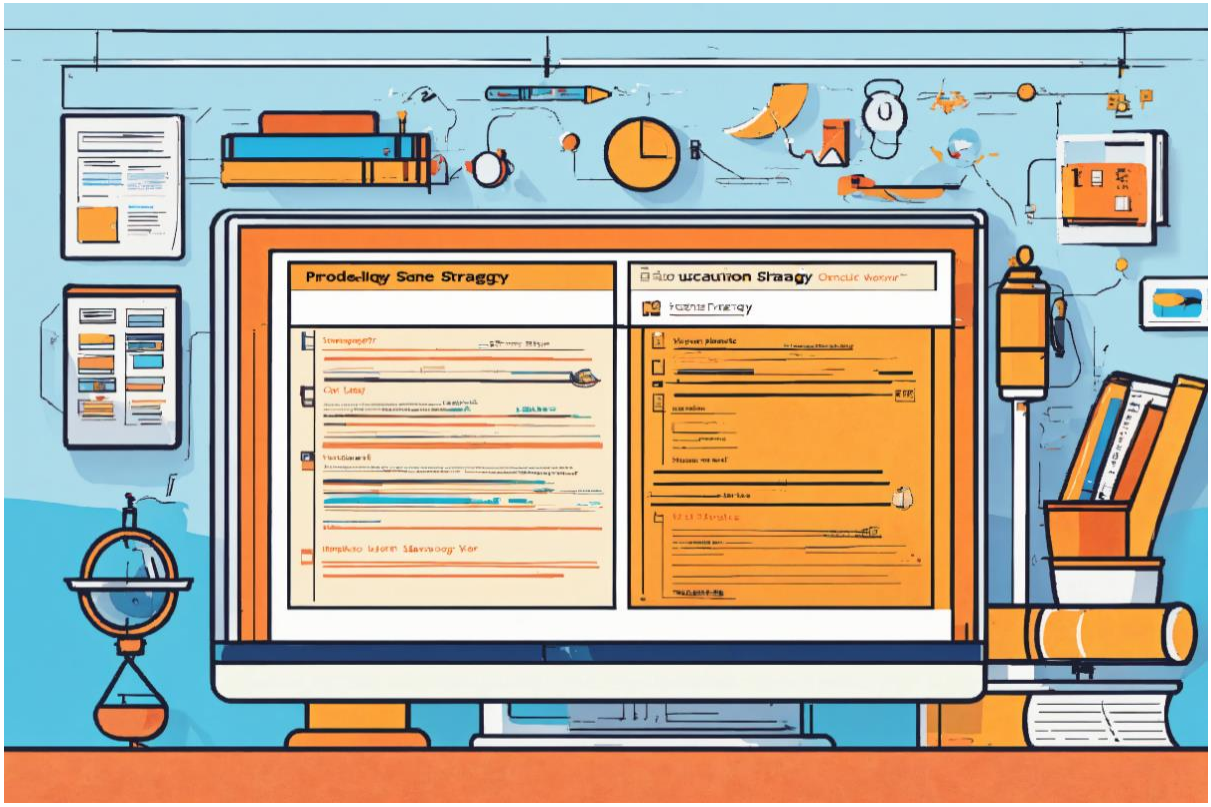
AdEase Time series

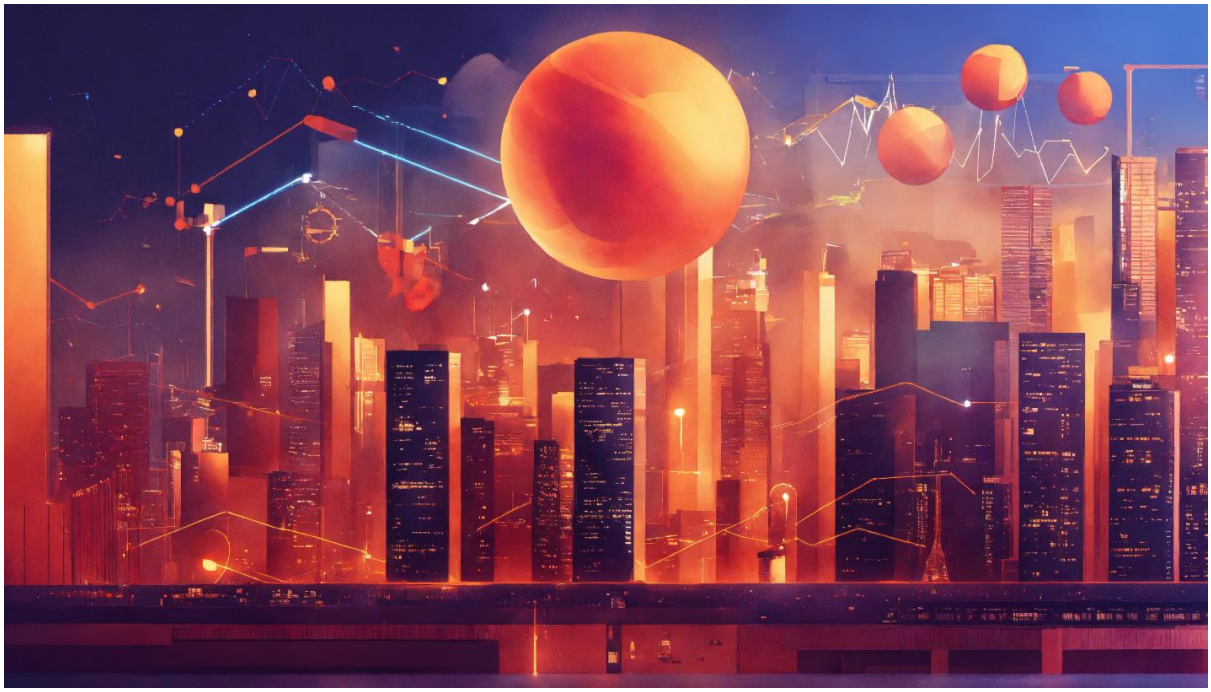
Ad Ease is an ads and marketing based company helping businesses elicit maximum clicks @ minimum cost. Ad Ease is an ad infrastructure to help businesses promote themselves easily, effectively, and economically. The interplay of 3 AI modules - Design, Dispense, and Decipher, come together to make it this an end-to-end 3 step process digital advertising solution for all.

You are working in the Data Science team of Ad ease trying to understand the per page view report for different Wikipedia pages for 550 days, and forecasting the number of views so that you can predict and optimize the ad placement for your clients. You are provided with the data of 145k Wikipedia pages and daily view count for each of them. Your clients belong to different regions and need data on how their ads will perform on pages in different languages.

Problem Statement



About Dataset



There are 2 dataset in this project

1. train_1.csv

- **Rows: Each row represents a unique article.**
- **Columns: The first column typically contains the page name, and subsequent columns represent dates.**
- **Values: The values in the cells are the number of visits the article received on the corresponding date.**

The page name has a specific format:

- **SPECIFIC NAME: The unique identifier for the article.**
- **LANGUAGE: The language edition of Wikipedia (e.g., "en" for English, "es" for Spanish).**
- **wikipedia.org: The main domain indicating that the data is from Wikipedia.**
- **ACCESS TYPE: The type of device used to access the page (e.g., "desktop", "mobile-web").**
- **ACCESS ORIGIN: The source of the request (e.g., "all-access", "spider").**

Page Name Format:

The page name has a specific format:

- **SPECIFIC NAME: The unique identifier for the article.**
- **LANGUAGE: The language edition of Wikipedia (e.g., "en" for English, "es" for Spanish).**
- **wikipedia.org: The main domain indicating that the data is from Wikipedia.**
- **ACCESS TYPE: The type of device used to access the page (e.g., "desktop", "mobile-web").**
- **ACCESS ORIGIN: The source of the request (e.g., "all-access", "spider").**

This indicates:

- **Article: Python (programming language)**
- **Language: English**
- **Domain: Wikipedia**
- **Access type: All devices**
- **Access origin: All agents (including both browser and spider requests)**

2. Exog_Campaign_eng.csv

- **Rows: Each row corresponds to a date.**
- **Columns: Typically, there are two columns:**

- **Date: The specific date.**

Purpose of this dataset is that this file indicates whether there was a campaign or significant event on specific dates that could affect the number of visits to English Wikipedia pages.

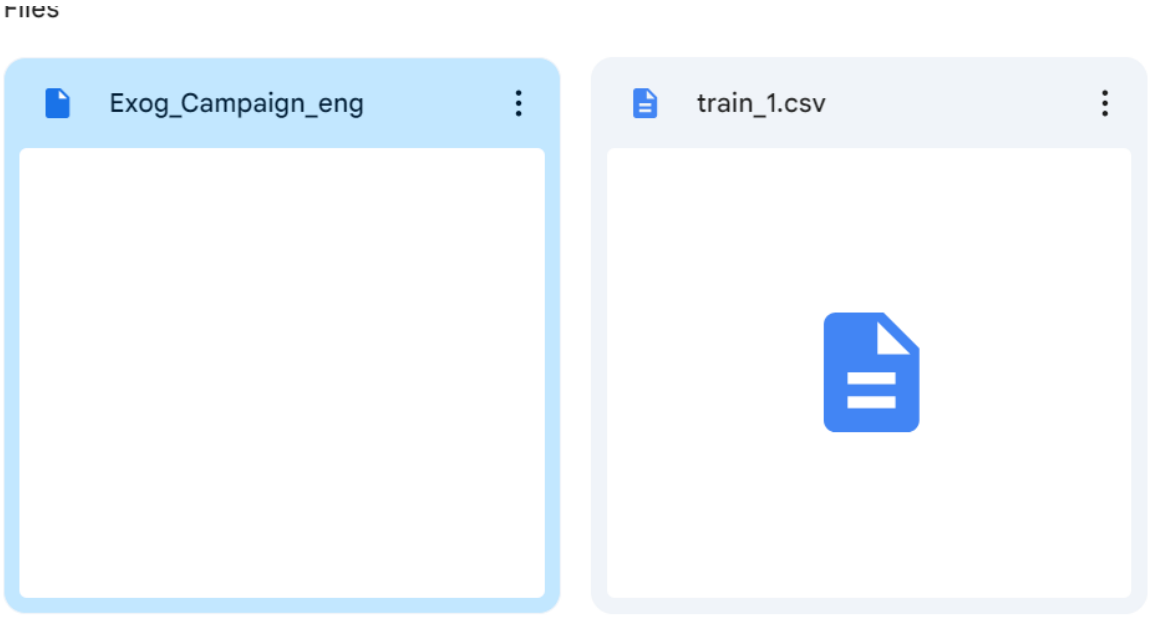
1: There was a campaign or significant event on that date.

0: There was no campaign or significant event on that date

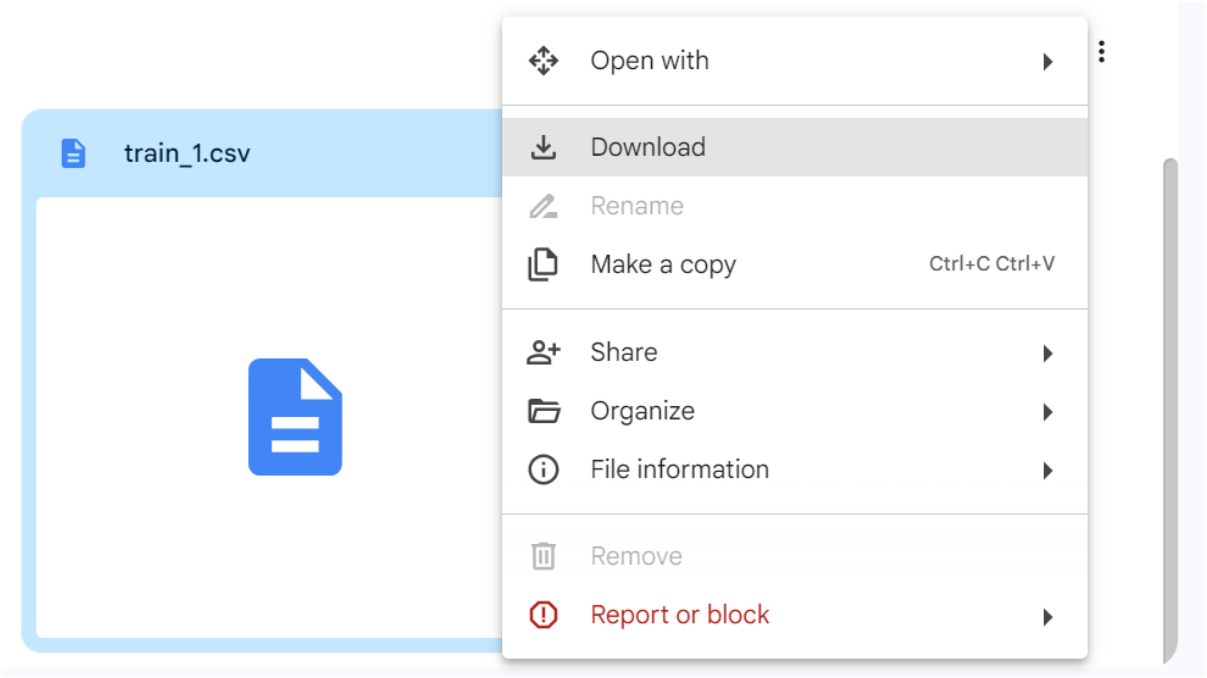
Steps:

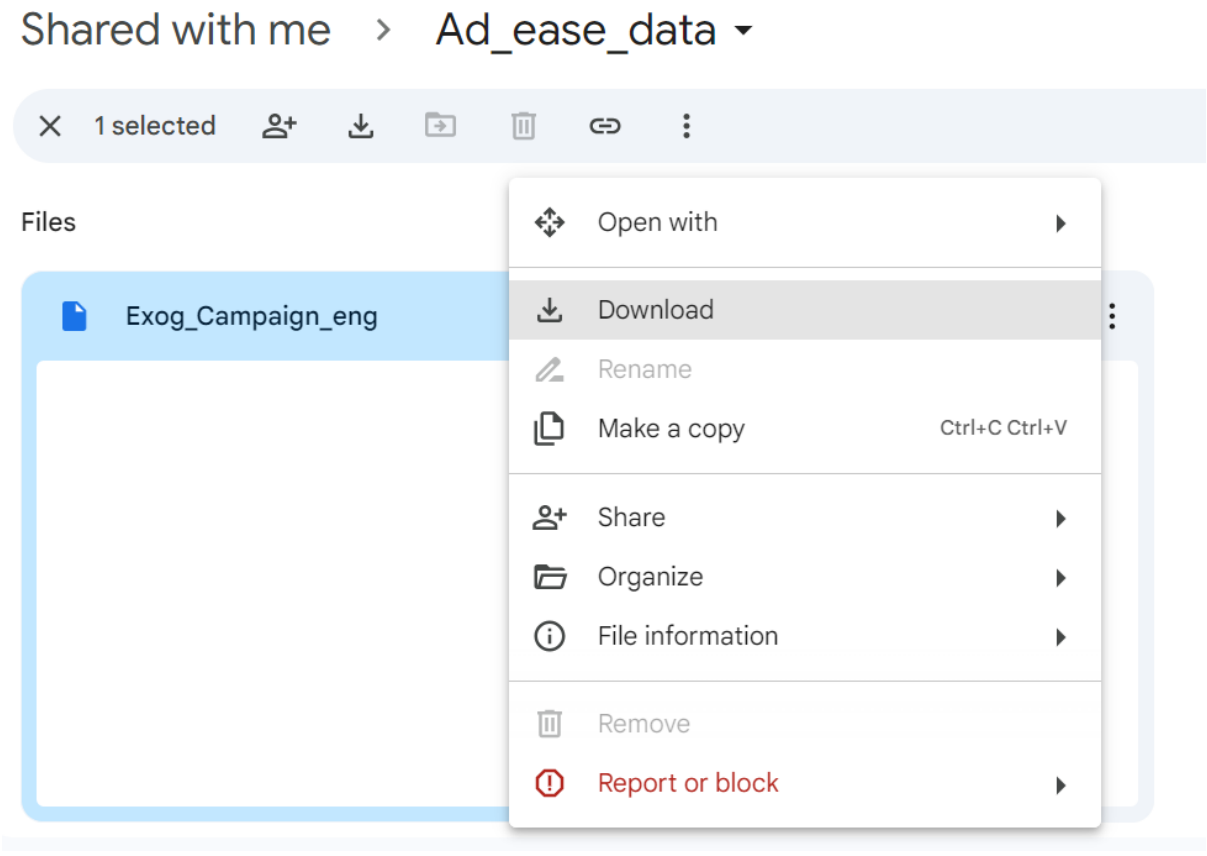
Step 1: click this link and save to your drive or on your computer

https://drive.google.com/drive/folders/1mdgQscjqnCtdg7LGltomyK0abN6lcHBb

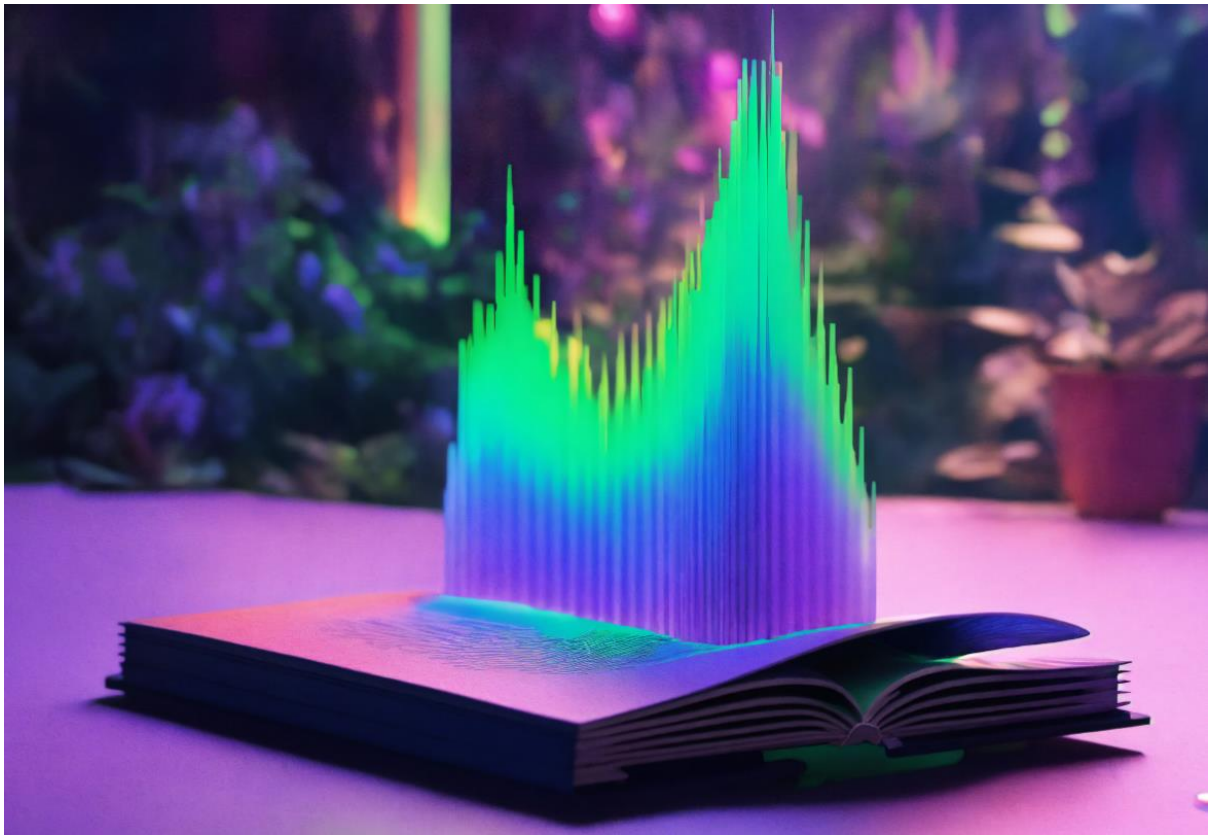


Click the 3 dot and Download both the file


train_1.csv
Open with
Download
Rename
Make a copy
Ctrl+C Ctrl+V
Share
Organize
File information
Remove
Report or block

