DATA ANALYTICS WITH COGNOS

WEBSITE TRAFFIC ANALYSIS – PHASE II

PRIMARY OBJECTIVES:

In phase II of Website Traffic Analysis, the primary focus is to predict future web traffic trends based on historic data from the collected dataset and evaluation of parameters such as bounce rate of the website.

ALGORITHM:

To achieve the above stated objective, we introduce an efficient machine learning algorithm called ARIMA, which is one of the commonly used methods in time series analysis and prediction.

ARIMA:

ARIMA, which stands for Auto Regressive Integrated Moving Average, is a popular time series forecasting algorithm used in machine learning and statistics. It is designed to model and predict time series data by capturing and accounting for the temporal dependencies and patterns within the data.

1. Auto Regressive (AR) Component:

- The "AR" component represents the autoregressive part of the model. It captures the relationship between the current observation and previous observations in the time series. In other words, it models the dependency of the current value on its own past values. The order of the autoregressive component is denoted as "p," and it signifies the number of past observations to consider.

2. Integrated (I) Component:

- The "I" component represents differencing, which is the process of making a non-stationary time series stationary. Stationarity is a crucial assumption for many time series models, including ARIMA. The "I" component indicates how many differences are needed to achieve stationarity. If the data is already stationary, "I" is set to 0.

3. Moving Average (MA) Component:

- The "MA" component represents the moving average part of the model. It models the relationship between the current observation and past white noise (random) error terms in the time series. The order of the moving average component is denoted as "q," and it specifies the number of past error terms to consider.

The order of the ARIMA model is typically denoted as (p, d, q), where "p" is the order of the autoregressive component, "d" is the order of differencing, and "q" is the order of the moving average component. The ARIMA model is trained on historical time series data.

Once the ARIMA model is trained, it can be used to make future predictions. The model utilizes the past observations and its own parameters to forecast future values in the time series.

WHY ARIMA?

ARIMA (Auto Regressive Integrated Moving Average) is well-suited for time series analysis for several reasons:

- Modelling Temporal Dependencies: ARIMA models capture temporal dependencies in time series data. They account for how the current value is related to past values, making them effective at modeling and forecasting sequences with patterns that repeat over time.

- ARIMA models provide interpretable coefficients for each component (AR, I, MA), which can help in understanding the temporal relationships within the data. This interpretability is valuable for making business decisions or gaining insights from the data.

ARIMA models are relatively simple and lightweight, making them computationally efficient and easy to implement. They don't require extensive computational resources compared to more complex models like deep neural networks. Popular libraries like `statsmodels` in Python and the "forecast" package in R provide tools for implementing ARIMA models.

APPLYING ARIMA TO WEB TRAFFIC PREDICTION:

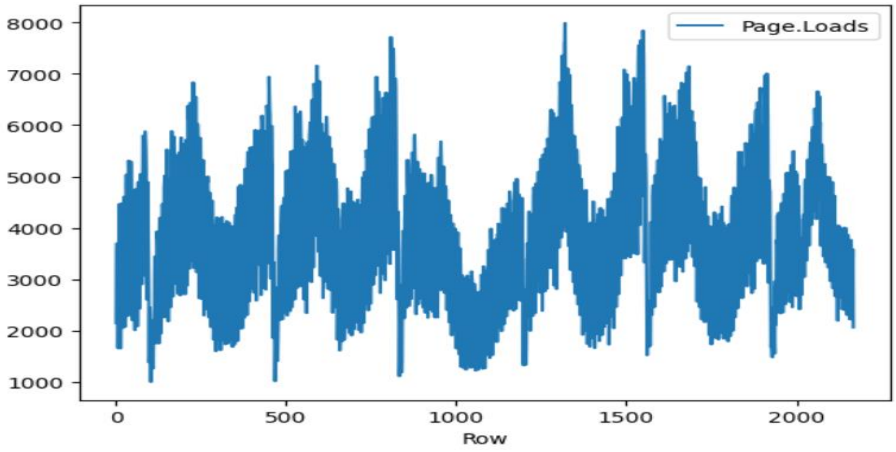
- **Data Collection**

  The primary step involves cleaning and preprocessing the collected data which includes page views, unique visitors, or other relevant metrics, ensuring that it's in a suitable format for time series analysis. This involves handling missing values and other discrepancies such as datatype mismatch etc…

