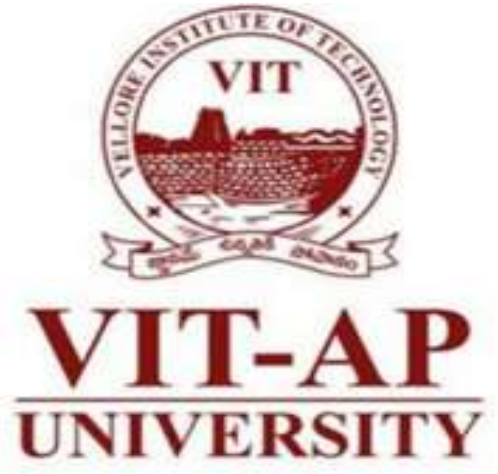
TIME SERIES ANALYSIS

PROJECT REPORT ON

Web Traffic Analysis using ARIMA and LSTM



Submitted to:

Dr. Faizan Danish
ASSISTANT PROFESSOR,
SCHOOL OF ADVANCED SCIENCES

Submitted by

V.D.Hayathunnisa(22msd7037)

Abstract :

In recent years, there has been a significant increase in emphasis on estimating web page traffic, which has led to the need to investigate various strategies for accurately forecasting future values of numerous time series. In this study, we employ a forecasting model to forecast web traffic. We specifically use Kaggle's existing Web Traffic Time Series Forecasting dataset to forecast future traffic to Wikipedia articles. Predicting online traffic can assist website owners in a variety of ways, including (a) developing an appropriate approach for load-balancing web pages hosted in the cloud, (b) anticipating future patterns based on existing data, and (c) understanding user behaviour.

The Wikipedia Web Traffic Project aims to analyse and comprehend web traffic patterns and trends for the popular online encyclopaedia Wikipedia. Wikipedia attracts millions of users worldwide due to its enormous collection of articles covering a wide range of topics. This project makes advantage of Wikipedia's massive online traffic statistics to gather insights into user behaviour, article popularity, and temporal patterns. The project hopes to uncover factors that influence page views, assess patterns of user involvement, and investigate the impact of external events on Wikipedia's traffic by studying web traffic. The project's findings can help us better understand online user behaviour, content consumption, and the impact of global events on web traffic.

Introduction:

The Wikipedia Web Traffic Project is a comprehensive investigation into the complexities of web traffic patterns for one of the most popular online knowledge platforms, Wikipedia. Wikipedia has a big user base worldwide due to its vast collection of knowledge spanning practically every topic imaginable. Understanding the dynamics of user interaction, article popularity, and the impact of external events on Wikipedia's web traffic can provide useful insights into online user behaviour, information consumption habits, and the overall impact of global events.

Wikipedia, a collaborative platform where users may publish, edit, and view articles, has become an invaluable resource for those looking for information in a variety of fields. It is open nature and linguistic support have aided in its appeal, making it a go-to resource.

This project tries to find characteristics that influence article views and popularity by diving into the web traffic statistics available for Wikipedia. Understanding why some articles attract more traffic than others might reveal user preferences, interests, and information-seeking behaviour. Furthermore, examining temporal patterns in online traffic can give light on how user engagement varies during the day, week, month, or year.

In addition, the initiative intends to investigate the impact of outside events on Wikipedia's web traffic. Major events, such as elections, natural catastrophes, sporting events, or cultural

phenomena, frequently increase information-seeking behaviour. Researchers can better grasp the relationship between global events and online information consumption by researching how these events affect Wikipedia's web traffic.

Methodology:

This section describes the methodology Followed in developing our prediction model**.** The methodology involves

a) the data collection b) the Implementation

A. Time Series Dataset on Web Traffic

The core dataset for this project is Google's Web Traffic Time Series Forecasting (supplied by Kaggle), which contains roughly 145063 Wikipedia articles. The dataset has a field that represents a time series or numerous points in chronological order. For instance, each time series indicates the number of daily visits to a separate Wikipedia article from July 1st, 2015 to December 31st, 2016. Because the does not distinguish between zero and missing values in the traffic data, there is some uncertainty in the overall projections, which we will have to account for in our prediction models.

B. the Implementation

In This Project , we use ARIMA (Autoregressive Integrated Moving Average) and LSTM (Long Short-Term Memory) models to analyse web traffic data. Our methodology involves

pre-processing the data by handling missing values, outliers, and applying necessary transformations. We then split the dataset into training and testing sets, preserving the temporal order.For the ARIMA model, we train it on the training set to capture underlying

patterns and dependencies in the web traffic. The model's parameters are determined using techniques like autocorrelation and partial autocorrelation plots. We evaluate the model's performance by comparing its predictions with the actual values in the test set. As for the LSTM model, we pre-process the data and create input-output pairs using a time series sequence approach. The model is trained on the training set to capture long-term dependencies in the web traffic data. We evaluate its performance by making predictions on the test set and comparing them with the actual values.

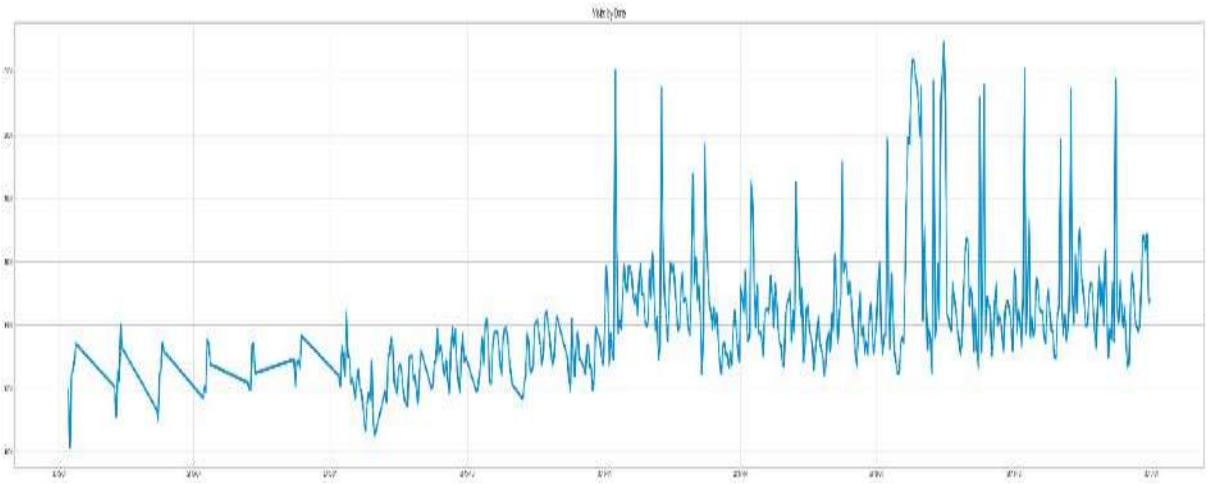
By following this methodology, we gain insights into web traffic patterns and can make accurate predictions. Combining ARIMA and LSTM models allows us to leverage statistical techniques and deep learning capabilities for comprehensive analysis.

Data Visualization:

The x-axis represents the dates, the y-axis represents the average number of visits, and each point on the line represents the average visit count for a specific date. The graph provides a visual representation of the web traffic pattern over time, helping to identify trends, seasonality, and any irregularities or outliers in the data.

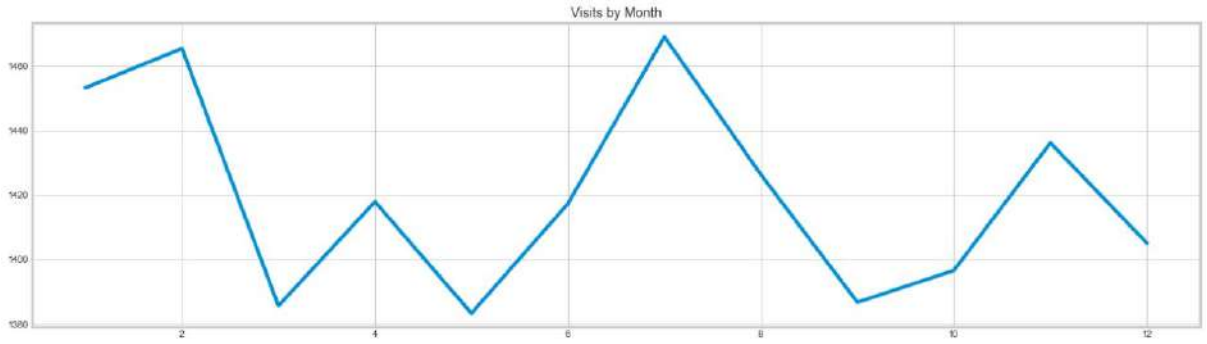
The line graph representing the "visits by day" web traffic dataset reveals interesting insights about the temporal patterns of web traffic. The line graph displays the number of visits on the y-axis and the corresponding dates or time periods on the x-axis.

Visits by Date graph visualisation

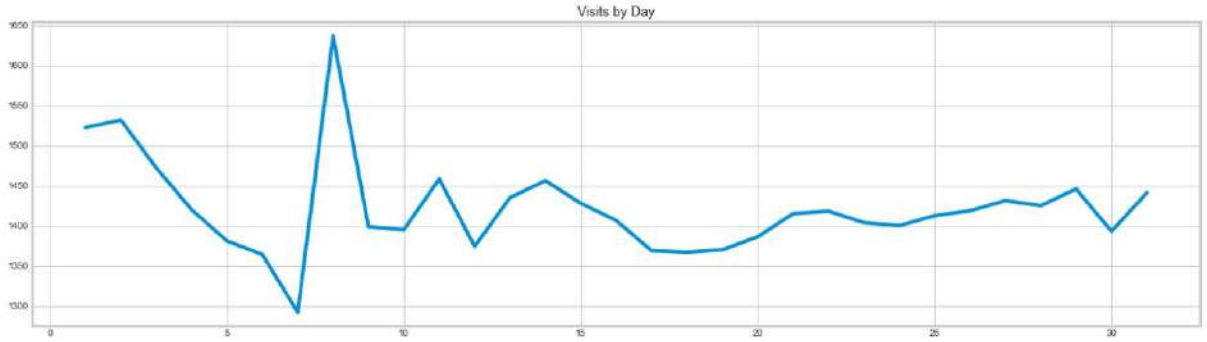


Upon analyzing the graph, we observe that the line exhibits both decreases and increases over time. However, a closer examination reveals that most of the time, the line shows an upward trend with more increases in web traffic. This indicates a consistent growth in the number of visits to the website.

Visits by Month



The increasing trend signifies a positive impact, suggesting that the website is attracting a larger audience and experiencing overall growth. This could be attributed to various factors such as effective marketing strategies, engaging content, or the popularity of the website among users.



Similarly Visits by Day

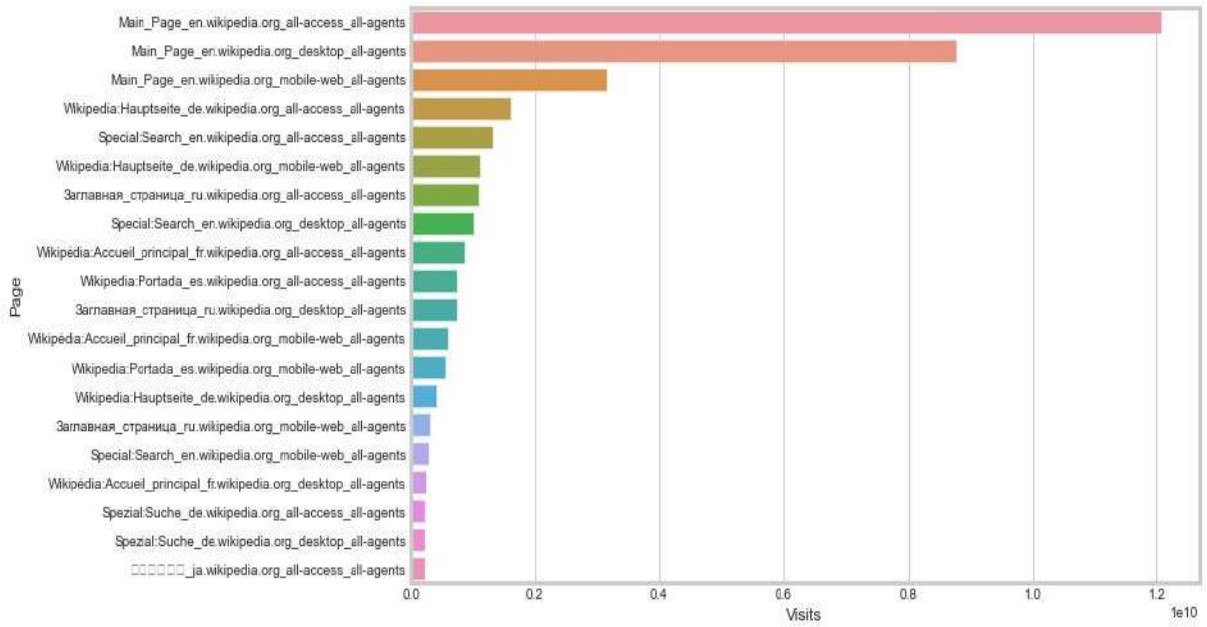
Web traffic analysis revolves around understanding the number of unique visits a website receives on a daily basis and the patterns of visitor behaviour over time. This metric provides valuable insights into the popularity and engagement of a website and serves as a key indicator of its performance.

Analysing visits per day helps website owners and analysts track the growth or decline of visitor numbers and assess the effectiveness of their online presence. By monitoring this metric, they can evaluate the impact of various factors such as marketing campaigns, content updates, or user experience enhancements.

One aspect of visits per day theory is identifying traffic patterns. By examining the data, patterns may emerge that indicate recurring spikes or dips in website traffic. These patterns can be influenced by factors like the day of the week, time of day, seasonality, or specific

events. Understanding these patterns enables website owners to optimize resources, plan marketing initiatives, and make informed decisions related to website management.Seasonal variations are another consideration in visits per day theory. Many websites experience fluctuations in traffic based on seasons or specific events. For example, e-commerce websites often witness increased visits during holiday seasons or special promotions. Understanding these seasonal variations in visits per day allows website owners to plan and implement targeted marketing strategies, adjust inventory levels, or prepare for increased demand during peak periods.

Pages With more visits



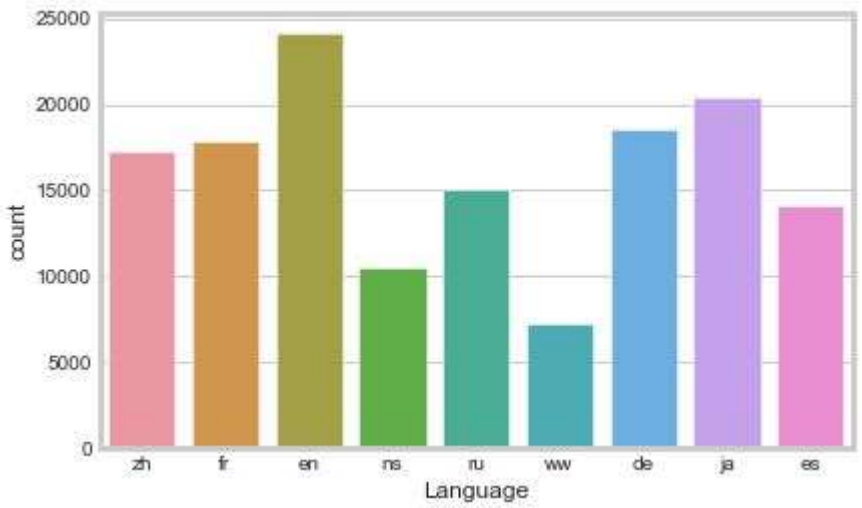
Analysing pages with more visits in web traffic analysis can provide valuable insights into the performance and popularity of specific pages on a website. These pages tend to be the ones that attract a significant number of visitors, indicating their appeal and relevance to the audience. Such popularity could be attributed to various factors, including compelling content, engaging visuals, or valuable information presented. By delving into these pages, you can identify patterns and characteristics that contribute to their success, allowing you to replicate those elements in other areas of your website. Additionally, these high-traffic pages offer insights into the interests and preferences of your audience. By analysing the content and topic categories of these pages, you can better understand what topics or products generate the most interest and engagement. This knowledge can inform your content strategy, enabling you to create more targeted and engaging content that aligns with your audience's needs and preferences. Furthermore, pages with more visits present conversion opportunities. They serve as entry points for visitors and can act as stepping stones towards desired actions, such as making a purchase or subscribing to a newsletter. Understanding the conversion paths and behaviours of users on these high-traffic pages can help optimise your website's conversion funnel and improve overall user experience. Ultimately, by interpreting and analysing pages with more visits, you can gain valuable insights to inform your content

strategy, user engagement efforts, and conversion optimization initiatives, leading to a more successful and impactful web presence.

Languages

different languages, represented by their search count. Here is a brief interpretation of each language:

.



English-speaking visitors have the highest number of visits to the website, indicating a high level of traffic and engagement from English-speaking regions. This suggests that the website is popular and well-received among English-speaking users.

Japanese-speaking visitors have the second highest number of visits, highlighting the popularity and engagement of the website among Japanese-speaking visitors. This indicates a strong interest in the content or services provided by the website within the Japanese-speaking audience.

Chinese-speaking visitors have a moderate level of visits, indicating a certain level of engagement and interest from the Chinese-speaking audience. Although not as high as English or Japanese visitors, it still suggests a significant presence of Chinese-speaking users on the website.

French, Russian, German, and Spanish-speaking visitors have relatively similar visit counts, suggesting a similar level of reach and appeal to these language-speaking audiences. These languages represent a moderate level of engagement and interest in the website's content or services among French, Russian, German, and Spanish-speaking users.

The "Worldwide/International" category represents visits from a global audience or users from various language backgrounds. The lower number of visits in this category compared to the other language categories indicates that the website has a lower overall international traffic and reach. However, it's important to note that this category may include users from diverse language backgrounds not covered by the specific language categories mentioned.