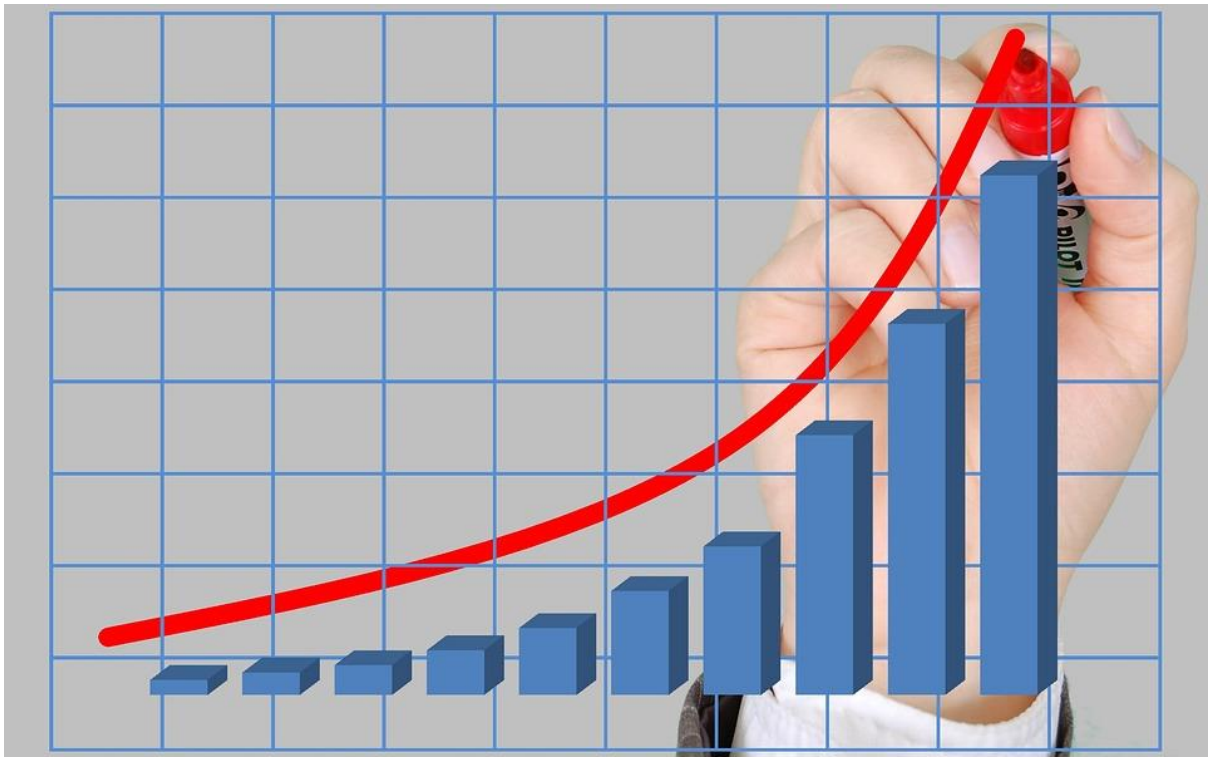

Shared with me
Ad_ease_data
1 selected
Files
Exog_Campaign_eng
Open with
Download
Rename
Make a copy
Ctrl+C Ctrl+V
Share
Organize
File information
Remove
Report or block

Step 2 : open colab notebook



Firstly you need to load both the dataset and then install a 2 package

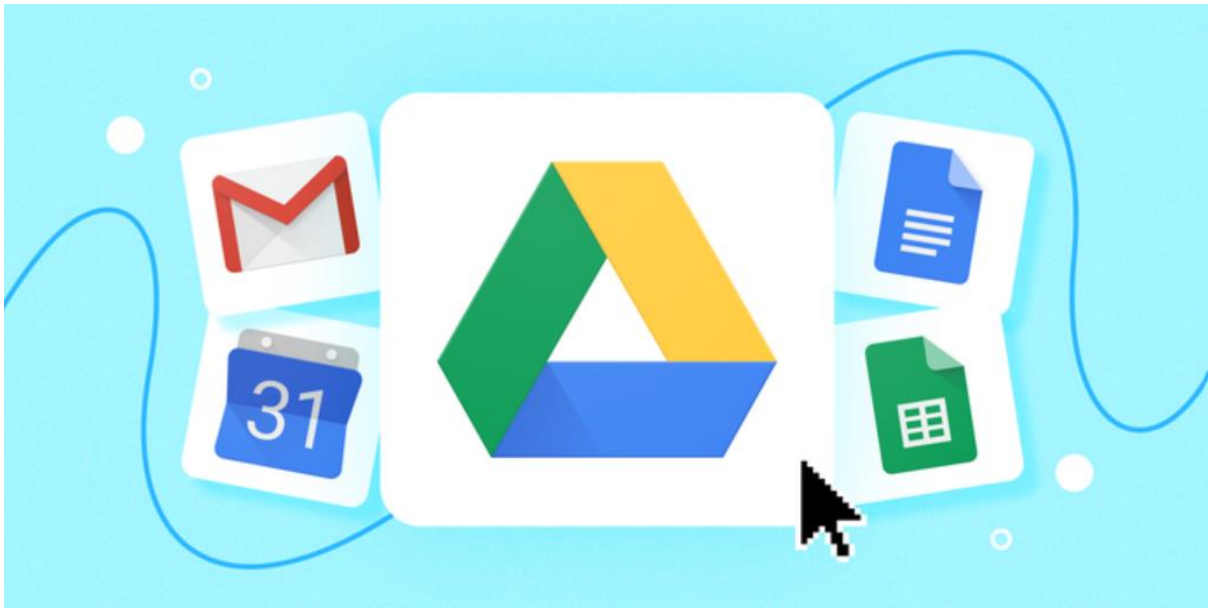
```
!pip install pmdarima
```



The “pmdarima” package, also known as Pyramid ARIMA, is a Python library that provides tools for time series analysis and forecasting. Some of the main uses of this package include:

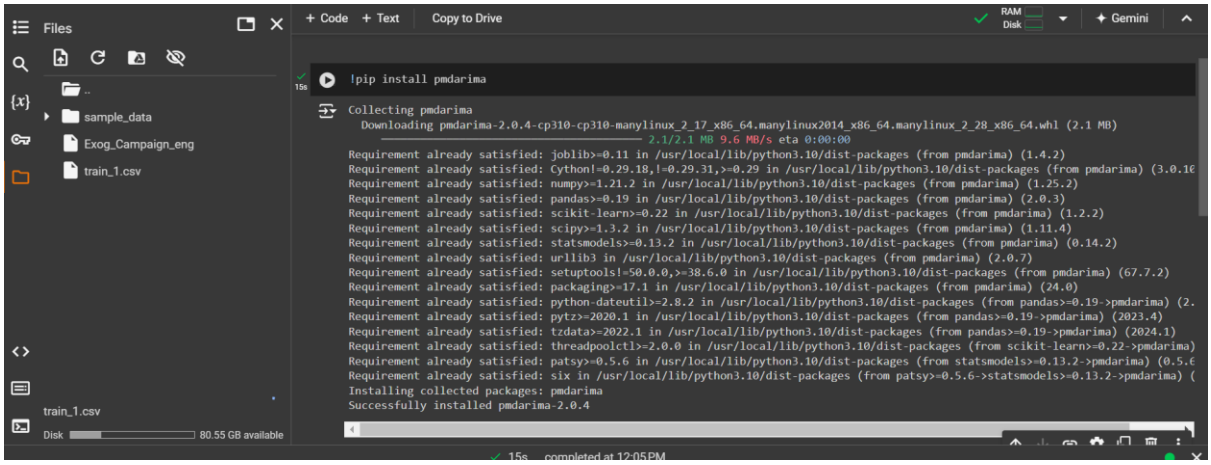
- **Automatic Time Series Forecasting: pmdarima offers automated model selection capabilities to determine the best ARIMA (AutoRegressive Integrated Moving Average) parameters for fitting time series data.**
- **Seasonal Decomposition: It allows users to decompose seasonal components from their time series data using methods like STL decomposition.**
- **Model Diagnostics: The package includes functions for evaluating the performance and diagnostic checks of fitted models such as AIC/BIC criteria, residual diagnostics, etc.**
- **Integration with scikit-learn API: pmdarima seamlessly integrates with scikit-learn’s pipeline functionality making it easy to incorporate into machine learning workflows.**
- **Support for SARIMAX Models: In addition to basic ARIMA modeling support, pmdarima extends its functionalities by supporting exogenous variables in SARIMAX models which can improve forecast accuracy when external factors influence your time series data.**

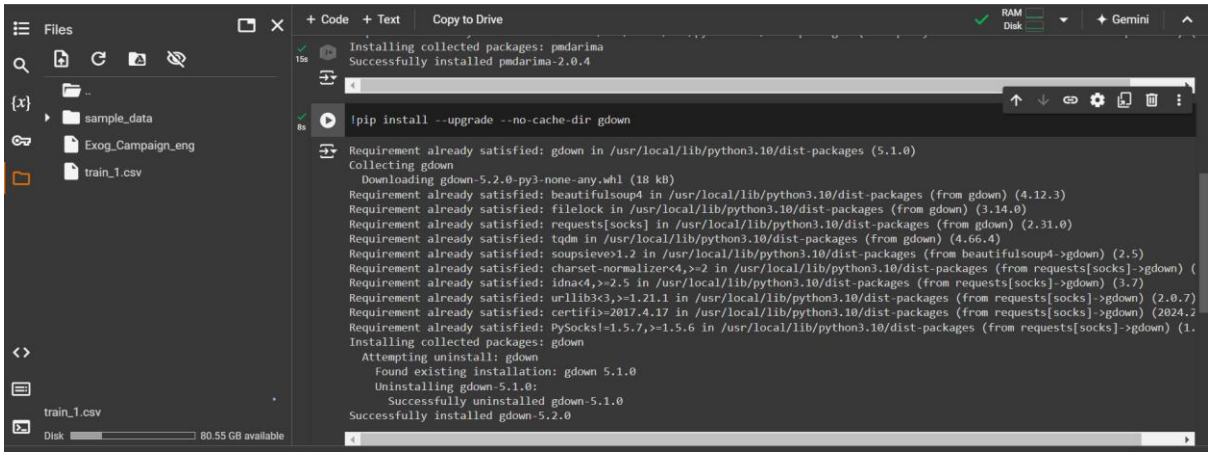
```
!pip install --upgrade --no-cache-dir gdown
```



This command will output the installation process, showing the progress and completion status. If the package is already installed, it will be upgraded to the latest version. If it's not installed, it will be installed afresh.

Screenshot





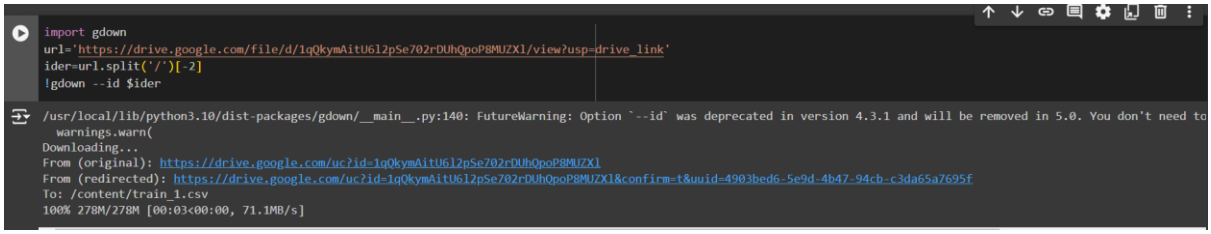
Step 3: copy the link which was given by scaler

https://drive.google.com/drive/folders/1mdgQscjqnCtdg7LGltomyK0abN6lcHBb

```
import gdown
url='https://drive.google.com/drive/folders/1mdgQscjqnCtdg7LGltomyK0abN6lcHBb'
ider=url.split('/')[-2]
!gdown --id $ider
```

This will download the specified file from Google Drive to the current working directory of your notebook or Python environment.

Screenshot



Step 4 : Importing libraries

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pmdarima as pm
from collections import Counter
import pandas as pd
import re
import os
import statsmodels.api as sm
import warnings
warnings.filterwarnings('ignore')
```

Step 5: Read The Data

```
df=pd.read_csv('/content/train_1.csv')
df.head()
```

Screenshot

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	...	2016-12-22	2016-12-23	2016-12-24	2016 12-2
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	...	32.0	63.0	15.0	26.(
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	...	17.0	42.0	28.0	15.(
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	...	3.0	1.0	1.0	7.(
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	...	32.0	10.0	26.0	27.(
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	48.0	9.0	25.0	13.(

5 rows × 551 columns

Step 6 : Inspect the Data

```
#inspect the data
print(df.head())
```

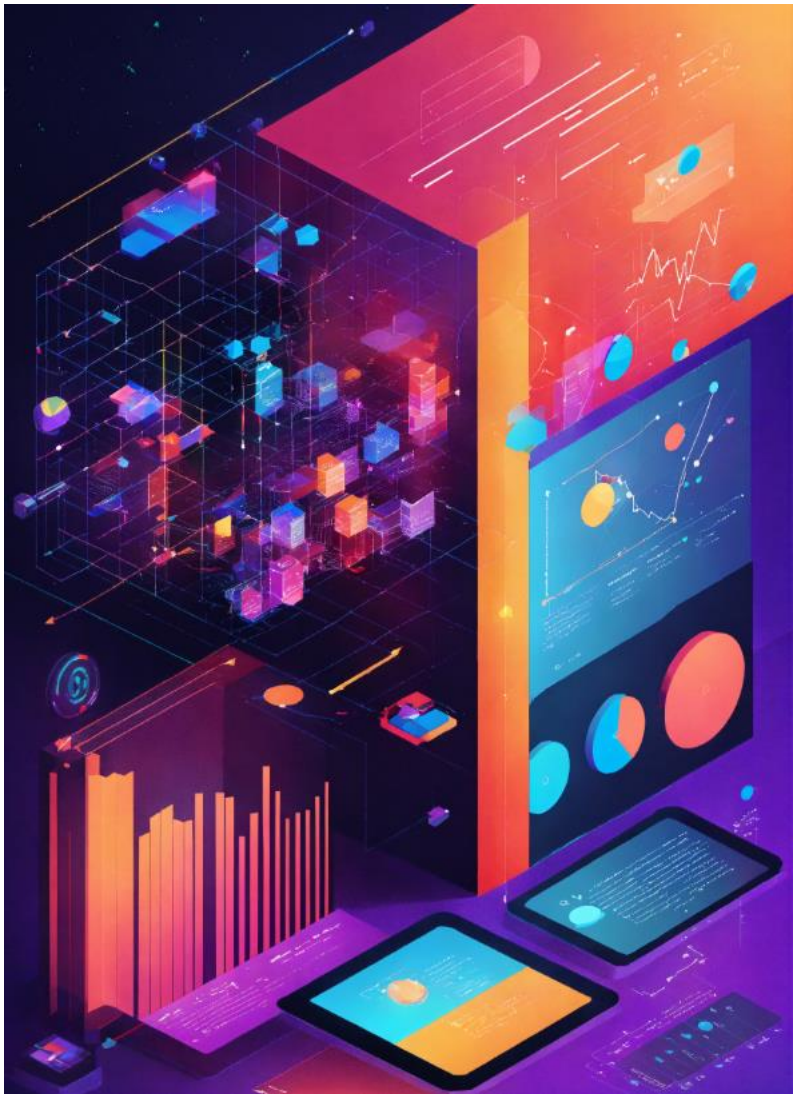
It is a convenient way to take a quick look at the beginning of a DataFrame to understand its structure and contents.

Screenshot

```
#inspect the data
print(df.head())
```

```
                                                Page  2015-07-01  2015-07-02  \
0            2NE1_zh.wikipedia.org_all-access_spider        18.0        11.0
1             2PM_zh.wikipedia.org_all-access_spider        11.0        14.0
2              3C_zh.wikipedia.org_all-access_spider         1.0         0.0
3          4minute_zh.wikipedia.org_all-access_spider        35.0        13.0
4  52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...         NaN         NaN

   2015-07-03  2015-07-04  2015-07-05  2015-07-06  2015-07-07  2015-07-08  \
0         5.0        13.0        14.0         9.0         9.0        22.0
1        15.0        18.0        11.0        13.0        22.0        11.0
2         1.0         1.0         0.0         4.0         0.0         3.0
3        10.0        94.0         4.0        26.0        14.0         9.0
4         NaN         NaN         NaN         NaN         NaN         NaN

   2015-07-09  ...  2016-12-22  2016-12-23  2016-12-24  2016-12-25  \
0        26.0  ...        32.0        63.0        15.0        26.0
1        10.0  ...        17.0        42.0        28.0        15.0
2         4.0  ...         3.0         1.0         1.0         7.0
3        11.0  ...        32.0        10.0        26.0        27.0
4         NaN  ...        48.0         9.0        25.0        13.0

   2016-12-26  2016-12-27  2016-12-28  2016-12-29  2016-12-30  2016-12-31
0        14.0        20.0        22.0        19.0        18.0        20.0
1         9.0        30.0        52.0        45.0        26.0        20.0
2         4.0         4.0         6.0         3.0         4.0        17.0
3        16.0        11.0        17.0        19.0        10.0        11.0
4         3.0        11.0        27.0        13.0        36.0        10.0

[5 rows x 551 columns]
```

Step 7 : Perform basic exploratory data analysis (EDA).



