INDEX

CHAPTER-1

CHAPTER-2

CHAPTER-3

CHAPTER-4

INTRODUCTION

India is a country that attracts millions of tourists every year and is considered one of the most popular tourist destinations in the world. These visitors are attracted to its natural beauty and cultural heritage. It has many historic sites as well like the Taj Mahal, Golden Temple, Red Fort, etc. With so much to offer, it's no surprise that tourism is one of the main sources of revenue for the country!

Tourism in India has been seen as a major tool to bring about socio-economic development to the people of the country. India's tourism industry has grown steadily in recent years. Not only this, India Tourism industry also provides employment opportunities to several people from all parts of the world. There are ample numbers of hotels, resorts, food joints and various other amenities available for tourists arriving in India.

No wonder Tourism is an important foreign exchange earner for India. The Department of Tourism, Ministry of Civil Aviation, Government of India is the apex body for the largest development and promotion of Indian tourism. To promote tourism in India, the government has been implementing a number of campaigns and schemes. These include Incredible India campaign along with a

number of planned events such as the International Travel Mart, Destination India Exhibitions worldwide and Indian Cultural Festivals Abroad to attract foreign tourists.

India is a vast country with a lot to offer. Therefore, it comes as no surprise that millions of people pay India a visit every year. The country offers a variety of sights and sounds to enjoy, from the vibrant cities to the peaceful countryside. With its diversity of tourist attractions, India is also known for yoga and meditation. From the snow-capped Himalayas to the tropical beaches of Goa, from the wildlife of Assam to the largest tea field of Kerala, there is something new in every corner of this country. Moreover, with major cities such as Mumbai, Bangalore, Chandigarh, and Delhi being so well connected by air, road, and rail networks from across the globe, it's easy to get around and explore some of the most iconic landmarks in India.

GOA TOURISM

TEMPLES IN INDIA

WILDLIFE OF INDIA

A country with a rich and colourful history, India is home to some of the world's most fascinating cultures. India is a country located in South Asia and the capital of India is New Delhi. The Government of India, the Constitutional Republic represents a highly diverse population of thousands of ethnic groups and the many languages spoken in the country.

CULTURE OF INDIA

India has a rich cultural heritage, reflected in its diverse population. The country's culture has been shaped by various external and internal influences. Over the centuries, India has seen a significant fusion of Hindus, Muslims, Jains, Sikhs and Buddhists. These religions are collectively known as Indian religions as they originated in India.

"Unity in diversity" – these are words that do not just hold meaning but can be applied to a country like India. From the time of Mauryas, Cholas and Mughals to the period of the British Empire, India has always been famous for its traditions and hospitality.

Due to warm relations and a sense of celebration, India has a special place in the global community. Its vibrant culture is an amalgamation of religions, festivals, food, arts, crafts and dance. The country is vibrant, eclectic, and attracts many tourists for its culture. Popularly known as the land of God, the country of India has everything from culture to values, customs and traditions that defines its beauty. Also, explore the beauty of Indian Tribes that adds colour and beauty to the culture of India.

FOREIGN TOURIST ARRIVALS AND FOREIGN EXCHANGE EARNINGS

These data is taken from Wikipedia.

Foreign tourist arrivals in India (1997–2020)[26]

Year	Arrivals (millions)	% change
1997	2.37	3.8
1998	2.36	−0.7
1999	2.48	5.2
2000	2.65	6.7
2001	2.54	−4.2
2002	2.38	−6.0
2003	2.73	14.3
2004	3.46	26.8
2005	3.92	13.3
2006	4.45	13.5

Foreign exchange earnings from tourism in India (1997–2020)[26]

Year	Earnings (US$ million)	% change	Earnings (₹ crores)	% change
1997	2,889	2.0	10,511	4.6
1998	2,948	2.0	12,150	15.6
1999	3,009	2.1	12,951	6.6
2000	3,460	15	15,626	20.7
2001	3,198	−7.6	15,083	−3.5
2002	3,103	−3.0	15,064	−0.1
2003	4,463	43.8	20,729	37.6
2004	6,170	38.2	27,944	34.8
2005	7,493	21.4	33,123	18.5

2007	5.08	14.3
2008	5.28	4.0
2009	5.17	−2.2
2010	5.78	11.8
2011	6.31	9.2
2012	6.58	4.3
2013	6.97	5.9
2014	7.68	10.2
2015	8.03	4.5
2016	8.80	9.7
2017	10.04	14.0
2018	10.56	5.2
2019	10.93	3.5
2020	2.74	-74.9

2006	8,634	15.2	39,025	17.8
2007	10,729	24.3	44,362	13.7
2008	11,832	10.3	51,294	15.6
2009	11,136	−5.9	53,754	4.8
2010	14,193	27.5	66,172	23.1
2011	16,564	16.7	83,036	25.5
2012	17,737	7.1	95,607	15.1
2013	18,445	4.0	107,563	12.5
2014	20,236	9.7	120,367	11.9
2015	21,071	4.1	134,844	12
2016	22,923	9.1	154,146	14.3
2017	27,310	19.1	177,874	15.4
2018	28,586	4.7	194,881	9.6
2019	30,058	5.1	211,661	8.6
2020	6,958	-76.8	50,136	-76.3

Source countries for foreign tourist arrivals in India in 2019[26]

Rank	Country	Number	Share in %
1	Bangladesh	2,577,727	
2	United States	1,512,032	
3	United Kingdom	1,000,292	
4	Australia	367,241	
5	Canada	351,859	
6	China (mainland)	339,442	
7	Malaysia	334,579	
8	Sri Lanka	330,861	
9	Germany	264,973	
10	Russia	251,319	
	Total of top 10	7,330,325	

OBJECTIVE OF PROJECT

This study aims to compare various quantitative models to forecast monthly foreign tourist arrivals (FTAs) to India . The models which are considered here include autoregressive integrated moving average (ARIMA) model. A model based on combination of single forecast values using simple average (SA) method has also been applied. The forecasting performance of these models have been compared under mean absolute percentage error (MAPE) and U-statistic (Ustat) criteria. Empirical findings suggest that the combination model gives better forecast of FTAs to India relative to other individual time series models considered here.

Keywords : Foreign tourist arrivals, Time series models, Forecast comparisons.

CONCLUSION:

Time series analysis of Indian tourist trends holds significant potential for understanding patterns, forecasting arrivals, policy formulation, and evaluating marketing strategies. By leveraging historical data and incorporating external factors, stakeholders can make informed decisions, allocate resources effectively, and promote sustainable tourism growth. Overcoming challenges related to data availability, quality, and modeling complexity is essential to ensure accurate predictions and meaningful insights. Time series analysis enables the tourism industry to adapt to changing trends.