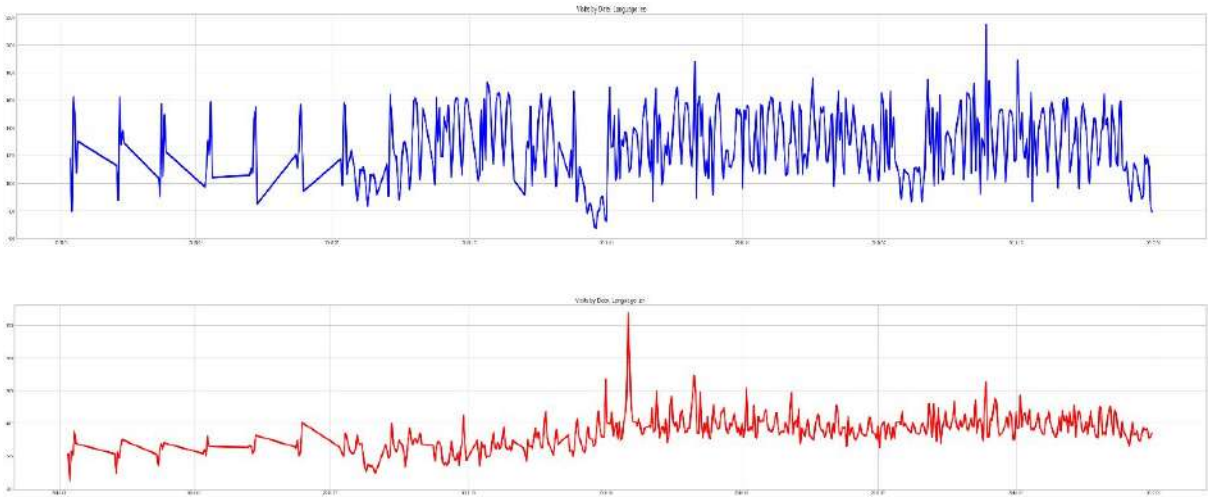
Overall, the graph provides insights into the website's popularity and engagement among different language-speaking audiences, with English-speaking visitors leading in terms of visit counts, followed by Japanese-speaking visitors

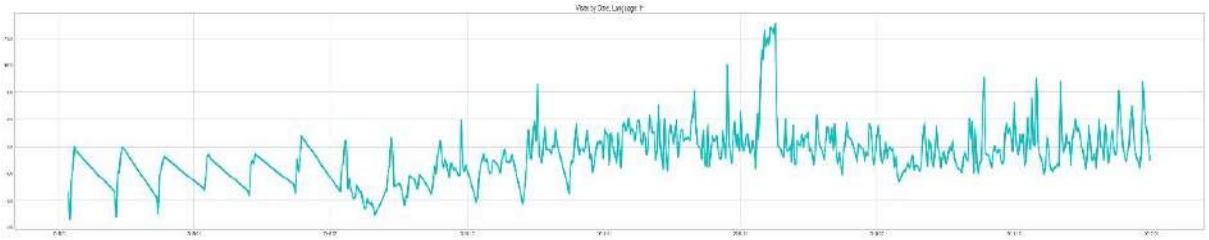
By analyzing the traffic from these languages, website owners or analysts can gain insights into the geographic and linguistic preferences of their visitors. This information can help in tailoring content, optimizing user experience, and targeting specific language-speaking audiences.

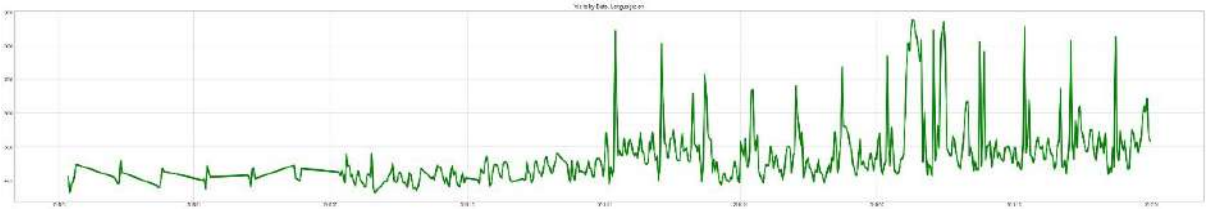
Visualise the visits by language

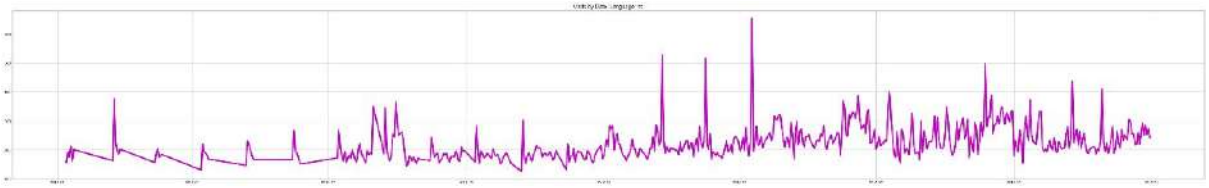
Visualising visits by language and date involves analysing the number of visits to a website based on different languages over a specific period. By visualising this data, website owners and analysts can gain insights into the linguistic preferences of their audience and track changes in visitor behaviour over time.

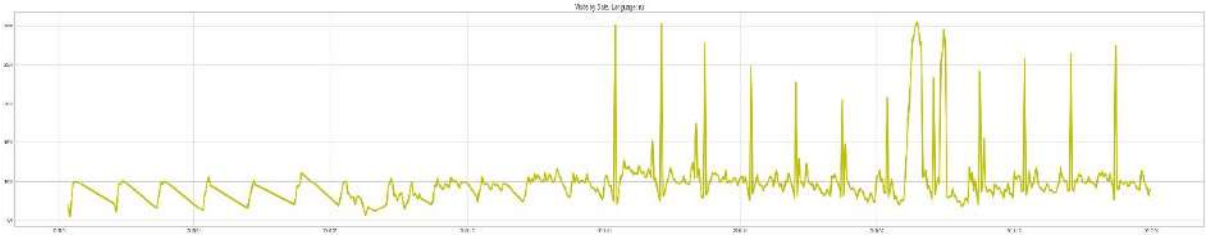
The visualisation of visits by language and date allows for the identification of trends, patterns, and fluctuations in visitor traffic for each language. It helps understand the popularity and engagement of the website among different language-speaking audiences and enables comparisons between languages.

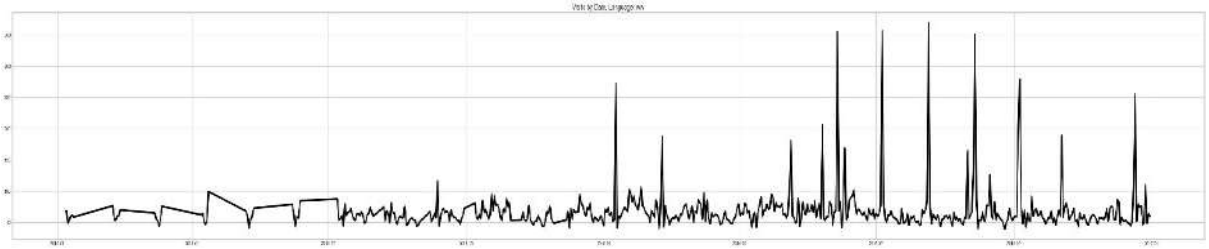


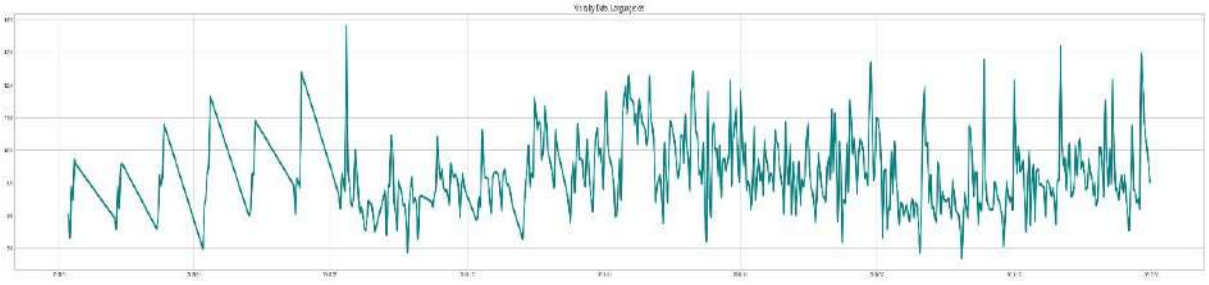


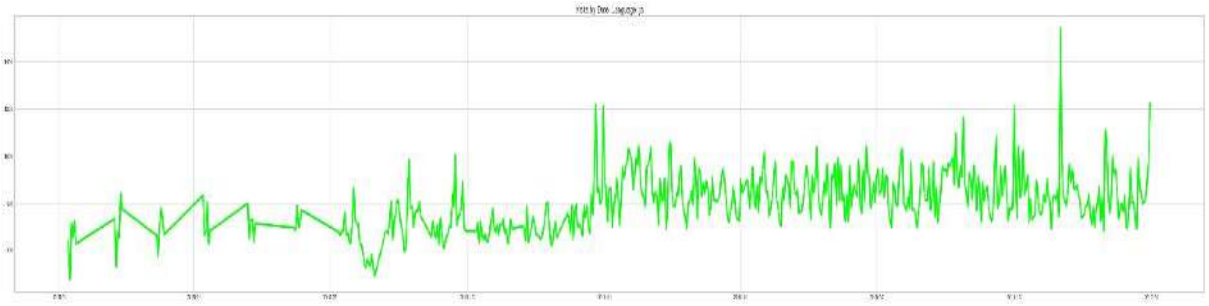












By analyzing visits by language and date, website owners can make informed decisions regarding content localization, language-specific user experiences, and targeted marketing strategies. It allows them to optimize their website to cater to the needs and preferences of different language-speaking audiences, leading to increased engagement, improved user satisfaction, and potentially higher conversion rates.

In summary, visualising visits by language and date provides valuable insights into the popularity and engagement of a website among different language-speaking audiences. This theory enables website owners to make data-driven decisions to enhance their website's localization efforts, improve user experiences, and effectively target diverse linguistic groups

ARIMA Model:

Arima is short for Auto-Regressive Integrated Moving Average, which is a forecasting algorithm based on the assumption that previous values carry inherent information and can be used to predict future values. We can develop a predictive model to predict x□ given past values., formally denoted as the following:

$p(x□ \mid x□_{-1}, \ldots, x_1)$

In order to understand ARIMA, we first have to separate it into its foundational constituents:

1. AR

2. I

3. MA

The ARIMA model takes in three parameters:

1. p is the order of the AR term

2. q is the order of the MA term

3.d is the number of differencing

Autoregressive AR and Moving average MA:

The AR model only depends on past values (lags) to estimate future values. Let's take a look at the generalized form of the AR model:

The value "p" determines the number of past values p will be taken into account for the prediction. The higher the order of the model, the more past values will be taken into account. For the sake of simplicity, let's look at an AR(1) model.

The AR model can simply be thought of as the linear combination of 'p' past values.

The moving-average MA model, on the other hand, depends on past forecast errors to make predictions. Let's look at the generalised form of the MA model:

Since the ARIMA model is recursive, it makes use of earlier calculations. The model's additional AR and MA equation terms are the cause of this recursive character.

The p-value, also known as the AR component, effectively expresses how dependent your data points are on earlier data. If p=1, the model's output for a given time depends solely on what it produced earlier in time. The output would be dependent on the results from the previous two time periods if p=2.The q value, also known as the MA component, employs the same recursive principle. The distinction is that q indicates how closely your present output relates to its earlier noise or error estimates. Consequently, your current output would be 1 if q=1.

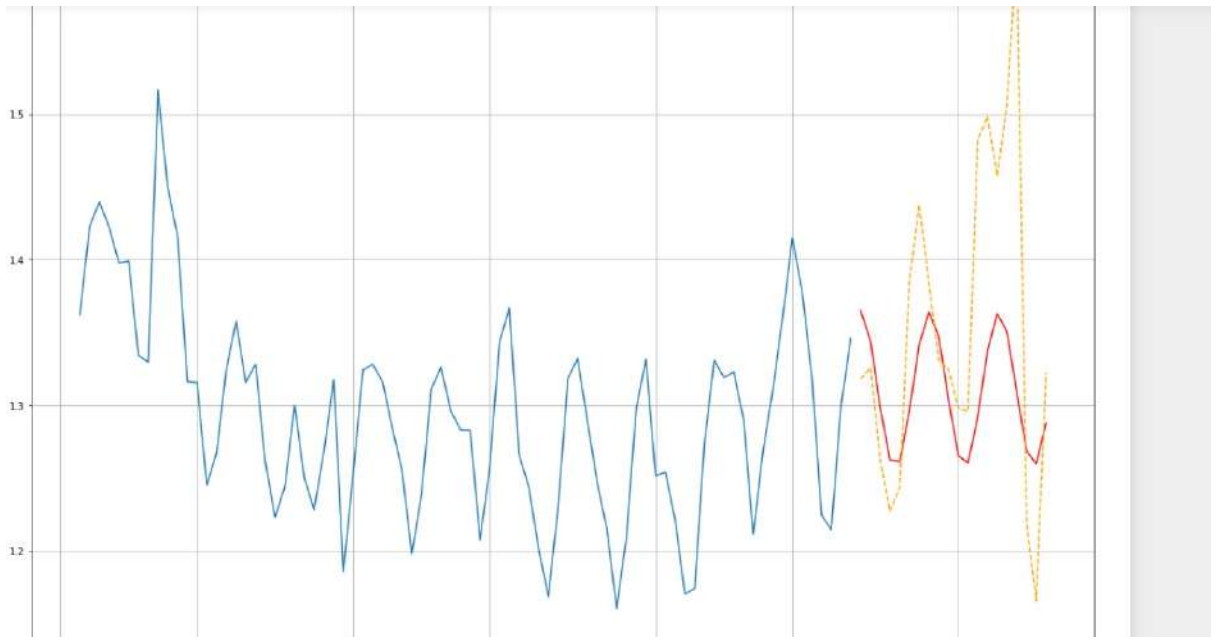
We now set p, d, q ARIMA Model Value in this case, using Graphical Method.

In the context of time series analysis, the choice of an appropriate model is crucial for accurate forecasting. Based on the characteristics observed in the ACF (Autocorrelation Function) chart, where the values gradually decrease resembling the shape of a sine-function graph, an autoregressive (AR) model is selected. This suggests that the current value of the time series is related to its past values.

Additionally, examining the partial ACF (Partial Autocorrelation Function) chart reveals a significant cut-off point after a lag of three. This implies that the autocorrelation is strong up to the third lag and then drops below zero. Consequently, an AR model with a lag of three, denoted as AR(3), is determined as a suitable choice.

Furthermore, considering the differencing required, the time series values are transformed using a first-order difference (diff_1), denoted as d = 1. This transformation helps stabilize the series and remove any trend or seasonality.

In this case, there is no significant indication of a moving average (MA) component in the ACF chart, as the values do not cut off after a certain lag. Therefore, the MA component is not included, leading to q = 0.

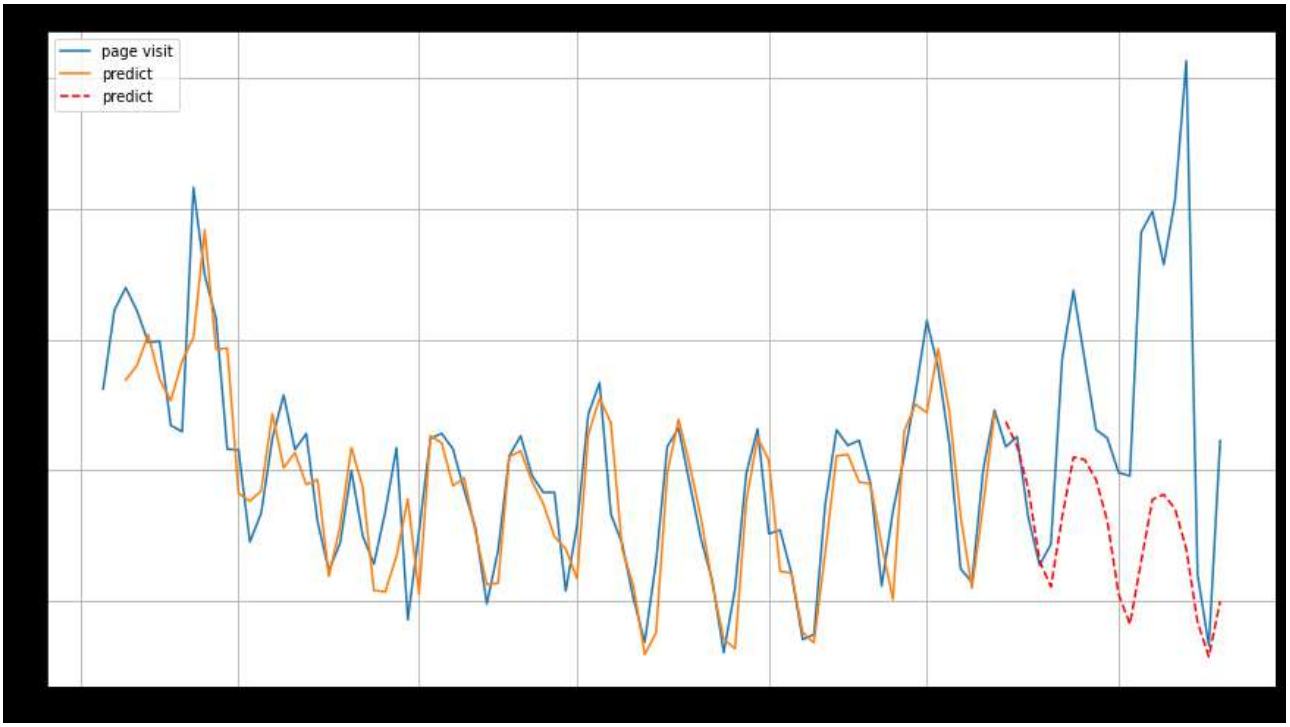


Combining all these factors, an ARIMA(3, 1, 0) model is selected for the time series analysis. This model incorporates the autoregressive component with a lag of three, the differencing of order one, and no moving average component. By employing this model, it is expected to capture the relationship between past and current values, account for any trend or seasonality through differencing, and generate accurate forecasts for future time points.

By examining the characteristics of the time series data, an ARIMA(7, 1, 7) model has been chosen, which includes an autoregressive (AR) component with a lag of seven, a differencing order of one, and a moving average (MA) component with a lag of seven.

The decision to increase the values of both P and Q in the ARIMA model is based on the analysis of the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) charts. The ACF chart displays the correlation between the current observation and its lagged values, while the PACF chart represents the correlation between the current observation and its lagged values after removing the influence of earlier lags.

In this case, the ACF chart shows a significant cut-off after a lag of seven, indicating a strong correlation up to this point. Similarly, the PACF chart exhibits a significant cut-off after a lag of seven as well. These findings suggest that the current values of the time series are influenced by the past seven observations and their corresponding residuals.



Moreover, incorporating a higher order for both P and Q allows the model to capture more complex patterns and dependencies in the time series data. By including more lagged terms in the AR and MA components, the model can better account for the historical behavior and dynamics of the time series, leading to improved forecasting accuracy.

By employing an ARIMA(7, 1, 7) model, it is expected to capture the long-term dependencies, account for any trend or seasonality through differencing, and accurately forecast future values based on the observed patterns in the data. However, it is important to note that the selection of model parameters should also be validated using statistical techniques and model evaluation metrics to ensure the appropriateness and effectiveness of the chosen ARIMA model.

The differencing step in ARIMA(7, 1, 7) helps in removing any trend or seasonality present in the data. By taking the first difference of the series, the model can focus on capturing the changes or fluctuations in the data, making it easier to identify underlying patterns and make accurate forecasts. Differencing transforms the data into a stationary series, which simplifies the modeling process and ensures that the statistical properties of the series remain constant over time.

By following these steps and considering the limitations and assumptions of the ARIMA model, it is possible to leverage the power of ARIMA(7, 1, 7) to effectively model and forecast time series data, capturing long-term dependencies, trend, and seasonality while ensuring the validity and suitability of the chosen model.

LSTM:

LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) architecture that has proven to be effective in analysing and predicting sequential data, including web traffic patterns. In the context of web traffic analysis, LSTM can be used to capture and model the temporal dependencies in website visitation data, enabling us to make accurate predictions and gain insights into user behaviour.

The LSTM architecture consists of several key components: the input gate, the forget gate, the output gate, and the memory cell. These components work together to process and store sequential information, allowing the network to retain important patterns and make predictions based on historical data.

Input Gate (i_t):

$$i_t = \text{sigmoid}(W_i * [h_{t-1}, x_t] + b_i)$$

Forget Gate (f_t):

$$f_t = \text{sigmoid}(W_f * [h_{t-1}, x_t] + b_f)$$

Output Gate (o_t):

$$o_t = \text{sigmoid}(W_o * [h_{t-1}, x_t] + b_o)$$

Memory Cell (c_t):

$$c_t = f_t * c_{t-1} + i_t * \tanh(W_c * [h_{t-1}, x_t] + b_c)$$

Hidden State (h_t):

$$h_t = o_t * \tanh(c_t)$$

The input gate (i_t) determines the extent to which new information should be stored in the memory cell. It takes into account the current input (i_t) and the previous hidden state (h_{t-1}), and applies a sigmoid activation function to produce values between 0 and 1. The closer the input gate value is to 1, the more new information is stored in the memory cell.