```
#Shape

print(df.shape)
```

The first element in the tuple, 61096, represents the number of rows (observations) in Data Frame. The second element, 551, represents the number of columns (features) in Data Frame.

It is a simple and effective way to check the dimensions of a Data Frame, providing insights into the amount of data you are working with and helping to ensure data integrity during preprocessing and analysis.

Now we need to find the summary of Data frame using info() to find :

- **Class type**
- **Range Index**
- **Columns**
- **Not null**
- **Data types**
- **Memory usage**

```
print(df.info())
```

- **Range Index have 61096 entries from 0 to 61095 (meaning there are 61096 rows in total).**
- **There are 551 columns in the Data Frame, listed from "Page" to "2016-12-31"**
- **It reveals that 550 columns have the data type float64 (which typically represents numerical data), and 1 column has the data type object (which can hold various data types like strings).**
- **The Memory usage is approximately 256.8 megabytes (MB).**

EDA

[8] 0s

```
#Shape

print(df.shape)
```

```
(61096, 551)
```

0s

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61096 entries, 0 to 61095
Columns: 551 entries, Page to 2016-12-31
dtypes: float64(550), object(1)
memory usage: 256.8+ MB
None
```

Steps 8 Checking missing Values

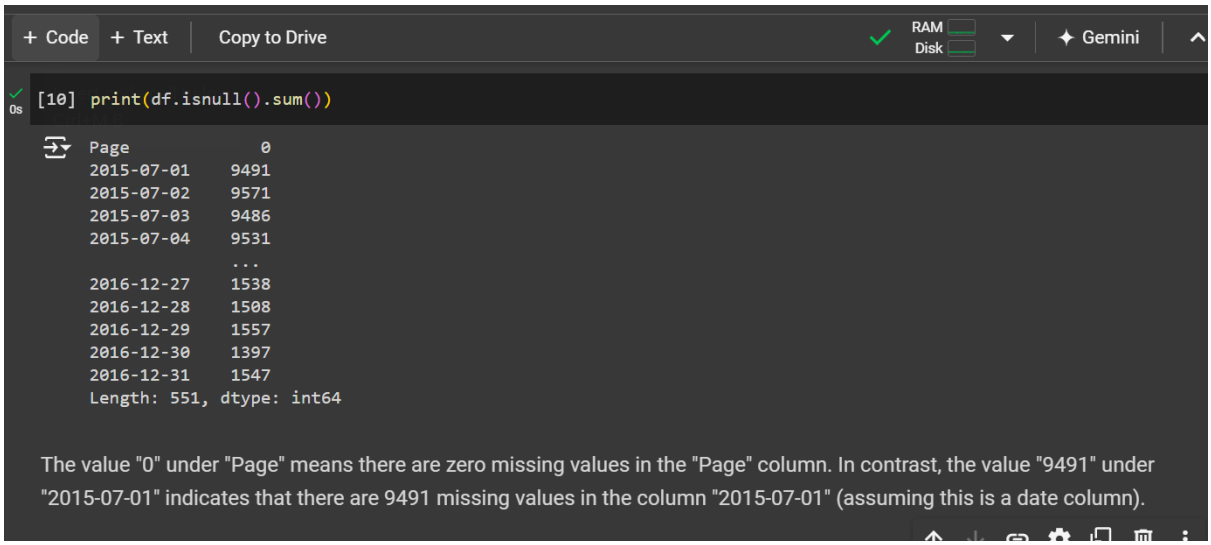
Steps 9: To check null values in each column of a data frame we use isnull()

```
print(df.isnull().sum())
```

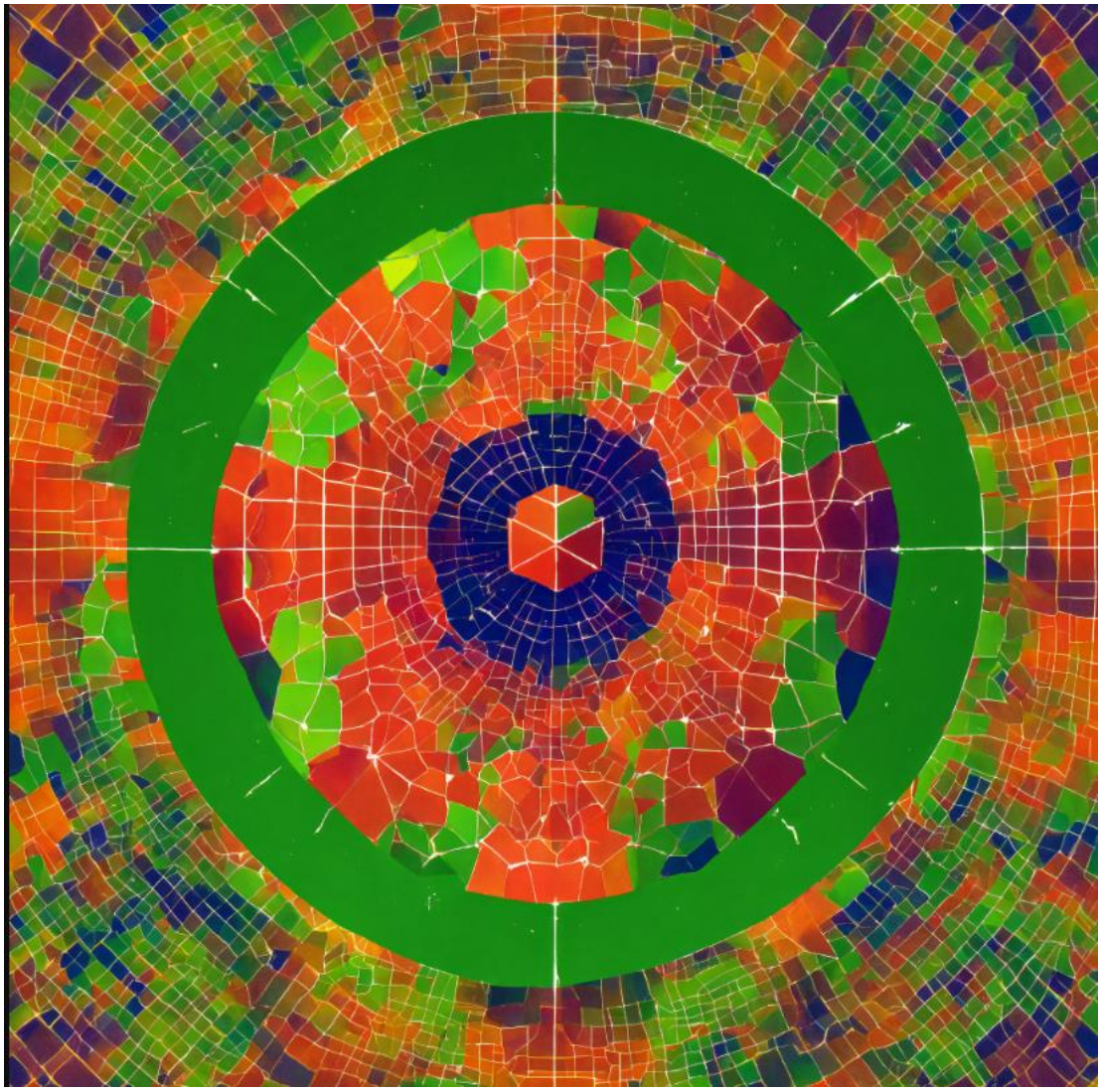
The value "0" under "Page" means there are zero missing values in the "Page" column.

In contrast, the value "9491" under "2015-07-01" indicates that there are 9491 missing values in the column "2015-07-01" (assuming this is a date column).

Screenshot

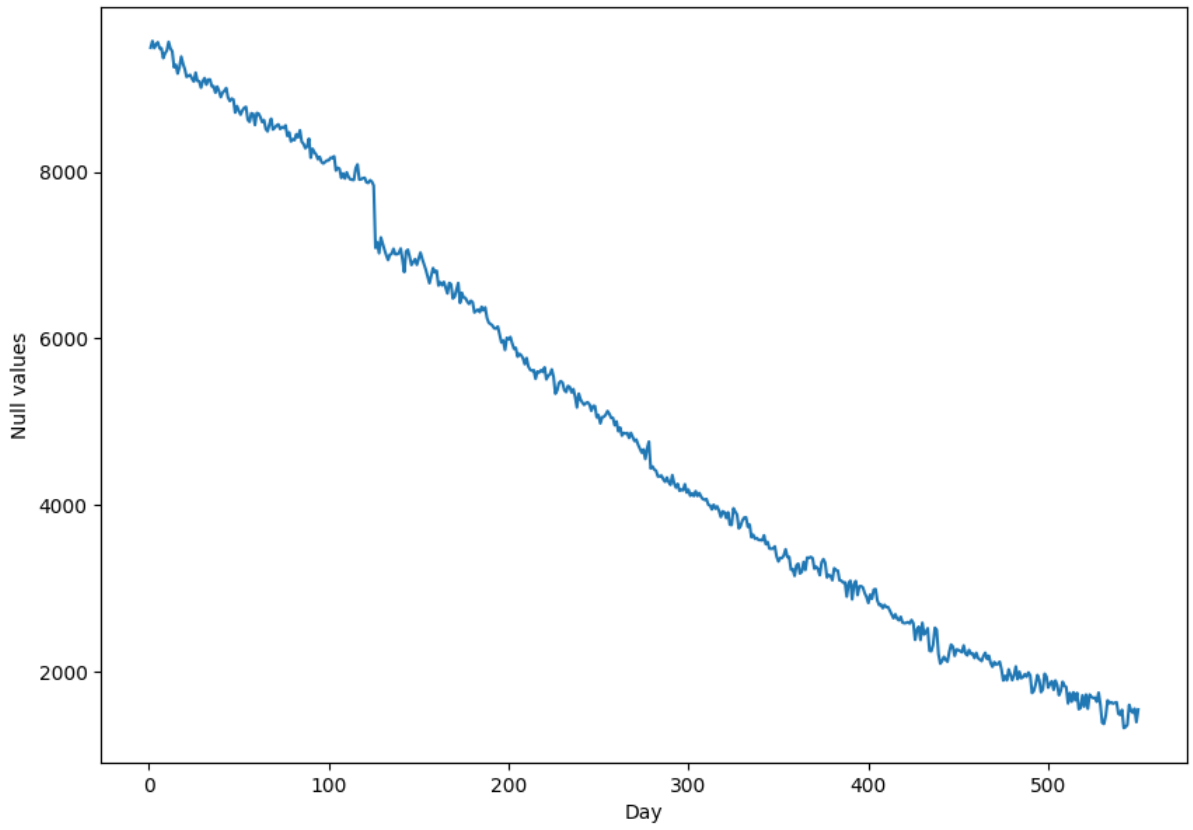


Step 10 : Visualization of Missing values (NAN) –The primary purpose of this code is to create a visualization that helps understand the distribution of missing values across different columns of the Data Frame. This can be crucial for data cleaning and preprocessing steps.



```
days = [r for r in range(1, len(df.columns))]
plt.figure(figsize=(10,7))
plt.xlabel('Day')
plt.ylabel('Null values')
plt.plot(days, df.isnull().sum()[1:])
```

Plot



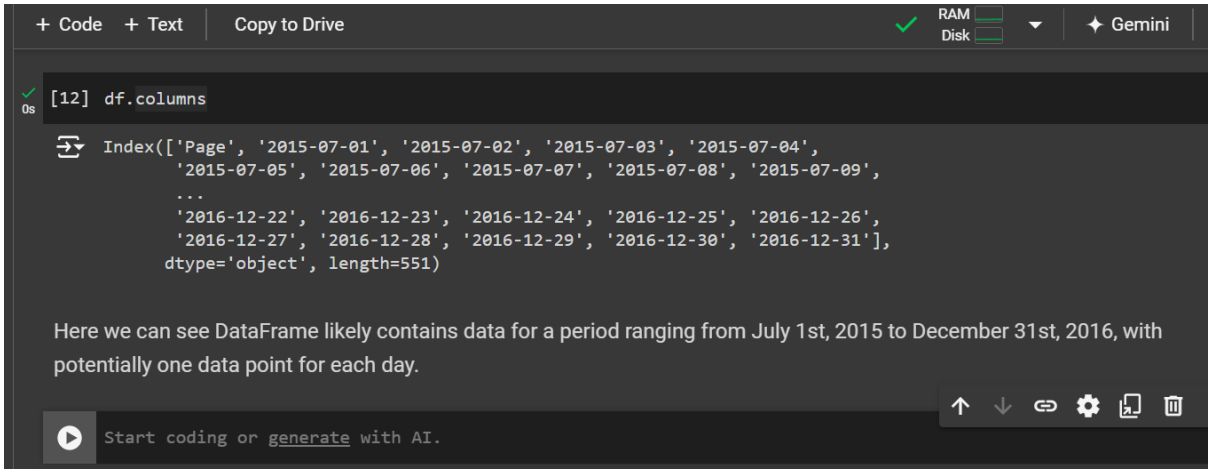
The line shows the variation in the number of missing values across the columns (days). A higher value on the y-axis indicates more missing data in that column (day).

x axis represent the Day and y axis represent the null values .

Steps 11: Now we need to find the labels of data frame for that we use

```
df.columns
```

Screenshot



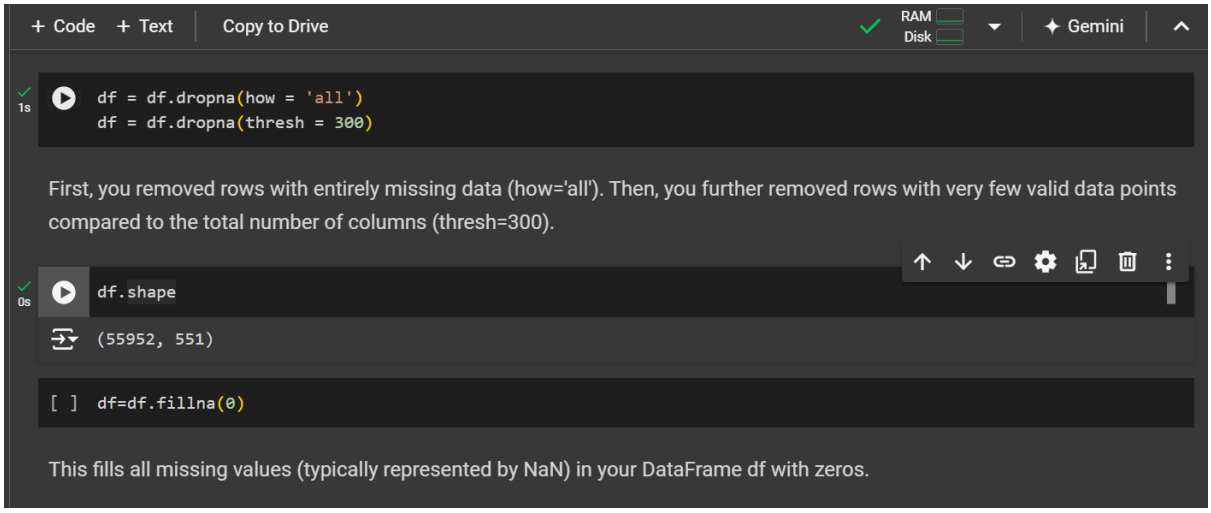
Step 12: Now we need to clean the missing values so that if a row has very few valid data points (less than 300 in this case) compared to the total number of columns (potentially 551 based on previous output), you might decide it's not informative enough to keep for analysis. This can help reduce the impact of rows with very little data. (Remove rows with very sparse data)

```
df = df.dropna(how = 'all')
df = df.dropna(thresh = 300)
```

```
df.shape
```

```
df=df.fillna(0)
```