CHAPTER=2

DATASET

Date	Tourist
31-01-2001	283750
28-02-2001	262306
31-03-2001	248965
30-04-2001	185338
31-05-2001	151098
30-06-2001	176716
31-07-2001	224432
31-08-2001	196517
30-09-2001	162326
31-10-2001	181605
30-11-2001	209685
31-12-2001	254544
31-01-2002	228150
28-02-2002	241133
31-03-2002	216839
30-04-2002	159789
31-05-2002	144571
30-06-2002	134566
31-07-2002	178231
31-08-2002	162594
30-09-2002	163089
31-10-2002	213267
30-11-2002	245661
31-12-2002	296474
31-01-2003	274215
28-02-2003	262692
31-03-2003	218473
30-04-2003	160941
31-05-2003	141508
30-06-2003	176324
31-07-2003	225359
31-08-2003	204940
30-09-2003	191339
31-10-2003	260569
30-11-2003	290583
31-12-2003	319271
31-01-2004	337345

29-02-2004	331697
31-03-2004	293185
30-04-2004	223884
31-05-2004	185502
30-06-2004	223122
31-07-2004	272456
31-08-2004	253301
30-09-2004	226773
31-10-2004	307447
30-11-2004	385238
31-12-2004	417527
31-01-2005	385977
28-02-2005	369844
31-03-2005	352094
30-04-2005	248416
31-05-2005	225394
30-06-2005	246970
31-07-2005	307870
31-08-2005	273856
30-09-2005	257184
31-10-2005	347757
30-11-2005	423837
31-12-2005	479411
31-01-2006	459489
28-02-2006	439090
31-03-2006	391009
30-04-2006	309208
31-05-2006	255008
30-06-2006	278370
31-07-2006	337332
31-08-2006	304387
30-09-2006	297891
31-10-2006	391399
30-11-2006	442413
31-12-2006	541571
31-01-2007	535631
28-02-2007	501692
31-03-2007	472494
30-04-2007	350550
31-05-2007	277017
30-06-2007	310364
31-07-2007	399866
31-08-2007	358446

30-09-2007	301892
31-10-2007	444564
30-11-2007	532428
31-12-2007	596560
31-01-2008	591337
29-02-2008	561393
31-03-2008	541478
30-04-2008	384203
31-05-2008	300840
30-06-2008	340159
31-07-2008	429456
31-08-2008	391423
30-09-2008	330874
31-10-2008	452566
30-11-2008	521247
31-12-2008	521990
31-01-2009	481308
28-02-2009	489787
31-03-2009	442062
30-04-2009	347544
31-05-2009	305183
30-06-2009	352353
31-07-2009	432900
31-08-2009	369707
30-09-2009	330707
31-10-2009	458849
30-11-2009	541524
31-12-2009	615775
31-01-2010	568719
28-02-2010	552152
31-03-2010	512152
30-04-2010	371956
31-05-2010	332087
30-06-2010	384642
31-07-2010	466715
31-08-2010	422173
30-09-2010	369821
31-10-2010	507093
30-11-2010	608178
31-12-2010	680004
31-01-2011	622713
28-02-2011	627719
31-03-2011	535613

30-04-2011	446511
31-05-2011	383439
30-06-2011	405464
31-07-2011	475544
31-08-2011	428490
30-09-2011	417478
31-10-2011	559641
30-11-2011	669767
31-12-2011	736843
31-01-2012	681002
29-02-2012	681193
31-03-2012	606456
30-04-2012	447581
31-05-2012	374476
30-06-2012	433390
31-07-2012	485808
31-08-2012	445632
30-09-2012	411562
31-10-2012	556488
30-11-2012	701185
31-12-2012	752972
31-01-2013	720321
28-02-2013	688569
31-03-2013	639530
30-04-2013	450580
31-05-2013	417453
30-06-2013	451223
31-07-2013	506427
31-08-2013	486338
30-09-2013	453561
31-10-2013	598095
30-11-2013	733923
31-12-2013	821581
31-01-2014	757786
28-02-2014	755678
31-03-2014	690441
30-04-2014	535321
31-05-2014	465043
30-06-2014	502028
31-07-2014	568871
31-08-2014	575750
30-09-2014	509142
31-10-2014	668398

30-11-2014	765497
31-12-2014	885144
31-01-2015	790854
28-02-2015	761007
31-03-2015	729154
30-04-2015	541551
31-05-2015	509869
30-06-2015	512341
31-07-2015	628323
31-08-2015	599478
30-09-2015	542600
31-10-2015	683286
30-11-2015	815947
31-12-2015	912723
31-01-2016	844533
29-02-2016	848782
31-03-2016	809107
30-04-2016	592004
31-05-2016	527466
30-06-2016	546972
31-07-2016	733834
31-08-2016	652111
30-09-2016	608177
31-10-2016	741770
30-11-2016	878280
31-12-2016	1021375
31-01-2017	964109
28-02-2017	931025
31-03-2017	885936
30-04-2017	717899
31-05-2017	622408
30-06-2017	663470
31-07-2017	779309
31-08-2017	719129
30-09-2017	719964
31-10-2017	866976
30-11-2017	997738
31-12-2017	1167840
31-01-2018	1045027
28-02-2018	1049259
31-03-2018	1021539
30-04-2018	745033
31-05-2018	606513

30-06-2018	683935
31-07-2018	806493
31-08-2018	785993
30-09-2018	719894
31-10-2018	890223
30-11-2018	1012569
31-12-2018	1191498
31-01-2019	1111040
28-02-2019	1090516
31-03-2019	978236
30-04-2019	774651
31-05-2019	615136
30-06-2019	726446
31-07-2019	818125
31-08-2019	800837
30-09-2019	751513
31-10-2019	945017
30-11-2019	1092440
31-12-2019	1226398
31-01-2020	1119250
29-02-2020	1018440
31-03-2020	328304
30-04-2020	2820
31-05-2020	3764
30-06-2020	8590
31-07-2020	12655
31-08-2020	19761
30-09-2020	28167
31-10-2020	41494
30-11-2020	70977
31-12-2020	90544
31-01-2021	94662
28-02-2021	110312
31-03-2021	133768
30-04-2021	78718
31-05-2021	19765
30-06-2021	36070
31-07-2021	72501
31-08-2021	92728
30-09-2021	115661
31-10-2021	191415
30-11-2021	263867
31-12-2021	317647

SOURCE OF DATA

Project took place after covid-19 pandemic. The data comes from the Indian Ministry of Tourism.

https://data.gov.in/catalog/number-foreign-tourists-india and

https://tourism.gov.in/about-us/about-ministry .

