib.pyplot as plt
# Calculate the moving average
window_size = 30  # Choose an appropriate window size
data['Moving Average'] = data['Unique.Visits'].rolling(window=window_size).mean()

# Plot the original time series data along with the smoothed data
plt.figure(figsize=(20, 6))  # Increase the breadth of the plot
plt.plot(data['Unique.Visits'], label='Original Data')
plt.plot(data['Moving Average'], label='Moving Average')
plt.title('Unique Visits with Moving Average')
plt.xlabel('Year')
plt.ylabel('Unique Visits')
plt.xticks(data.index[::365], pd.to_datetime(data.index[::365]).year)
plt.legend()
plt.show()
```

Output



The 30-day **moving average** smooths the daily fluctuations, revealing the general trend of visitor counts over time. The peaks and troughs that were visible in the raw data are less prominent, and a smoother trendline is now visible.

Moving averages are useful for detecting overall trends, but they do not capture short-term changes well. For forecasting purposes, a more sophisticated model that can incorporate both long-term and short-term patterns will be needed.

5) Moving Average Model Evaluation

```
# Calculate RMSE
rmse = math.sqrt(mean_squared_error(data_for_evaluation['Unique.Visits'],
data_for_evaluation['Moving Average']))

# Calculate MAE
mae = mean_absolute_error(data_for_evaluation['Unique.Visits'],
data_for_evaluation['Moving Average'])

# Calculate MSE
mse = mean_squared_error(data_for_evaluation['Unique.Visits'],
data_for_evaluation['Moving Average'])

print("RMSE:", rmse)
print("MAE:", mae)
print("MSE:", mse)
```

Output

```
RMSE: 800.4623435190808
MAE: 658.0654038041785
MSE: 640739.963392059
```

These error metrics indicate that while the moving average captures general trends, it doesn't accurately predict daily visitor counts. The high RMSE and MAE values indicate that the predictions deviate significantly from the actual values, likely because moving averages over smooth the data and fail to account for short-term fluctuations.

Technical Terms:

- **RMSE (Root Mean Squared Error)**: This measures the average magnitude of the error between predicted and actual values. The lower the RMSE, the better the model is at predicting actual values. RMSE penalizes large errors more than smaller ones due to the squaring of errors.
- **MAE (Mean Absolute Error)**: This measures the average of the absolute differences between predicted and actual values. Unlike RMSE, it treats all errors equally and doesn't penalize large errors more than small ones.
- **MSE (Mean Squared Error)**: Similar to RMSE, but it's the average of the squared differences between predicted and actual values. MSE also penalizes large errors more heavily.

6) Exponential Smoothing

Technical Terms:

- **Exponential Smoothing:** This is a technique that smooths data by applying **decreasing weights to older observations**. The most recent observations are given higher weight, which makes the model more responsive to recent changes.
- **Alpha (α)**: This is the smoothing parameter in exponential smoothing that controls **how much weight is given to recent observations**. A higher alpha means that more weight is given to recent observations, making the model more responsive to recent changes
- **Exponential smoothing** is a more versatile method than the moving average because it can adapt to both long-term trends and short-term fluctuations, depending on the value of alpha. It offers better potential for **short-term forecasting**, but the choice of alpha is crucial

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import SimpleExpSmoothing

# Apply exponential smoothing with different smoothing parameters
alpha_values = [0.1, 0.3, 0.5, 0.7]  # Example smoothing parameters

for alpha in alpha_values:
    # Fit the model
    model = SimpleExpSmoothing(data['Unique.Visits']).fit(smoothing_level=alpha)

    # Get the smoothed values
    data['Exponential Smoothing_' + str(alpha)] = model.fittedvalues

    # Plot the original data, moving average, and exponential smoothing
    plt.figure(figsize=(20, 6))
    plt.plot(data['Unique.Visits'], label='Original Data')
    plt.plot(data['Moving Average'], label='Moving Average')
    plt.plot(data['Exponential Smoothing_' + str(alpha)], label='Exponential
Smoothing (alpha = ' + str(alpha) + ')')
    plt.title('Unique Visits with Exponential Smoothing (alpha = ' + str(alpha) +
')')
    plt.xlabel('Year')
    plt.ylabel('Unique Visits')
    plt.xticks(data.index[::365], pd.to_datetime(data.index[::365]).year)
    plt.legend()
    plt.show()
```

Code Explanation

- **from statsmodels.tsa.holtwinters import SimpleExpSmoothing**:

Imports **Simple Exponential Smoothing** from **statsmodels**, a method for smoothing time series data by applying a weighted average, where more weight is given to recent observations.

- **alpha_values = [0.1, 0.3, 0.5, 0.7]**:

Creates a list of **smoothing parameters** (alpha), which control the degree of smoothing. Lower alpha values smooth more, while higher values react more to recent changes.

- **for alpha in alpha_values:**:

Starts a loop to apply exponential smoothing with each value of alpha (0.1, 0.3, 0.5, 0.7).

- **model = SimpleExpSmoothing(data['Unique.Visits']).fit(smoothing_level=alpha)**:

Applies **Simple Exponential Smoothing** to the 'Unique.Visits' data. The model is fitted using the alpha value from the loop to control how much weight recent data has.