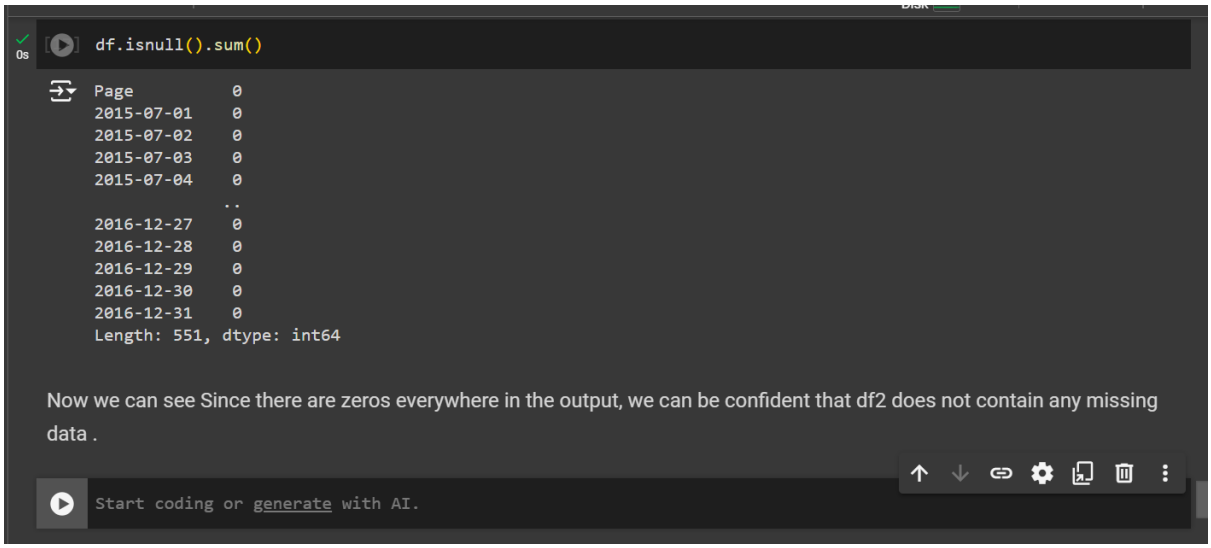
Screenshot



Step 13: Now checking null values again

df.isnull().sum()

Screenshot



Step 14: It's important to note that copying a DataFrame can create a new object in memory, especially if the original DataFrame is large. So, if memory usage is a concern, it's crucial to consider if creating a copy is truly necessary. In some cases, working with views of the original Data Frame might be a more memory-efficient approach.

```
df2 = df.copy()
```

Step 15: For Analyse Wikipedia pages based on language we need to perform Data Formatting

Data formatting is a crucial step in data preprocessing, where raw data is transformed into a structured format suitable for analysis. This often involves converting data types, handling missing values, normalizing data, and ensuring consistency

```
# Data formating
import re #Regular Expressions (RegEx)
def lang(Page):
  val = re.search('[A-Za-z]{2}.wikipedia.org_',Page)
  if val:
    return val[0][0:2]

  return 'no_lang'

df2['Language'] = df2['Page'].apply(lambda x: lang(str(x)))
```

Steps 16: Unique Values

Unique values -This helps in understanding the distribution of languages across the Wikipedia pages in the dataset.It provides a quick summary of the distribution of values in the specified column, which is helpful for exploratory data analysis (EDA).

```
# count of each langauge or Number of wiki pages in each language
df2["Language"].value_counts()
```

Screenshot

Unique values

Add text cell

```
# count of each langauge or Number of wiki pages in each language
df2["Language"].value_counts()
```

```
Language
en         13871
fr         12670
no_lang     8750
zh          8084
ja          4691
de          4317
ru          3569
Name: count, dtype: int64
```

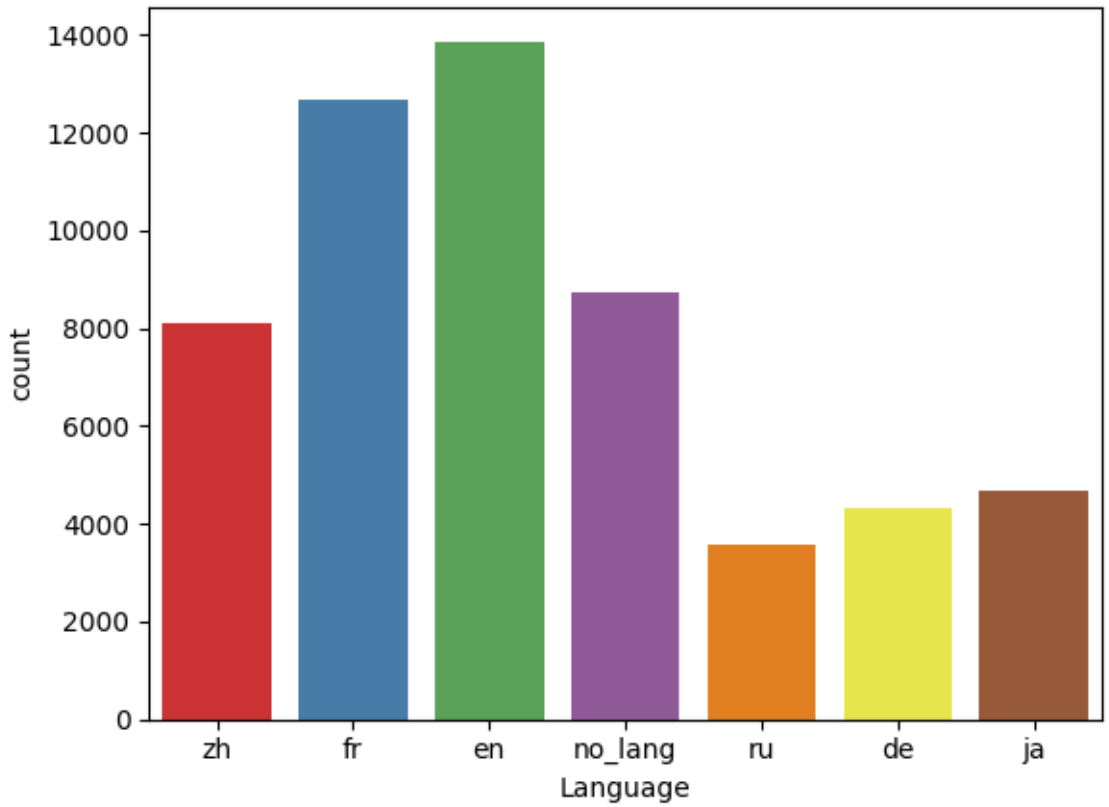
- Index: The first column, titled "Language", lists all the unique languages found in the "Language" column. In this case, it shows entries like "en" (presumably English), "fr" (French), "no_lang" (likely representing entries with no language specified), and others.
- Value Counts: The second column, titled "count" (and with a data type of "int64" indicating integers), represents the number of times each language appears in the "Language" column.
- For example, the value "13871" under "en" signifies that there are 13871 wiki pages in English in your DataFrame df2.

Step 17: Bar Chart Visualization.

Now we need to Visualize through bar chart for better understandings It provides a more clear and informative way to explore the distribution of languages in Data Frame compared to just looking at raw counts. It leverages human perception to make comparisons and identify patterns in the data more easily.

```
# Count of each language or number of wiki pages in each language.
sns.countplot(data = df2, x = "Language", palette = 'Set1')
```

Plots



We can see in the plot that the most common languages are English (en), French (fr), and "no_lang" (which likely represents entries with no language specified). There are also wiki pages in other languages, such as Chinese (zh), Japanese (ja), German (de), and Russian (ru).

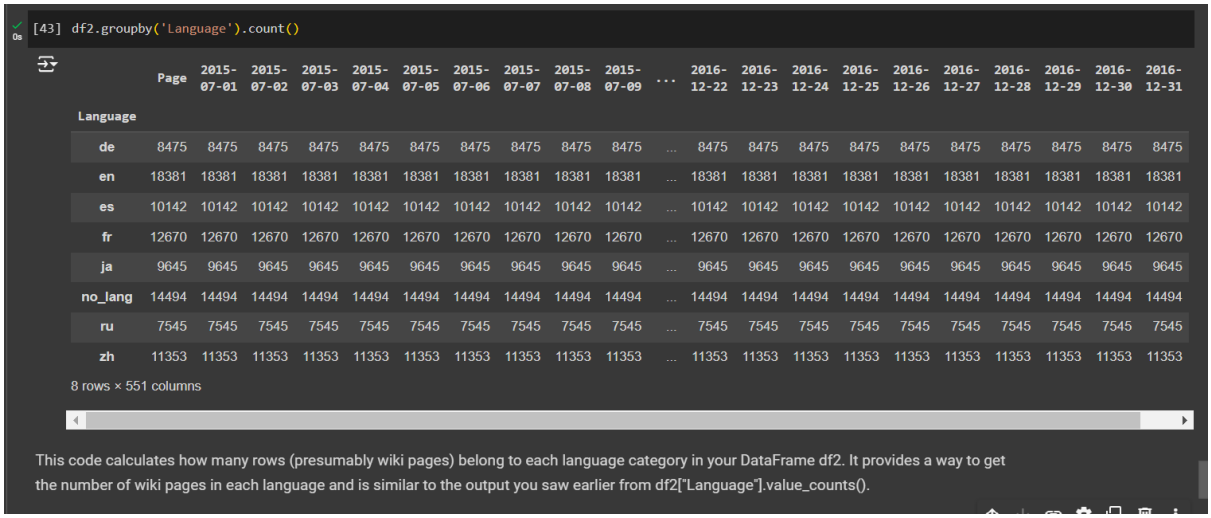
Distribution: The number of wiki pages varies considerably between languages. English has the most wiki pages, followed by French and then "no_lang". The other languages have fewer wiki pages.

Step 18 : Groupby

Now we need to calculates how many rows (wiki pages) belong to each language category in your DataFrame df2.

```
df2.groupby('Language').count()
```

Screenshot



```
[43] df2.groupby('Language').count()
```

Language	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	...	2016-12-22	2016-12-23	2016-12-24	2016-12-25	2016-12-26	2016-12-27	2016-12-28	2016-12-29	2016-12-30	2016-12-31
de	8475	8475	8475	8475	8475	8475	8475	8475	8475	8475	...	8475	8475	8475	8475	8475	8475	8475	8475	8475	8475
en	18381	18381	18381	18381	18381	18381	18381	18381	18381	18381	...	18381	18381	18381	18381	18381	18381	18381	18381	18381	18381
es	10142	10142	10142	10142	10142	10142	10142	10142	10142	10142	...	10142	10142	10142	10142	10142	10142	10142	10142	10142	10142
fr	12670	12670	12670	12670	12670	12670	12670	12670	12670	12670	...	12670	12670	12670	12670	12670	12670	12670	12670	12670	12670
ja	9645	9645	9645	9645	9645	9645	9645	9645	9645	9645	...	9645	9645	9645	9645	9645	9645	9645	9645	9645	9645
no_lang	14494	14494	14494	14494	14494	14494	14494	14494	14494	14494	...	14494	14494	14494	14494	14494	14494	14494	14494	14494	14494
ru	7545	7545	7545	7545	7545	7545	7545	7545	7545	7545	...	7545	7545	7545	7545	7545	7545	7545	7545	7545	7545
zh	11353	11353	11353	11353	11353	11353	11353	11353	11353	11353	...	11353	11353	11353	11353	11353	11353	11353	11353	11353	11353

8 rows × 551 columns

This code calculates how many rows (presumably wiki pages) belong to each language category in your DataFrame df2. It provides a way to get the number of wiki pages in each language and is similar to the output you saw earlier from df2["Language"].value_counts().

Steps 19 : check the data integrity checks, and debugging purposes.

```
df2.head()
```

Screenshot

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	...	2016-12-23	2016-12-24	2016-12-25	2016-12-26	2016-12-27	2016-12-28	2016-12-29	2016-12-30	2016-12-31
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	...	63.0	15.0	26.0	14.0	20.0	22.0	19.0	18.0	20.0
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	...	42.0	28.0	15.0	9.0	30.0	52.0	45.0	26.0	20.0
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	...	1.0	1.0	7.0	4.0	4.0	6.0	3.0	4.0	17.0
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	...	10.0	26.0	27.0	16.0	11.0	17.0	19.0	10.0	11.0
5	5566_zh.wikipedia.org_all-access_spider	12.0	7.0	4.0	5.0	20.0	8.0	5.0	17.0	24.0	...	27.0	8.0	17.0	32.0	19.0	23.0	17.0	17.0	3.0

5 rows × 552 columns

Step 20: Data Preprocessing

```
# Mean number of clicks for each language on each day

df2 = df2.drop("Page", axis = 1)
df2_language = df2.groupby("Language").mean().transpose()
```

Here df2 is is presumably data frame.

In 1st line the code removes the "Page" column from df2 and returns a new DataFrame without that column.

In 2nd line it groups the DataFrame df2 by the "Language" column and calculates the mean of the numeric columns for each language group and then transposes the resulting DataFrame, swapping rows and columns.

Step 21: inspect the data

```
df2_language.head()
```

Screenshot

```
[63] df2_language.head()
```

Language	de	en	es	fr	ja	no_lang	ru	zh
2015-07-01	763.765926	3767.328604	1127.485204	499.092872	614.637160	102.733545	663.199229	272.498521
2015-07-02	753.362861	3755.158765	1077.485425	502.297852	705.813216	107.663447	674.677015	272.906778
2015-07-03	723.074415	3565.225696	990.895949	483.007553	637.451671	101.769629	625.329783	271.097167
2015-07-04	663.537323	3711.782932	930.303151	516.275785	800.897435	86.853871	588.171829	273.712379
2015-07-05	771.358657	3833.433025	1011.759575	506.871666	768.352319	96.254105	626.385354	291.977713

Step 22: Reseting Index(Clean up)

By default, when you reset the index, the old index is added as a new column in the DataFrame, and a new sequential index is created.

inplace=True means the operation is done in place.

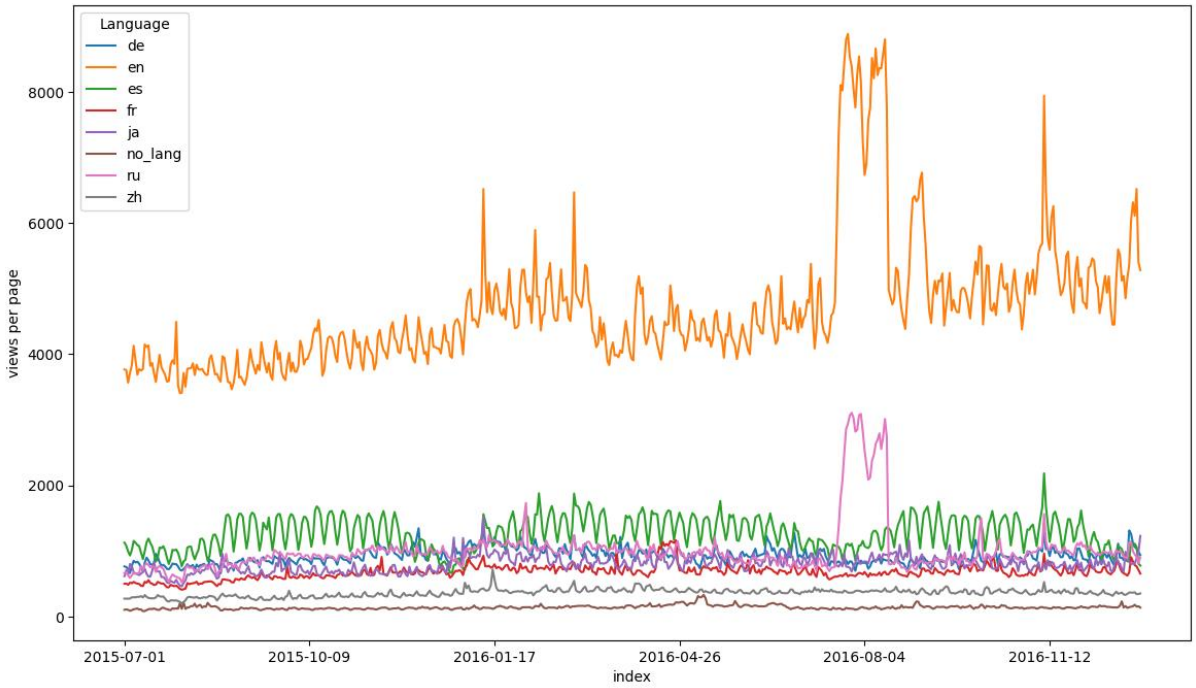
The code resets the index of df2_language to a default integer index and then sets the 'index' column as the new index of the DataFrame. This might be done to clean up the Data Frame's structure or to prepare it for further analysis or operations where the 'index' column holds important information.

```
df2_language.reset_index(inplace = True)
df2_language.set_index('index', inplace = True)
```

Step 23: Lets visualize a plot by showing the mean number of views per page over different days for each language.

```
# Mean number of views Vs Days

df2_language.plot(figsize = (14,8))
plt.ylabel("views per page")
```

Plot



Here we can see the line graph where the x-axis shows dates ranging from July 2015 to November 2016 and the y-axis shows the average number of views per page.

The blue line shows the average number of views per page over time. It appears to have an upward trend, though there is some fluctuation. This suggests that on average, pages were viewed more often later in the time period than they were earlier.