==========

TIME SERIES

Have you ever wanted to predict the future? Perhaps you've wondered whether the stock market will go up or down tomorrow, or if the demand for your company's products will increase or decrease over time. Time series analysis can help you answer these questions and more. It's a powerful tool for understanding patterns in data that change over time, and it has countless applications in fields like finance, meteorology, and marketing. We'll explore the key concepts and techniques of time series analysis, and show how we can use it to gain insights and make predictions that can help make better decisions. So, lets understand these concepts in more detail.

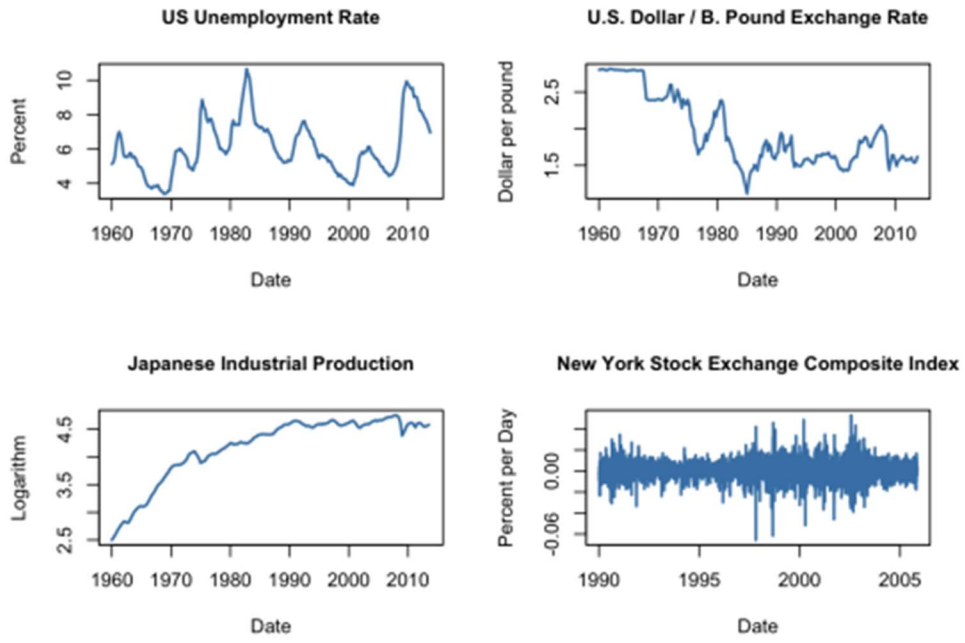
TABLE OF CONTENTS

INTRODUCTION TO TIME SERIES ANALYSIS

Definition:

In simple terms, time series analysis is the study of a sequence of data points ordered in time. This data can be used to identify patterns, trends, and relationships over time.

Examples: Weather records, economic indicators and patient health evolution metrics



❖ Objectives:

The objectives of time series analysis include understanding how variables change over time, identifying the factors that influence these changes, and predicting future values of the time series variable.

❖ Assumptions:

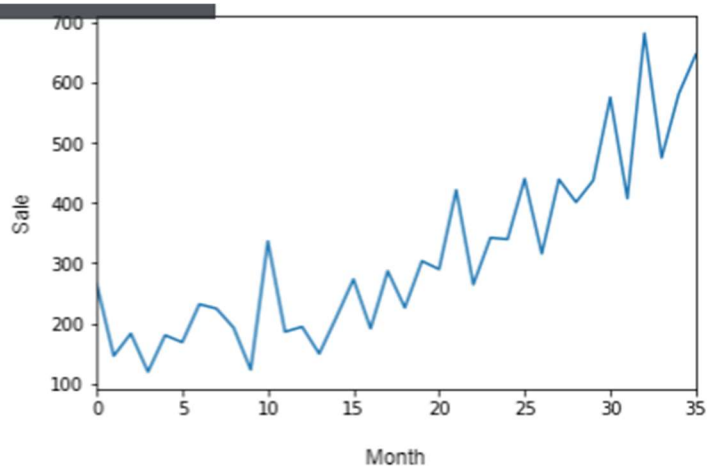
The only assumption in time series analysis is that the data is stationary, meaning that the statistical properties of the data are not changing over time. More precisely, a stationary time series is one in which the mean, variance, and autocorrelation structure of the data are constant over time.This assumption ensures that the analysis is reliable and accurate.

❖ What is Time Series?

Any data recorded with some fixed interval of time is called as time series data. This fixed interval can be hourly, daily, monthly or yearly. e.g. hourly temp reading, daily changing fuel prices, monthly electricity bill, annul company profit report etc. In time series data, time will always be independent variable and there can be one or many dependent variable. Sales forecasting time series with shampoo sales for every month will look like this, In above example since there is only one variable dependent on time so its called as univariate time series. If there are multiple dependent variables, then its called as multivariate time series. Objective of time series analysis is to understand how change in time affect the dependent variables and accordingly predict values for future time intervals.

Sales forecasting time series with shampoo sales for every month will look like this,

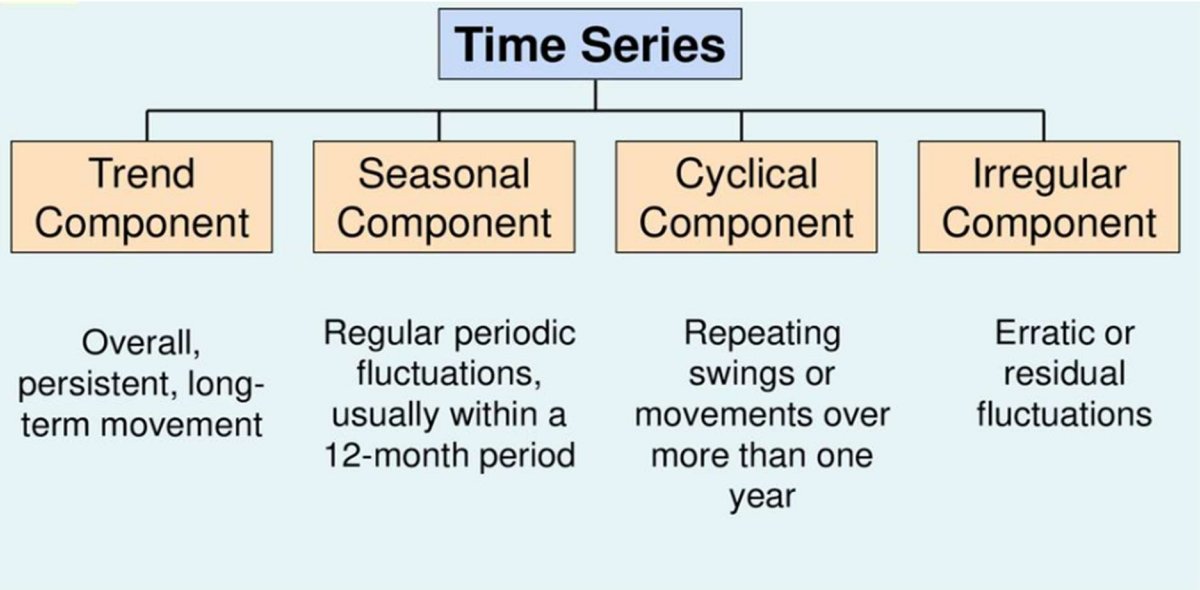
Month	Sale
01-01-2018	266
02-01-2018	145.9
03-01-2018	183.1
04-01-2018	119.3
05-01-2018	180.3
...	...
...	...
...	...
08-03-2020	407.6
09-03-2020	682
10-03-2020	475.3
11-03-2020	581.3
12-03-2020	646.9