- **data['Exponential Smoothing_' + str(alpha)] = model.fittedvalues**:

Stores the **smoothed values** from the model in a new column of the data DataFrame. The column name indicates which alpha value was used (e.g., 'Exponential Smoothing_0.1').

- **plt.figure(figsize=(20, 6))**:

Creates a new figure with specific dimensions (20 inches wide by 6 inches high) to plot the data.

- **plt.plot(data['Unique.Visits'], label='Original Data')**:

Plots the original **time series data** (website visitors) with a label 'Original Data'.

- **plt.plot(data['Moving Average'], label='Moving Average')**:

Plots the **moving average** of the original data for comparison with exponential smoothing.

- **plt.plot(data['Exponential Smoothing_' + str(alpha)], label='Exponential Smoothing (alpha = ' + str(alpha) + ')')**:

  Plots the **exponential smoothing** line corresponding to the current alpha value in the loop, allowing for comparison with the original data and moving average.

- **plt.title('Unique Visits with Exponential Smoothing (alpha = ' + str(alpha) + ')')**:

Sets the **plot title**, specifying the alpha value used for smoothing in the current plot.

- **plt.xlabel('Year')**:

Labels the x-axis as **'Year'**, representing the time period for the data.

- **plt.ylabel('Unique Visits')**:

Labels the y-axis as **'Unique Visits'**, showing the number of website visitors.

- **plt.xticks(data.index[::365], pd.to_datetime(data.index[::365]).year)**:

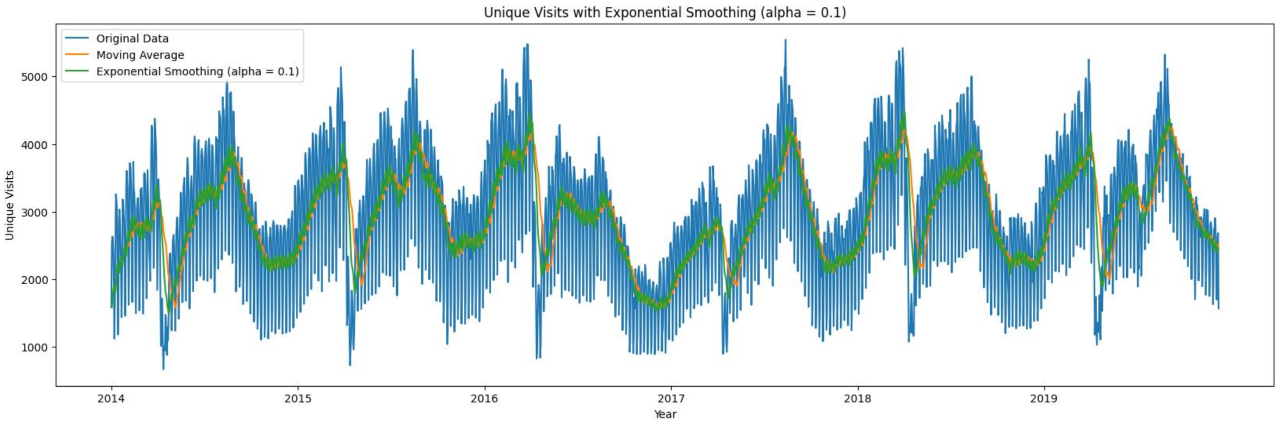
Adjusts the **x-axis ticks** to display **years** instead of daily data, making the plot less cluttered.

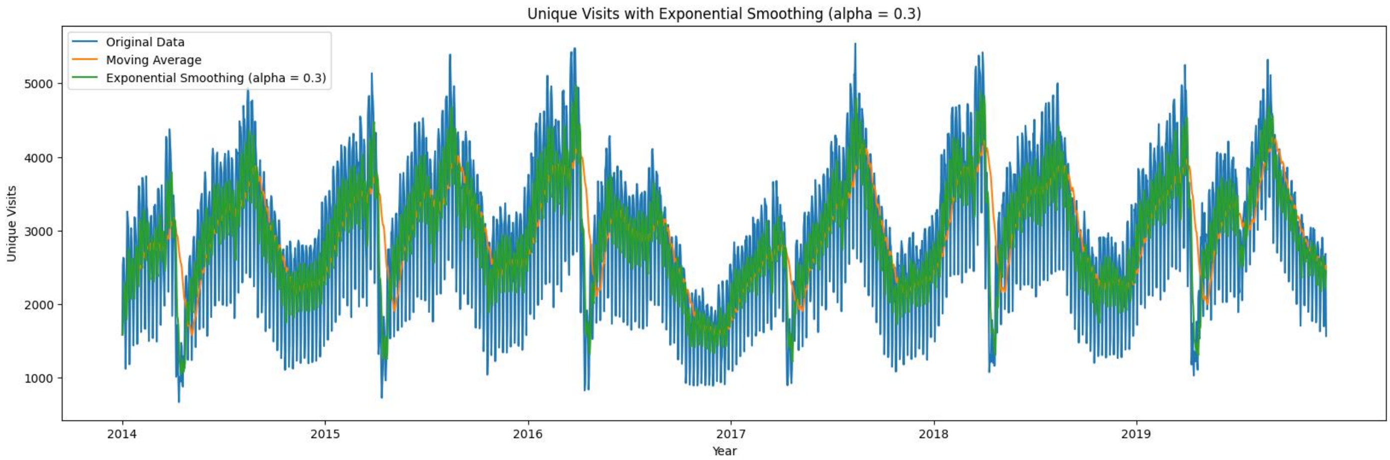
- **plt.legend()**:

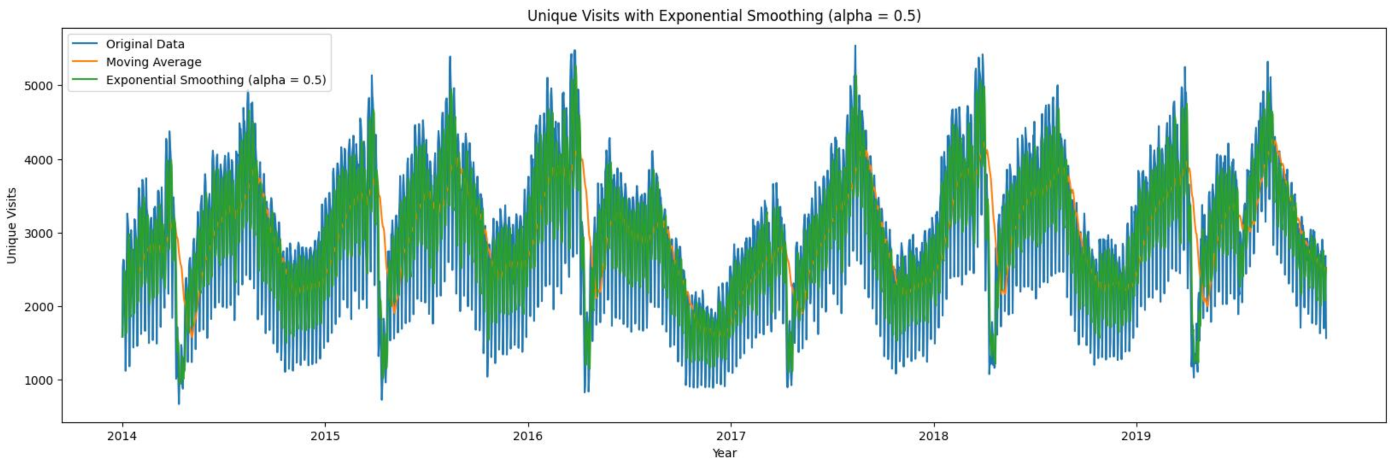
Adds a **legend** to the plot, distinguishing between the original data, moving average, and different exponential smoothing lines.

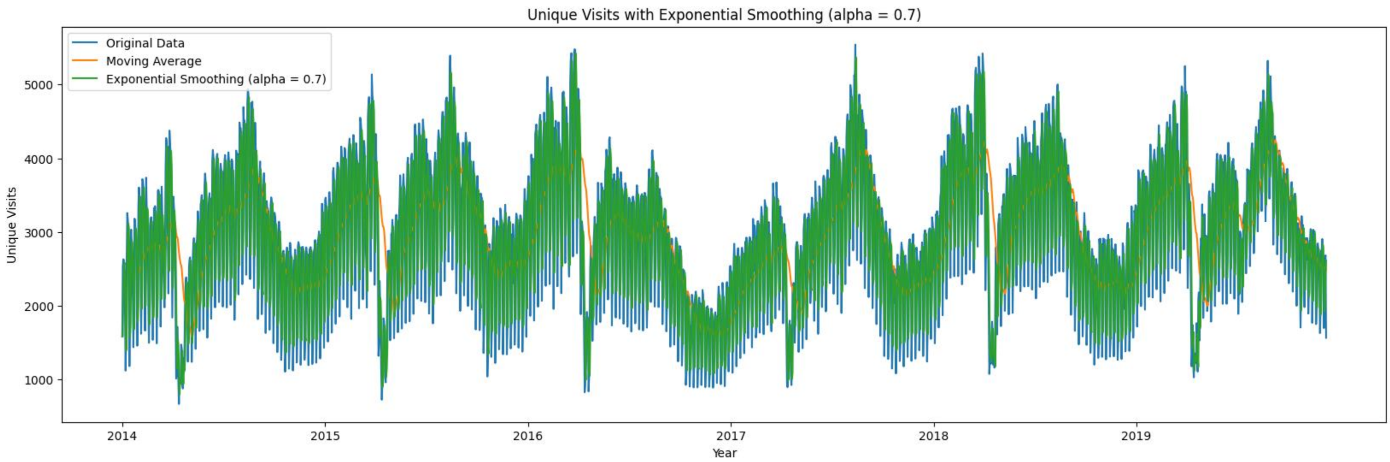
- **plt.show()**:

Displays the plot with the original data, moving average, and smoothed data for the current alpha value


Unique Visits with Exponential Smoothing (alpha = 0.1)
Original Data
Moving Average
Exponential Smoothing (alpha = 0.1)
Unique Visits
Year