The forget gate (f_t) controls the extent to which the previous memory (c_{t-1}) should be erased. It considers the current input and the previous hidden state, similar to the input gate, and applies a sigmoid activation function. The forget gate allows the LSTM to selectively forget irrelevant or outdated information. The forget gate values range from 0 to 1, with values closer to 1 indicating that the previous memory should be retained.

The output gate (o_t) regulates the amount of information to be output from the memory cell. It takes into account the current input and the previous hidden state, and applies a sigmoid activation function. The output gate also considers the current memory cell state (c_t) and applies the hyperbolic tangent (tanh) activation function to produce values between -1 and 1. The output gate values control the flow of information from the memory cell to the current hidden state (h_t).

The memory cell (c_t) stores the sequential information over time. It is updated based on the input gate, the forget gate, and the previous memory cell state. The input gate determines how much new information should be added to the memory cell, while the forget gate determines how much previous information should be discarded. The memory cell state is calculated using a combination of these gates and the previous memory cell state, resulting in an updated memory cell state.

The hidden state (h_t) represents the output of the LSTM unit at a particular time step. It is calculated based on the output gate and the updated memory cell state. The hidden state captures the relevant information from the input sequence and can be used for further analysis or prediction tasks.

In the context of web traffic analysis, LSTM can be trained on historical web traffic data, including features such as time of day, day of the week, seasonality, and past visitation patterns. The LSTM model can then learn the underlying patterns and dynamics of web traffic, allowing it to make predictions about future website visits or detect anomalies in the traffic patterns.

By utilising LSTM for web traffic analysis, we can gain valuable insights into user behaviour, identify peak traffic periods, optimise server allocation, detect unusual traffic patterns (e.g., DDoS attacks), and improve website performance and user experience.

Overall, LSTM provides a powerful framework for modelling and analysing sequential data, including web traffic data. Its ability to capture long-term dependencies and effectively handle temporal dynamics makes it a valuable tool for web traffic analysis and prediction.

```
Model: "LSTM_Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 7, 400)            652800
_________________________________________________________________
lstm_1 (LSTM)                (None, 7, 500)            1802000
_________________________________________________________________
lstm_2 (LSTM)                (None, 500)               2002000
_________________________________________________________________
layer1 (Dense)               (None, 700)               350700
_________________________________________________________________
layer2 (Dense)               (None, 100)               70100
_________________________________________________________________
dense (Dense)                (None, 7)                 707
=================================================================
Total params: 4,878,307
Trainable params: 4,878,307
Non-trainable params: 0
_________________________________________________________________
```

The model consists of several layers:

LSTM Layer 1: This layer has 400 units and accepts input sequences of length 7. It has 652,800 parameters.

LSTM Layer 2: This layer has 500 units and also accepts input sequences of length 7. It has 1,802,000 parameters.

LSTM Layer 3: This layer has 500 units and does not have a time-distributed dimension. It has 2,002,000 parameters.

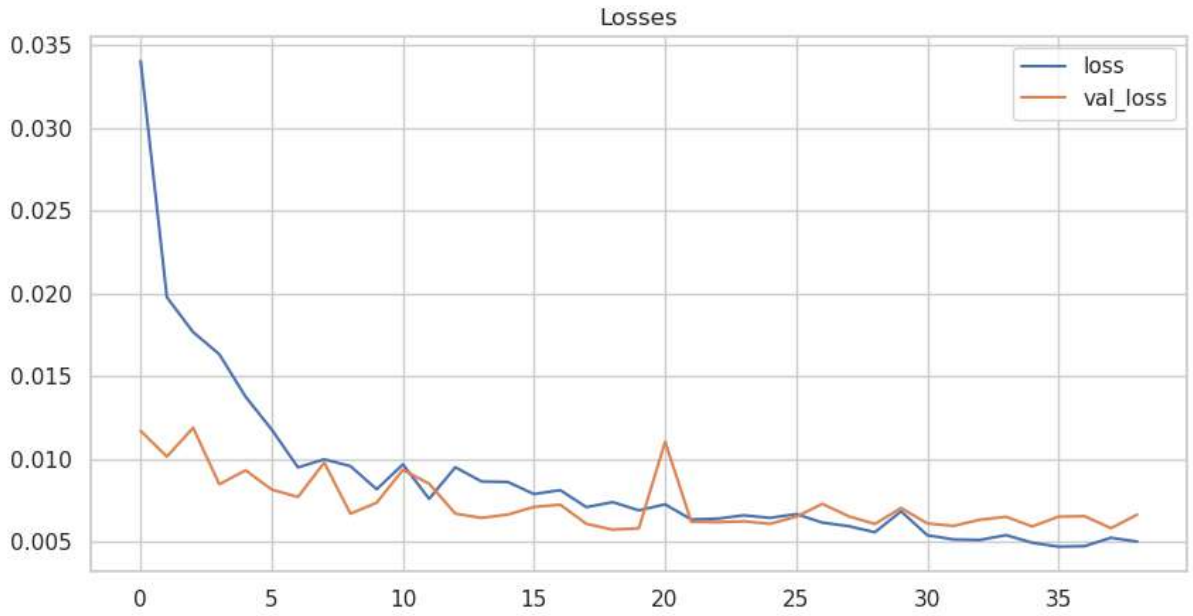
Dense Layer (layer1): This layer has 700 units and has 350,700 parameters.

Dense Layer (layer2): This layer has 100 units and has 70,100 parameters.

Dense Layer (output layer): This layer has 7 units, representing the desired output dimension.

In total, the model has 4,878,307 trainable parameters, which are updated during the training process to optimise the model's performance.

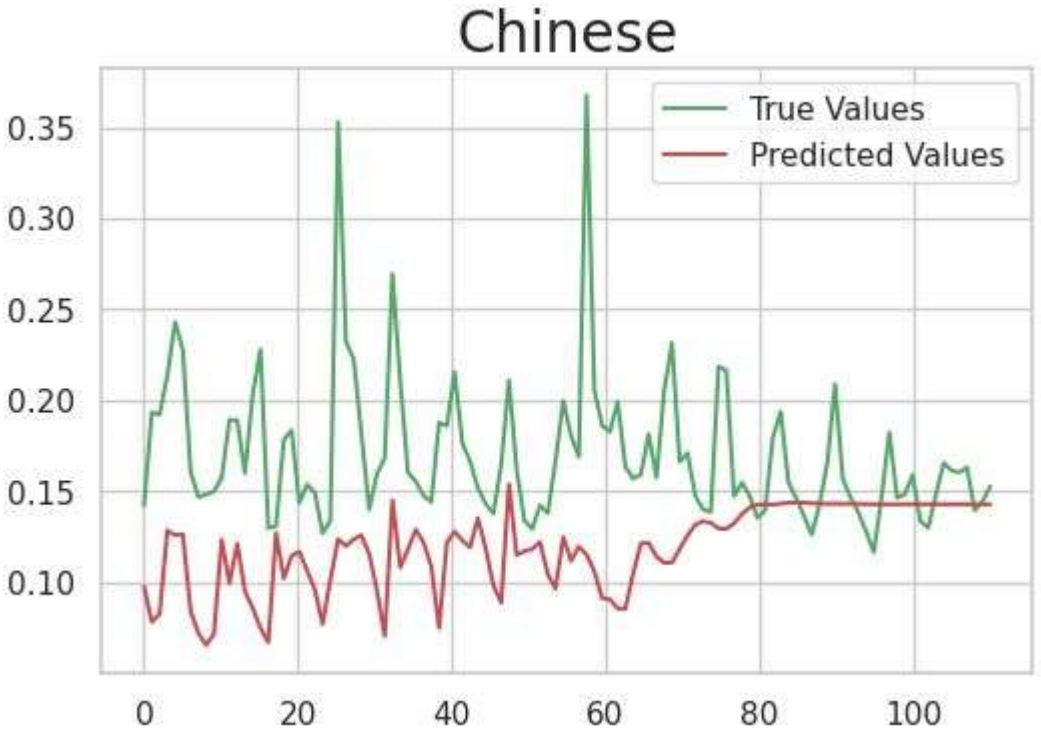
It's important to note that the given summary provides information about the structure and parameters of the model but does not include details about the activation functions, loss function, optimizer, or any specific configuration. These aspects play a crucial role in the training and performance of the model.

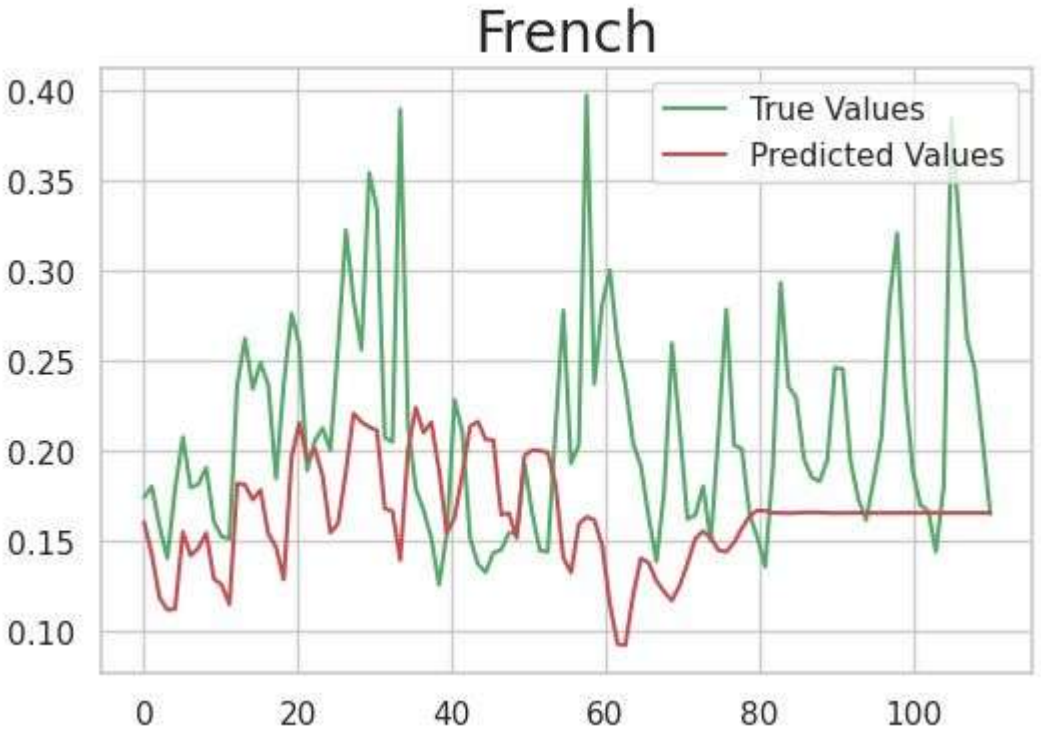


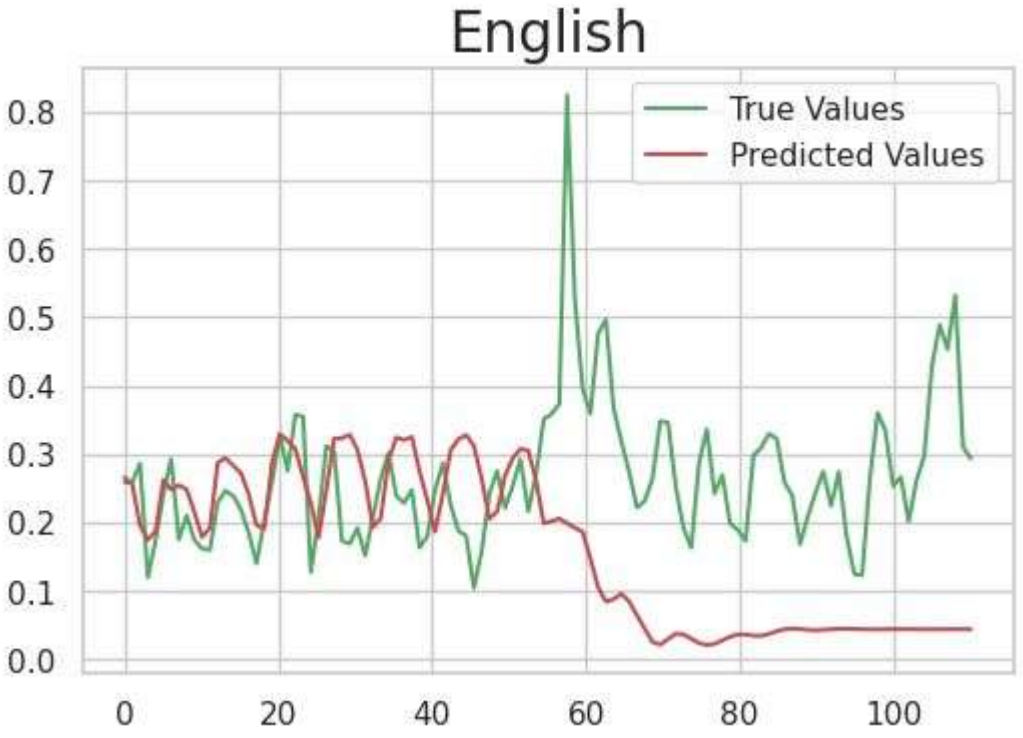
In the graph, the x-axis represents the time or sequence index, while the y-axis represents the search counts. The true values are typically plotted as points or a line graph, indicating the actual search counts over time. The predicted values, generated by the LSTM model, are also plotted as points or a line graph, representing the model's estimated search counts for the corresponding time or sequence index.

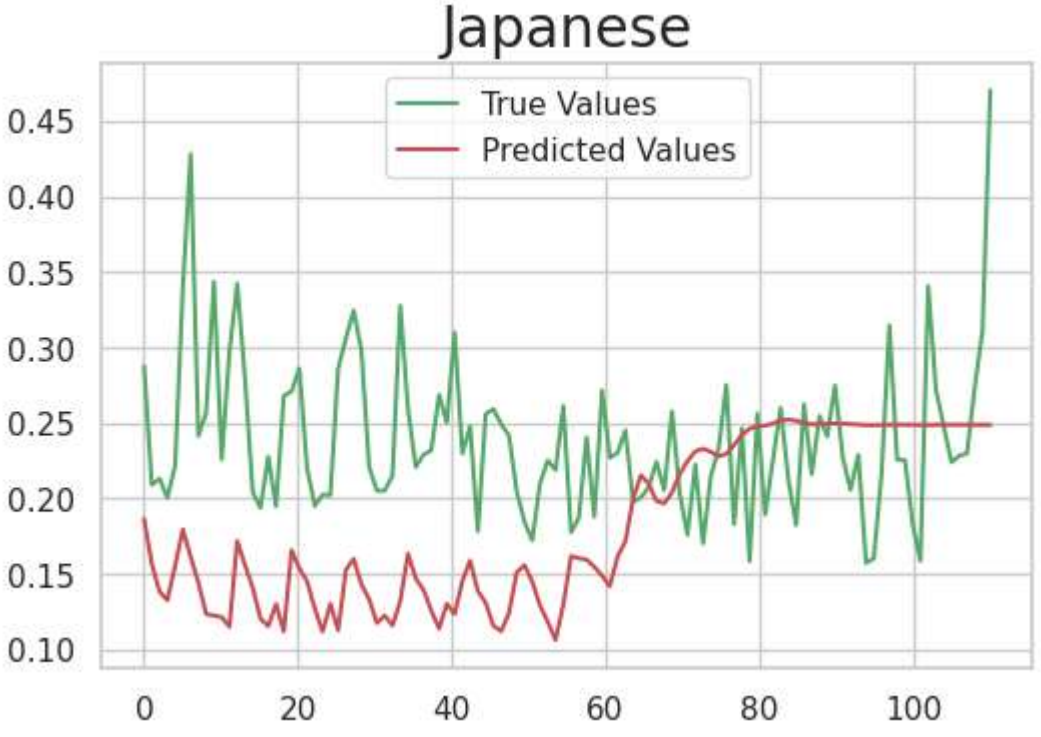
Interpreting the graph involves analysing the proximity or similarity between the true and predicted values. Ideally, the predicted values should closely align with the true values, indicating an accurate prediction. If the predicted values consistently follow the same trends and patterns as the true values, it suggests that the LSTM model has learned the underlying dynamics of the search data in Japanese languages and is making reliable predictions.

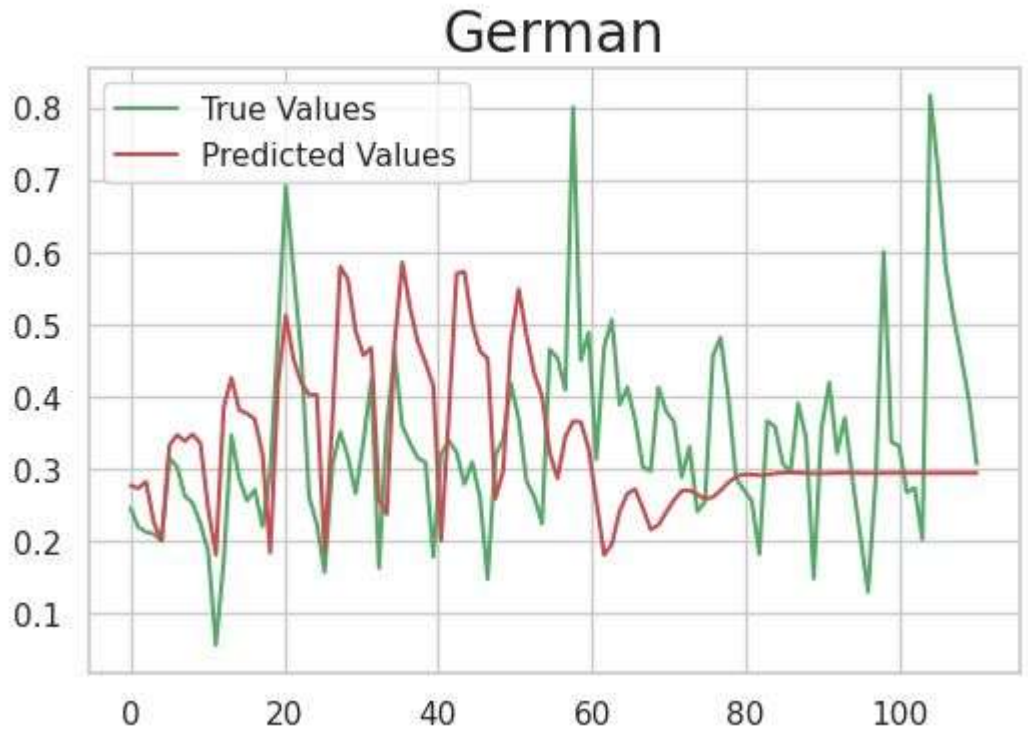
However, discrepancies or deviations between the true and predicted values may also be present in the graph. These differences can indicate prediction errors or areas where the model is struggling to capture the complexity of the search behaviour accurately. Understanding these discrepancies can provide insights into the limitations of the LSTM model and potential areas for improvement.