In above example since there is only one variable dependent on time so its called as univariate time series. If there are multiple dependent variables, then its called as multivariate time series. Objective of time series analysis is to understand how change in time affect the dependent variables and accordingly predict values for future time intervals.

❖ COMPONENTS OF TIME SERIES ANALYSIS

Time series data can be decomposed into four key components: trend, seasonality, cyclical, and irregularity.

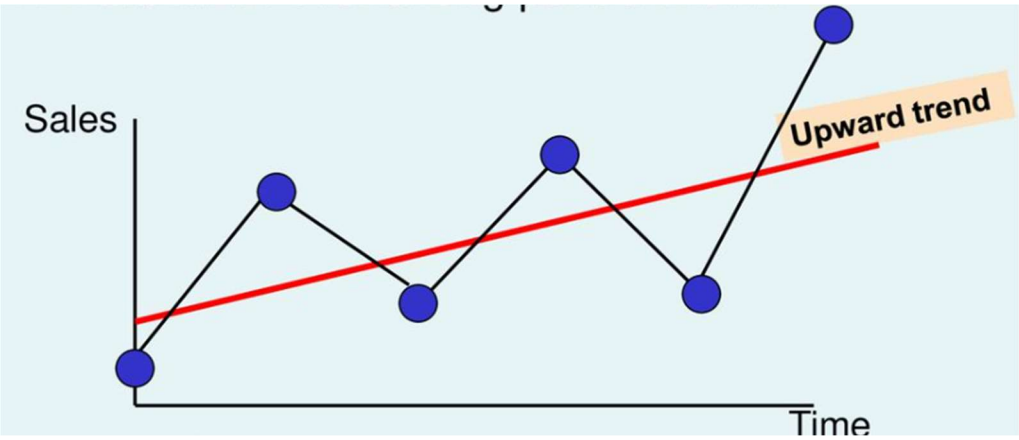


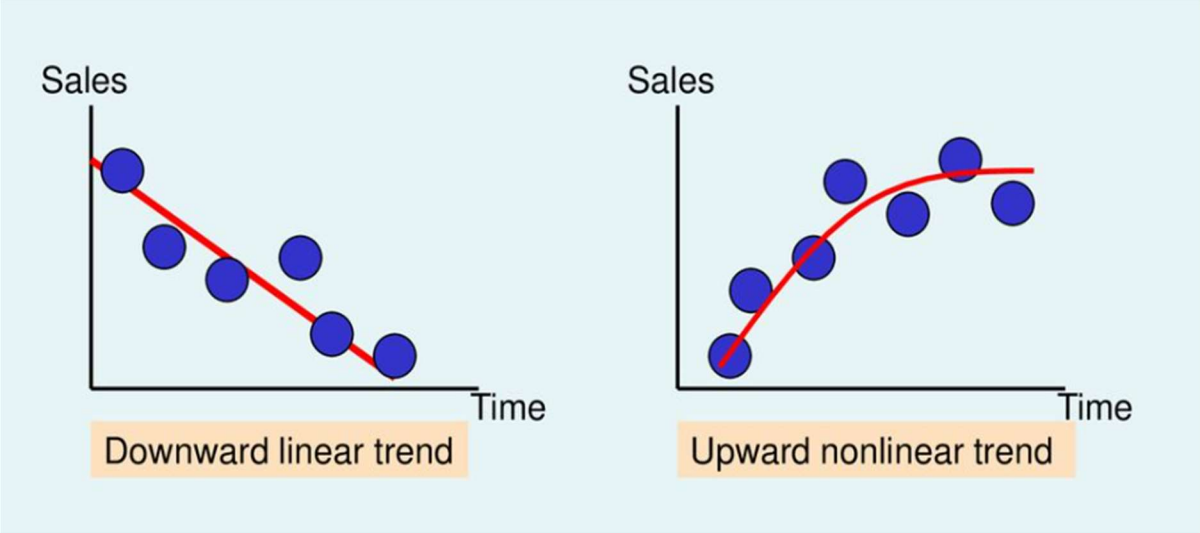
❖ TREND

A trend is a long-term pattern in the data that reflects the overall direction of the series. Trends can be upward, downward, or flat. For example, a company's stock prices might show an upward trend over several years, reflecting the overall growth of the company. We can use techniques like linear regression or moving averages to identify and model a trend. For example = we take dependent variable sales on time then we can see the following results including trends:

- ❖ Long run increase or decrease over time.
- ❖ Data taken over a long period of time.

- Trends can be upward or downward.
- Trends can be linear or non linear.