Unique Visits with Exponential Smoothing (alpha = 0.3)
Original Data
Moving Average
Exponential Smoothing (alpha = 0.3)
Unique Visits
Year


Unique Visits with Exponential Smoothing (alpha = 0.5)
Original Data
Moving Average
Exponential Smoothing (alpha = 0.5)
Unique Visits
Year


Unique Visits with Exponential Smoothing (alpha = 0.7)
Original Data
Moving Average
Exponential Smoothing (alpha = 0.7)
Unique Visits
Year

Inference:

- Higher alpha values (0.7) produce a curve that follows the actual data more closely, but can overreact to short-term fluctuations. Lower alpha values (0.1) smooth the data too much, causing the model to miss significant short-term changes.
- Exponential smoothing with a mid-range alpha (e.g., 0.3) provides a balance between capturing long-term trends and responding to short-term changes, making it more suitable for **forecasting** than a simple moving average.

Why Alpha=0.3 chosen?

Choosing **alpha = 0.3** for **exponential smoothing** strikes a balance between responsiveness to recent data and overall trend smoothing. Here's why **alpha = 0.3** might be better compared to other values:

Balanced Sensitivity:

- **Alpha = 0.1** is too low, causing the model to overly smooth the data, which results in **slower responses to recent changes** and misses short-term fluctuations.
- **Alpha = 0.7** is too high, making the model **overly responsive to short-term noise** and potentially overfitting to random variations in the data.**Alpha = 0.3** provides a **middle ground**, capturing recent trends without overreacting to noise.

Reduced Overfitting:

Higher alpha values like **0.5 or 0.7** tend to capture too much noise from short-term fluctuations, which can lead to **overfitting**. **Alpha = 0.3** smooths these fluctuations while still giving enough weight to recent data.

Good Forecasting:

Alpha = 0.3 maintains enough **historical trend** while being responsive to **new data**, making it effective for time series with moderate fluctuations, like your daily website visitor data.

7) Model Evaluation of Expo Smoothening (alpha = 0.3)

```
# Extract the actual and predicted values
actual_values = data['Unique.Visits']
predicted_values = data['Exponential Smoothing_0.3']

# Calculate RMSE
rmse = math.sqrt(mean_squared_error(actual_values, predicted_values))

# Calculate MSE
mse = mean_squared_error(actual_values, predicted_values)

# Calculate MAE
mae = mean_absolute_error(actual_values, predicted_values)

print("RMSE:", rmse)
print("MSE:", mse)
print("MAE:", mae)
```

These metrics show that **exponential smoothing** (especially with an alpha of 0.3) performs slightly better than the moving average model. The RMSE and MAE values are lower than for the moving average, indicating better prediction accuracy.

8) **Augmented Dickey-Fuller (ADF) Test** – For stationary data testing

Technical Terms:

- **ADF Test (Augmented Dickey-Fuller Test)**: This is a statistical test used to determine whether a time series is **stationary** (i.e., its statistical properties, such as mean and variance, don't change over time). A stationary series is essential for many time-series models like ARIMA to perform well.
- **Stationarity**: A time series is stationary if its statistical properties (mean, variance, etc.) do not change over time. A stationary series is easier to model and predict compared to non-stationary series.

```
# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/PA Internal Test/daily_website_visitor_DS7_final.csv', index_col='Date', parse_dates=True)
# Perform the Augmented Dickey-Fuller test
result = adfuller(data['Unique.Visits'])

# Print the test results
print('ADF Statistic:', result[0])
print('p-value:', result[1])
print('Critical Values:', result[4])

# Interpret the results
if result[1] <= 0.05:
    print("The time series is likely stationary.")
else:
    print("The time series is likely non-stationary.")
```

Output

```
ADF Statistic: -4.475968574445406
p-value: 0.00021726409300080015
Critical Values: {'1%': -3.4334094211542983, '5%': -2.8628915360971003, '10%': -2.5674894918770197}
The time series is likely stationary.
```

Inference:

- Since the **p-value** is less than 0.05, we reject the null hypothesis of non-stationarity. This means that the data is **stationary**, and no additional transformation (like differencing) is needed to stabilize the time series.

Why Models like ARIMA need stationary data???

ARIMA models require stationary data because the fundamental assumption behind ARIMA is that the statistical properties of the data (like mean, variance, and autocovariance) are constant over time. Here's why:

- Predictability: Stationarity ensures that the relationship between values is **consistent over time**, making it easier to model future values based on past data. If the data is non-stationary (i.e., trends, seasonality, or variance changes over time), ARIMA **struggles to find meaningful patterns.**
- Mathematical Simplicity: ARIMA models **rely on mathematical properties** like autocorrelations to predict future values. These properties are stable only in stationary data. Non-stationary data introduces changing patterns that ARIMA cannot handle effectively.
- Error Behavior: In non-stationary data, **residuals (errors) often exhibit trends or seasonality**, violating the assumption of constant variance. This makes it difficult for ARIMA to produce accurate predictions.
- Differencing to Make Data Stationary: If the data is non-stationary, differencing (I in ARIMA) is used to remove trends and make the series stationary. This ensures that the model works with stable, predictable data patterns.