Step 24 : Check stationarity

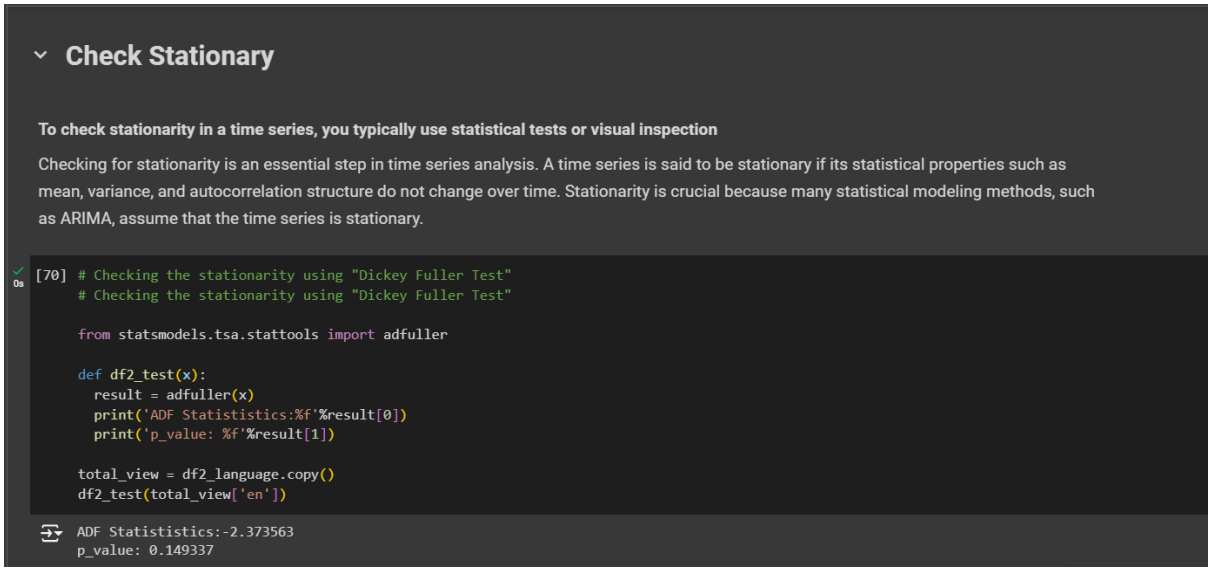
There are some methods for this:

- **Visual Inspection: Plot the time series data and look for any obvious trends or seasonality. If the mean and variance appear to be relatively constant over time, it suggests stationarity. You can use tools like rolling statistics plots to visualize this.**
- **Summary Statistics: Calculate summary statistics such as mean and variance over different time periods. If these statistics remain relatively constant over time, it indicates stationarity.**
- **Augmented Dickey-Fuller (ADF) Test: This is a statistical test that tests the null hypothesis that a unit root is present in a time series dataset, indicating non-stationarity. If the p-value obtained from the test is below a certain significance level (commonly 0.05), you reject the null hypothesis, suggesting stationarity.**
- **Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test: Similar to the ADF test, the KPSS test also checks for stationarity but under the null hypothesis that the data is stationary around a deterministic trend. If the p-value is above a certain significance level, you fail to reject the null hypothesis, suggesting stationarity.**

- **Dickey-Fuller Generalized Least Squares (DF-GLS) Test: This test is a modification of the ADF test that allows for serial correlation and heteroscedasticity in the errors. It provides a more robust test for stationarity.**

```
# Checking the stationarity using "Dickey Fuller Test"
# Checking the stationarity using "Dickey Fuller Test"

from statsmodels.tsa.stattools import adfuller

def df2_test(x):
  result = adfuller(x)
  print('ADF Statistics:%f'%result[0])
  print('p_value: %f'%result[1])

total_view = df2_language.copy()
df2_test(total_view['en'])
```

Screenshot



Why Check for Stationarity?

- **Model Assumptions: Many time series models assume that the series is stationary.**

- **Predictive Accuracy: Non-stationary data can lead to inaccurate predictions.**
- **Statistical Inference: The statistical properties of a stationary series are easier to infer and analyze.**

Types of Stationarity

- **Strict Stationarity: A time series is strictly stationary if the joint distribution of any set of observations is invariant under time shifts.**
- **Weak Stationarity (or Second-Order Stationarity): A time series is weakly stationary if its mean, variance, and autocorrelation structure remain constant over time.**

Methods to Check for Stationarity

- **Visual Inspection: Plot the time series and check for trends or seasonal patterns.**
- **Summary Statistics: Divide the series into segments and check if the mean and variance are constant across segments.**
- **Autocorrelation Function (ACF): Plot the ACF and check if the correlations die down quickly.**
- **Statistical Tests: Use formal statistical tests to check for stationarity.**

Step 24: Time series Decomposition

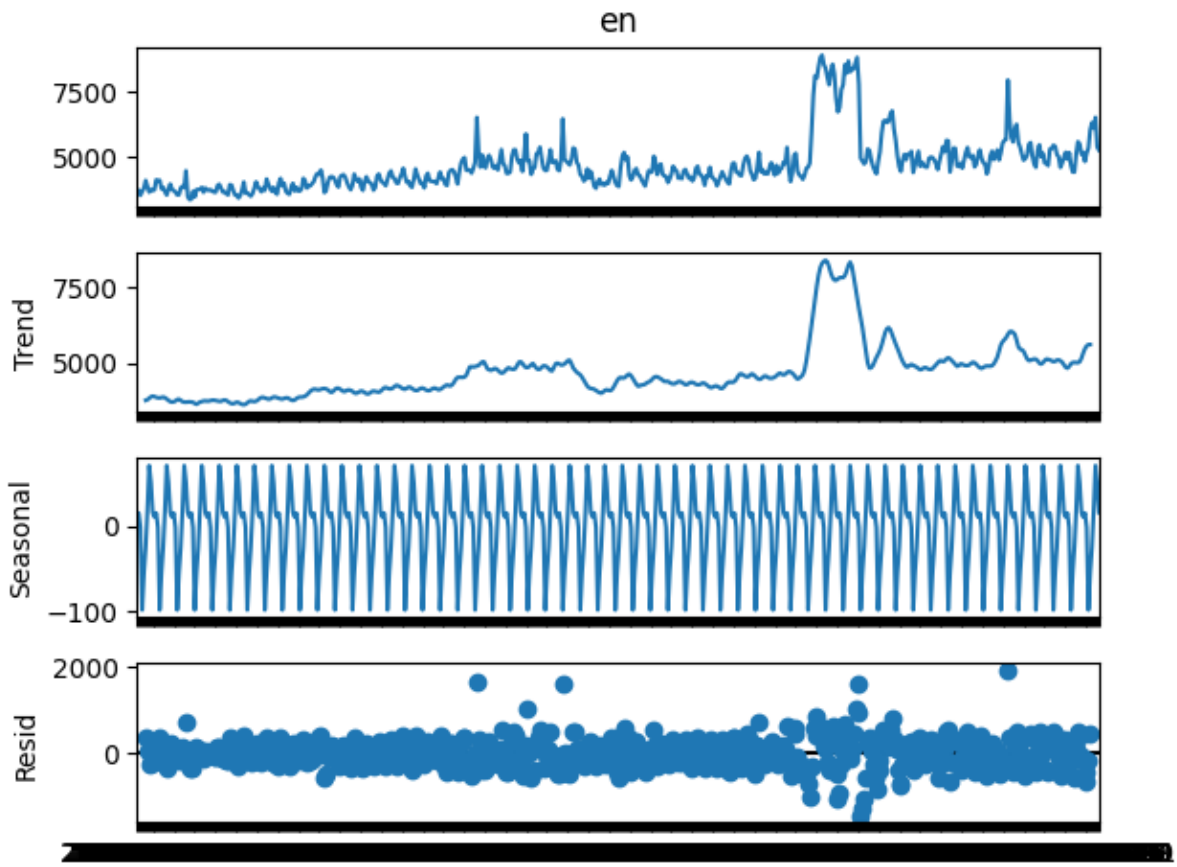
Time series decomposition is a valuable technique for understanding the underlying patterns and components within time series data. By decomposing a time series into its constituent parts, analysts can gain insights into the trend, seasonality, and noise present in the data, which can aid in forecasting and decision-making processes.

```
total_view = df2_language.copy()
```

```
ts = total_view["en"]
```

```
from statsmodels.tsa.seasonal import seasonal_decompose
import statsmodels.api as sm

tsn = pd.Series(total_view['en'],index = total_view.index)

decomposition = sm.tsa.seasonal_decompose(tsn, model =
"additive", period = 10)

decomposition.plot()
plt.show()
```

Plot

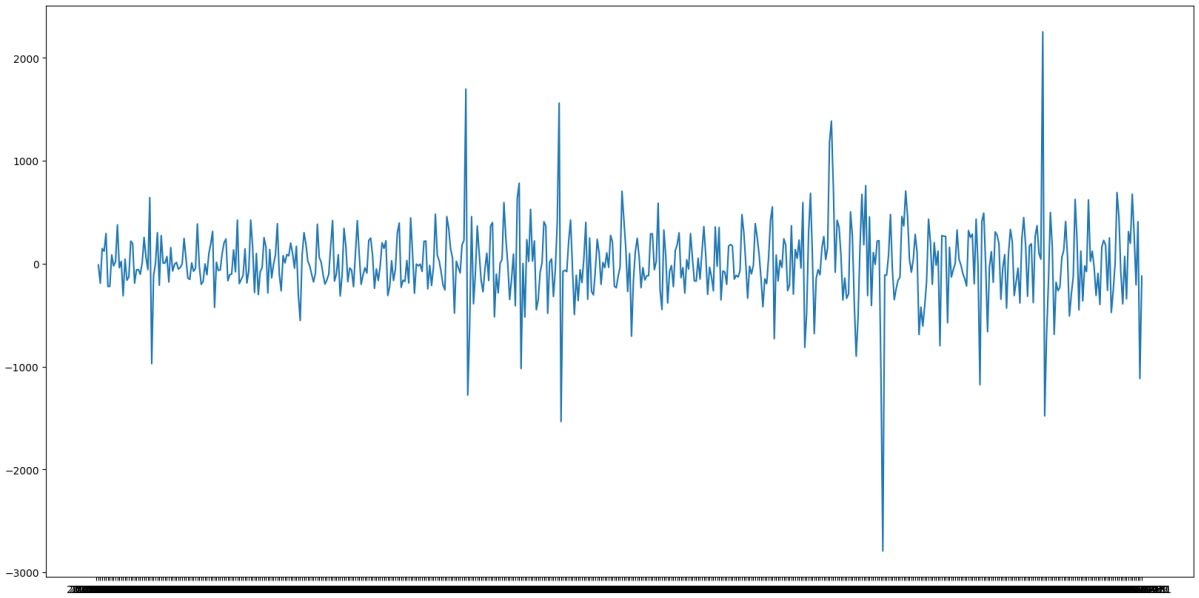


Steps 25 : Removing seasonality & trend using differencing

After differencing, you should inspect the differenced series to ensure that the trend and seasonality have been adequately removed. You can then use the differenced series for further analysis, such as forecasting or modeling, which assumes stationarity in the data.

```
ts_diff = ts - ts.shift(1)
plt.figure(figsize = (20,10))
plt.plot(ts_diff)
plt.show()
```

Plot

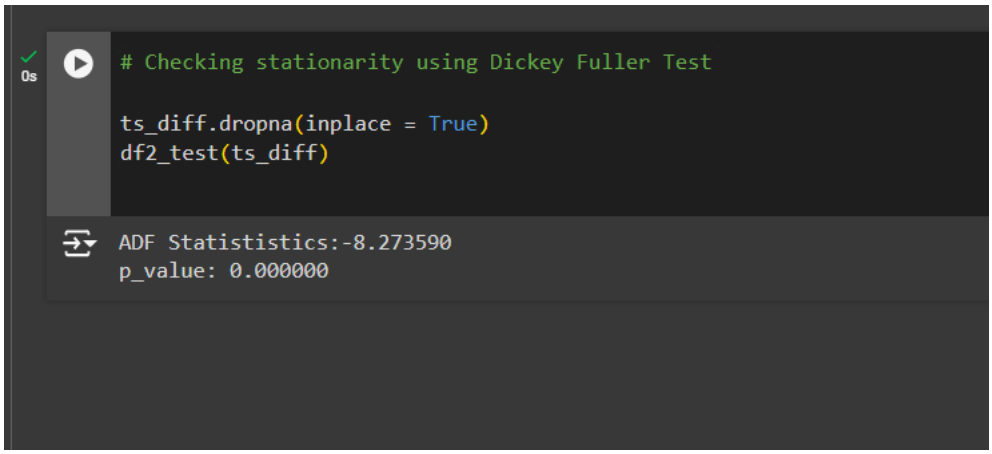


The graph displays a line that fluctuates around zero. It starts at a high positive value, then quickly goes negative, oscillates around zero several times, and ends at a negative value.

Steps : 26 checking stationarity using Dickey Fuller Test

```
# Checking stationarity using Dickey Fuller Test

ts_diff.dropna(inplace = True)
df2_test(ts_diff)
```