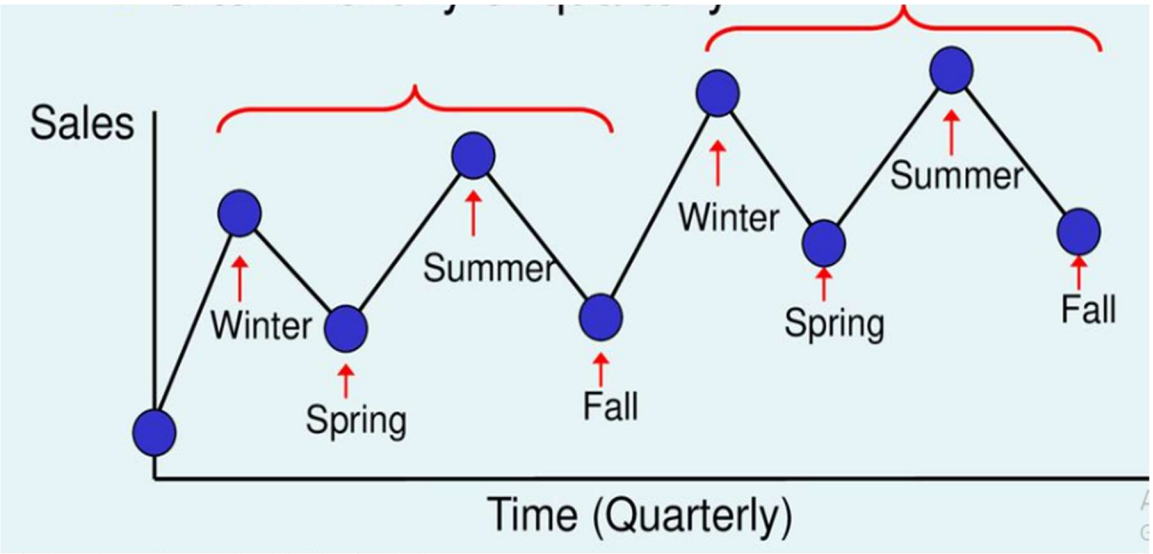


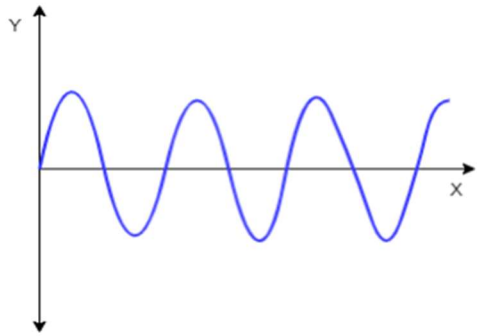
❖ SEASONALITY:

Seasonality refers to patterns in the data that repeat at regular intervals. For example, sales of winter coats might increase every year in the fall and winter months, reflecting seasonal demand.

We can use techniques like seasonal decomposition or Fourier analysis to identify and model seasonality ,again we take the same example of sales to know seasonality. We can see the following result including seasonality:

- ❖ Short term regular wave like pattern
- ❖ Observed within one year.
- ❖ Often monthly or quarterly..



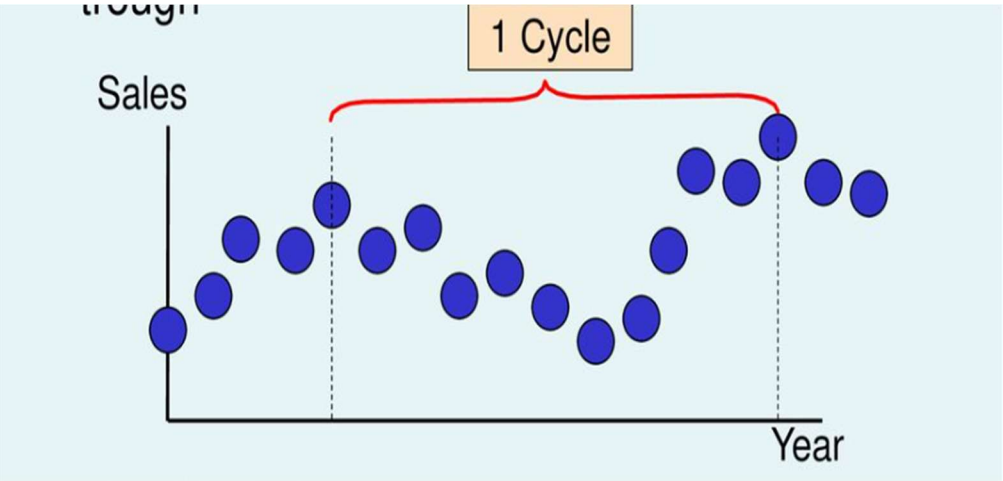


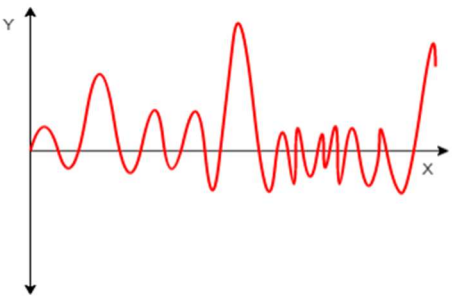
Here x is time and y is sales,from above diagram we can see the the seasonal component in the sales data.

❖ CYCLICAL

Cyclical patterns are similar to trends but are not fixed like seasonality. Instead, cyclical patterns occur when data experiences a rise and fall that is not of a fixed frequency, like the business cycle in economics. We can use techniques like spectral or wavelet analysis to identify and model cyclical patterns.

- ❖ Long term wave like pattern
- ❖ Regularly occur but may very in length.
- ❖ Often measured peak to peak and trough to trough.

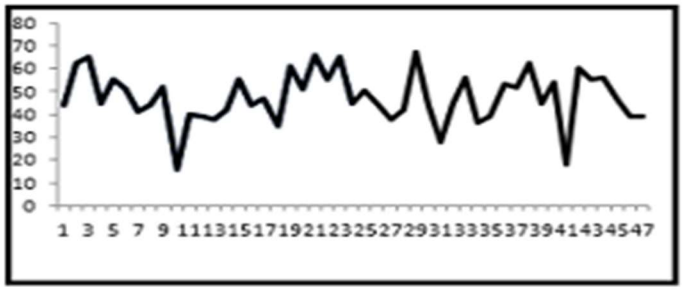




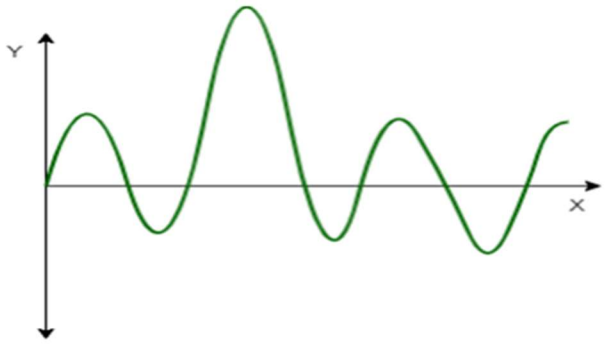
From above we can see the cycle of data.

❖ IRREGULARITY:

Irregularity, or noise, refers to the unpredictable fluctuations in the data that are not explained by trends, seasonality, or cyclical patterns. Randomness is important because it can affect the accuracy of predictions and models. We can use techniques like residual analysis or time series forecasting models to identify and model irregularity.