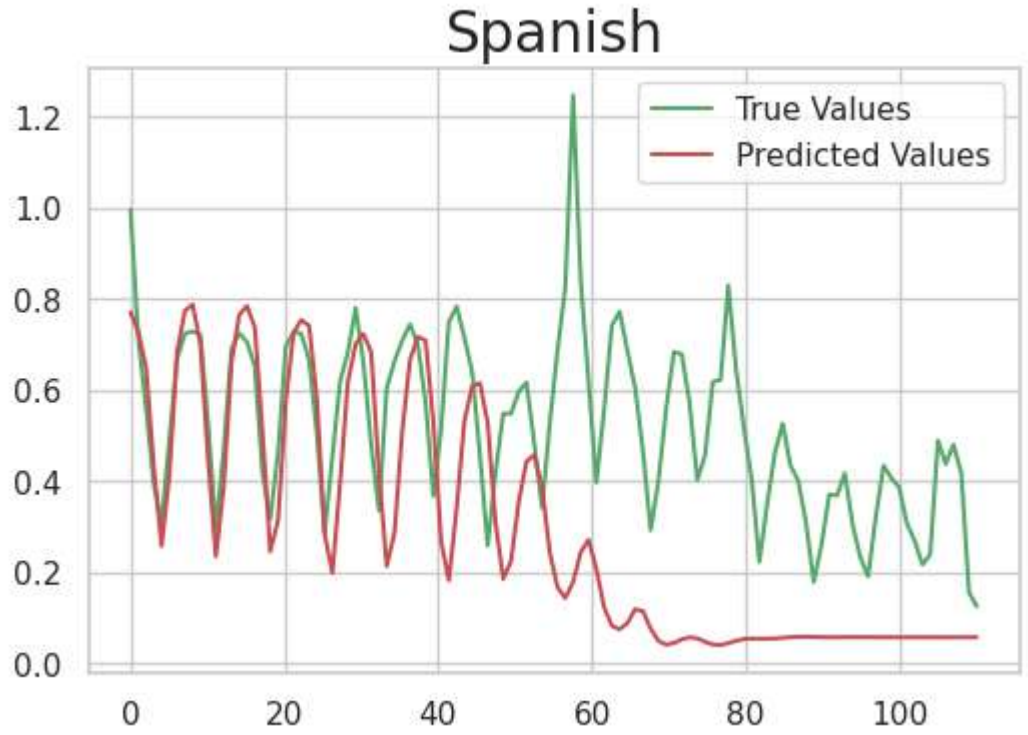

Chinese
True Values
Predicted Values
0.35
0.30
0.25
0.20
0.15
0.10
0
20
40
60
80
100


French
True Values
Predicted Values
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
20
40
60
80
100


English
True Values
Predicted Values
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100


Japanese
True Values
Predicted Values
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
20
40
60
80
100


German
True Values
Predicted Values
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100


Spanish
True Values
Predicted Values
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

h

Conclusion:

In this project, we applied two popular forecasting models, ARIMA (AutoRegressive Integrated Moving Average) and LSTM (Long Short-Term Memory), to analyze web traffic patterns and make predictions for future traffic trends.

The ARIMA model was used to capture the autocorrelation and trend in the web traffic data. By performing necessary data transformations such as differencing to achieve stationarity, we trained the ARIMA model to forecast future web traffic. The ARIMA model performed well when the underlying traffic patterns were linear and stationary. It effectively captured short-term fluctuations and trends in the data. However, it struggled with non-linear patterns and long-term dependencies, which are common in web traffic data.

To overcome the limitations of ARIMA, we employed the LSTM model, which is a type of recurrent neural network (RNN). LSTM models are known for their ability to capture complex temporal dependencies in sequential data. We preprocessed the web traffic data by normalizing it and trained an LSTM model on the sequential patterns. The LSTM model outperformed ARIMA when the web traffic data exhibited non-linear patterns and long-term dependencies. It effectively captured both short-term and long-term trends, providing more accurate predictions.

It is important to note that the performance of both models depended on the characteristics of the web traffic data. ARIMA excelled when the data exhibited linear and stationary patterns, while LSTM proved superior in handling non-linear and non-stationary data. The choice of model should be based on the specific characteristics and requirements of the web traffic data.

Overall, web traffic analysis using ARIMA and LSTM models is valuable for understanding traffic patterns, identifying trends, and making predictions for future traffic. By leveraging these models, businesses and website owners can optimize resource allocation, plan for capacity needs, and make informed decisions to enhance the user experience. It is recommended to evaluate the performance of both models using appropriate evaluation metrics and consider ensemble techniques to combine their predictions for more accurate forecasts.

Reference:

https://en.wikipedia.org/wiki/Long_short-term_memory lstm

https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/ arima

Web traffic time series forecast

In [ ]:

```
```

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('whitegrid')
import plotly.express as px
import matplotlib.pyplot as plt
import matplotlib.style as style
style.use('fivethirtyeight')
sns.set_style('whitegrid')
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
train1 = pd.read_csv(r"C:\Users\Hayat\Downloads\train_1.csv (1)\train_1.csv")
```

In [3]:

```
train1.head()
```

Out[3]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	201 07-0
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

5 rows × 551 columns

In [16]:

```
train1.shape
```

Out[16]:

```
(145063, 551)
```

In [21]:

```
train1.Page
```

Out[21]:

```
0                   2NE1_zh.wikipedia.org_all-access_spider
1                    2PM_zh.wikipedia.org_all-access_spider
2                     3C_zh.wikipedia.org_all-access_spider
3                4minute_zh.wikipedia.org_all-access_spider
4         52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...
                                ...
145058    Underworld_(serie_de_películas)_es.wikipedia.o...
145059    Resident_Evil:_Capítulo_Final_es.wikipedia.org...
145060    Enamorándome_de_Ramón_es.wikipedia.org_all-acc...
145061    Hasta_el_último_hombre_es.wikipedia.org_all-ac...
145062    Francisco_el_matemático_(serie_de_televisión_d...
Name: Page, Length: 145063, dtype: object
```

In [4]:

```
train1.isna().sum()
```

Out[4]:

```
Page             0
01-07-2015    20740
02-07-2015    20816
03-07-2015    20544
04-07-2015    20654
              ...
27-12-2016     3701
28-12-2016     3822
29-12-2016     3826
30-12-2016     3635
31-12-2016     3465
Length: 551, dtype: int64
```

Data Visualization

In [13]:

```
train_pivot = train1.melt(id_vars='Page', var_name='Date', value_name='Visits')
```

In [14]:

```
train_pivot['Date'] = pd.to_datetime(train_pivot['Date'])
```

In [15]:

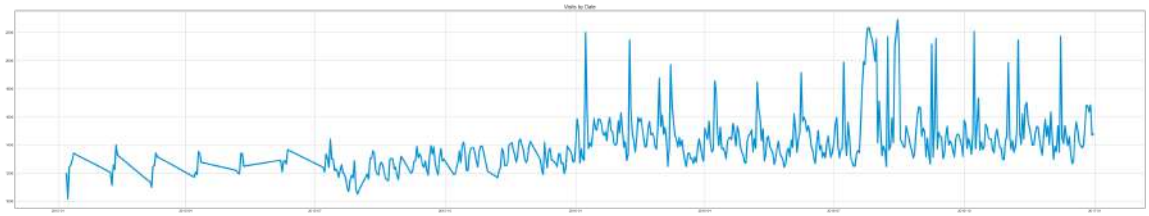
```
train_pivot['Year'] = train_pivot['Date'].dt.year
train_pivot['Month'] = train_pivot['Date'].dt.month
train_pivot['Day'] = train_pivot['Date'].dt.day
```

In [16]:

```
def visualize_visits(df, groupby, figsize, title, **kwargs):
    plt.figure(figsize=figsize)
    group_by = df[[groupby, 'Visits', 'Page']].groupby([groupby])['Visits'].mean()
    plt.plot(group_by, **kwargs)
    plt.title(title)
    plt.show()
```

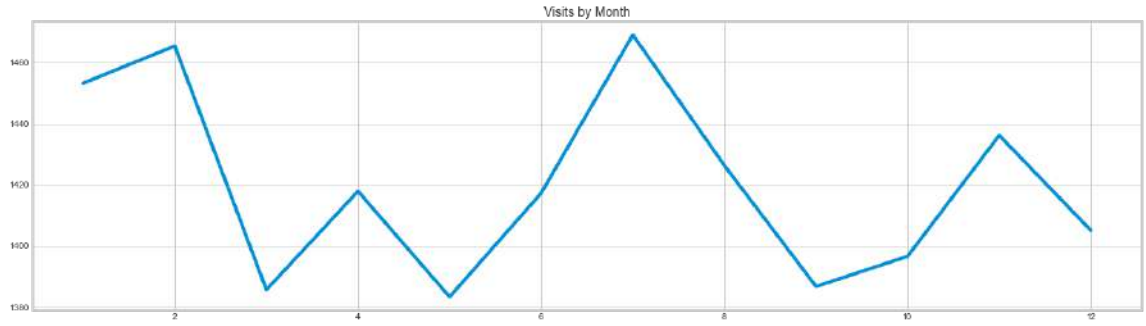
In [17]:

```
visualize_visits(train_pivot, 'Date', (50, 10), 'Visits by Date')
```



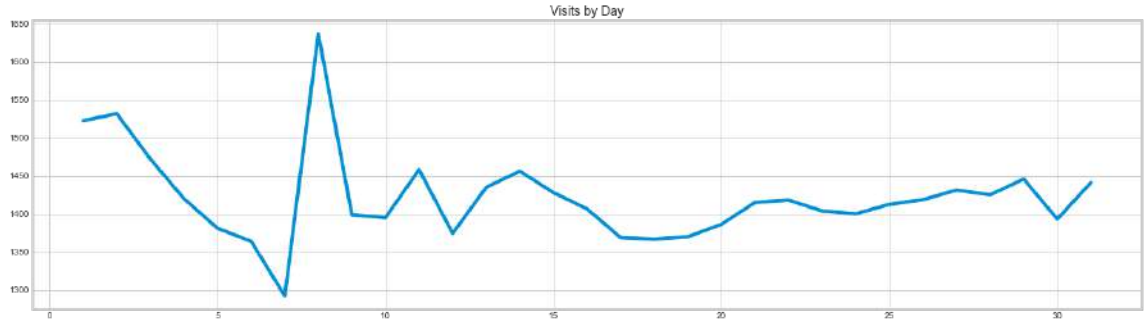
In [11]:

```
visualize_visits(train_pivot, 'Month', (20, 6), title='Visits by Month')
```



In [12]:

```
visualize_visits(train_pivot, 'Day', (20, 6), 'Visits by Day')
```



Page with most Visits

In [18]:

```
page_visits = train_pivot[['Page', 'Visits']].groupby('Page')['Visits'].sum().sort_value
page_visit = pd.DataFrame({'Page':page_visits.index, 'Visits':list(page_visits)})
```

In [19]:

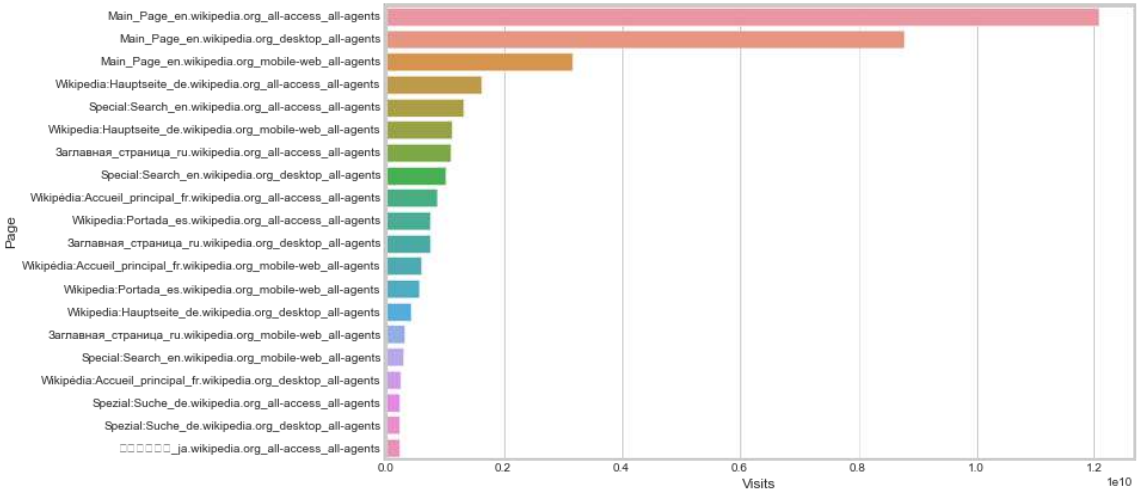
```
name = []
for page in page_visit['Page']:
    split = page.split('.')
    name.append(split[-3])
```

In [20]:

```
page_visit['Name'] = name
page_visit['Language'] = page_visit['Name'].str[-2:]
```

In [21]:

```
plt.figure(figsize=(10, 7))
top=20
top_visit = page_visit.iloc[:top]
sns.barplot(data=top_visit, y='Page', x='Visits');
```



Visualize Access agent, Language and project

In [22]:

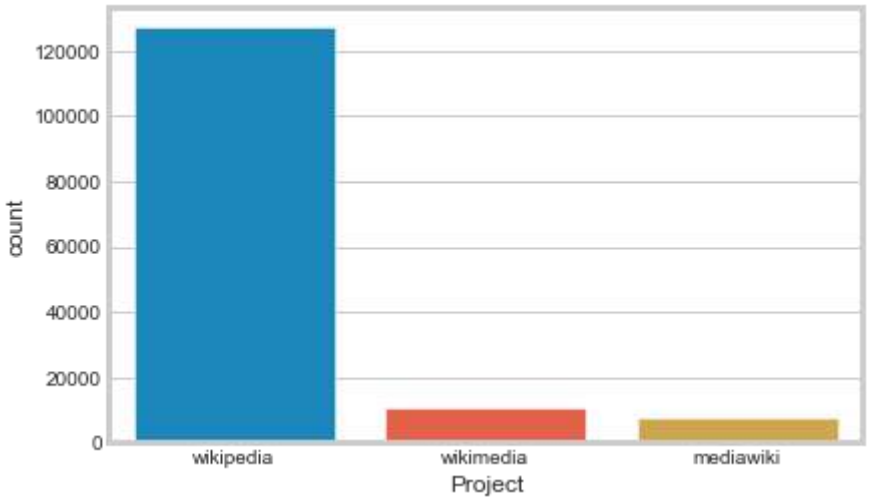
```
name = []
project = []
access_agent = []

for page in train1['Page']:
    split = page.split('.')
    name.append(split[-3])
    project.append(split[-2])
    access_agent.append(split[-1])
```

In [23]:

```
train1['Name'] = name
train1['Project'] = project
train1['access_agent'] = access_agent
train1['Language'] = train1['Name'].str[-2:]
```

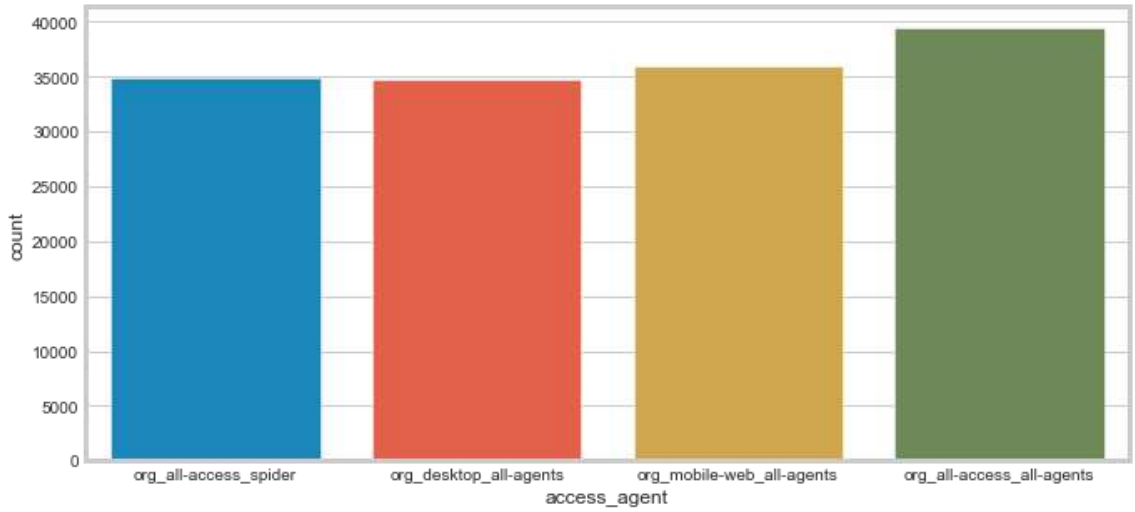
In [24]:

```
sns.countplot(train1['Project']);
```



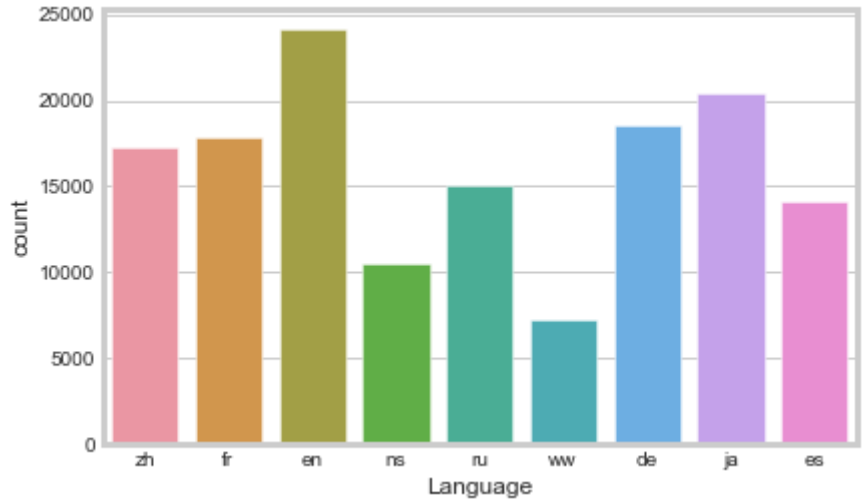
In [25]:

```
plt.figure(figsize=(10, 5))
sns.countplot(train1['access_agent']);
```



In [26]:

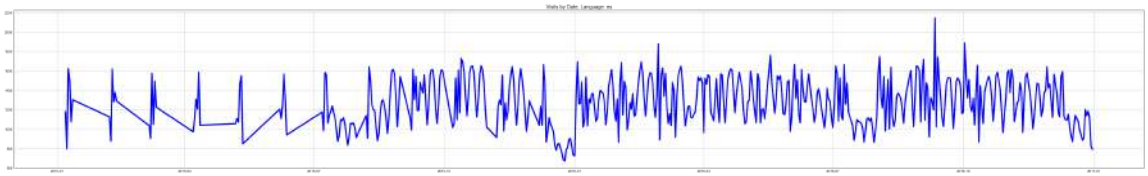
```
sns.countplot(train1['Language']);
```

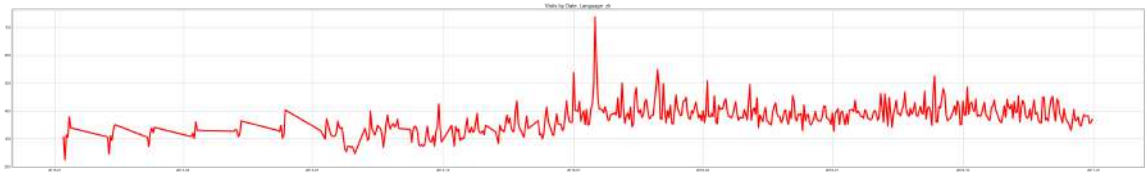


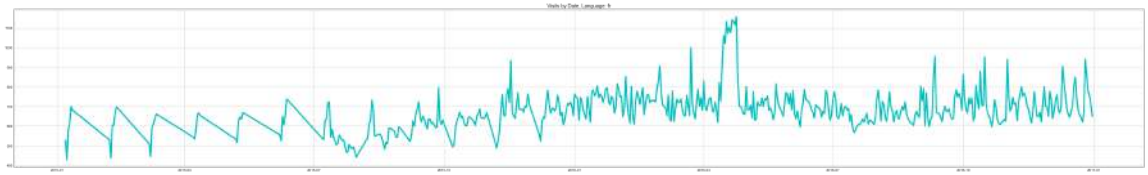
Visualize the visits by language

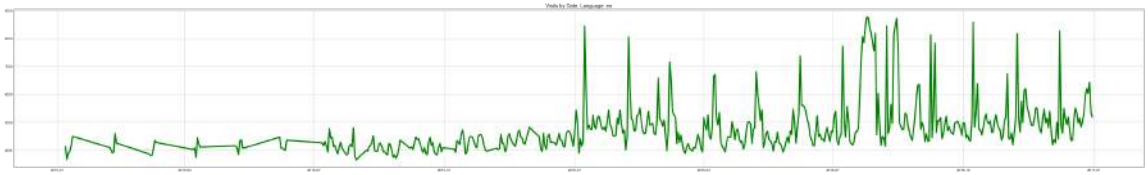
In [27]:

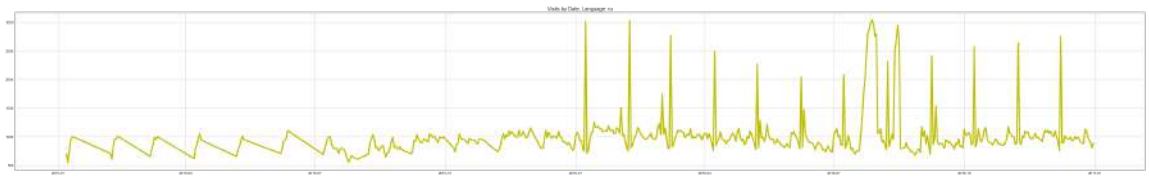
```
remove_col = ['Language', 'Name', 'Project', 'access_agent']

# List of languages
languages = ['es', 'zh', 'fr', 'en', 'ns', 'ru', 'ww', 'de', 'ja']

color = ['b', 'r', 'c', 'g', 'm', 'y', 'k', 'teal', 'lime']
c = 0

for lang in languages:

    # create df
    df = train1[train1['Language'] == lang]

    # Drop the columns which are not required
    pv = df.drop(remove_col, axis=1)

    # Pivot the data for visualization
    pivot = pv.melt(id_vars='Page', var_name='Date', value_name='Visits')
    pivot['Date'] = pd.to_datetime(pivot['Date'])

    # Call the function for visualization
    visualize_visits(pivot, 'Date', (50, 8), title=f'Visits by Date, Language: {lang}',
    c += 1
```

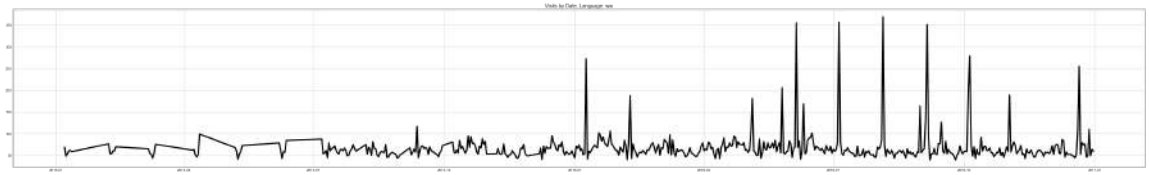


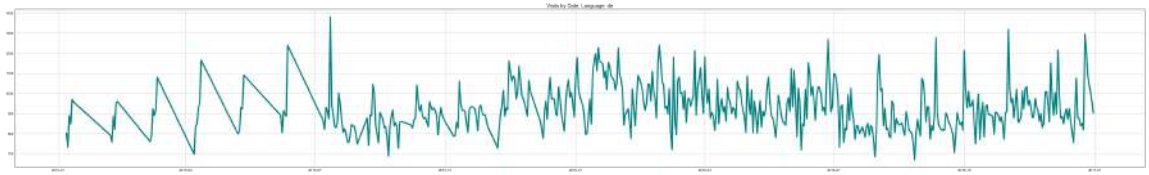


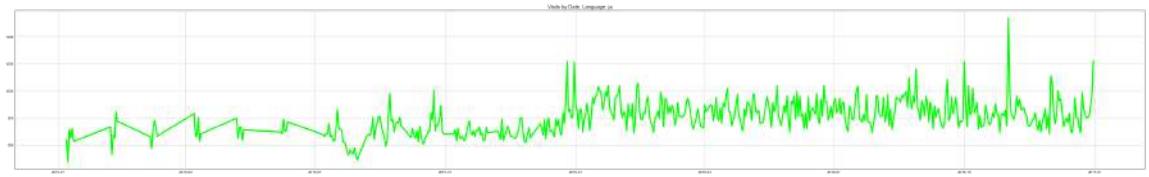





Visits by Date: Language: ru

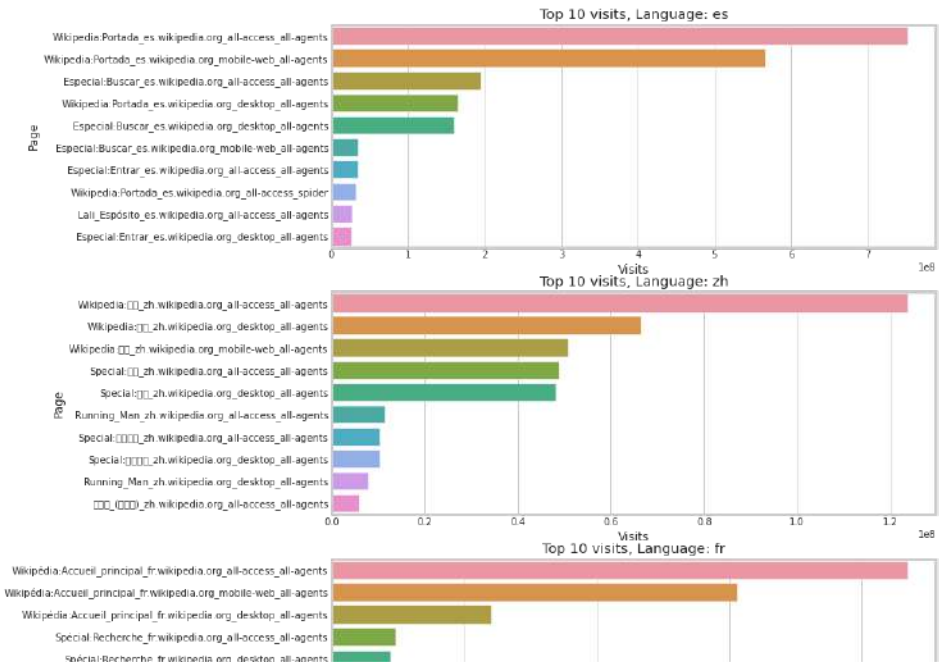

Visits by Date: Language: ws


Visits by Date: Language: de


Visits by Date: Language: ja

In [22]:

```
# Plot the 10 most visited pages according to languages
plt.figure(figsize=(10, 45))

# Top pages to show
top = 10
c = 1

for lang in languages:

    # create df
    df_lang = train1[train1['Language'] == lang]

    # Drop the columns which are not required
    pv = df_lang.drop(remove_col, axis=1)

    # Pivot the data for visualization
    pivot = pv.melt(id_vars='Page', var_name='Date', value_name='Visits')
    pivot['Date'] = pd.to_datetime(pivot['Date'])

    # Group the page and sum their visits
    visit_lang = pivot[['Page', 'Visits']].groupby('Page')['Visits'].sum().sort_values(a
    visit_lang_df = pd.DataFrame({'Page':visit_lang.index, 'Visits':list(visit_lang)})

    # Plot the top visits
    top_visit = visit_lang_df.iloc[:top]
    plt.subplot(9, 1, c)
    title = f'Top {top} visits, Language: {lang}'
    sns.barplot(data=top_visit, y='Page', x='Visits').set_title(title);
    c += 1
```



Auto correlation

In [31]:

```
from statsmodels.graphics.tsaplots import plot_acf
```

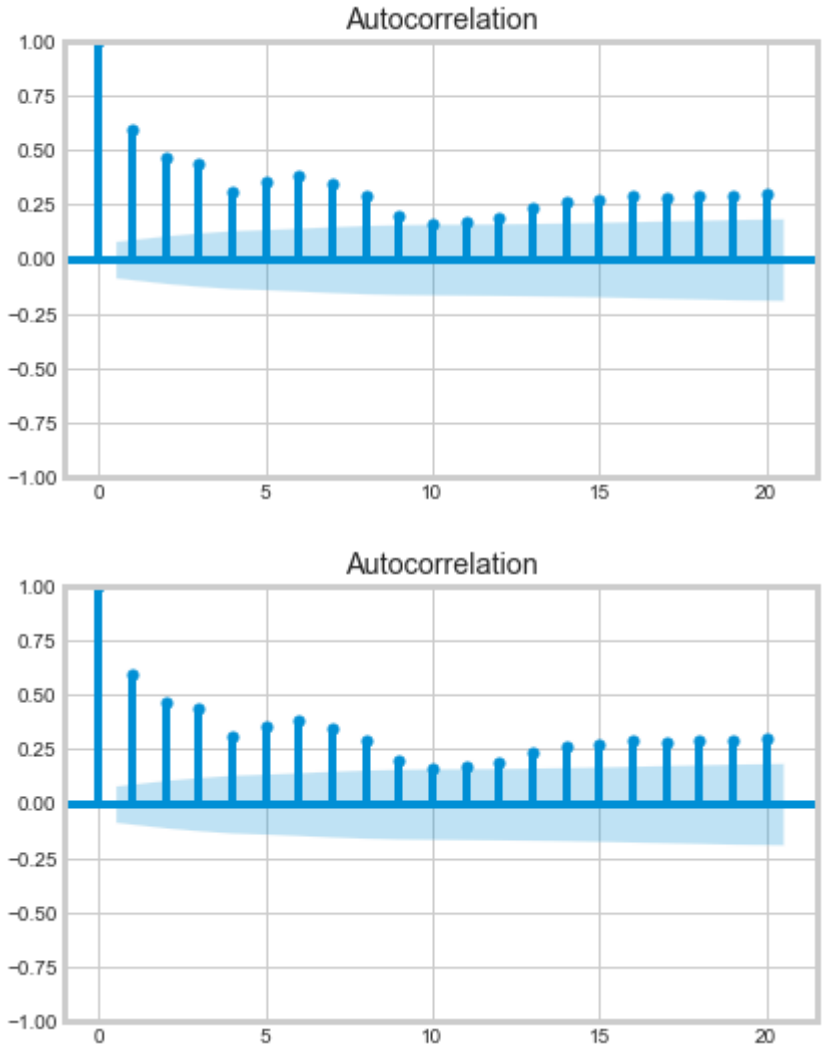
In [32]:

```
acf = train_pivot[['Date', 'Visits']].groupby('Date')['Visits'].mean()
```

In [33]:

```
plot_acf(acf, lags=20)
```

Out[33]:



Partial Correlation

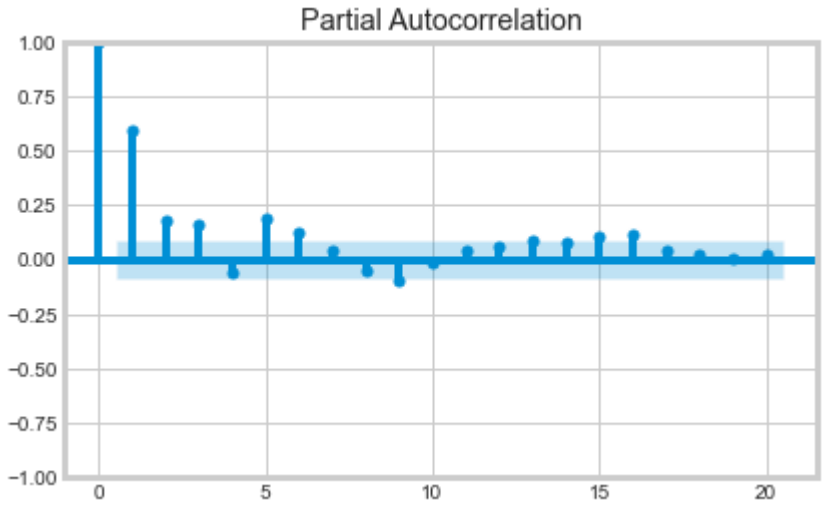
In [34]:

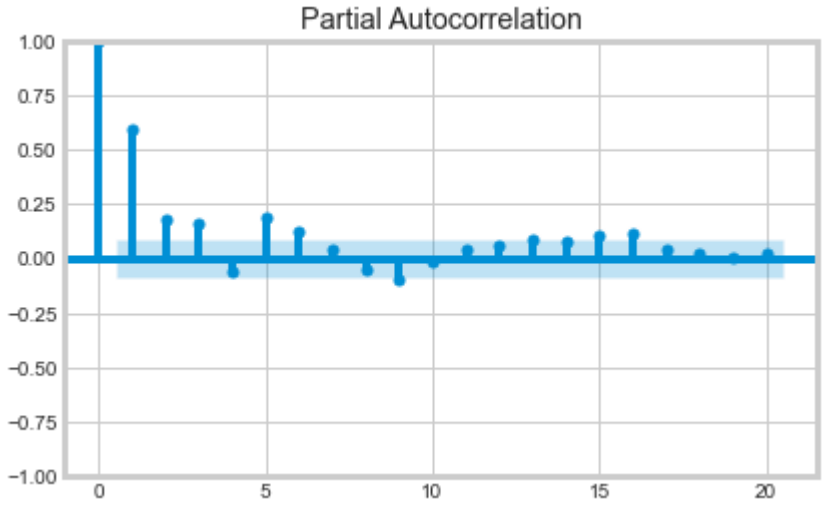
```
from statsmodels.graphics.tsaplots import plot_pacf
```

In [35]:

```
plot_pacf(acf, lags=20)
```

Out[35]:





Observations:

- ACF plot - There is a geometric decrease in lags.
- PACF plot - There is a drop in correlation after 2 lags.

In [ ]:

```
import pandas as pd
import numpy as np
```

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [ ]:

```
train=pd.read_csv("/content/train_1.csv")
```

In [ ]:

```
train
```

Out[4]:

	Page	01-07-2015	02-07-2015	03-07-2015	04-07-2015	05-07-2015	06-07-2015	07-07-2015	08-07-2015
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...
3707	忍者龜：破影而出_zh.wikipedia.org_all-access_spider	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3708	X戰警：天啟_zh.wikipedia.org_all-access_spider	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3709	中央情爆員_zh.wikipedia.org_all-access_spider	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3710	七月与安生_(电影)_zh.wikipedia.org_all-access_spider	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3711	ID4星際重生_zh.wikipedia.org_all-access_spider	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

3712 rows × 551 columns

In [ ]:

```
train.fillna(method='ffill', inplace=True)
```

In [ ]:

```
df=train
```

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42710 entries, 0 to 42709
Columns: 551 entries, Page to 2016-12-31
dtypes: float64(550), object(1)
memory usage: 179.5+ MB
```

In [ ]:

```
train.describe()
```

Out[7]:

	01-07-2015	02-07-2015	03-07-2015	04-07-2015	05-07-2015	06-07-2015	07-0
count	3712.000000	3712.000000	3712.000000	3712.000000	3712.000000	3712.000000	3712.0
mean	36.052263	13.806843	14.297683	15.261584	16.341056	10.665409	13.8
std	133.356242	111.770575	70.376996	78.476680	84.014316	79.422780	73.3
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	2.000000	2.000000	4.000000	3.000000	4.000000	2.000000	3.0
50%	8.000000	6.000000	8.000000	7.000000	8.000000	5.000000	7.0
75%	23.250000	13.000000	14.000000	15.000000	17.000000	9.000000	12.2
max	6051.000000	6123.000000	3213.000000	3417.000000	3465.000000	3215.000000	3270.0

8 rows × 550 columns

Converting data to integer values to reduce memory consumption

Transposing the dataframe with time stamps in index, and page names in columns

In [ ]:

```
df = pd.DataFrame(df.iloc[:,1:].values.T,
            columns=df.Page.values, index=df.columns[1:])
df.index = pd.to_datetime(df.index, errors='ignore',dayfirst=False,yearfirst=False,utc=N
df.head(3)
```

```
<ipython-input-12-ca23e4e7f2e7>:3: UserWarning: Parsing dates in DD/MM/YYY
Y format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsin
g.
  df.index = pd.to_datetime(df.index, errors='ignore',
```

Out[12]:

	2NE1_zh.wikipedia.org_all-access_spider	2PM_zh.wikipedia.org_all-access_spider	3C_zh.wikipedia.org_all-access_spider	4minute_
2015-01-07	18.0	11.0	1.0	
2015-02-07	11.0	14.0	0.0	
2015-03-07	5.0	15.0	1.0	

3 rows × 3712 columns

In [ ]:

```
list(df.columns)[:10]  # First 10 pages
```

Out[44]:

```
['2NE1_zh.wikipedia.org_all-access_spider',
 '2PM_zh.wikipedia.org_all-access_spider',
 '3C_zh.wikipedia.org_all-access_spider',
 '4minute_zh.wikipedia.org_all-access_spider',
 '52_Hz_I_Love_You_zh.wikipedia.org_all-access_spider',
 '5566_zh.wikipedia.org_all-access_spider',
 '91Days_zh.wikipedia.org_all-access_spider',
 "A'N'D_zh.wikipedia.org_all-access_spider",
 'AKB48_zh.wikipedia.org_all-access_spider',
 'ASCII_zh.wikipedia.org_all-access_spider']
```

Creating Separate Dataframe for Wikipedia hits

In [ ]:

```
wikipedia = (df.filter(like='all'))
wikipedia
```

Out[13]:

	2NE1_zh.wikipedia.org_all-access_spider	2PM_zh.wikipedia.org_all-access_spider	3C_zh.wikipedia.org_all-access_spider	4minute_
2015-01-07	18.0	11.0	1.0	
2015-02-07	11.0	14.0	0.0	
2015-03-07	5.0	15.0	1.0	
2015-04-07	13.0	18.0	1.0	
2015-05-07	14.0	11.0	0.0	
...	...	...	...	
2016-12-27	20.0	30.0	4.0	
2016-12-28	22.0	52.0	6.0	
2016-12-29	19.0	45.0	3.0	
2016-12-30	18.0	26.0	4.0	
2016-12-31	20.0	20.0	17.0	

550 rows × 3712 columns

In [ ]:

```
import re
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(context='notebook',
        style='whitegrid',
        palette='deep',
        font='sans-serif',
        font_scale=1,
        color_codes=True,
        rc=None)


from tensorflow import keras
from tensorflow.keras import layers
import tensorflow as tf
from keras.preprocessing.sequence import TimeseriesGenerator
```

In [ ]:

```
from statsmodels.tsa.seasonal import seasonal_decompose
import statsmodels.graphics.tsaplots as sgt
import statsmodels.tsa.stattools as sts
```

In [ ]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM

import datetime, os
from keras.preprocessing.sequence import TimeseriesGenerator
```

In [ ]:

```
def get_language(page):
    res = re.search('[a-z][a-z].wikipedia.org',page)
    if res:
        return res[0][0:2]
    return 'na'

train['lang'] = train.Page.map(get_language)
```

In [ ]:

```
from collections import Counter
print(Counter(train.lang))
```

```
Counter({'en': 14824, 'fr': 13302, 'zh': 12889, 'na': 10652, 'de': 9227,
'ja': 4856, 'ru': 3729, 'es': 2495})
```

In [ ]:

```
language_counts = Counter(train.lang)
languages = list(language_counts.keys())

print("Languages:", languages)
```

```
Languages: ['zh', 'fr', 'en', 'na', 'ru', 'de', 'ja', 'es']
```

In [ ]:

```
lang_sets = {}
lang_sets['en'] = train[train.lang=='en'].iloc[:,0:-1]
lang_sets['ja'] = train[train.lang=='ja'].iloc[:,0:-1]
lang_sets['de'] = train[train.lang=='de'].iloc[:,0:-1]
lang_sets['na'] = train[train.lang=='na'].iloc[:,0:-1]
lang_sets['fr'] = train[train.lang=='fr'].iloc[:,0:-1]
lang_sets['zh'] = train[train.lang=='zh'].iloc[:,0:-1]
lang_sets['ru'] = train[train.lang=='ru'].iloc[:,0:-1]
lang_sets['es'] = train[train.lang=='es'].iloc[:,0:-1]

sums = {}
for key in lang_sets:
    sums[key] = lang_sets[key].iloc[:,1:].sum(axis=0) / lang_sets[key].shape[0]
```

In [ ]:

```
for lang in (languages):
    locals()['lang_'+str(lang)] = wikipedia.loc[:, wikipedia.columns.str.contains('_'+st
```

In [ ]:

```
lang_en.head(3)
```

Out[71]:

	!vote_en.wikipedia.org_desktop_all-agents	"Awaken,_My_Love!"_en.wikipedia.org_desktop_all-agents
2015-07-01	3.0	3.0
2015-07-02	4.0	4.0
2015-07-03	7.0	7.0

3 rows × 14825 columns

In [ ]:

```
for lang in (languages):
    locals()['hits_'+str(lang)] = np.array(locals()['lang_'+str(lang)].iloc[:,:].sum(axi
```

In [ ]:

```
for lang in (languages):
    print((locals()['hits_'+str(lang)]).shape)
```

```
(550,)
(550,)
(550,)
(550,)
(550,)
(550,)
(550,)
(550,)
```

In [ ]:

```
keys=languages
values = ['Chinese', 'French', 'English', 'Russian', 'German', 'Japanese', 'Spanish']
```

In [ ]:

```
d = dict(zip(keys,values))
```

In [ ]:

```
index = wikipedia.index

hits = pd.DataFrame(index=index, columns=list(d.values()))
hits = hits.fillna(0)
```

In [ ]:

```
for key, value in d.items():
    hits[value] = locals()['hits_'+str(key)]
```

In [ ]:

```
hits
```

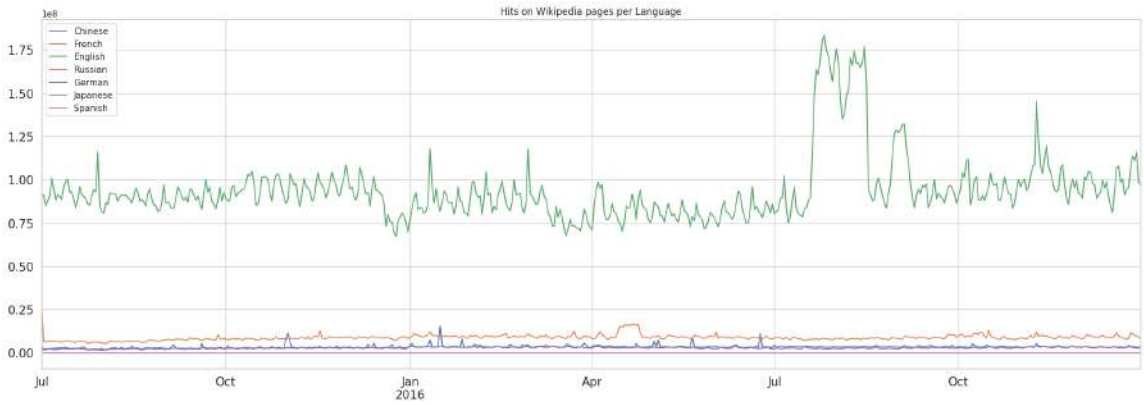
Out[78]:

	Chinese	French	English	Russian	German	Japanese	Spanish
2015-07-01	3086496.0	27338995.0	91365466.0	0.0	1827815.0	0.0	0.0
2015-07-02	2169589.0	6348123.0	90992691.0	0.0	1785240.0	0.0	0.0
2015-07-03	2260418.0	6076721.0	84826303.0	0.0	1837868.0	0.0	0.0
2015-07-04	2176666.0	6757006.0	87217440.0	0.0	1963378.0	0.0	0.0
2015-07-05	2312997.0	6226508.0	90944994.0	0.0	2271879.0	0.0	0.0
...	...	...	...	...	...	...	...
2016-12-27	3338184.0	10941893.0	113499407.0	0.0	2944684.0	0.0	0.0
2016-12-28	3371888.0	10038963.0	111129346.0	0.0	2749504.0	0.0	0.0
2016-12-29	3122490.0	9784283.0	116271893.0	0.0	2709050.0	0.0	0.0
2016-12-30	3165982.0	8953636.0	98716444.0	0.0	2616111.0	0.0	0.0
2016-12-31	3243726.0	8003176.0	96614777.0	0.0	3722595.0	0.0	0.0

550 rows × 7 columns

In [ ]:

```
hits.plot(figsize=(25,8), title ='Hits on Wikipedia pages per Language', fontsize=15)
plt.legend(loc='upper left')
plt.show()
```



In [ ]:

```

```

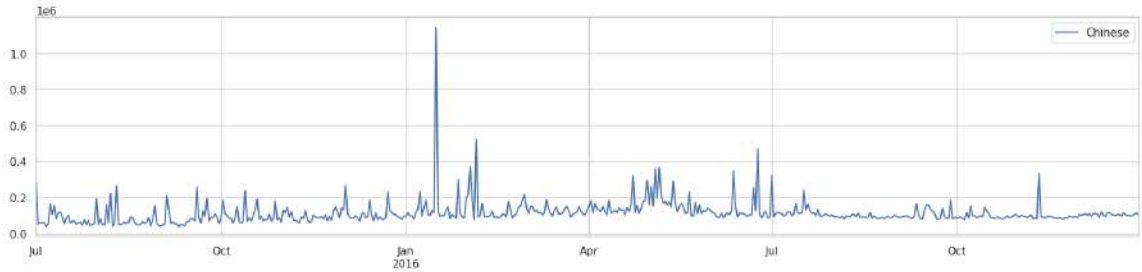
In [ ]:

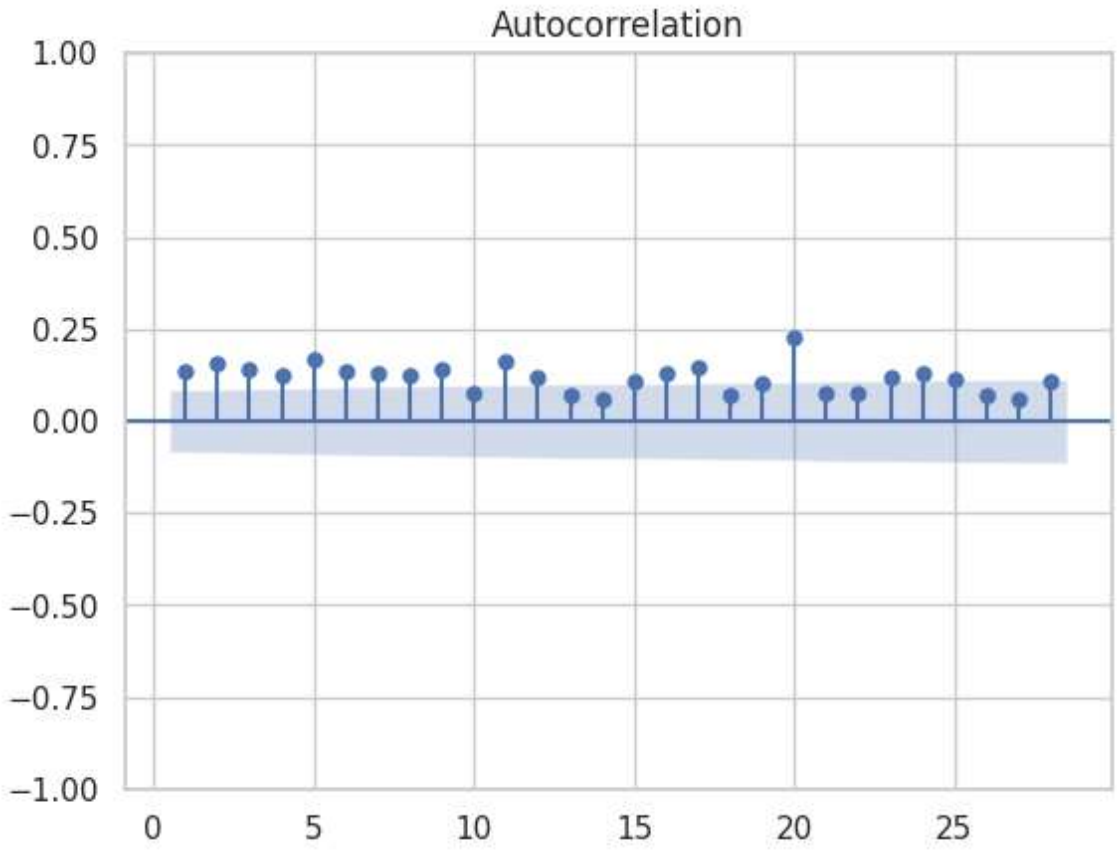
```

```

In [ ]:

```
plt.rcParams["figure.dpi"] = 100
hits.iloc[:,0:1].plot(figsize=(20,4))
sgt.plot_acf(np.array(hits.iloc[:,0:1]),
             ax=None,
             lags=None,
             alpha=0.05,
             use_vlines=True,
             unbiased=False,
             fft=False,
             missing='none',
             title='Autocorrelation',
             zero=False,  # Not including the 1st term as its acf w.r.t. itself will alwa
             vlines_kwargs=None)
plt.show()
```

```
<ipython-input-39-ade8dd1d527b>:3: FutureWarning: the 'unbiased'' keyword
is deprecated, use 'adjusted' instead.
  sgt.plot_acf(np.array(hits.iloc[:,0:1]),
```



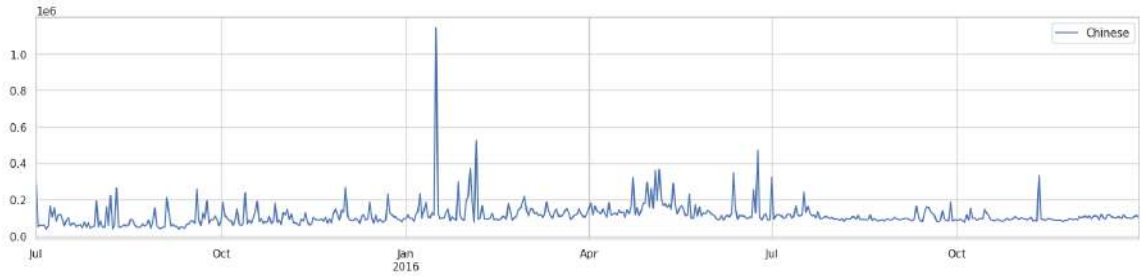


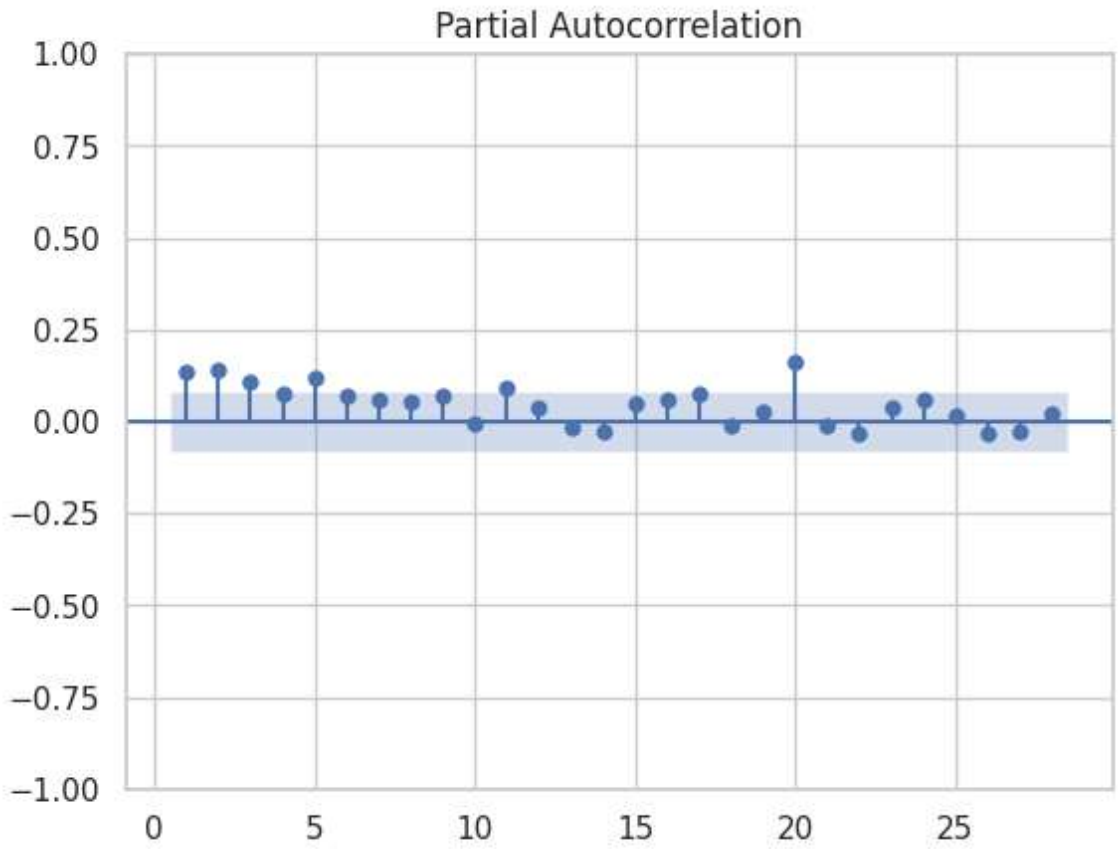
In [ ]:

```
```

In [ ]:

```
plt.rcParams["figure.dpi"] = 100
hits.iloc[:,0:1].plot(figsize=(20,4))
sgt.plot_pacf(np.array(hits.iloc[:,0:1]),
              ax=None,
              lags=None,
              alpha=0.05,
              method='ols',
              use_vlines=True,
              title='Partial Autocorrelation',
              zero=False,    # Not including the 1st term as its pacf w.r.t. itself will a
              vlines_kwargs=None)
plt.show()
```





Test-Train Split

In [ ]:

```
brk = 0.8
data_split = int(len(hits)*brk)
data_split
```

Out[79]:

```
440
```

In [ ]:

```
X, y = hits.iloc[:data_split,:], hits.iloc[data_split:,:]
```

In [ ]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)

scaled_X = scaler.transform(X)
scaled_y = scaler.transform(y)
```

In [ ]:

```
print(scaled_X.max(), scaled_X.min())
print(scaled_y.max(), scaled_y.min())
```

```
1.0 0.0
0.6730716086514008 0.0
```

In [ ]:

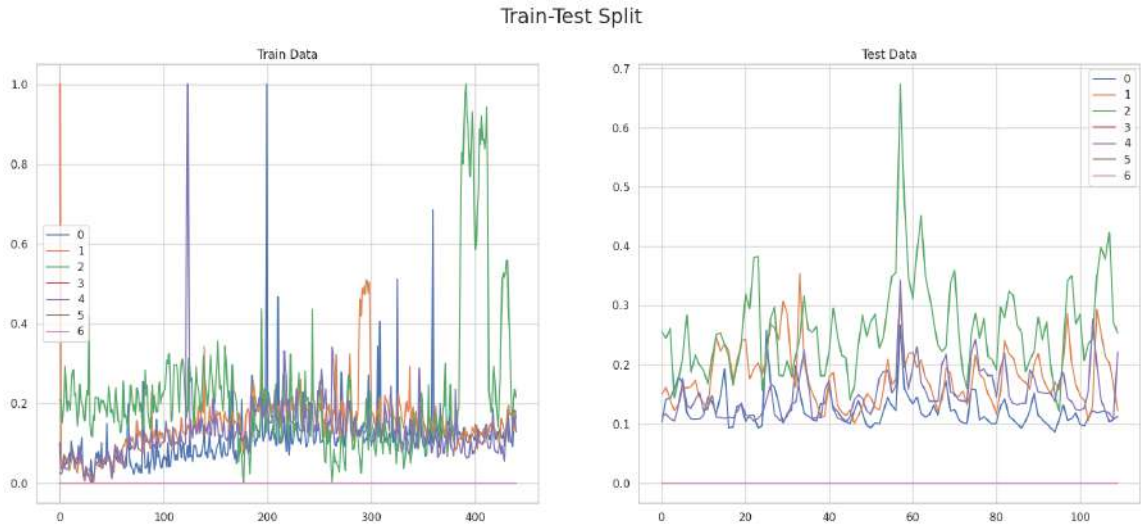
```
X_df = (pd.DataFrame(scaled_X))
y_df = (pd.DataFrame(scaled_y))
```

In [ ]:

```
```

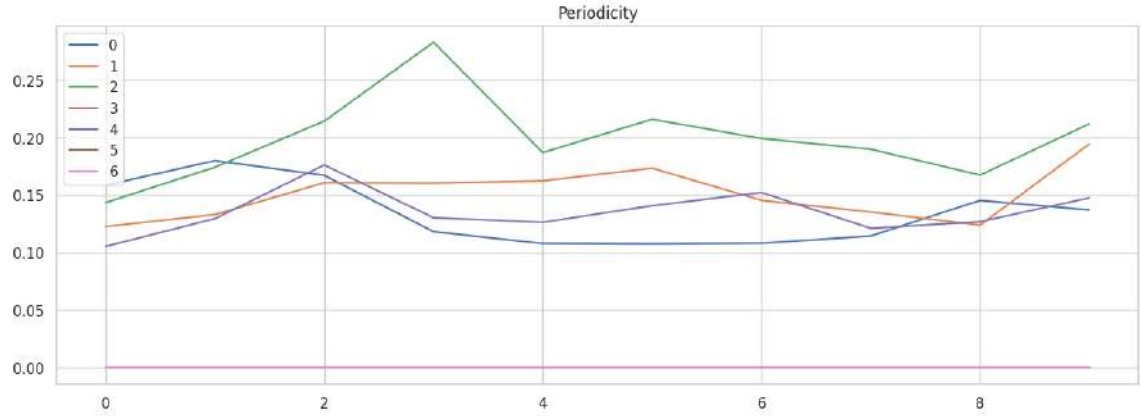
In [ ]:

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20,8), dpi=100)
plt.suptitle('Train-Test Split', fontsize=20)
X_df.plot(ax=axes[0], title='Train Data')
y_df.plot(ax=axes[1], title='Test Data')

plt.show()
```



In [ ]:

```
pd.DataFrame(scaled_y[3:13,:]).plot(figsize=(15,5), title='Periodicity')
plt.show()
```



In [ ]:

```

```

ARIMA Model

In [ ]:

```
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(train.values, order=(3, 1, 0))
model_fit = model.fit()
model_fit.summary()
```

Out[15]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	80
Model:	ARIMA(3, 1, 0)	**Log Likelihood**	-1338.979
Date:	Sat, 14 Jan 2023	**AIC**	2685.958
Time:	06:12:56	**BIC**	2695.436
Sample:	0	**HQIC**	2689.756
	- 80		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.0510	0.098	0.520	0.603	-0.141	0.243
ar.L2	-0.0218	0.040	-0.543	0.587	-0.101	0.057
ar.L3	-0.0405	0.049	-0.835	0.404	-0.136	0.055
sigma2	2.776e+13	2.92e-16	9.5e+28	0.000	2.78e+13	2.78e+13

Ljung-Box (L1) (Q):	0.17	**Jarque-Bera (JB):**	1.22
Prob(Q):	0.68	**Prob(JB):**	0.54
Heteroskedasticity (H):	0.86	**Skew:**	0.26
Prob(H) (two-sided):	0.71	**Kurtosis:**	3.33

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 1.37e+43.
Standard errors may be unstable.

In [ ]:

```
import itertools
p = range(0, 3)
d = range(1, 3)
q = range(0, 3)

pdq = list(itertools.product(p, d, q))

aic = []
for i in pdq:
    model = ARIMA(train_df.values, order=(i))
    model_fit = model.fit()
    print(f'ARIMA({i}) AIC : {round(model_fit.aic,2)}')
    aic.append(round(model_fit.aic, 2))
```

```
ARIMA((0, 1, 0)) AIC : 2682.88
ARIMA((0, 1, 1)) AIC : 2683.13
ARIMA((0, 1, 2)) AIC : 2688.78
ARIMA((0, 2, 0)) AIC : 2694.53
ARIMA((0, 2, 1)) AIC : 2696.87
ARIMA((0, 2, 2)) AIC : 2702.79
ARIMA((1, 1, 0)) AIC : 2683.11
ARIMA((1, 1, 1)) AIC : 2684.95
ARIMA((1, 1, 2)) AIC : 2714.18
ARIMA((1, 2, 0)) AIC : 2696.3

/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/statespace/sarimax.
py:966: UserWarning: Non-stationary starting autoregressive parameters fou
nd. Using zeros as starting parameters.
  warn('Non-stationary starting autoregressive parameters'
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/statespace/sarimax.
py:978: UserWarning: Non-invertible starting MA parameters found. Using ze
ros as starting parameters.
  warn('Non-invertible starting MA parameters found.'