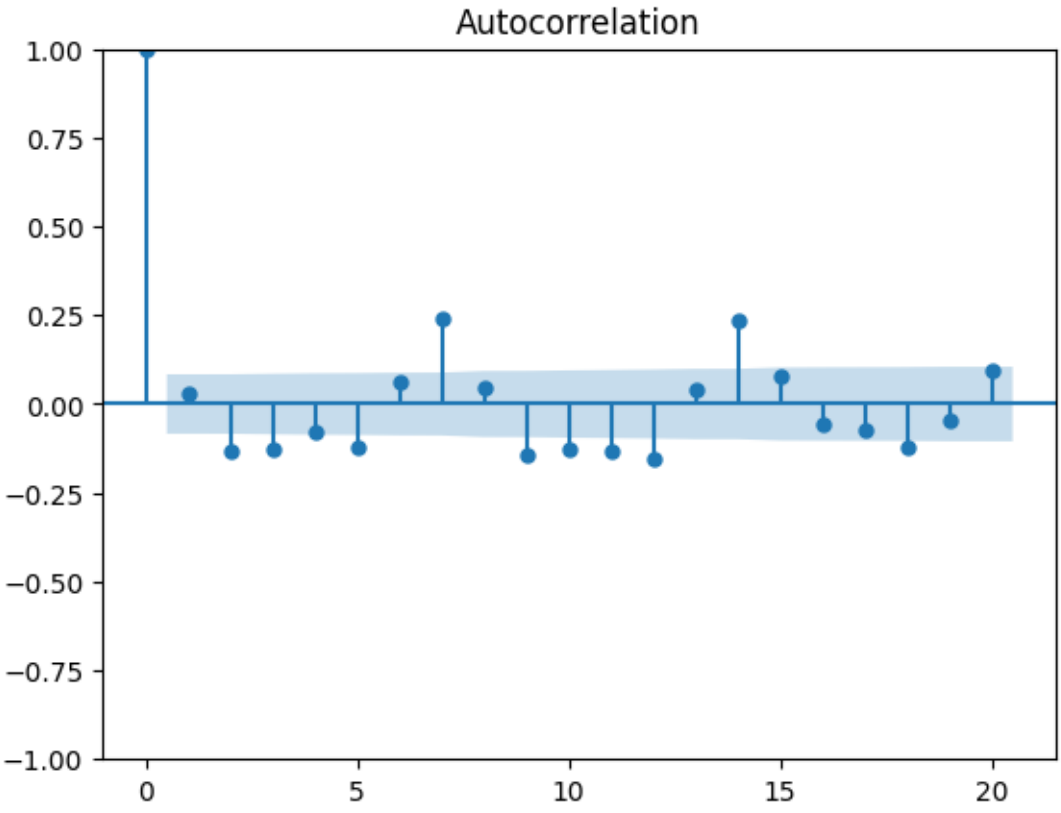
Screenshot

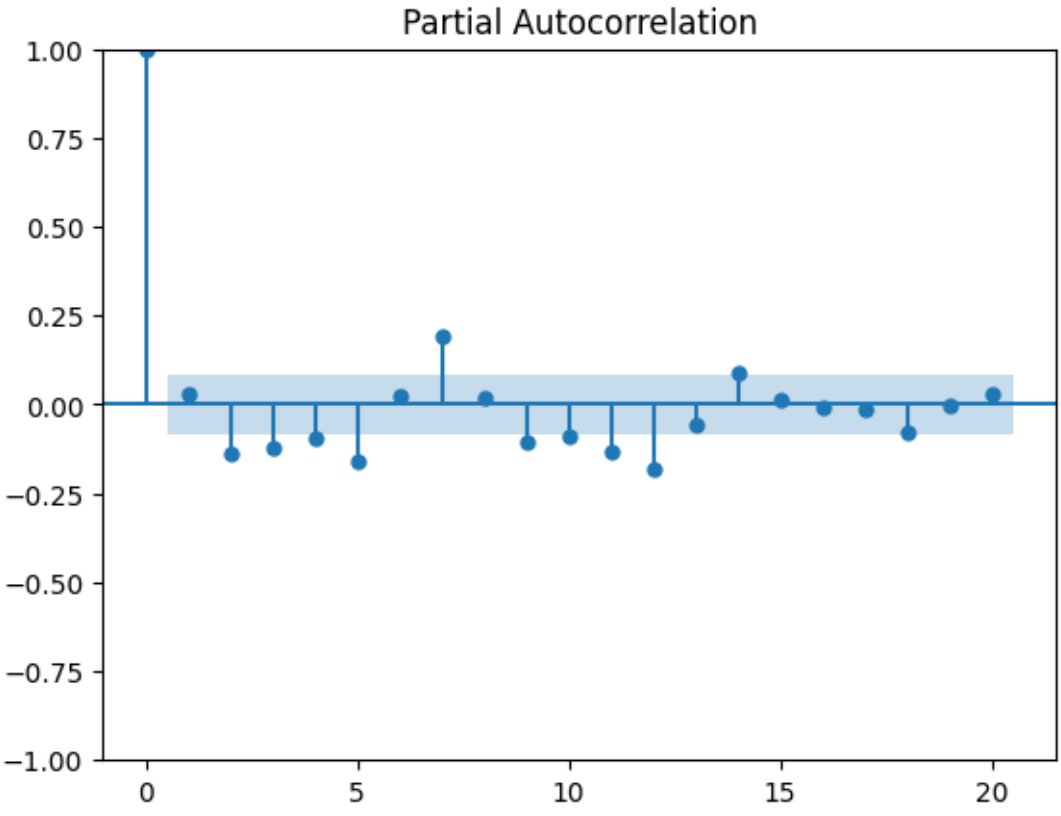


Step 27 : Auto correlation & Partial Autocorrelation Plots

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
acf = plot_acf(ts_diff,lags = 20)
pacf = plot_pacf(ts_diff, lags = 20)
```

Plots

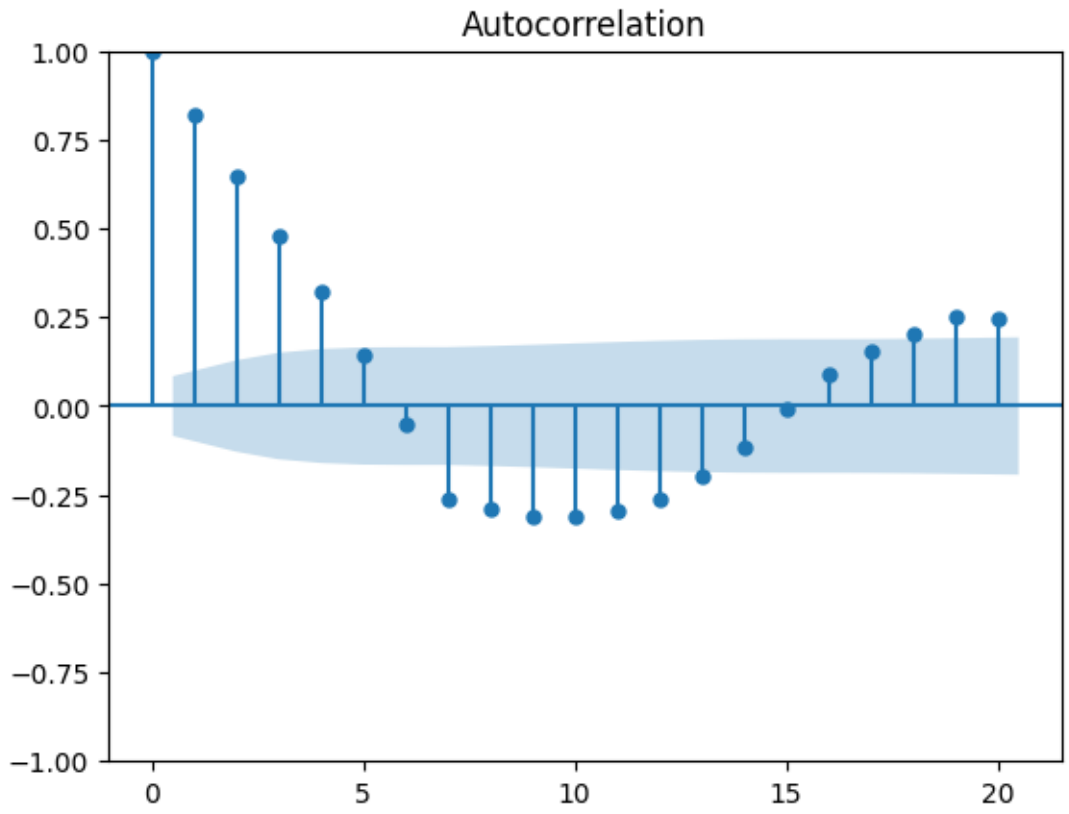

Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20

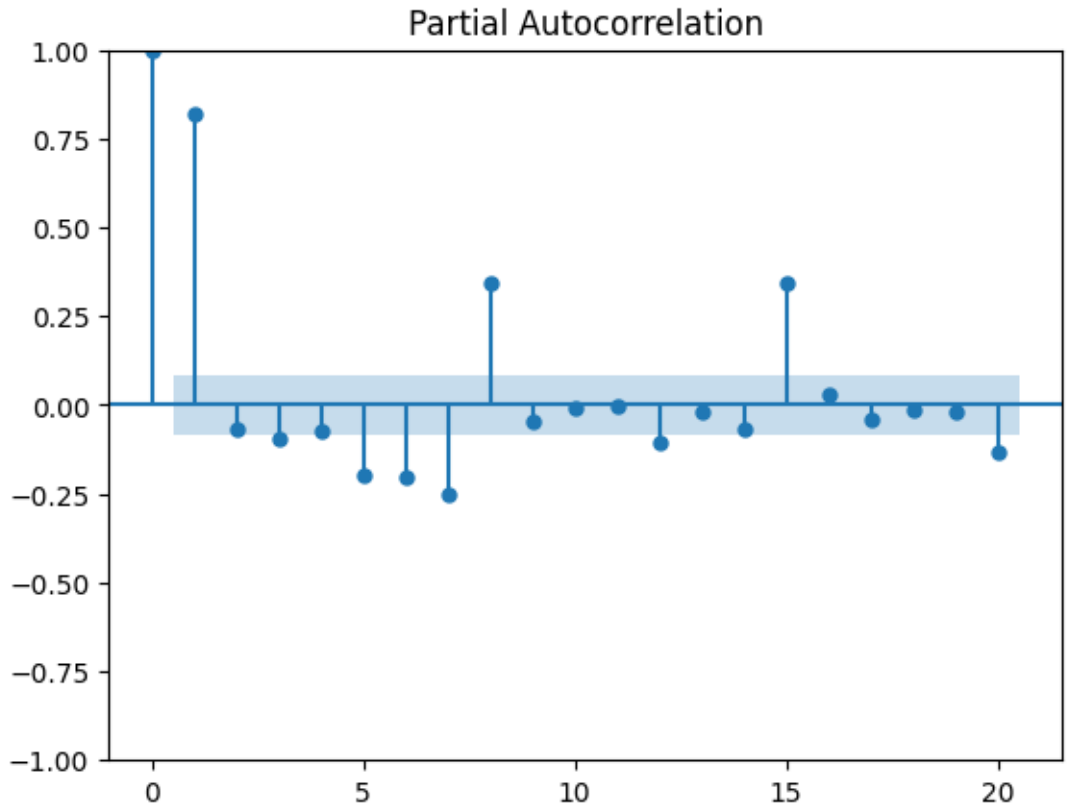

Partial Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20

- In this plot the ACF values appear to be relatively small for all lags, and most of them fall within the confidence interval lines (the dashed lines above and below the x-axis). This suggests that there is little or no autocorrelation in the time series data. In other words, the values in the time series are not predictable based on past values.
- The PACF values cut off at around lag 20. However, for the lags that are shown, the PACF values appear to die down quickly to zero. This suggests that there may be significant autocorrelation at lags 1 and 2, but not at higher lags.

```
ts_diff = ts - ts.shift(7)
ts_diff.dropna(inplace = True)
acf = plot_acf(ts_diff,lags = 20)
pacf = plot_pacf(ts_diff, lags = 20)
```

Plot





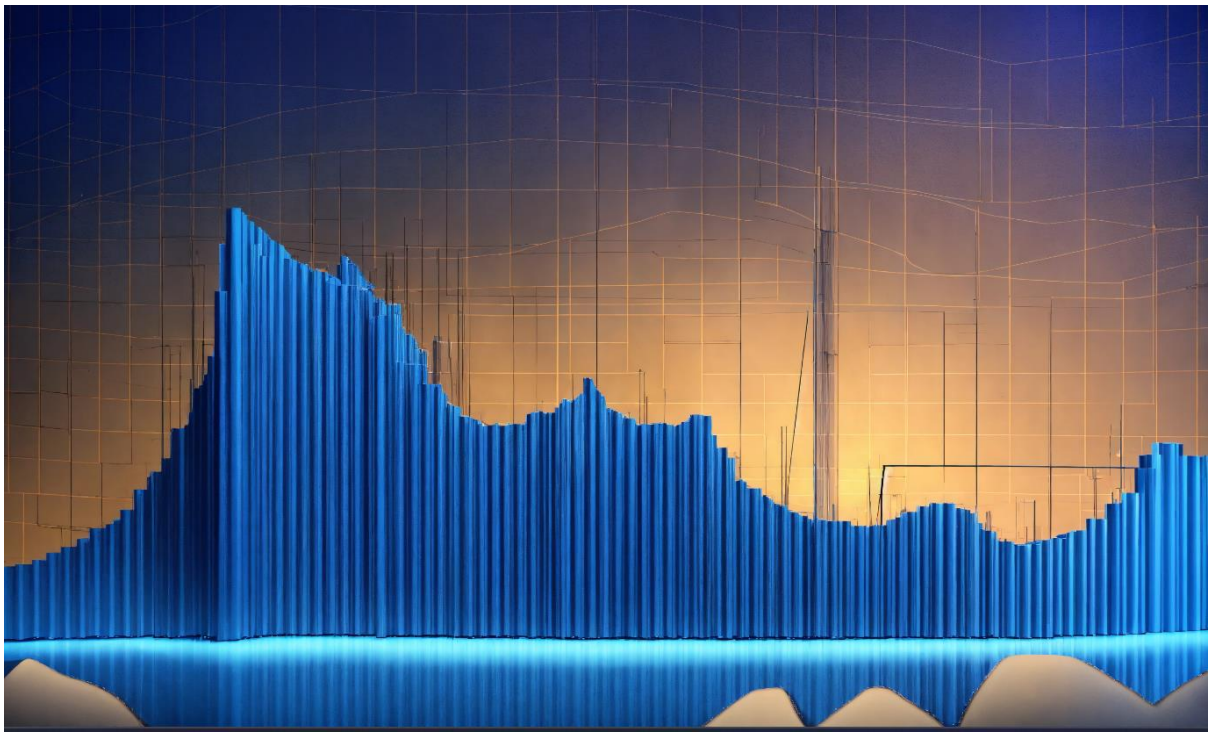
The PACF plot suggests that there might be significant autocorrelation at lower lags in the time series data and the ACF plot suggests that there is weak or no autocorrelation in the time series data.

Step 28: Spitting Test

Splitting data into training and testing sets is crucial for assessing the performance and generalization ability of machine learning models, preventing overfitting, and tuning model parameters for optimal performance on unseen data.

```
train = ts[:-30]
test = ts[-30:]
```

Step 29 : ARIMA



ARIMA stands for Autoregressive Integrated Moving Average Model. It belongs to a class of models that explains a given time series based on its own past values -i.e.- its own lags and the lagged forecast errors. The equation can be used to forecast future values. Any 'non-seasonal' time series that exhibits patterns and is not a random white noise can be modeled with ARIMA models.

So, ARIMA, short for AutoRegressive Integrated Moving Average, is a forecasting algorithm based on the idea that the information in the past values of the time series can alone be used to predict the future values

It is specified by three order parameters: (p, d, q), where:

- **p is the order of the AR term**
- **q is the order of the MA term**
- **d is the number of differencing required to make the time series stationary**

- **AR(p) Autoregression – a regression model that utilizes the dependent relationship between a current observation and observations over a previous period. An auto regressive (AR(p)) component refers to the use of past values in the regression equation for the time series.**

- I(d) Integration – uses differencing of observations (subtracting an observation from observation at the previous time step) in order to make the time series stationary. Differencing involves the subtraction of the current values of a series with its previous values d number of times.

- MA(q) Moving Average – a model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations. A moving average component depicts the error of the model as a combination of previous error terms. The order q represents the number of terms to be included in the model.

```
#performance matrics
from sklearn.metrics import mean_squared_error as mse,
mean_absolute_error as mae, mean_absolute_percentage_error as
mape

def performance(actual, predicted):
  print("MAE:", round(mae(actual, predicted),3))
  print("RMSE:", round(mse(actual, predicted)**0.5,3))
  print("MAPE:", round(mape(actual, predicted),3))
```

Types of ARIMA Model

- o ARIMA : Non-seasonal Autoregressive Integrated Moving Averages
- o SARIMA : Seasonal ARIMA
- o SARIMAX : Seasonal ARIMA with exogenous variables

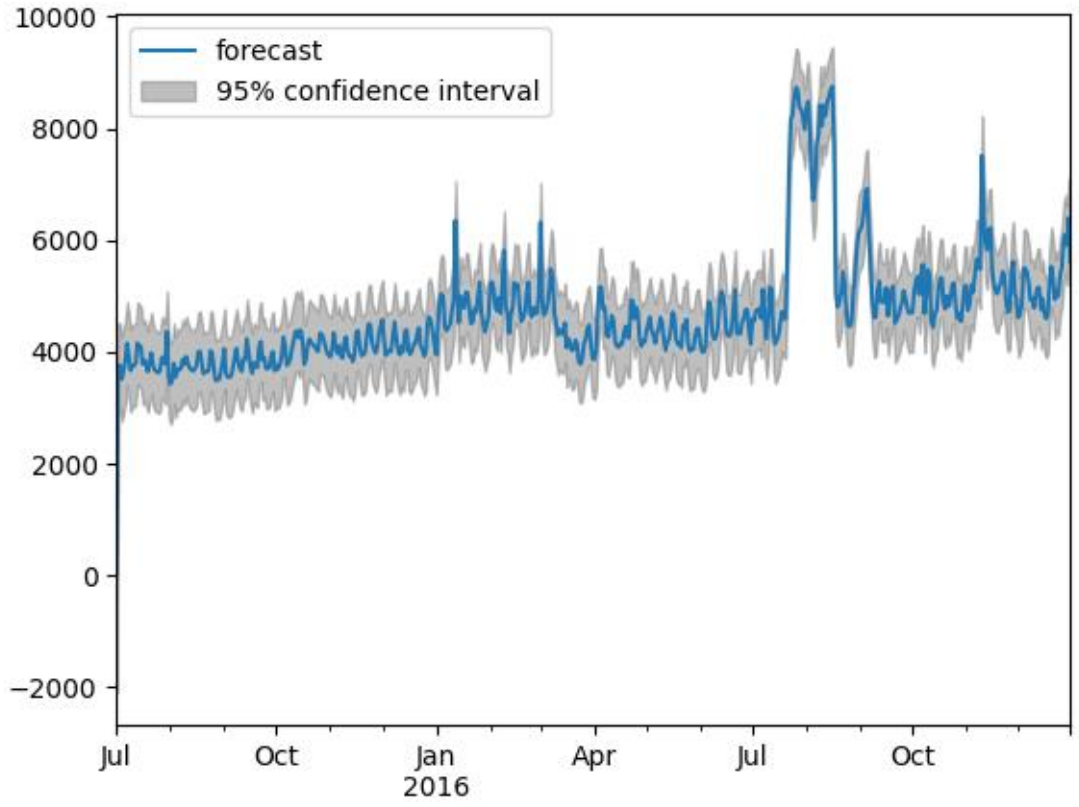
If a time series, has seasonal patterns, then we need to add seasonal terms and it becomes SARIMA, short for Seasonal ARIMA.

```
# Importing ARIMA
from statsmodels.tsa.arima.model import ARIMA
# Importing pandas data frame
from pandas import DataFrame
```

```
# Developing the model for entire data set
model=ARIMA(ts, order = (4,1,3))
model_fit = model.fit()
```

```
# Models prediction with 95% confidence interval

from statsmodels.graphics.tsaplots import plot_predict

plot_predict(model_fit,dynamic = False)
plt.show()
```

PLOT



- **Forecast: The exact value of the forecast is not shown, but it can be assumed to be somewhere between the upper and lower confidence interval lines.**
- **Confidence level: The confidence interval is 95%. This means that if we were to make the same forecast many times, 95% of the time the actual outcome would fall within the shaded area.**

- **Certainty: The width of the confidence interval reflects the level of certainty associated with the forecast. A narrow confidence interval indicates a more certain forecast, while a wide confidence interval indicates a less certain forecast. In this case, the confidence interval appears to be relatively wide, which suggests that there is a significant degree of uncertainty associated with the forecast.**

Step 30: Multistep forecasting

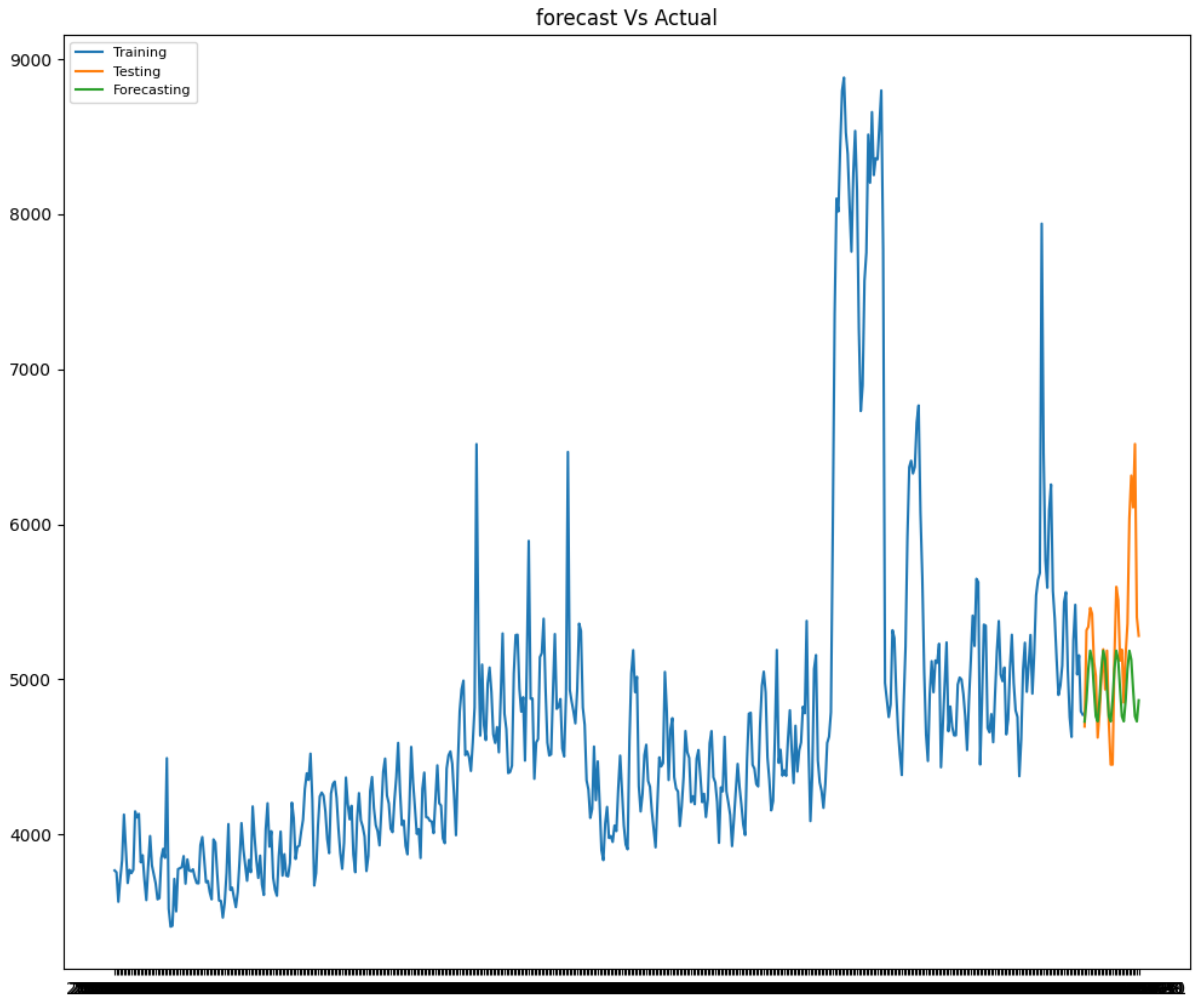
Multi-step forecasting extends traditional time series forecasting by predicting multiple future time steps, enabling decision-makers to plan and make informed decisions based on predictions of future trends and patterns in the data.

- **Multi-step forecasting is widely used in various fields:**
- **Energy demand forecasting**
- **Stock market prediction**
- **Sales forecasting**
- **Weather forecasting**
- **Supply chain optimization**
- **Long-term planning in finance and economics.**