(d) Irregular



EXAMPLE:

Let's say we are analyzing the monthly sales data of a clothing store. By plotting the data over time, we notice an upward trend in sales over several years. To model this trend, we can use linear regression to fit a straight line to the data. However, we also notice that sales of winter clothing increase during the fall and winter months, indicating seasonality. To model this seasonality, we can use seasonal decomposition to separate the data into its seasonal and non-seasonal components. Additionally, we observe fluctuations that do not repeat at regular intervals, such as changes in fashion trends, representing the cyclical component. We can use techniques like spectral or wavelet analysis to model this cyclical pattern.

❖DATA TYPES AND LIMIT

There are two main types of time series data:

- ❖ stationary and
- ❖ non-stationary.

- **Stationary time series data:**

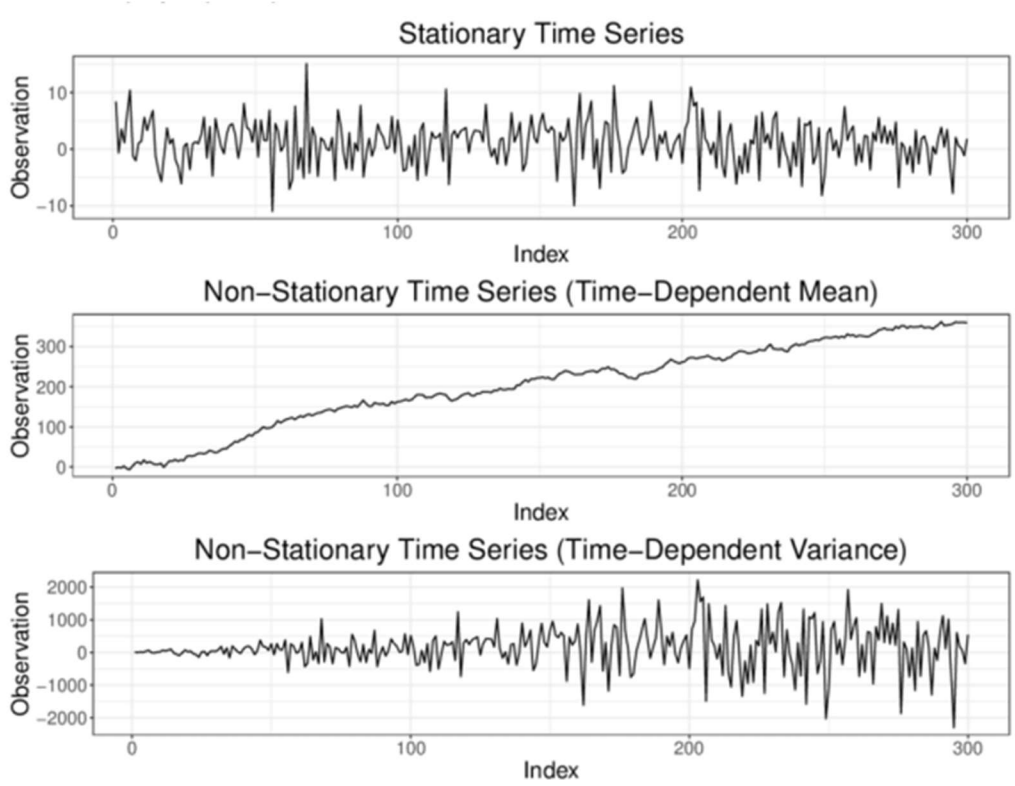
A stationary time series data is one where the statistical properties of the series, such as its mean, variance, and covariance, are constant over time. This means that the mean value of the data

remains constant during the analysis, and the variance remains constant with respect to the time frame. Additionally, the covariance, which measures the relationship between two variables, remains constant over time. In other words, the data does not have any trend, seasonality, cyclical, or irregularity component

- **Non-stationary time series data:**

It is one where the statistical properties of the series change over time. This can occur when the mean, variance, or covariance changes with respect to time. Non-stationary data can have trend, seasonality, cyclical, and irregularity components and can be more challenging to analyze than stationary data. Identifying whether a dataset is stationary or non-stationary is a crucial step in time series analysis, as it determines the appropriate modeling techniques to use.

For example: If the data is non-stationary, it may be necessary to first transform the data to make it stationary before applying any analysis techniques.



❖LIMITATIONS OF TIME SERIES ANALYSIS

- Missing values are not supported by TSA, similar to other models.
- The data points must have a linear relationship for proper analysis.
- Data transformations are mandatory, which can be expensive.
- TSA models typically work best on univariate data and may not perform as well with multivariate datasets.

- TSA assumes that the observations are equally spaced in time, which may not always be the case in real-world scenarios.

- Time series analysis is sensitive to outliers, and the presence of outliers in the data can significantly affect the results of the analysis.

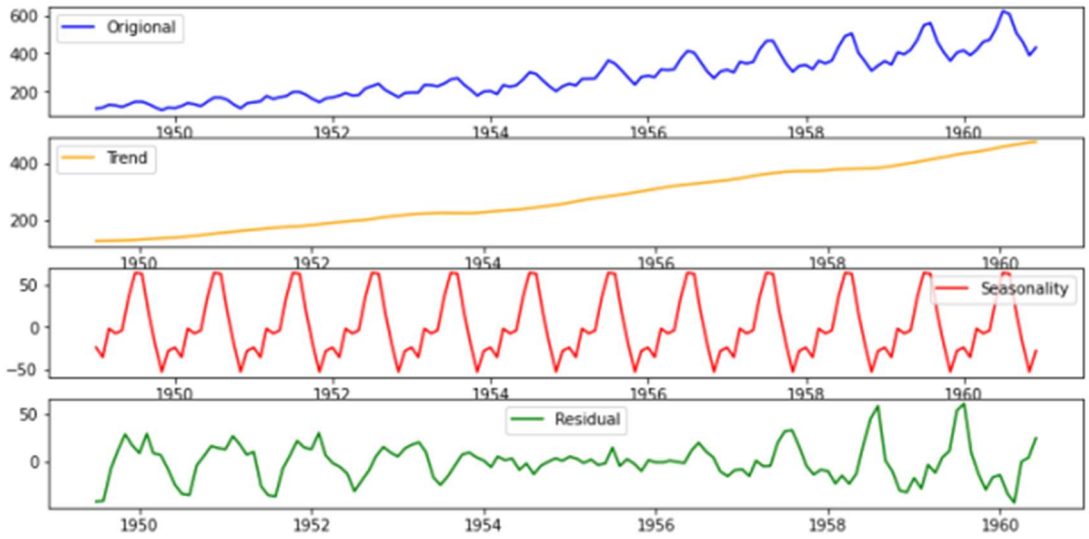
❖ TIME SERIES ANALYSIS

Time series analysis involves the study of patterns in data over time. Let us go through a brief overview of various techniques and tests used for time series analysis.