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2167 entries, 0 to 2166
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Row                2167 non-null   int64
 1   Day.Of.Week        2167 non-null   int64
 2   Page.Loads         2167 non-null   int64
 3   Unique.Visits      2167 non-null   int64
 4   First.Time.Visits  2167 non-null   int64
 5   Returning.Visits   2167 non-null   int64
dtypes: int64(6)
memory usage: 101.7 KB
```

- **Time Series Exploration**

  The next step is to visualize the website traffic data to identify trends, seasonality, and any other patterns. This initial exploration helps in understanding the data.

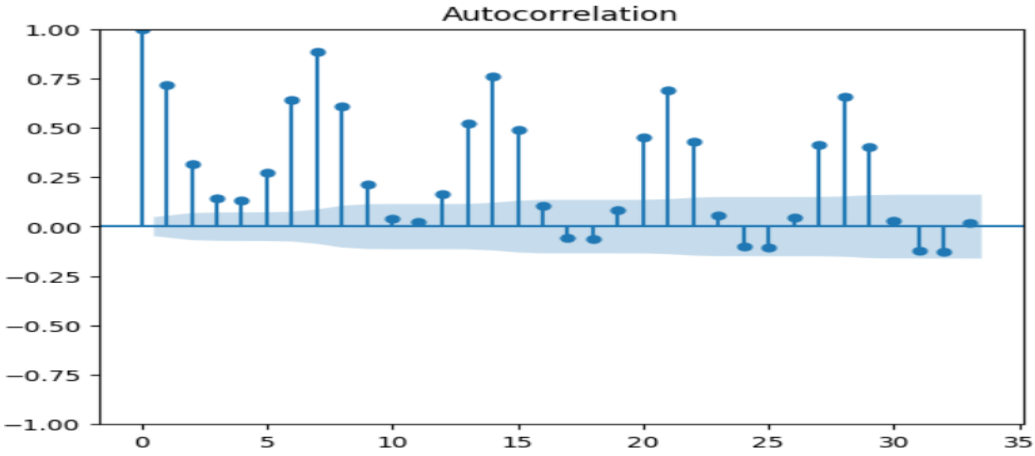


- **Stationarity Check**

In time series analysis, ACF (Auto Correlation Function) and PACF (Partial Auto Correlation Function) are two important tools used to understand the autocorrelation or temporal dependence within a time series.

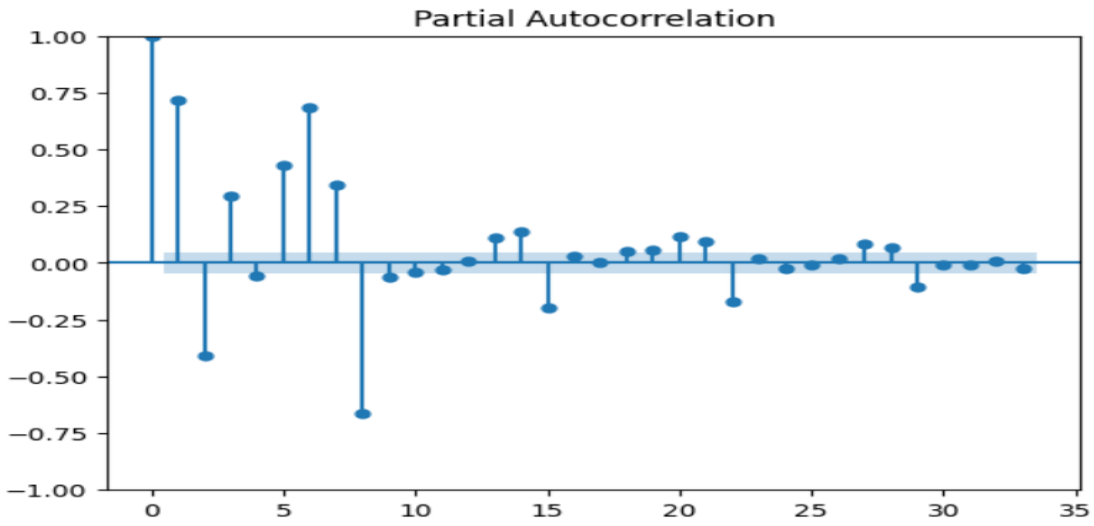
1. **Auto Correlation Function (ACF)**

The ACF measures the linear relationship between a time series and its lagged values. It computes the correlation coefficient between the time series at a given time point and the same time series at previous time points (lags). The ACF plot shows how the current value of a time series is related to its past values at different lags. A strong positive or negative correlation at a specific lag suggests that past observations at that lag have an impact on the current observation.



2. Partial Auto Correlation Function (PACF)

The PACF measures the correlation between a time series and its lagged values while controlling for the influence of intervening lags. It helps identify the direct (partial) effect of a specific lag on the current value, excluding the effects of the intervening lags. The PACF plot is useful for identifying the order (p) of the autoregressive (AR) component in an ARIMA model. It indicates which lags have a direct influence on the current value.



Both ACF and PACF plots are commonly used in the process of selecting appropriate orders (p and q) for an ARIMA model, where:

- p represents the order of the autoregressive (AR) component (number of lags to consider in the past).
- q represents the order of the moving average (MA) component (number of past forecast errors to consider).

Next we use the Augmented Dickey-Fuller (ADF) test or other methods to check whether the time series is stationary.

The ADF test is primarily used to check for the presence of a unit root in a time series, which is indicative of non-stationarity. To determine whether to reject the null hypothesis, you compare the ADF statistic to critical values at a chosen significance level (alpha). Common significance levels are 0.05 and 0.01.