What is Autocorrelation & Lag?

Autocorrelation is a statistical measure that assesses the correlation of a time series with its own past values (lags). It shows how current values are related to previous values over different time intervals.

Values close to 1 indicate a strong positive correlation, meaning high values are followed by high values.

Values close to -1 indicate a strong negative correlation, meaning high values are followed by low values.

Values near 0 suggest no significant correlation.

Lag refers to the time interval between observations in a time series. For example, if you're examining values from one day to the next, a lag of 1 means comparing today's value with yesterday's value.

In summary:

- **Autocorrelation** quantifies the relationship between a series and its past values.
- **Lag** is the time difference between those observations.

9) ARIMA & its working

ARIMA stands for **AutoRegressive Integrated Moving Average**. It is a popular statistical method used for time series forecasting. The model is characterized by three components: p, d, and q.

- **p**: The number of lag observations in the model (autoregressive part).
- **d**: The number of times that the raw observations are differenced (integrated part).
- **q**: The size of the moving average window.

1. Autoregressive (AR) Component:

- **What It Does**: The AR component looks at past values to help predict the current value.
- **How It Works**: Imagine you're trying to guess the score of a sports game. If the team has been scoring a lot in recent games, it's likely they'll score again. So, past performance influences your prediction for today's game.
- **Key Idea**: If the model shows that past scores (or data points) positively affect current scores, it suggests that there's a kind of "momentum" – meaning, what happened before is important for what's happening now.

2. Moving Average (MA) Component:

- **What It Does**: The MA component focuses on past mistakes in predictions (forecast errors) to improve future predictions.
- **How It Works**: Let's say you predicted that a team would score 3 goals, but they actually scored 5. That's a mistake (an error). In your next prediction, you might consider that error. If you were off by +2 goals last time, you might adjust your next guess upward to account for that mistake.
- **Key Idea**: If the model incorporates past errors, it suggests that those mistakes matter. By learning from what went wrong, the model can make better predictions in the future.

3. Integrated (I) Component:

What It Does:

- The Integrated part of ARIMA helps make a time series data stationary, which means its statistical properties (like the mean and variance) do not change over time.

Why It Matters:

- Many forecasting models, including ARIMA, work best with stationary data because they can't reliably predict future values if the data has trends or seasonal patterns that change over time.

How It Works:

1. **Differencing**: The main technique used in the Integrated part is called differencing. This involves subtracting the previous observation from the current observation to eliminate trends

10) What are ACF & PACF plots

ACF (AutoCorrelation Function) Plot

- **What It Is**: Imagine you have a series of numbers (like daily temperatures). The ACF plot helps you see how today's temperature is related to yesterday's, the day before yesterday's, and so on.
- **Layman's Terms**: It shows you how much today's value depends on its past values.

PACF (Partial AutoCorrelation Function) Plot

- **What It Is**: The PACF plot also looks at how today's temperature relates to past temperatures, but it removes the influence of the days in between.
- **Layman's Terms**: It shows you how much today's value depends on just the immediate past value, ignoring the effects of all the earlier days.

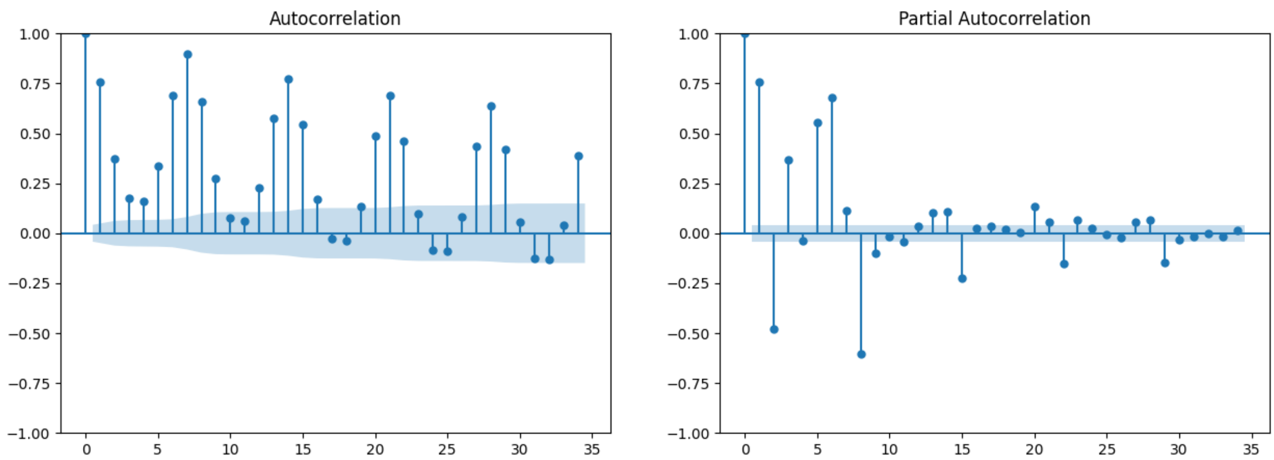
Summary

- **ACF**: How today relates to all past values.
- **PACF**: How today relates to just the last value, ignoring the others.