```
# Developing the model from the train data

model = ARIMA(train, order = (4,1,3))
fitted = model.fit()

# Forecasting for next 30 periods
fc = fitted.forecast(30, alpha = 0.05) # 95% confidence interval

# Above code returns an array and hence convert it into series
fc_series = pd.Series(fc, index = test.index)

# Plot to compare the predicted values with actual test values
plt.figure(figsize = (12,10), dpi = 100)

plt.plot(train, label = "Training")
plt.plot(test, label = "Testing")
plt.plot(fc_series, label = "Forecasting")

plt.title("forecast Vs Actual")
```

```
plt.legend(loc = "upper left", fontsize = 8)
```

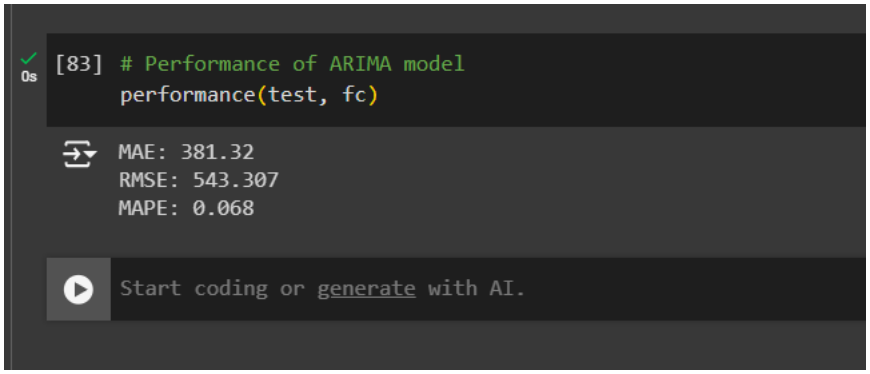
Plot



The curve starts at a high precision value and then decreases gradually as the recall increases. This suggests that the model is initially good at identifying positive cases, but as it tries to identify more positive cases, it starts to include more false positives.

```
# Performance of ARIMA model
performance(test, fc)
```

Screenshot



Step 31: SARIMA (Seasonal Autoregressive Integrated Moving Average)

- **SARIMA is a time series forecasting model that extends the ARIMA model to account for seasonality in the data.**
- **SARIMAX allows you to leverage additional information from external sources, potentially improving the accuracy of forecasts.**
- **It provides flexibility in modeling complex time series data with both endogenous and exogenous components.**
- **ARIMAX models are relatively interpretable compared to some machine learning approaches, making it easier to understand the underlying relationships in the data.**

```
Df3 = pd.read_csv('/content/Exog_Campaign_eng')
```

```
print(df3.head())
```

```
df3.shape
```

```
df3['Exog'].to_numpy()
```

Screenshot

```
[94] df3= pd.read_csv('/content/Exog_Campaign_eng')
```

```
[95] print(df3.head())
```

```
   Exog
0     0
1     0
2     0
3     0
4     0
```

```
[96] df3.shape
```

```
(550, 1)
```

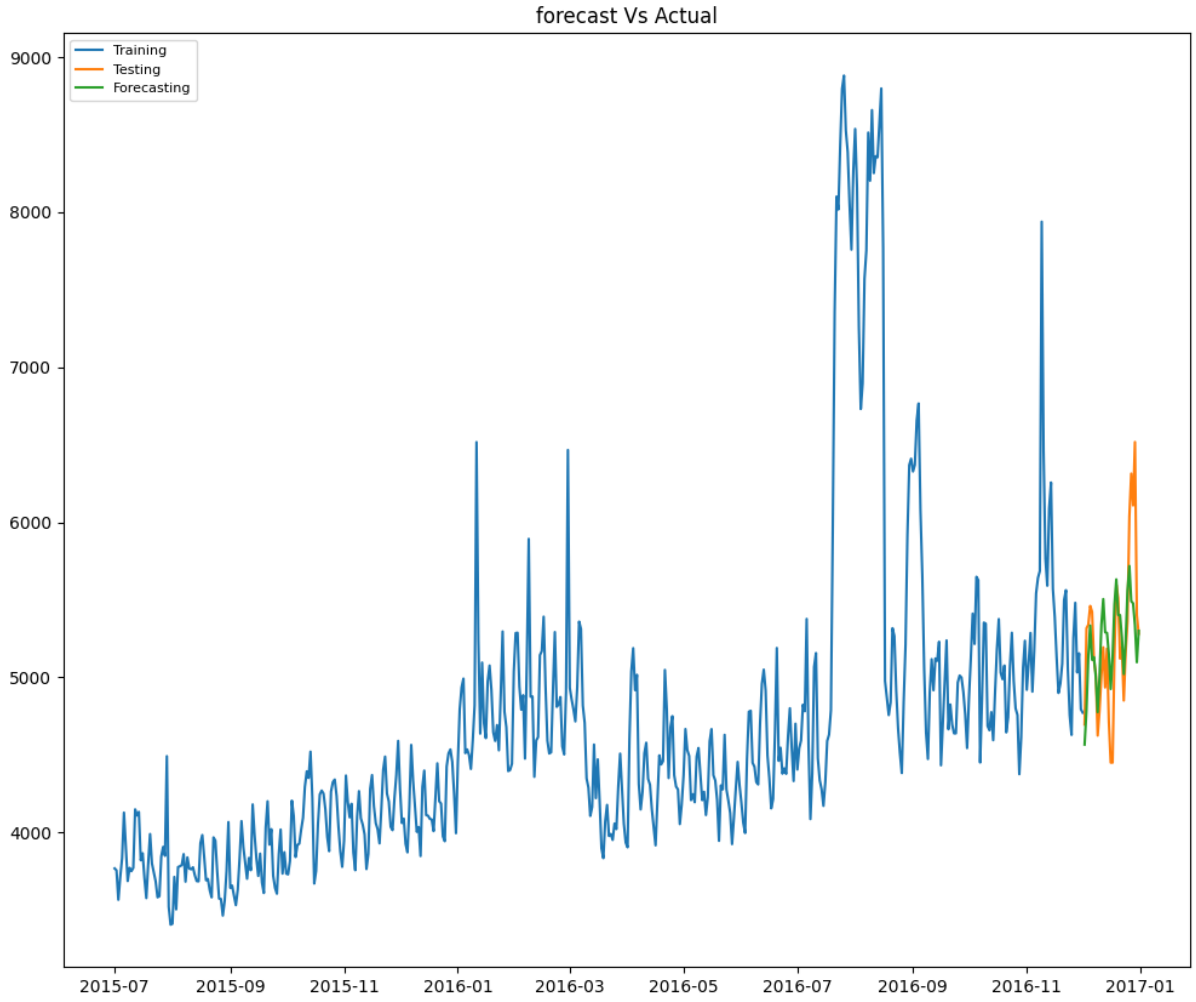
```
df3['Exog'].to_numpy()
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0])
```

```
# Developing SARIMAX model
import statsmodels.api as sm

model = sm.tsa.statespace.SARIMAX(train, df3 = df3[:-30], order =
(4,1,3), seasonal_order = (1,1,1,7))
results = model.fit()

fc = results.forecast(30, exog = pd.DataFrame(df3[-30:]))

# Pandas series
fc_series = pd.Series(fc)

# Plot
plt.figure(figsize = (12,10), dpi = 100)

train.index = train.index.astype('datetime64[ns]')
test.index = test.index.astype('datetime64[ns]')

plt.plot(train, label = "Training")
plt.plot(test, label = "Testing")
```

```
plt.plot(fc_series, label = "Forecasting")

plt.title("forecast Vs Actual")
plt.legend(loc = "upper left", fontsize = 8)
```

Plot

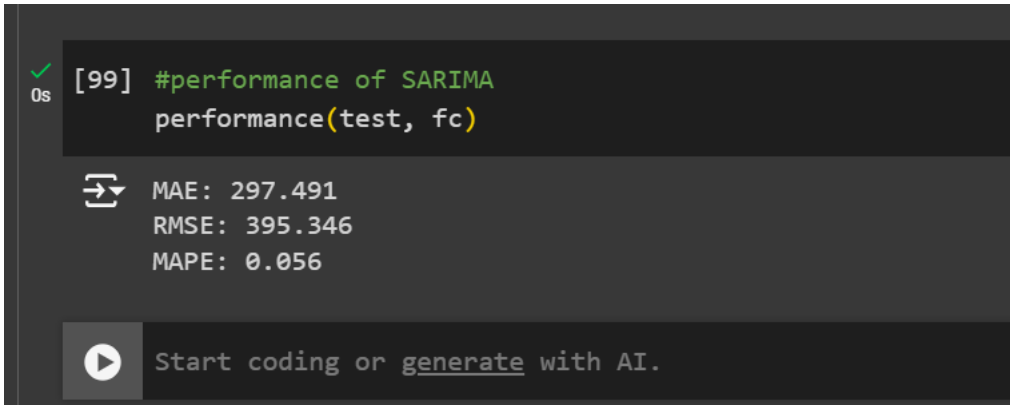


- The x-axis shows the date, ranging from July 2015 to November 2016.
- The y-axis shows the forecasted value, but the label is cut off so it's difficult to say what units it's measured in.
- The graph shows three lines:

- A solid blue line represents the forecasted value. This is the value that the model predicts for each date.
- Two dashed blue lines, above and below the solid line, represent the upper and lower bounds of the confidence interval. The confidence interval is a range of values that the forecast is likely to fall within, with a certain level of certainty (usually 95%).

```
#performance of SARIMA
performance(test, fc)
```

Screenshot



Steps 32 Facebook Prophet

Facebook Prophet is an open-source software library developed by Facebook's Core Data Science team for forecasting time series data. It's particularly useful for situations where the data exhibits trends, seasonality, and potential holidays that might influence the data.

```
ts.head()
```

Screenshot

```
[100] ts.head() #ts is time series of data

index
2015-07-01    3767.328604
2015-07-02    3755.158765
2015-07-03    3565.225696
2015-07-04    3711.782932
2015-07-05    3833.433025
Name: en, dtype: float64

Start coding or generate with AI.
```

```
ts_new = ts.reset_index()
```

```
ts_new["holiday"] = df3["Exog"]
```

```
ts_new.rename({"index":"ds", "en":"y"}, axis = 1, inplace = True)
ts_new.head()
```

Screenshot

```
[101] ts_new = ts.reset_index()

[102] ts_new["holiday"] = df3["Exog"]

ts_new.rename({"index":"ds", "en":"y"}, axis = 1, inplace = True)
ts_new.head()
```

	ds	y	holiday
0	2015-07-01	3767.328604	0
1	2015-07-02	3755.158765	0
2	2015-07-03	3565.225696	0
3	2015-07-04	3711.782932	0
4	2015-07-05	3833.433025	0

The code snippet prepares a new data frame ts_new that is compatible with Facebook Prophet for time series forecasting. It creates the specific column names ("ds" and "y") that Prophet expects and incorporates potential holiday information from another data frame (df3).

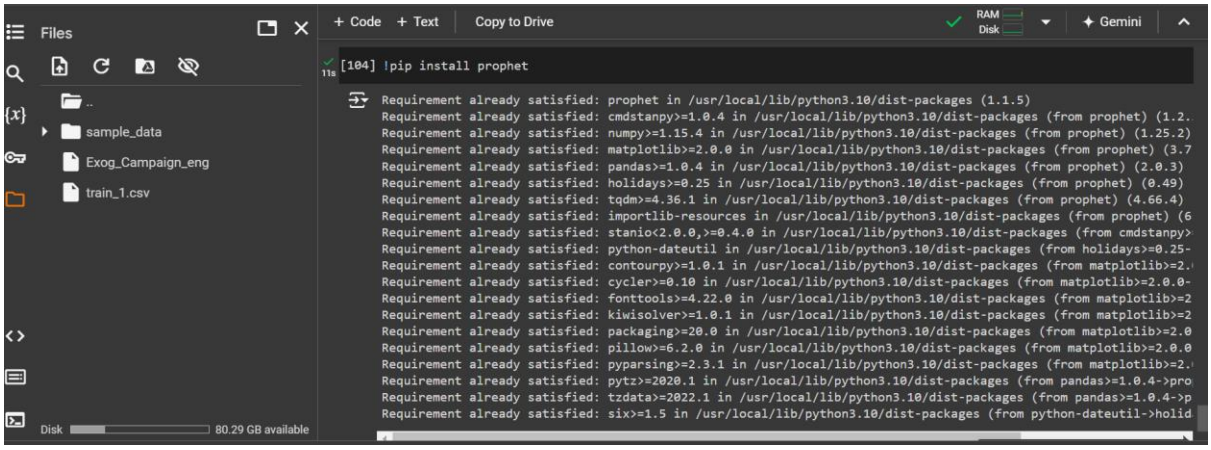
Step 33: fbprophet without exog data

For this we need to install a new package

```
!pip install prophet
```

!pip install prophet, it instructs pip to download the Prophet package from the Python Package Index (PyPI) repository and install it in your Python environment, making the Prophet library available for use in Python scripts

Screenshot

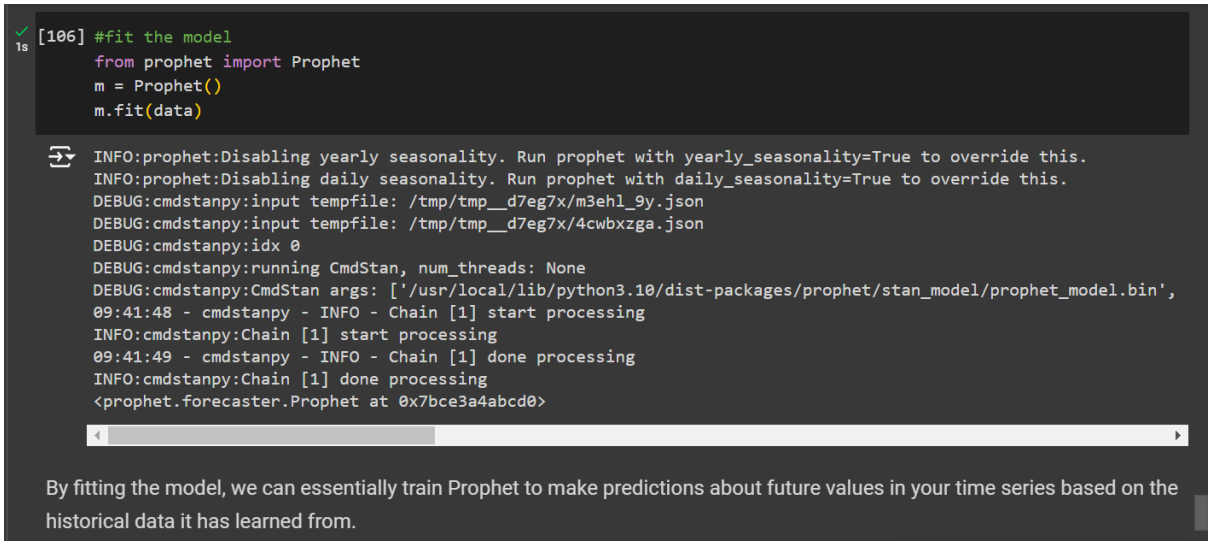



```
data = pd.DataFrame({
    'ds': pd.date_range(start='2020-01-01', periods=100, freq='D'),
    'y': range(100)
})
```

It creates a pandas DataFrame named data that contains 100 days of time series data, ranging from January 1st, 2020, to March 17th, 2020 (inclusive).

```
#fit the model
from prophet import Prophet
m = Prophet()
m.fit(data)
```

Screenshot



Step 34: Make future predictions:

Create a Data Frame with future dates for which you want to make predictions.