```
from statsmodels.tsa.stattools import adfuller
adf_test = adfuller(df_train[param])
print(f'p-value: {adf_test[1]}')
```

```
p-value: 0.00022288390389911994
```

If the data is not stationary, we apply differencing to make it stationary, otherwise we proceed with further steps. Here p-value is 0.0002 which is less than 0.05 (under the threshold), hence we continue without differencing.

- Model Selection

  The next step is to choose an appropriate ARIMA model order (p, d, q). This involves determining the number of autoregressive (AR) terms (p), differencing order (d), and moving average (MA) terms (q).

  Manual estimation of parameters:

  In our project, manually we have chosen the parameters to be p = 2, d = 1, q = 0 to train the ARIMA model using training data set. The entire data set is split into two sets, one containing 1667 entries in the training data set and another containing 500 entries in the test data set.

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(df_train[param], order=(2,1,0))
model_fit = model.fit()
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:              Page.Loads   No. Observations:                 1667
Model:                 ARIMA(2, 1, 0)   Log Likelihood                -17.660
Date:                Wed, 11 Oct 2023   AIC                             41.321
Time:                        21:31:22   BIC                             57.575
Sample:                             0   HQIC                            47.344
                                - 1667
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3110      0.022     14.157      0.000       0.268       0.354
ar.L2         -0.4765      0.025    -18.869      0.000      -0.526      -0.427
sigma2         0.0598      0.002     25.125      0.000       0.055       0.064
===================================================================================
Ljung-Box (L1) (Q):                   1.76   Jarque-Bera (JB):                46.37
Prob(Q):                              0.18   Prob(JB):                         0.00
Heteroskedasticity (H):               0.85   Skew:                            -0.34
Prob(H) (two-sided):                  0.05   Kurtosis:                         2.53
===================================================================================
```

Model selection is also done through analysis, experimentation, or automated tools like auto-ARIMA.

```
import pmdarima as pm
auto_arima = pm.auto_arima(df_train[param], stepwise=False, seasonal=False)
auto_arima
```

```
ARIMA(5,1,0)(0,0,0)[0] intercept
```

- Model Training
  The selected ARIMA model is then trained on a portion of the historical data. This allows the model to estimate its parameters based on past traffic patterns.