11) ACF & PACF plots

```
#  acf and pacf plots

import matplotlib.pyplot as plt
# ACF and PACF plots
fig, axes = plt.subplots(1, 2, figsize=(15, 5))
plot_acf(data['Unique.Visits'], ax=axes[0])
plot_pacf(data['Unique.Visits'], ax=axes[1])
plt.show()
```

Output



Arima has p,d,q components. We get the p value from Partial autocorrelation graph, q value from autocorrelation graph & d by how many time you difference your data to make it stationary

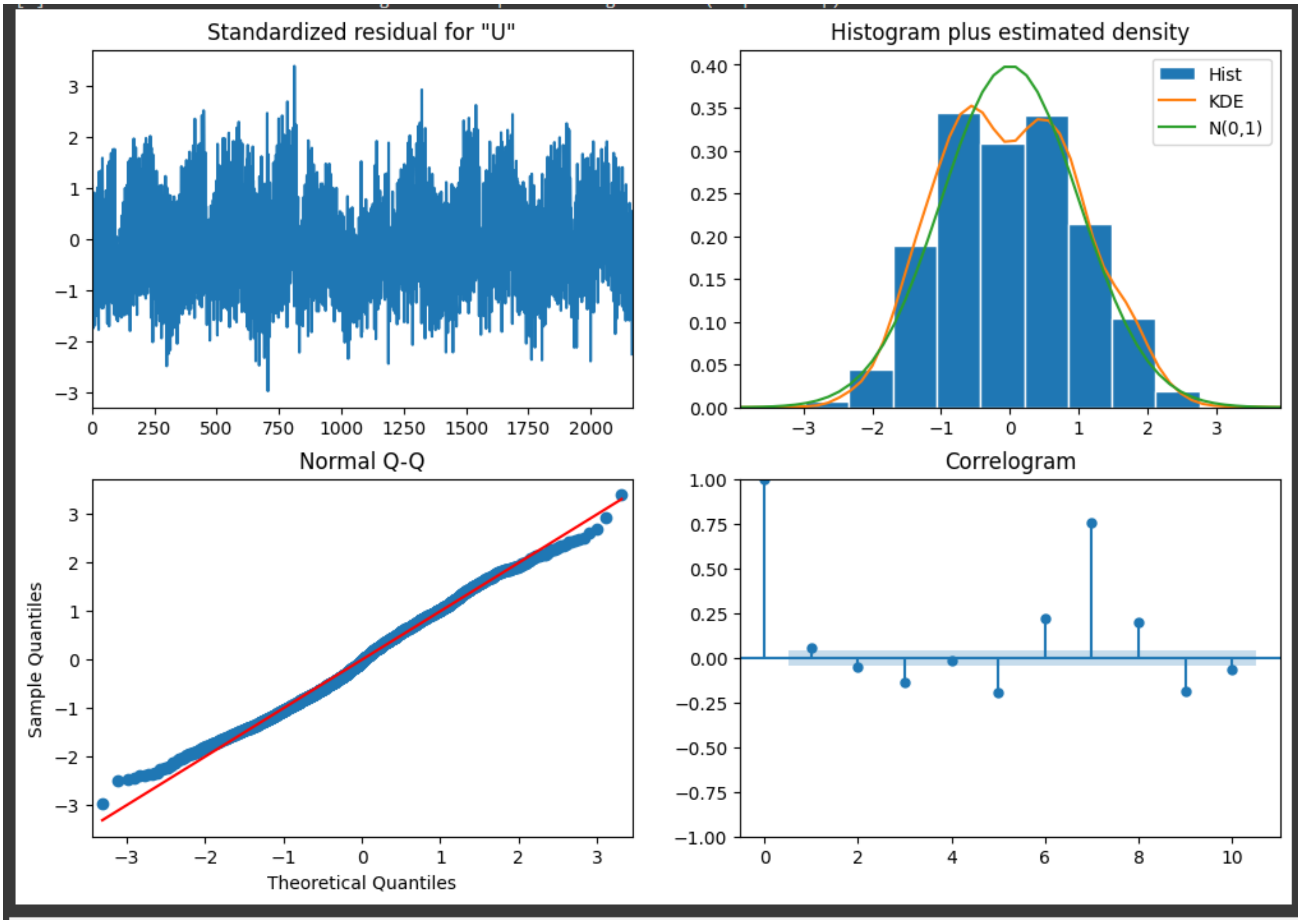
How to calculate:
P – Number of times the spike touches the value **1.00 in PACF**
(it also means post how many spikes the graph starts to lag so here after 1 spike graph starts to lag)
Q - Same in ACF plot
D – Number of times you differentiate data to make it stationary in our case 0
So our ARIMA components are (p,d,q) – (1,0,1)

12) Build & Fit Arima Model

```
# Define the parameters based on analysis
p = 1  # Based on PACF plot
d = 0  # Based on differencing (already differentiated)
q = 1  # Based on ACF plot

# Build and fit the ARIMA model
model = ARIMA(data['Unique.Visits'], order=(p, d, q))
arima_result = model.fit()

# Display the summary of the model
print(arima_result.summary())

# Diagnostics plots
arima_result.plot_diagnostics(figsize=(12, 8))
plt.show()
```