```
#make future predictions
future = m.make_future_dataframe(periods=30)
future.tail()
```

Screenshot

```
[107] #make future predictions
      future = m.make_future_dataframe(periods=30)
      future.tail()
```

	ds
125	2020-05-05
126	2020-05-06
127	2020-05-07
128	2020-05-08
129	2020-05-09

These dates will typically start from the last date in your original data and extend for the specified number of periods (30 days in this case).

Step 35: Visualize the Results

```
#visualise the results
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Screenshot

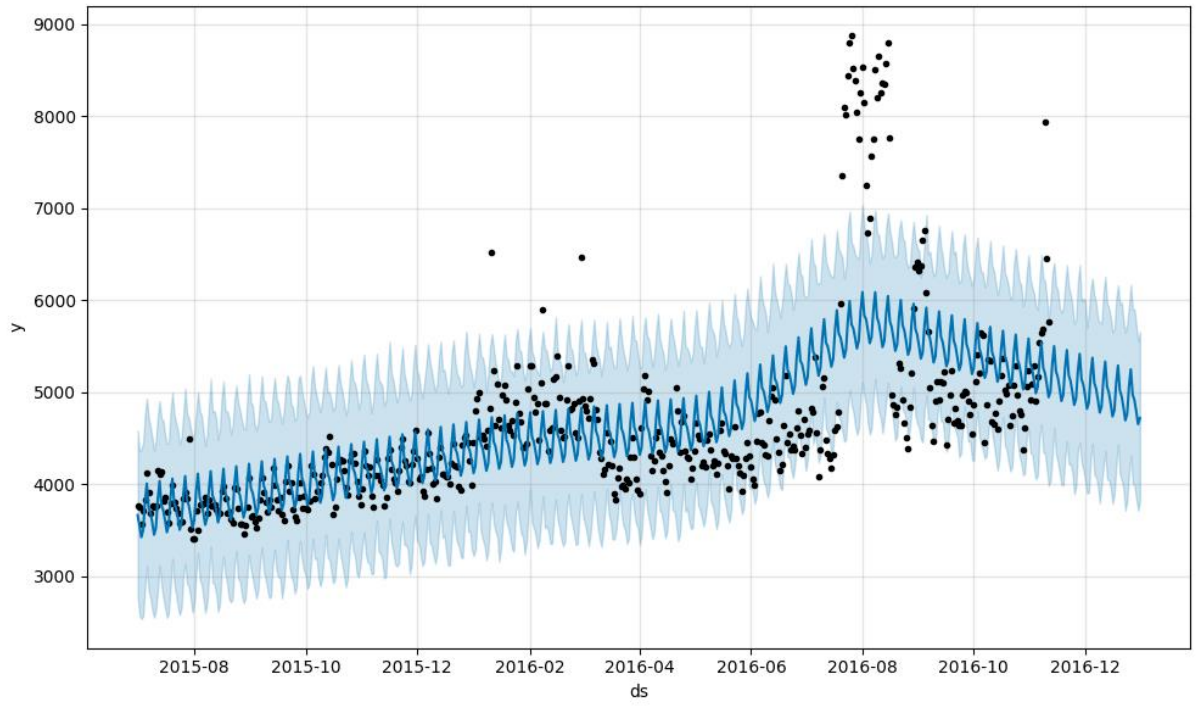
```
[108] #visualise the results
      forecast = m.predict(future)
      forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

	ds	yhat	yhat_lower	yhat_upper
125	2020-05-05	124.999952	124.995893	125.003635
126	2020-05-06	125.999941	125.995679	126.003608
127	2020-05-07	126.999964	126.995292	127.003904
128	2020-05-08	127.999992	127.995476	128.003973
129	2020-05-09	129.000011	128.994939	129.004170

Here ds means date stamp. These dates will typically start from the last date in your original data and extend for the specified number of periods (30 days in this case).

```
m = Prophet()
m.fit(ts_new.loc[:,["ds","y"]][:-50])
future = m.make_future_dataframe(periods = 50, freq = "D",)
forecast = m.predict(future)
m.plot(forecast);
```

Plot



The time series plot in the image is a helpful way to visualize the Prophet model's predictions for future values. It not only shows the expected value (yhat) but also conveys the level of uncertainty associated with those predictions (prediction interval).

Step 36: Extract column of forecast

```
forecast.columns
```

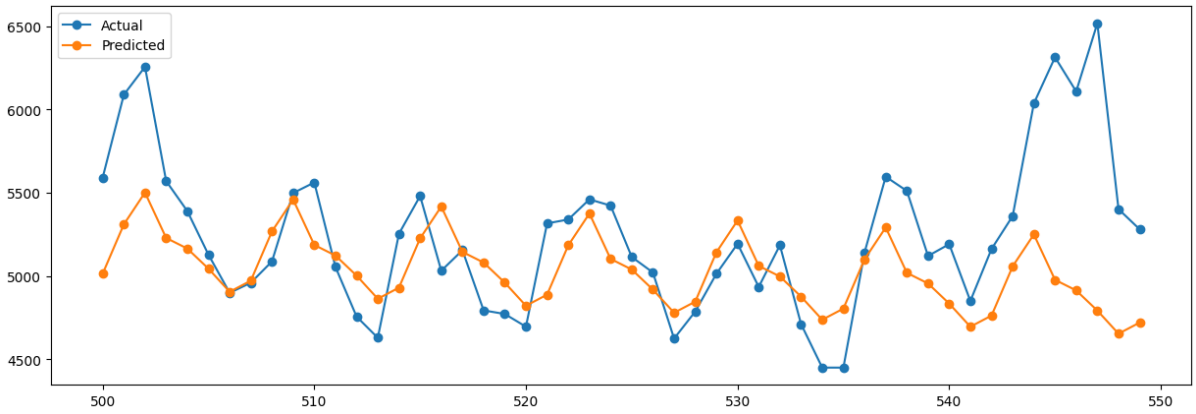
Screenshot

```
forecast.columns
```

```
Index(['ds', 'trend', 'yhat_lower', 'yhat_upper', 'trend_lower', 'trend_upper',
       'additive_terms', 'additive_terms_lower', 'additive_terms_upper',
       'weekly', 'weekly_lower', 'weekly_upper', 'multiplicative_terms',
       'multiplicative_terms_lower', 'multiplicative_terms_upper', 'yhat'],
      dtype='object')
```

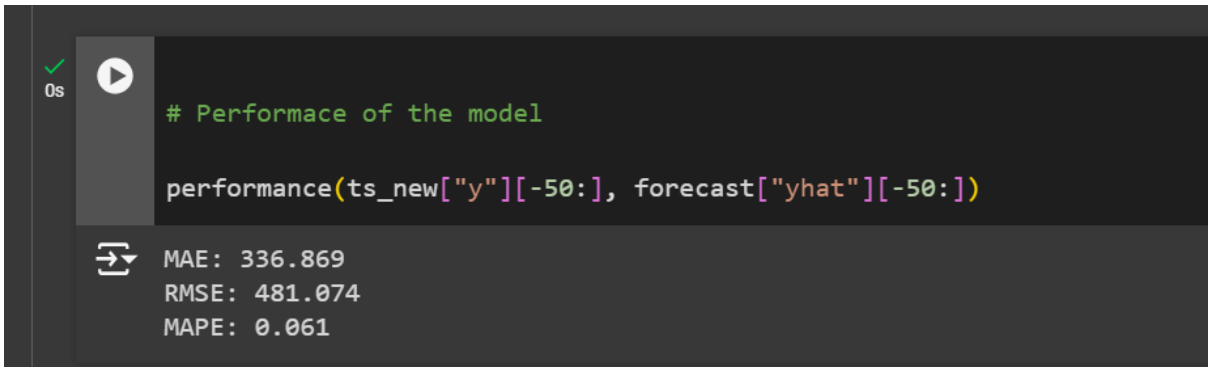
```
ts_new["y"][-50:].plot(style = '-o', figsize = (15,5), label = "Actual")
forecast["yhat"][-50:].plot(style = '-o', figsize = (15,5), label = "Predicted")
plt.legend()
```

Plot




```
# Performace of the model

performance(ts_new["y"][-50:], forecast["yhat"][-50:])
```

Screenshot

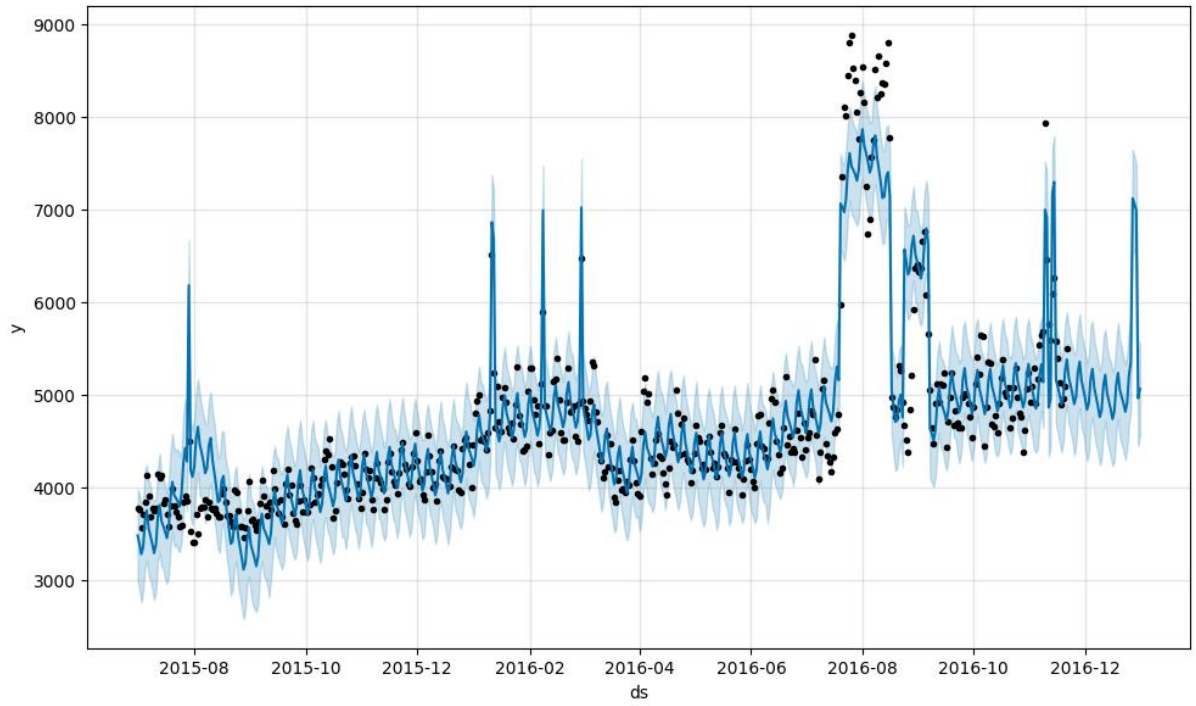



```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

Step 37: fbprophet with exogenous variable

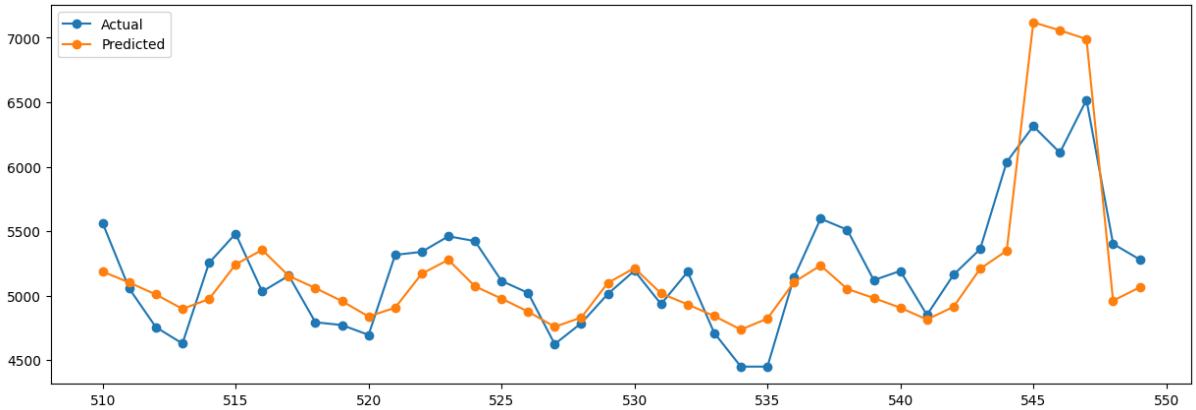
```
m2 = Prophet(yearly_seasonality = True, weekly_seasonality = True)
m2.add_regressor("holiday") # adding holidays data in the m2
m2.fit(ts_new[:-40])
forecast2 = m2.predict(ts_new)
fig = m2.plot(forecast2)
```

Plot



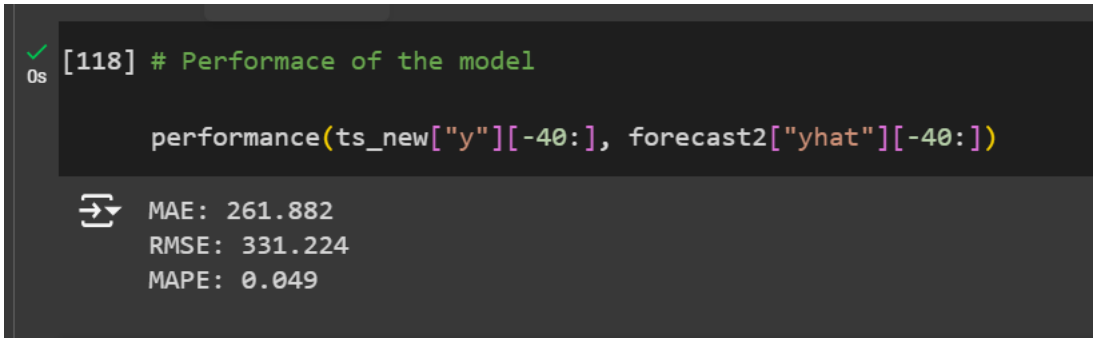

```
# Comaprision of Predicted values with comparsion
ts_new["y"][-40:].plot(style = '-o', figsize = (15,5), label = "Actual")
forecast2["yhat"][-40:].plot(style = '-o', figsize = (15,5), label =
"Predicted")
plt.legend()
```

Plot



```
# Performace of the model

performance(ts_new["y"][-40:], forecast2["yhat"][-40:])
```

Screenshot




```
# Prepare data
data = pd.Series(range(100), index=pd.date_range(start='2020-01-01', periods=100, freq='D'))
```

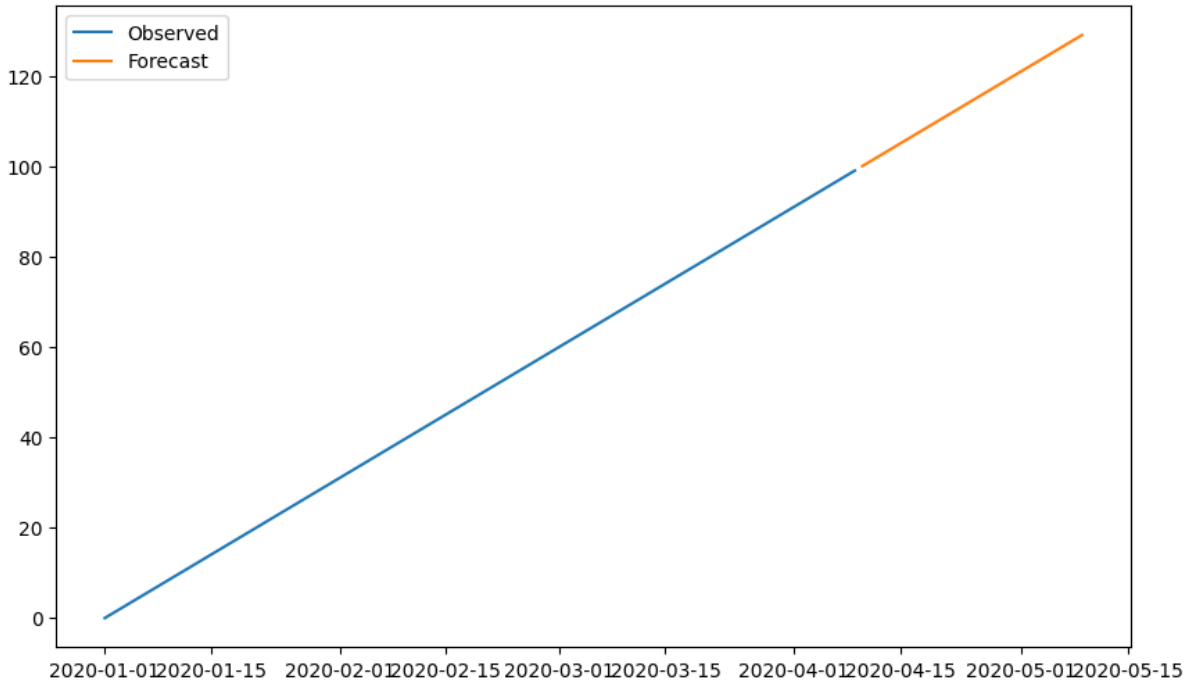
```
# Fit the SARIMAX model
model = SARIMAX(data, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
results = model.fit()
```

```
# Make future predictions
forecast = results.get_forecast(steps=30)
forecast_index = pd.date_range(start=data.index[-1] + pd.Timedelta(days=1), periods=30, freq='D')
forecast_values = forecast.predicted_mean
```

Steps: 38 Visualize the Result

```
# Visualize the results
plt.figure(figsize=(10, 6))
plt.plot(data.index, data, label='Observed')
plt.plot(forecast_index, forecast_values, label='Forecast')
plt.legend()
plt.show()
```

Plot:



Steps 39 : Insights And Recommendations

Recommendations

- **Ads should be shown only during Sunday, Monday, Tuesday and Wednesday as these days have highest and average viewers on page and it's periodic by osbservations of predicted data**

- **Ads budget should be divided in proportion to percentage of viewers page-language wise, for example english-pages have the most viewers and chinese-pages have least viewers so maximum part of budget should be allocated for english page and minimum part of budget should be allocated for chinese page**

Insights

- **Viewers on page is very low during Thursday, Friday, Saturday**
- **Viewers on page is very high during Sunday, Monday, Tuesday**
- **Viewers on page is average on Wednesday**
- **Most viewed page by languges are english(33.65%), spanish(21.44%), russian(15%), japanese(10.8%), french(7.13%), german(7%), chinese(5%)**