- **Decomposition of Time Series:**

  Decomposition of time series refers to the process of breaking down a time series into its individual components, namely trend, seasonality, cyclical, and irregularity. This technique is used to understand and analyze the underlying patterns and structures in the data. Decomposition of time series can be done using various techniques, such as moving averages, exponential smoothing, and regression analysis. Once the individual components are identified, they can be modeled and analyzed

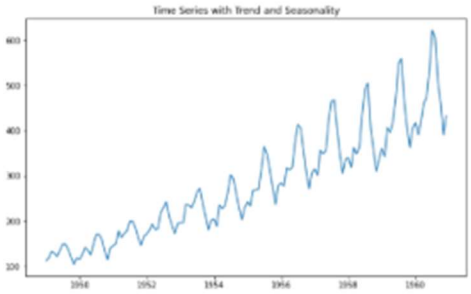
separately to gain insights into the behavior and patterns of the time series data.

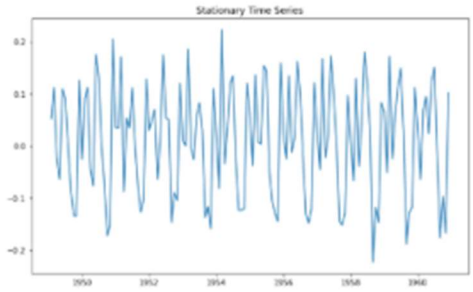


STATIONARITY

Stationary Data: In time series analysis, it is important to work with stationary data to make accurate forecasts and identify patterns. Stationary time series data is one where the statistical values, such as the mean and standard deviation, are constant over time. This means that the data does not have any trend or seasonality, and the statistical properties of the data should not be a function of time. In other words, the data should have a constant mean, variance, and covariance. By removing trends and

seasonality from the time series data, it can be transformed into a stationary series, which is easier to analyze and model.





❖TEST FOR STATIONARITY

One simple way to identify stationarity is by visually examining the plot of the time series data for any noticeable trend or seasonality. However, for real-world data, more advanced techniques like rolling statistics and the Augmented Dickey Fuller test can be utilized to determine the stationarity of the data.

❖ ROLLING STATISTICS

Rolling statistics is a time series analysis technique that involves calculating statistical metrics, such as the mean and standard deviation, within a defined window size as it moves across the

series. The size of the window is typically set based on the frequency of the data, such as daily or monthly. For stationary time series data, the mean and standard deviation should remain constant over time, which can be verified by using rolling statistics. If there are no significant changes in the calculated mean and standard deviation values within the defined window size, then the data is likely to be stationary.

==========

❖ AUGMENTED DICKEY FULLER (ADF) TEST

The Augmented Dickey Fuller (ADF) test is a statistical test used to determine whether a time series is stationary or non-stationary. The test is based on the idea of regression, where the dependent variable is the time series itself and the independent variable is a lagged version of the series.

The null hypothesis of the ADF test is that the time series is non-stationary. ADF test will return 'p-value' and 'Test Statistics' output values.

- **P-Value:**
  - Null Hypothesis (H0): Series is non-stationary Alternate
  - Alternative Hypothesis (HA): Series is stationary ○ p-value >0.05 Fail to reject (H0) ○ p-value <= 0.05 Accept (H1)

- **Test statistics: More negative this value is, the more likely we have stationary series. Also, this value should be smaller than critical values(1%, 5%, 10%).**

For e.g. If test statistic is smaller than the 5% critical values, then we can say with 95% confidence that this is a stationary series The ADF test is a widely used technique in time series analysis and is particularly useful when working with real-world data that may have trend or seasonality components. It helps to ensure that the time series is stationary before applying any further analysis or modeling techniques.

❖ CONVERTING NON-STATIONARY DATA TO STATIONARY DATA

To make time series data stationary, it is important to account for its characteristics such as trend and seasonality. This can be achieved by making the mean and variance of the time series constant. Below are some techniques that are commonly used to make non-stationary data stationary.

❖ DIFFERENCING

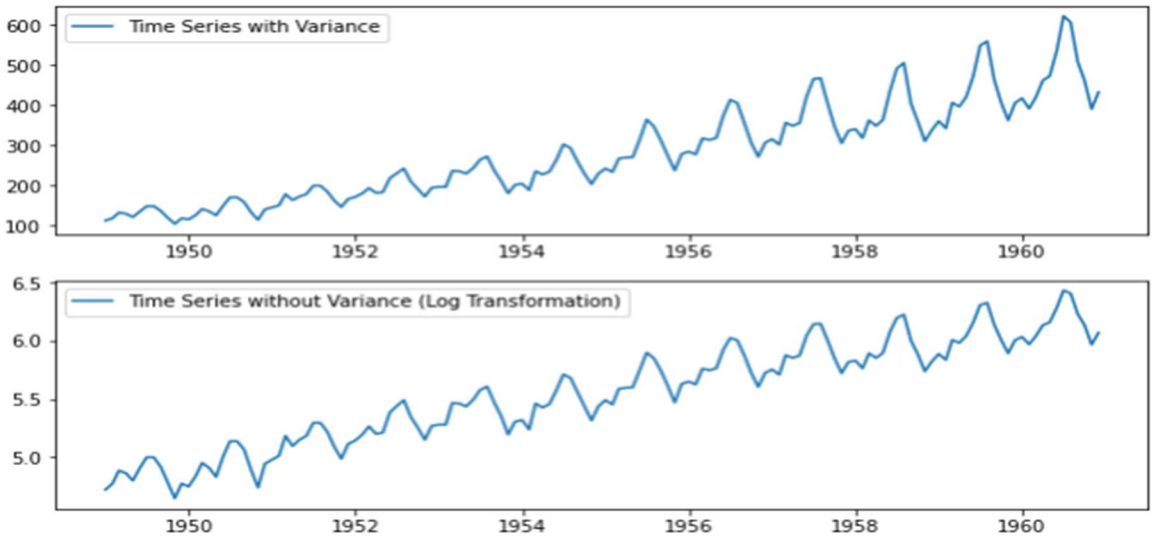
The differencing technique helps in removing trend and seasonality from time series data. It is performed by subtracting the previous observation from the current observation. The differenced data will contain one less data point than the original data. Differencing reduces the number of observations and stabilizes the mean of a time series.

Difference= pervious observation – current observation

After performing differencing, it is recommended to plot the data to visualize the change. If there is not sufficient improvement, we can perform second or third-order differencing.