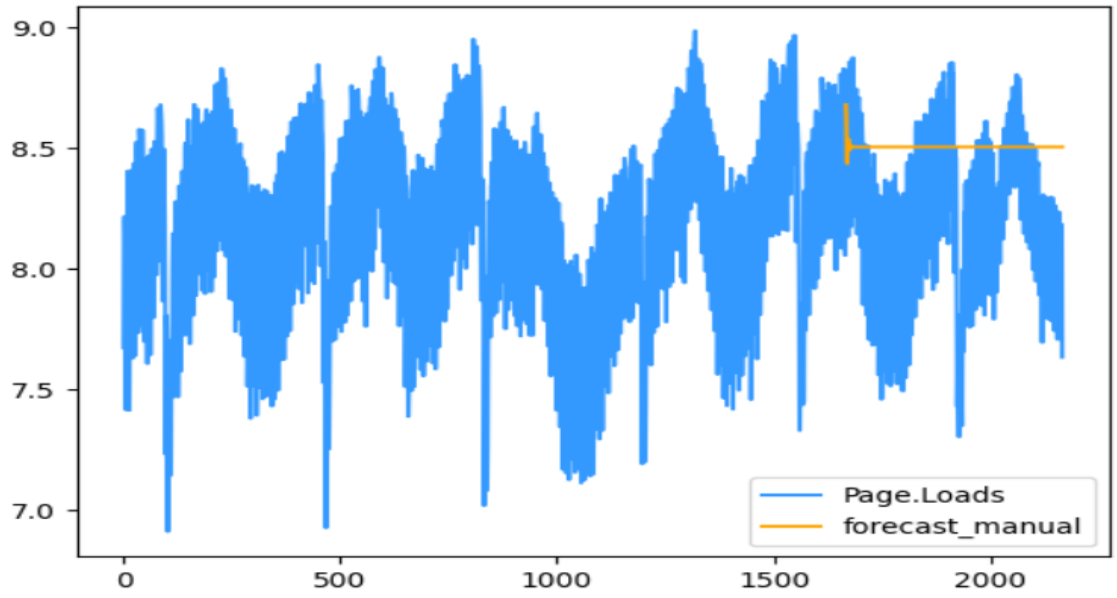
Dep. Variable:	y	**No. Observations:**	1667
Model:	SARIMAX(5, 1, 0)	**Log Likelihood**	791.511
Date:	Wed, 11 Oct 2023	**AIC**	-1569.021
Time:	22:06:08	**BIC**	-1531.094
Sample:	0	**HQIC**	-1554.966
	- 1667		
Covariance Type:	opg		

- Model Validation
  The performance of the ARIMA model is then validated over a holdout dataset.

- Forecasting
  The trained ARIMA model is employed to make traffic predictions for future periods. These predictions can be valuable for planning and resource allocation.

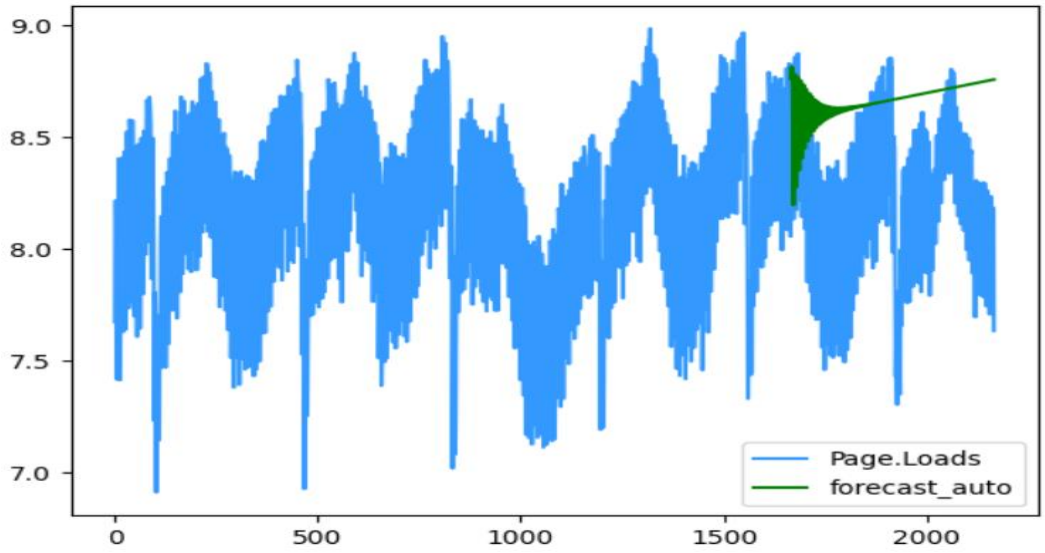
  Manual Parameter based prediction:

```
forecast_test = model_fit.forecast(len(df_test))

df['forecast_manual'] = [None]*len(df_train) + list(forecast_test)

df[[param,'forecast_manual']].plot(color=['#3399ff','orange'])
```