Output

```
                               SARIMAX Results
==============================================================================
Dep. Variable:           Unique.Visits   No. Observations:                 2167
Model:                  ARIMA(1, 0, 1)   Log Likelihood              -16613.783
Date:                 Tue, 24 Sep 2024   AIC                          33235.565
Time:                         17:09:29   BIC                          33258.289
Sample:                              0   HQIC                         33243.875
                                - 2167
Covariance Type:                   opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       2943.6465     48.255     61.002      0.000    2849.069    3038.224
ar.L1          0.5728      0.023     24.964      0.000       0.528       0.618
ma.L1          0.7467      0.017     43.423      0.000       0.713       0.780
sigma2      2.682e+05      1e+04     26.723      0.000    2.49e+05    2.88e+05
===================================================================================
Ljung-Box (L1) (Q):                   7.33   Jarque-Bera (JB):                32.89
Prob(Q):                              0.01   Prob(JB):                         0.00
Heteroskedasticity (H):               0.90   Skew:                             0.11
Prob(H) (two-sided):                  0.17   Kurtosis:                         2.44
===================================================================================
```



Explanation

- A p-value < 0.05 is typically considered statistically significant. In this case, all p-values are 0.000, indicating strong significance.
- AR (AutoRegressive) term: Represents the dependency between an observation and a certain number of lagged observations. AR(1) of **0.5728** suggests that each data point is **moderately positively correlated with its immediate predecessor.**
- MA (Moving Average) term: Represents the dependency between an observation and a residual error from a moving average model applied to lagged observations. MA(1) of 0.7467 indicates a **strong influence** of the **previous period's prediction error on the current value.**
- AIC (Akaike Information Criterion): Purpose: AIC **estimates the quality of each model** relative to each of the other models. It rewards goodness of fit but also **includes a penalty** for the number of parameters used in the model.
- BIC (Bayesian Information Criterion): Purpose: Similar to AIC, BIC also evaluates models but with a **stronger penalty for the number of parameters,** especially as the sample size increases
- Lower AIC and BIC values are preferred. We also ran a model with components (2,0,2) its AIC and BIC values were high than (1,0,1) proving this to be a better fit

Detailed Inference of our output

The significance of all terms suggests that each component of the model contributes meaningfully to explaining the variation in Unique Visits. This increases confidence in the model's structure and its ability to capture important patterns in the data.

1. Interpretation of AR and MA Terms:
   - The AR(1) term of 0.5728 indicates moderate positive autocorrelation. This means that about **57.28%** of the **previous period's deviation from the mean carries over to the current period.** It suggests a consistent, moderately strong trend in the data.
   - The MA(1) term of 0.7467 is relatively high, indicating that about **74.67%** of the **previous period's forecast error is incorporated into the current period's** forecast. This suggests the model efficiently adjusts for **recent prediction errors**, enhancing its **short-term forecasting accuracy**.
2. Residual Analysis:
   - The Q-Q plot and histogram of residuals approximate a **normal distribution**. This near-normality of residuals is crucial because it supports the **validity of various statistical tests** and confidence intervals derived from the model.
3. Homoscedasticity:

   - The heteroskedasticity test yielded a p-value of **0.17**, which is **greater** than the typical significance level of 0.05. This suggests that the variance of the residuals is relatively constant across different levels of the predicted values (homoscedasticity). **Basically data has homoscedasticity**.
   - Homoscedasticity is an important assumption in time series analysis. When met, it indicates that the **model's predictive power is consistent across** the range of predictions, enhancing the reliability of forecasts and statistical inferences drawn from the model.

These strengths collectively suggest that your ARIMA(1,0,1) model provides a solid foundation for understanding and forecasting the Unique Visits time series. It captures significant short-term dependencies, adjusts well to recent forecast errors, and largely meets key statistical assumptions. This makes it a valuable tool for both interpreting past trends and generating short-term forecasts of Unique Visits.

13) MAE,MSE,RMSE values of ARIMA model

```
# Calculate MSE, MAE, and RMSE
mse = mean_squared_error(data['Unique.Visits'], predictions)
mae = mean_absolute_error(data['Unique.Visits'], predictions)
rmse = np.sqrt(mse)

print(f"MSE: {mse}")
print(f"MAE: {mae}")
print(f"RMSE: {rmse}")
```

Output

```
MSE: 267604.4310135119
MAE: 430.95719857956766
RMSE: 517.3049690593663
```

14) Comparison

Model	Moving Average (30)	Exponential Smoothening (0.3)	ARIMA Model (1,0,1)
RMSE values	800.46	772.99	**517.30**

RMSE values less for ARIMA hence reinforcing it to be a better model as compared to other models

Why RMSE is Better than MAE and MSE??

1. **Sensitivity to Outliers:**
   - **RMSE (Root Mean Square Error)**: Squares the errors before averaging, which means it gives more weight to larger errors. This sensitivity can be useful if you want to penalize significant errors more heavily.
   - **MAE (Mean Absolute Error)**: Treats all errors equally, which can sometimes overlook the impact of larger errors. It's more robust to outliers but may not capture the true impact of significant deviations.
   - **MSE (Mean Squared Error)**: Like RMSE, it squares the errors, but since it doesn't take the square root, its scale is not as interpretable as RMSE. MSE can be inflated by large errors, which is useful for certain analyses but may not always be practical.
2. **Interpretability:**
   - RMSE is in the same units as the original data, making it more interpretable than MSE. For example, if you're measuring visits to a website, RMSE tells you the average error in terms of actual visits.