ARIMA((1, 2, 1)) AIC : 2684.19
ARIMA((1, 2, 2)) AIC : 2687.43
ARIMA((2, 1, 0)) AIC : 2684.84
ARIMA((2, 1, 1)) AIC : 2685.41
ARIMA((2, 1, 2)) AIC : 2656.23
ARIMA((2, 2, 0)) AIC : 2696.85
ARIMA((2, 2, 1)) AIC : 2683.57
ARIMA((2, 2, 2)) AIC : 2694.85

/opt/conda/lib/python3.7/site-packages/statsmodels/base/model.py:606: Conv
ergenceWarning: Maximum Likelihood optimization failed to converge. Check
mle_retvals
  ConvergenceWarning)
```

In [ ]:

```
# ARIMA((2, 1, 2)) is best aic

model = ARIMA(train_df.values, order=(2, 1, 2))
model_fit = model.fit()
model_fit.summary()
```

Out[17]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	80
Model:	ARIMA(2, 1, 2)	**Log Likelihood**	-1323.117
Date:	Sat, 14 Jan 2023	**AIC**	2656.235
Time:	06:12:58	**BIC**	2668.082
Sample:	0	**HQIC**	2660.981
	- 80		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.2645	0.012	107.294	0.000	1.241	1.288
ar.L2	-0.9961	0.012	-86.563	0.000	-1.019	-0.974
ma.L1	-1.2507	0.050	-25.086	0.000	-1.348	-1.153
ma.L2	0.9943	0.076	13.122	0.000	0.846	1.143
sigma2	1.444e+13	3.04e-15	4.75e+27	0.000	1.44e+13	1.44e+13

Ljung-Box (L1) (Q):	4.92	**Jarque-Bera (JB):**	2.96
Prob(Q):	0.03	**Prob(JB):**	0.23
Heteroskedasticity (H):	0.44	**Skew:**	-0.12
Prob(H) (two-sided):	0.04	**Kurtosis:**	3.92

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 7.16e+42.
Standard errors may be unstable.

In [ ]:

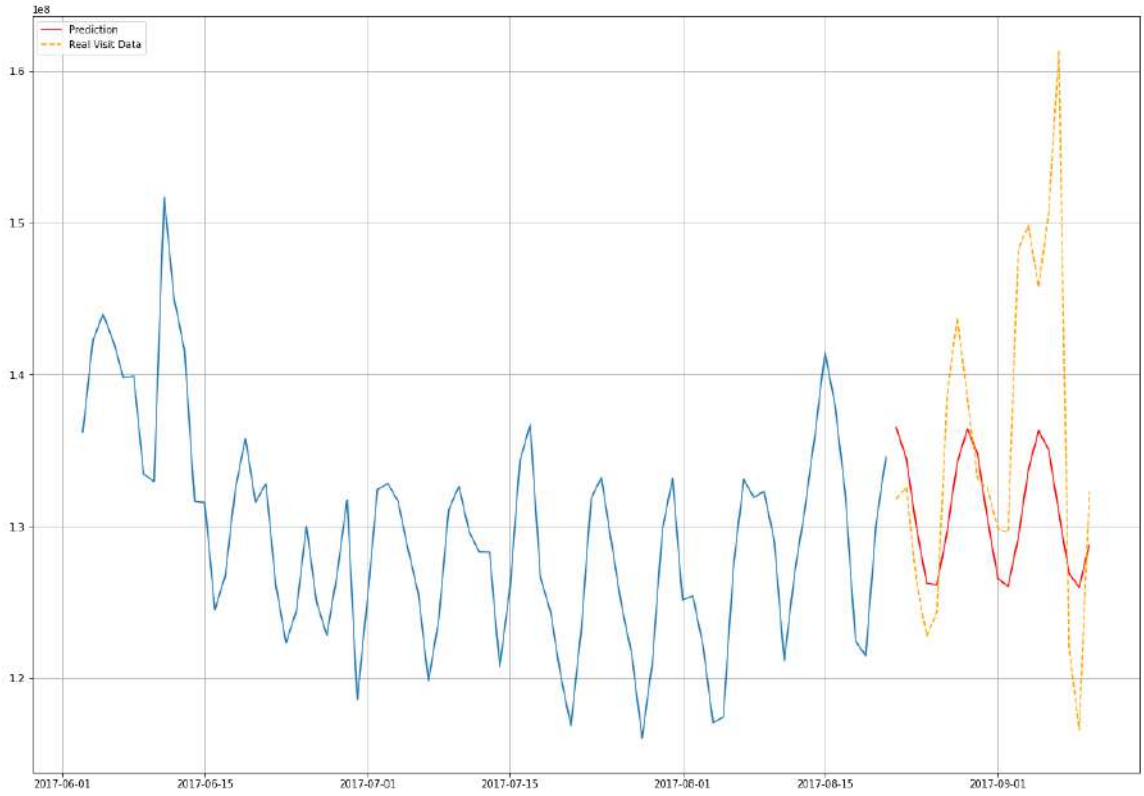
```
prediction = model_fit.forecast(len(test_df))
prediction_value = prediction
prediction_index = list(test_df.index)
```

In [ ]:

```
```

In [ ]:

```
fig, axes = plt.subplots(figsize=(20, 14))


axes.plot(train_df)
axes.plot(prediction_index, prediction_value,color='r', label='Prediction')
axes.plot(test_df, linestyle='--', color='orange', label='Real Visit Data')
axes.legend(loc='upper left')
axes.grid()
```



In [ ]:

```
```

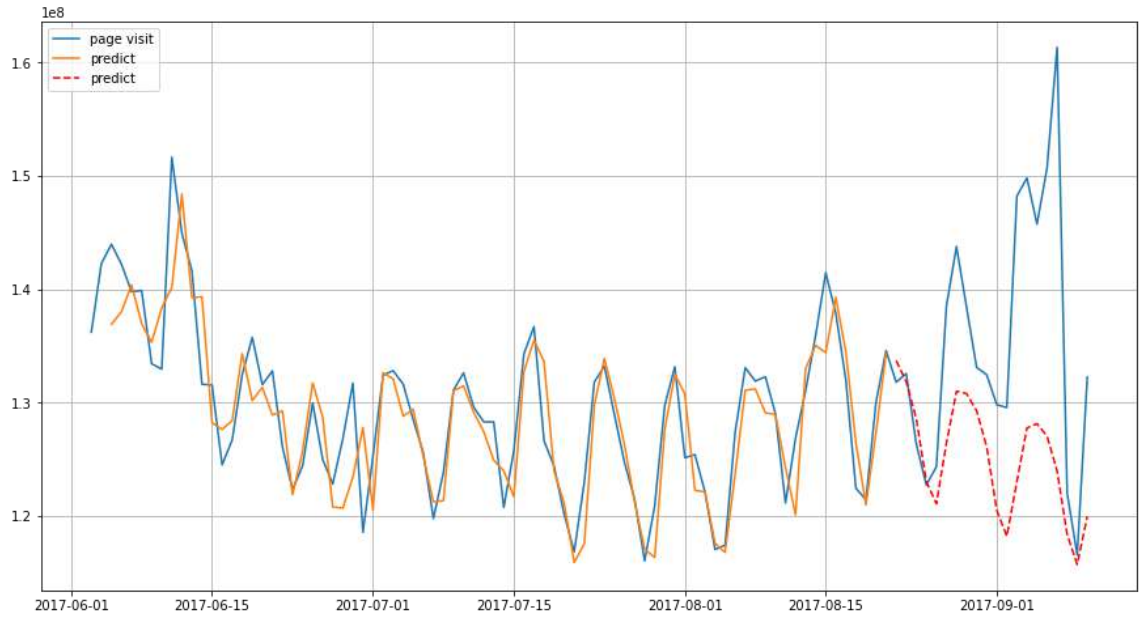
In [ ]:

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(train_df, order=(7,1,7))
model_fit = model.fit()

fig, axes = plt.subplots(figsize=(15, 8))
axes.plot(df, label='page visit')
axes.plot(model_fit.fittedvalues[2:], label='predict')
axes.plot(pd.DataFrame(model_fit.forecast(steps = len(test_df))), label='predict', color

axes.grid()
axes.legend()
```

```
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:4
71: ValueWarning: No frequency information was provided, so inferred frequ
ency D will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:4
71: ValueWarning: No frequency information was provided, so inferred frequ
ency D will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:4
71: ValueWarning: No frequency information was provided, so inferred frequ
ency D will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/statespace/sarimax.
py:966: UserWarning: Non-stationary starting autoregressive parameters fou
nd. Using zeros as starting parameters.
  warn('Non-stationary starting autoregressive parameters'
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/statespace/sarimax.
py:978: UserWarning: Non-invertible starting MA parameters found. Using ze
ros as starting parameters.
  warn('Non-invertible starting MA parameters found.'
```

Out[20]:

```
<matplotlib.legend.Legend at 0x7f8830d35dd0>
```



LSTM Model

In [ ]:

```
print(scaled_X.shape)
print(scaled_y.shape)
print('No. of features = '+str(scaled_X.shape[1]))
print('No. of train instances = '+str(scaled_X.shape[0]))
print('No. of test instances = '+str(scaled_y.shape[0]))
```

```
(440, 7)
(110, 7)
No. of features = 7
No. of train instances = 440
No. of test instances = 110
```

In [ ]:

```
length = 7
batch = 1

n_features = scaled_X.shape[1]
n_features
```

Out[87]:

```
7
```

In [ ]:

```
generator = TimeseriesGenerator(data = scaled_X,
                                targets = scaled_X,
                                length = length,
                                sampling_rate=1,
                                stride=1,
                                start_index=0,
                                end_index=None,
                                shuffle=False,
                                reverse=False,
                                batch_size=batch)
```

In [ ]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM

import datetime, os

model = Sequential(layers=None, name="LSTM_Model")

model.add(LSTM( units = 400,activation='tanh',
                input_shape=( length, n_features),
                recurrent_activation='sigmoid',
                use_bias=True,
                kernel_initializer='glorot_uniform',
                recurrent_initializer='orthogonal',
                bias_initializer='zeros',
                unit_forget_bias=True,
                kernel_regularizer=None,
                recurrent_regularizer=None,
                bias_regularizer=None,
                activity_regularizer=None,
                kernel_constraint=None,
                recurrent_constraint=None,
                bias_constraint=None,
                dropout=0.0,
                recurrent_dropout=0.0,
                implementation=2,
                return_sequences=True,
                return_state=False,
                go_backwards=False,
                stateful=False,
                time_major=False,
                unroll=False
            ) )
model.add(LSTM(units = 500, return_sequences=True))

model.add(LSTM(units = 500, return_sequences=False))

model.add(Dense(700, activation="relu", name="layer1"))

model.add(Dense(100, activation="relu", name="layer2"))


model.add(Dense( units = n_features,
                activation='relu',
                use_bias=True,
                kernel_initializer='glorot_uniform',
                bias_initializer='zeros',
                kernel_regularizer=None,
                bias_regularizer=None,
                activity_regularizer=None,
                kernel_constraint=None,
                bias_constraint=None))

model.compile(optimizer='adam', loss='mse')
```

In [ ]:

```
model.summary()
```

```
Model: "LSTM_Model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 7, 400)            652800

 lstm_4 (LSTM)               (None, 7, 500)            1802000

 lstm_5 (LSTM)               (None, 500)               2002000

 layer1 (Dense)              (None, 700)               350700

 layer2 (Dense)              (None, 100)               70100

 dense_1 (Dense)             (None, 7)                 707

=================================================================
Total params: 4,878,307
Trainable params: 4,878,307
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss',
                           min_delta=0,
                           patience=20,
                           verbose=1,
                           mode='auto',
                           baseline=None,

                           restore_best_weights=False)
```

In [ ]:

```
validation_generator = TimeseriesGenerator(scaled_y,scaled_y, length=length, batch_size=
```

In [ ]:

```
f%%time
history = model.fit(generator,
                    steps_per_epoch=None,
                    epochs=500,
                    verbose=1,
                    callbacks=[early_stop],
                    validation_data = validation_generator,
                    validation_steps=None,
                    validation_freq=1,
                    class_weight=None,
                    max_queue_size=10,
                    workers=1,
                    use_multiprocessing=False,
                    shuffle=True,
                    initial_epoch=0)
```

```
Epoch 1/500
433/433 [==============================] - 66s 135ms/step - loss: 0.0126 -
val_loss: 0.0025
Epoch 2/500
433/433 [==============================] - 57s 131ms/step - loss: 0.0056 -
val_loss: 0.0024
Epoch 3/500
433/433 [==============================] - 57s 131ms/step - loss: 0.0048 -
val_loss: 0.0025
Epoch 4/500
433/433 [==============================] - 56s 130ms/step - loss: 0.0031 -
val_loss: 0.0019
Epoch 5/500
433/433 [==============================] - 57s 131ms/step - loss: 0.0029 -
val_loss: 0.0016
Epoch 6/500
433/433 [==============================] - 55s 128ms/step - loss: 0.0028 -
val_loss: 0.0016
Epoch 7/500
433/433 [==============================] - 57s 132ms/step - loss: 0.0026 -
val_loss: 0.0021
Epoch 8/500
433/433 [==============================] - 58s 134ms/step - loss: 0.0031 -
val_loss: 0.0014
Epoch 9/500
433/433 [==============================] - 57s 132ms/step - loss: 0.0053 -
val_loss: 0.0027
Epoch 10/500
433/433 [==============================] - 58s 134ms/step - loss: 0.2044 -
val_loss: 0.0023
Epoch 11/500
433/433 [==============================] - 60s 139ms/step - loss: 0.0072 -
val_loss: 0.0017
Epoch 12/500
433/433 [==============================] - 59s 135ms/step - loss: 0.0071 -
val_loss: 0.0024
Epoch 13/500
433/433 [==============================] - 61s 140ms/step - loss: 0.0072 -
val_loss: 0.0017
Epoch 14/500
433/433 [==============================] - 58s 135ms/step - loss: 0.0071 -
val_loss: 0.0028
Epoch 15/500
433/433 [==============================] - 60s 139ms/step - loss: 0.0070 -
val_loss: 0.0024
Epoch 16/500
433/433 [==============================] - 59s 137ms/step - loss: 0.0071 -
val_loss: 0.0023
Epoch 17/500
433/433 [==============================] - 60s 139ms/step - loss: 0.0071 -
val_loss: 0.0021
Epoch 18/500
433/433 [==============================] - 59s 136ms/step - loss: 0.0071 -
val_loss: 0.0020
Epoch 19/500
433/433 [==============================] - 59s 137ms/step - loss: 0.0070 -
val_loss: 0.0025
Epoch 20/500
433/433 [==============================] - 58s 133ms/step - loss: 0.0071 -
val_loss: 0.0020
Epoch 21/500
```

```
433/433 [==============================] - 59s 136ms/step - loss: 0.0071 -
val_loss: 0.0023
Epoch 22/500
433/433 [==============================] - 57s 132ms/step - loss: 0.0071 -
val_loss: 0.0022
Epoch 23/500
433/433 [==============================] - 58s 134ms/step - loss: 0.0070 -
val_loss: 0.0019
Epoch 24/500
433/433 [==============================] - 58s 133ms/step - loss: 0.0070 -
val_loss: 0.0019
Epoch 25/500
433/433 [==============================] - 59s 137ms/step - loss: 0.0072 -
val_loss: 0.0029
Epoch 26/500
433/433 [==============================] - 58s 133ms/step - loss: 0.0071 -
val_loss: 0.0021
Epoch 27/500
433/433 [==============================] - 58s 133ms/step - loss: 0.0070 -
val_loss: 0.0021
Epoch 28/500
433/433 [==============================] - 60s 139ms/step - loss: 0.0070 -
val_loss: 0.0020
Epoch 28: early stopping
CPU times: user 40min 13s, sys: 53.3 s, total: 41min 7s
Wall time: 32min 30s
```

In [ ]:

```
print(history.history.keys())
```

```
dict_keys(['loss', 'val_loss'])
```

In [ ]:

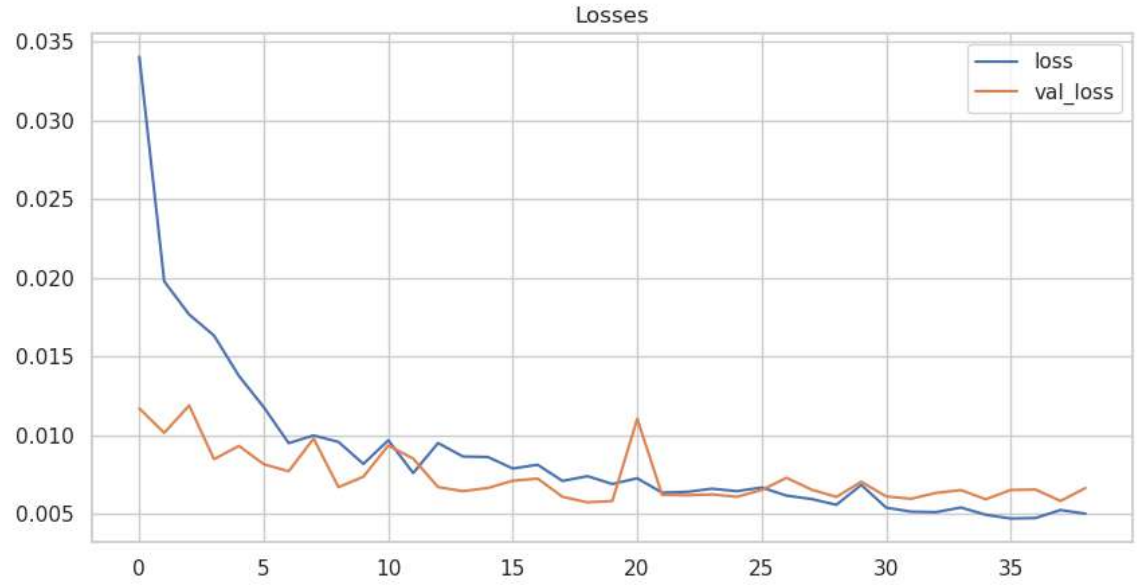
```
losses = pd.DataFrame(model.history.history)
```

In [ ]:

```
```

In [ ]:

```
plt.rcParams["figure.dpi"] = 100
losses.plot(figsize=(10,5))
plt.title('Losses')
plt.show()
```



In [ ]:

```
%%time
test_predictions = []

first_eval_batch = scaled_X[-length:]
current_batch = first_eval_batch.reshape((1, length, n_features))

for i in range(len(scaled_y)):

    current_pred = model.predict(current_batch,verbose=0)[0]

    test_predictions.append(current_pred)

    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],axis=1)
```

```
CPU times: user 12.5 s, sys: 218 ms, total: 12.7 s
Wall time: 13.9 s
```

In [ ]:

```
print(np.array(test_predictions).shape)
print(scaled_y.shape)
```

```
(110, 7)
(110, 7)
```

In [ ]:

```
print(np.array(test_predictions).max(), np.array(test_predictions).min())
print(scaled_y.max(), scaled_y.min())
```

```
0.22898214 0.0
0.6730716086514008 0.0
```

In [ ]:

```
true_predictions = scaler.inverse_transform(test_predictions)
print(true_predictions.shape)
```

```
(110, 7)
```

In [ ]:

```
print(np.array(true_predictions).max(), np.array(true_predictions).min())
print(np.array(y).max(), np.array(y).min())
```

```
93757903.4251278 0.0
145476988.0 0.0
```

In [ ]:

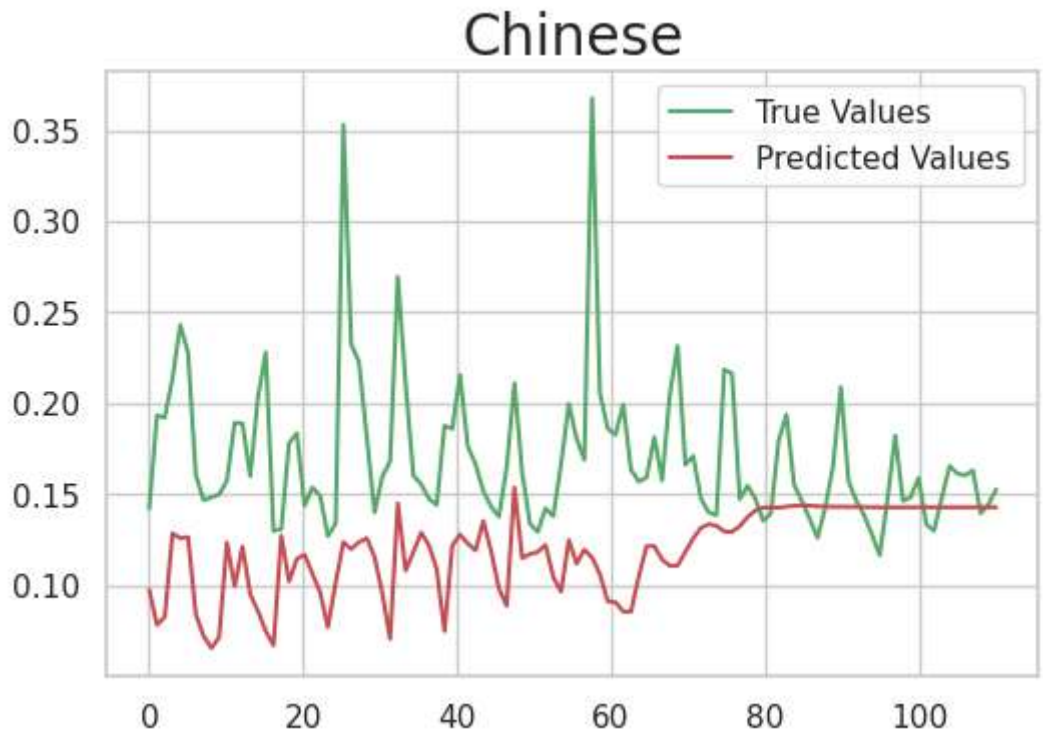
```
t_l = len(scaled_y)
t_l
```

Out[109]:

```
110
```

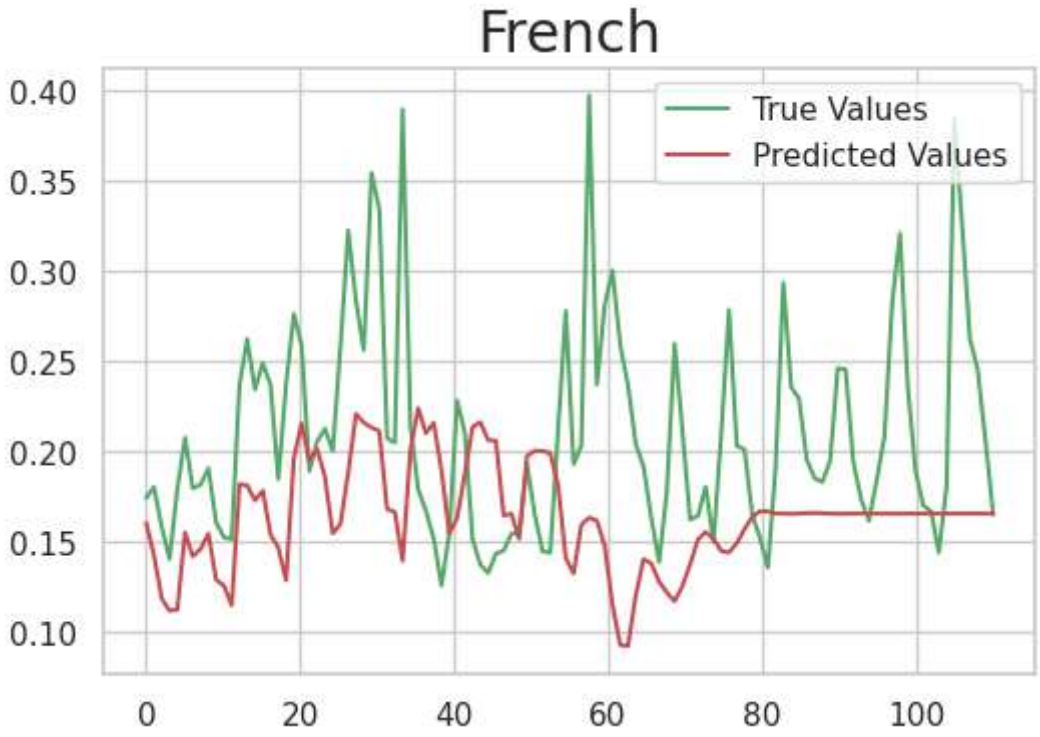
In [ ]:

```
plt.figure(dpi=100)
plt.plot(np.linspace(0,t_l,t_l), scaled_y[:,0:1] , label='True Values',c='g')
plt.plot(np.linspace(0,t_l,t_l), np.array(test_predictions)[:,0:1], label='Predicted Val
plt.title(hits.columns[0], fontsize=20)
plt.legend()
plt.show()
```



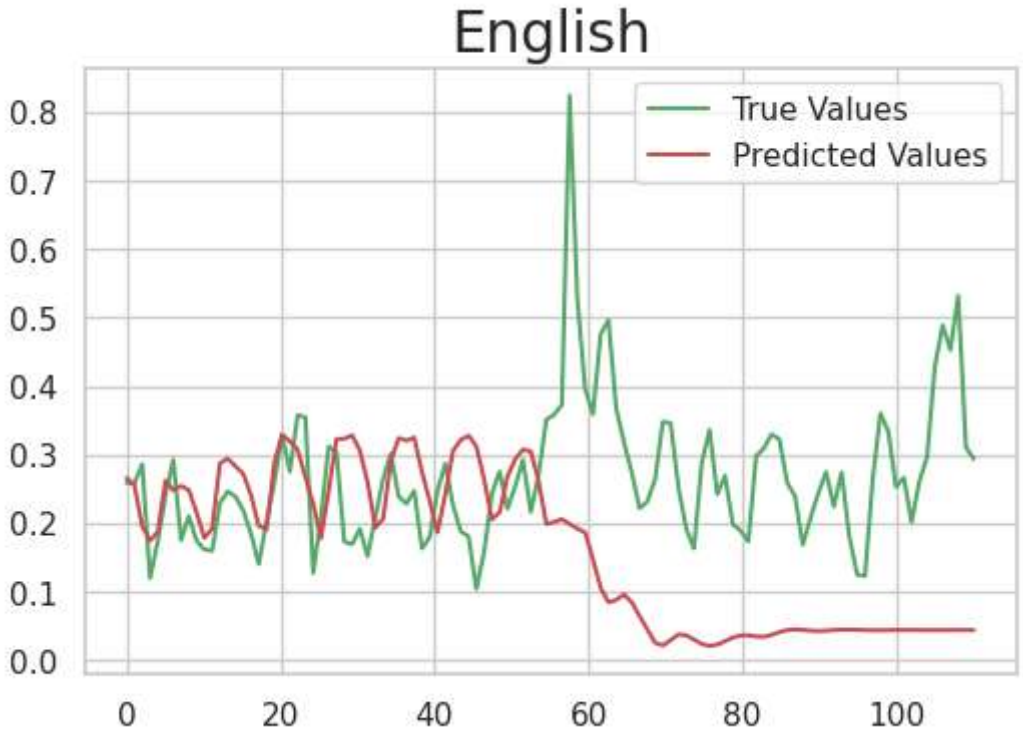
In [ ]:

```
plt.plot(np.linspace(0,t_l,t_l), scaled_y[:,1:2] , label='True Values',c='g')
plt.plot(np.linspace(0,t_l,t_l), np.array(test_predictions)[:,1:2], label='Predicted Val
plt.title(hits.columns[1], fontsize=20)
plt.legend()
plt.show()
```



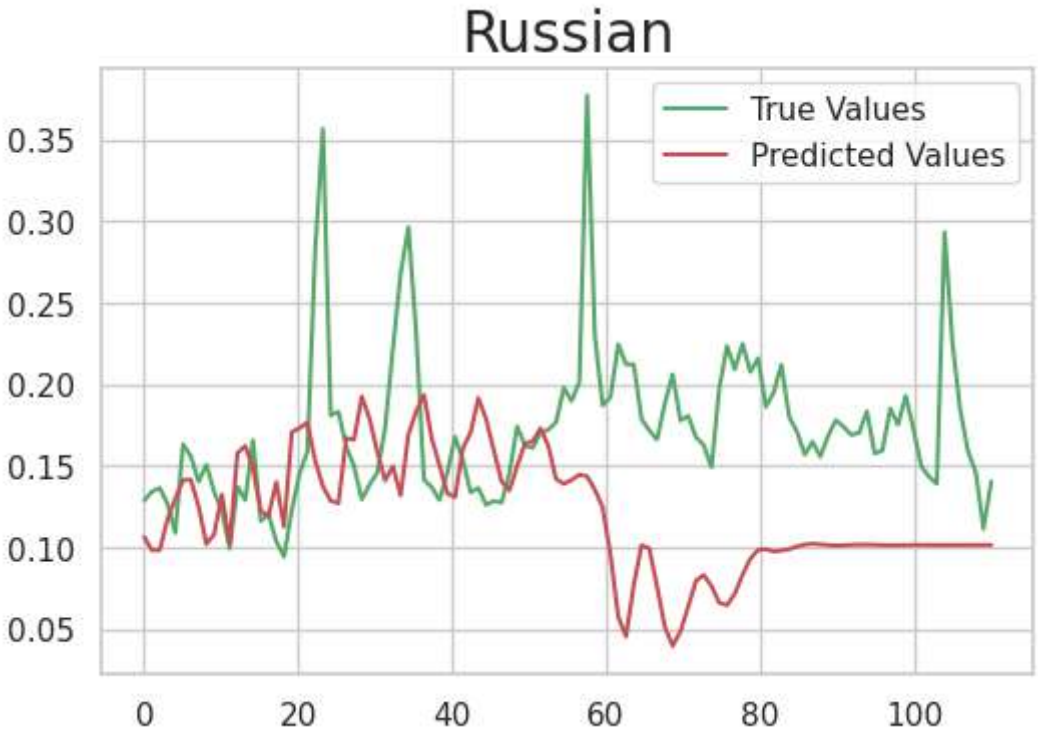
In [ ]:

```
plt.plot(np.linspace(0,t_l,t_l), scaled_y[:,2:3] , label='True Values',c='g')
plt.plot(np.linspace(0,t_l,t_l), np.array(test_predictions)[:,2:3], label='Predicted Val
plt.title(hits.columns[2], fontsize=20)
plt.legend()
plt.show()
```



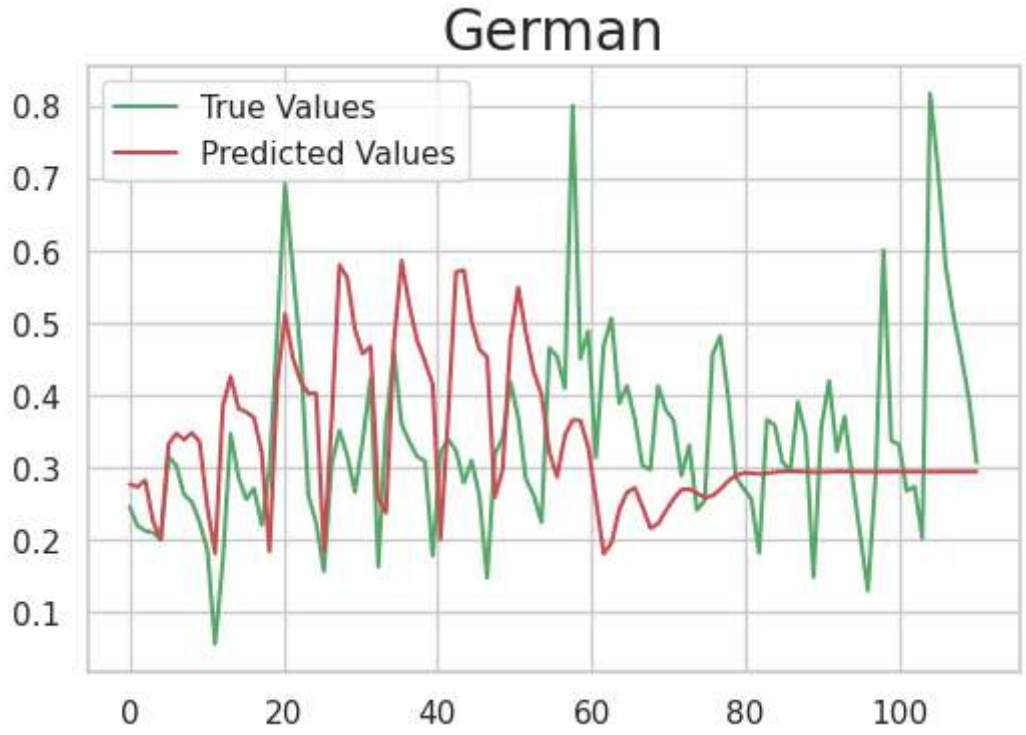
In [ ]:

```
plt.plot(np.linspace(0,t_l,t_l), scaled_y[:,3:4] , label='True Values',c='g')
plt.plot(np.linspace(0,t_l,t_l), np.array(test_predictions)[:,3:4], label='Predicted Val
plt.title(hits.columns[3], fontsize=20)
plt.legend()
plt.show()
```



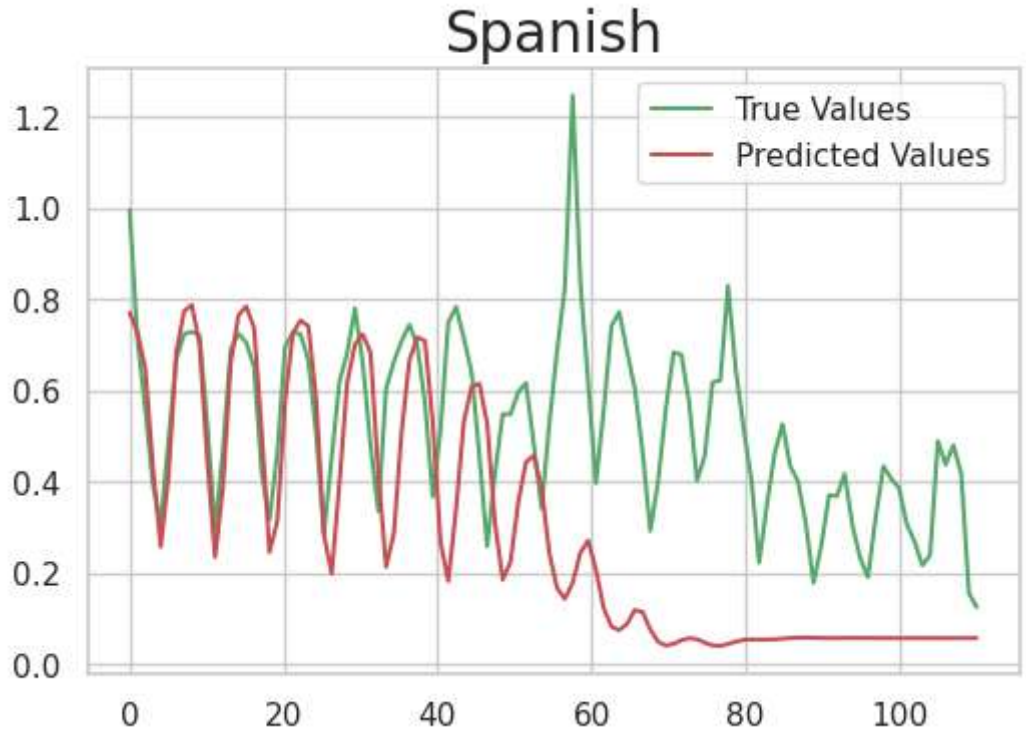
In [ ]:

```
plt.plot(np.linspace(0,t_l,t_l), scaled_y[:,4:5] , label='True Values',c='g')
plt.plot(np.linspace(0,t_l,t_l), np.array(test_predictions)[:,4:5], label='Predicted Val
plt.title(hits.columns[4], fontsize=20)
plt.legend()
plt.show()
```



In [ ]:

```
plt.plot(np.linspace(0,t_l,t_l), scaled_y[:,6:7] , label='True Values',c='g')
plt.plot(np.linspace(0,t_l,t_l), np.array(test_predictions)[:,6:7], label='Predicted Val
plt.title(hits.columns[6], fontsize=20)
plt.legend()
plt.show()
```



In [ ]:

```
plt.plot(np.linspace(0,t_l,t_l), scaled_y[:,5:6] , label='True Values',c='g')
plt.plot(np.linspace(0,t_l,t_l), np.array(test_predictions)[:,5:6], label='Predicted Val
plt.title(hits.columns[5], fontsize=20)
plt.legend()
plt.show()
```