Auto – ARIMA based prediction:

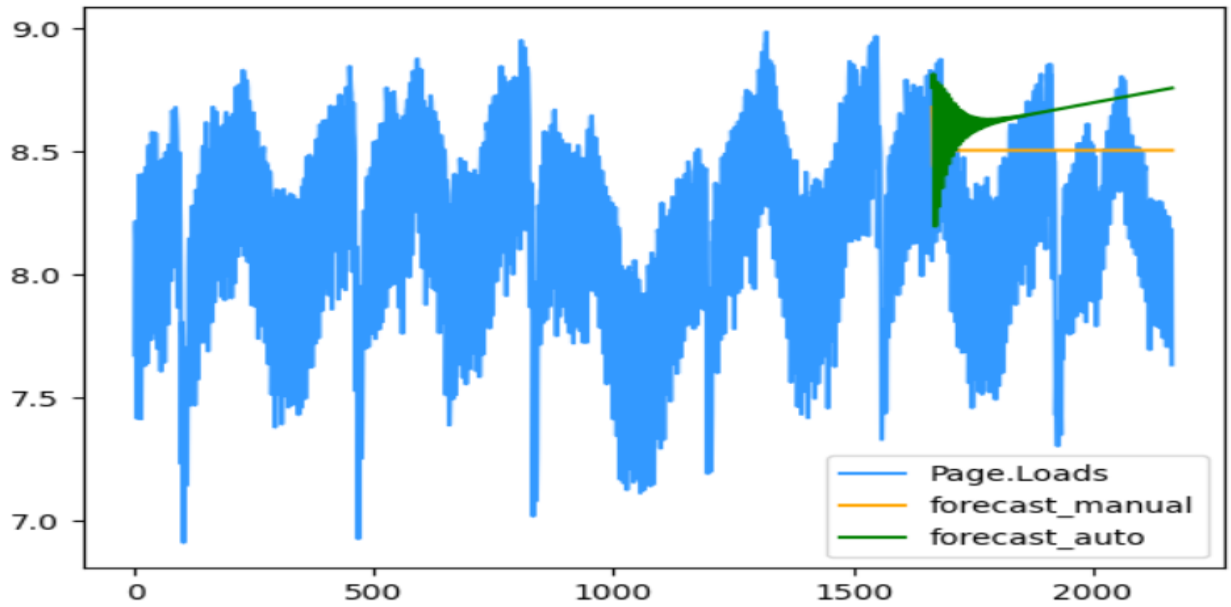
```
forecast_test_auto = auto_arima.predict(n_periods=len(df_test[param]))
df['forecast_auto'] = [None]*len(df_train[param]) + list(forecast_test_auto)

df[[param,'forecast_auto',]].plot(color=['#3399ff','green'])
```



ARIMA model parameters may need tuning and refinement, and different orders can be experimented with to achieve better forecasting accuracy.

MODEL EVALUATION:

The performance of the ARIMA model is assessed using various metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE).



Evaluation metrics:

- Mean Absolute Error (MAE): The average absolute difference between the actual and forecasted values.
- Mean Squared Error (MSE): The average of the squared differences between actual and forecasted values.
- Root Mean Squared Error (RMSE): The square root of the MSE, providing error in the same units as the data.
- Mean Absolute Percentage Error (MAPE): The average percentage difference between actual and forecasted values.