Conclusions:

1. **Performance Comparison**: The RMSE values indicate that the ARIMA model (517) has the lowest error compared to both the Moving Average (800) and Exponential Smoothing (773). This suggests that ARIMA provides the most accurate forecasts among the three methods for this dataset.
2. **Model Suitability**: Since ARIMA effectively captures both the trend and seasonality in the data, as well as making adjustments for past errors, it performs better in this context. The significantly lower RMSE implies that ARIMA is better at minimizing prediction errors.

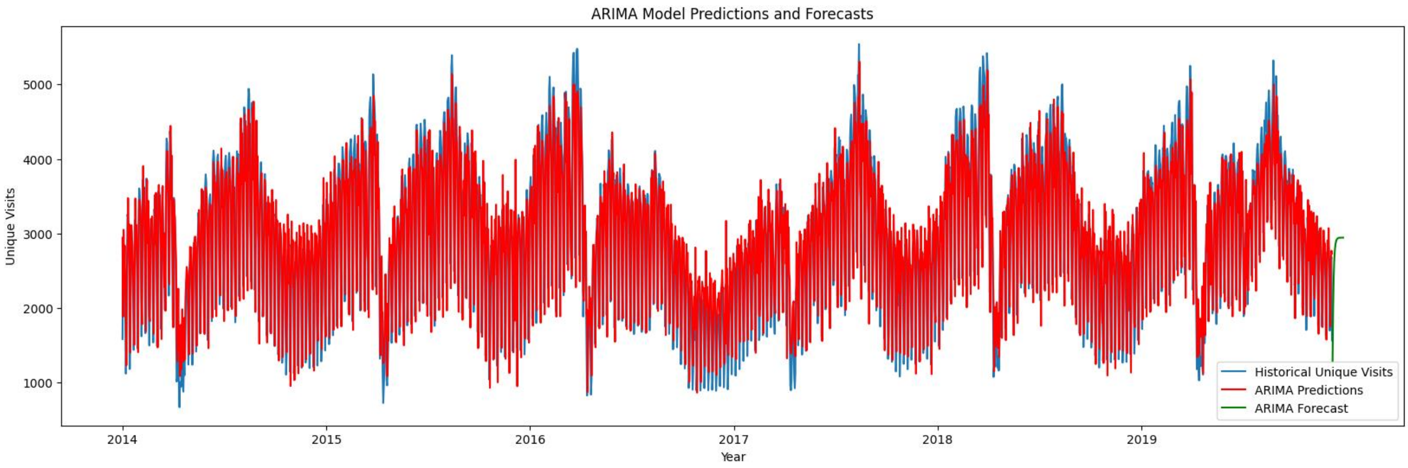
15) Forecasting values from best fit model (ARIMA)

```
# Forecast future values
forecast = arima_result.forecast(steps=20)
print(forecast)   # Print the forecasted values
```

Output

```
2167    1282.445018
2168    1992.141512
2169    2398.642326
2170    2631.478352
2171    2764.842452
2172    2841.230902
2173    2884.984771
2174    2910.046164
2175    2924.400860
2176    2932.622959
2177    2937.332423
2178    2940.029916
2179    2941.574989
2180    2942.459978
2181    2942.966883
2182    2943.257228
2183    2943.423533
2184    2943.518789
2185    2943.573350
2186    2943.604601
```

```
import pandas as pd
import matplotlib.pyplot as plt
plt.figure(figsize=(20, 6))
plt.plot(data['Unique.Visits'], label='Historical Unique Visits')
plt.plot(predictions, label='ARIMA Predictions', color='red')
plt.plot(forecast.index, forecast, label='ARIMA Forecast', color='green')

plt.title('ARIMA Model Predictions and Forecasts')
plt.xlabel('Year')
plt.ylabel('Unique Visits')

# Show year labels every 365 days, consider extending the x-axis based on the
forecast period.
plt.xticks(data.index[::365], pd.to_datetime(data.index[::365]).year)
plt.legend()
plt.show()
```



Key observations:

1. Seasonal pattern: There's a clear recurring pattern, suggesting strong seasonality in the data.
2. Trend: The overall trend appears relatively stable, with some fluctuations over the years.
3. Forecasting: The model provides both predictions (red) that closely follow the historical data (blue) and forecasts (green) for future periods.
4. Volatility: The data shows considerable short-term variability within each seasonal cycle.
5. Model fit: The ARIMA predictions seem to capture the general pattern of the historical data well, indicating a good model fit.
6. Forecast uncertainty: The future forecast (green line) shows less volatility than the historical data, which is typical for ARIMA forecasts as they tend towards the mean over time.

This ARIMA model appears to be effectively capturing both the seasonal patterns and overall trends in the data, providing a basis for short-term forecasting of unique visits.