Manual method:

```
mae - manual: 0.29704959231892286
mape - manual: 0.03701199782139722
mse - manual: 0.15254232756192615
rmse - manual: 0.39056667492494307
```

Auto ARIMA method:

```
mae - auto: 0.4017289158308398
mape - auto: 0.04997490341769592
mse - auto: 0.24986339528970478
rmse - auto: 0.49986337662375785
```

The above evaluation metrics show that the values predicted using manually estimated p, d and q parameters are more closer to the original values in the test data set than those predicted using auto – ARIMA estimated parameters. The error values shown above show the deflection of the model's prediction from the original trend of the historical data.

COMPARISON OF ARIMA WITH OTHER MODELS:

A comparison of ARIMA with other ML models in terms of their efficiency for website traffic prediction:

ARIMA model	Other ML based prediction models
Well-suited for capturing linear temporal dependencies and seasonality in time series data.	Flexibility: ML models can capture a wide range of data patterns, including non-linear relationships and complex interactions
Simple and interpretable, making it easy to understand the model's parameters	Ability to handle large and high-dimensional data. Suitable for handling diverse types of data sources beyond time series data.
Often effective for data with straightforward trends and seasonality.	Capacity to incorporate external factors and feature engineering for more accurate predictions.

May not perform well on complex, non-linear, or irregular data patterns.	Complexity: Some ML models, particularly deep learning models, can be complex and computationally intensive, requiring significant data and resources.
Assumes that the data is stationary, which might not hold true for all website traffic data.	May require substantial preprocessing and feature engineering.
Limited ability to incorporate external factors or exogenous variables that can influence website traffic	May not provide as much interpretability as ARIMA, making it challenging to explain model decisions.

1.ARIMA vs. Exponential Smoothing:

Efficiency Comparison:

- **Complexity:** ARIMA models can be more complex due to the need for manual parameter tuning. Exponential smoothing is relatively simpler as it automatically assigns weights.
- **Handling Complexity:** ARIMA is better suited for time series data with complex patterns, such as multiple seasonalities and trends.
- **Ease of Use:** Exponential smoothing may be more user-friendly for beginners as it doesn't require the same level of parameter selection and tuning.
- **Performance on Simple Data:** Exponential smoothing may perform well on time series data with simple patterns, while ARIMA might be overkill.

BOUNCE RATE ANALYSIS:

The bounce rate is a crucial metric that measures the percentage of visitors who navigate away from a website after viewing only a single page or spending a very brief amount of time on that page. In essence, it quantifies how many users "bounce" off your website without engaging further by visiting additional pages or taking any action. Understanding and managing bounce rates is essential for optimizing website performance and user experience.

Bounce Rate is typically defined as the percentage of single-page visits (i.e., visitors who leave your site after viewing only one page) over the total number of visits to your website.

Bounce Rate = (Number of Single-Page Visits) / (Total Number of Visits)

```
total_page_loads = df['Page.Loads'].sum()
total_bounces = df['First.Time.Visits'].sum()

bounce_rate = (total_bounces / total_page_loads) * 100
print(f"Bounce Rate: {bounce_rate:.2f}%")
```

```
Bounce Rate: 93.56%
```

OBSERVATIONS MADE FROM BOUNCE RATES:

- A high bounce rate indicates that visitors are not finding what they expected or that the website's content or design needs improvement.
- It also suggests that visitors are not engaging with the user interface of the website.
- Relevance of Landing Page: If the landing page doesn't match the visitor's expectations or search intent, they are more likely to bounce.
- Website Speed: Slow-loading pages can lead to higher bounce rates as users become frustrated and abandon the site.
- User Experience: A cluttered, confusing, or unattractive website design can deter visitors from exploring further.
- Content Quality: Low-quality, irrelevant, or uninformative content can lead to quick exits.

CONCLUSION:

ARIMA is particularly effective in modelling time series data with temporal dependencies, seasonality, and trends. It is widely used in various domains, including finance, economics, stock market prediction, demand forecasting, and more.