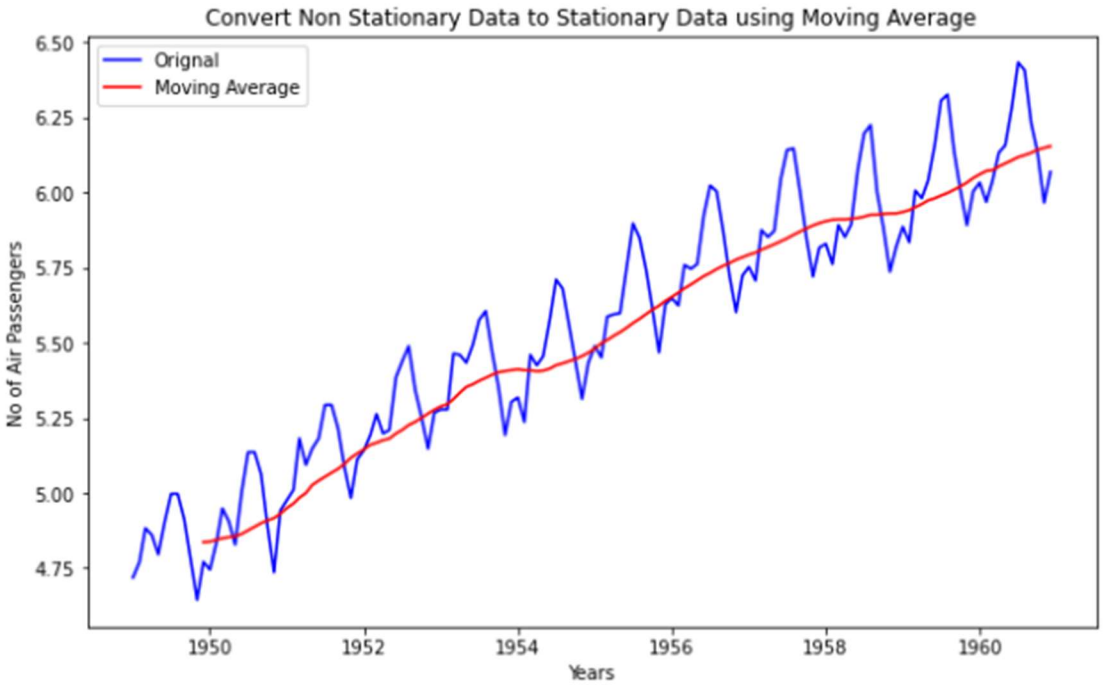
❖ TRANSFORMATION

Transformation Another technique to stabilize the variance across time is to apply a power transformation to the time series. Log, square root, and cube root are the most commonly used transformation techniques. Depending on the growth of the time series, we can choose the appropriate transformation method. For example, a time series with a quadratic growth trend can be made linear by taking the square root. If differencing doesn't work, we may want to use one of the above transformation techniques to remove the variation from the series.



❖ MOVING AVERAGE:

❖ Moving average is a time series technique that involves creating a new series by taking averages of data points from the original series. To calculate the average, two or more raw data points can be used, which is also known as the 'window width'. The averages are calculated from the start to the end for each set of w consecutive values, giving it the name moving averages. This technique can also be utilized for time series forecasting.



❖ WEIGHTED MOVING AVERAGES(WMA)

Weighted Moving Averages (WMA) is a method used in technical analysis to give more importance to recent data points while assigning less weight to data points that are far in the past. To calculate WMA, each value in the data set is multiplied by a pre-determined weight, and the resulting products are summed up. There are several ways to assign weights, and one of the commonly used methods is Exponential Weighted Moving Average, where weights are assigned to past values based on a decay factor.

❖ CENTERED MOVING AVERAGES (CMA)

A centered moving average calculates the value of the moving average at time t by centering the window around time t and computing the average of the w values within the window. For instance, a centered moving average with a window of 3 is calculated as follows

$$CMA(t) = mean(t-1, t, t+1)$$

CMA is a useful technique for visualizing time series data.

❖ TRAILING MOVING AVERAGES (TMA)

In contrast to centered moving averages, trailing moving averages calculate the average of the last w values rather than averaging over a window centered around a time period of interest. For example, a trailing moving average with a window of 3 would be calculated as: TMA is useful for forecasting.

$$TMA(t) = mean(t-2, t-1, t)$$

==========

❖CORRELATION

The most significant aspect of values in a time series is their dependence on previous values. We can determine the correlation between time series observations and previous time steps, known as lags. Because the correlation of time series observations is calculated with the values of the same series at previous times, this is referred to as autocorrelation or serial correlation. To illustrate this point, let's consider the example of fish prices, where

- P(t) represents the fish price of today,
- P(t-1) represents the fish price of the previous month, and
- P(t-2) represents the fish price of the month before that.

A time series of fish prices can be represented as P(t-n), ... P(t-3), P(t-2), P(t-1), P(t). By analyzing fish prices over the past few months, we can predict the fish price for today. All past and future data points are interrelated in time series, and ACF and PACF functions assist in determining the correlation in it.

❖ AUTO CORRELATION FUNCTION (ACF)

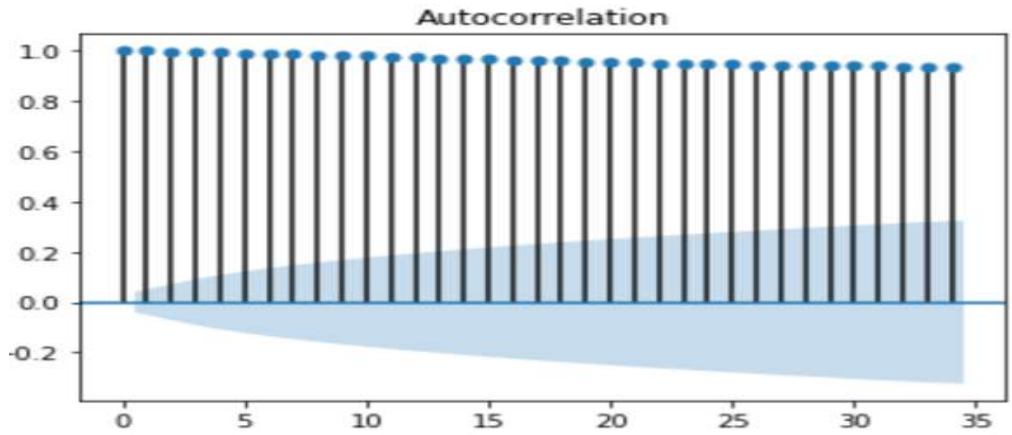
Autocorrelation is the correlation between a time series with a lagged version of itself. The ACF starts at a lag of 0, which is the correlation of the time series with itself and therefore results in a correlation of 1.

- ACF determines the correlation between points based on the number of time steps separating them.
- For example, in the fish price example, we can determine the correlation between the current month's fish price P(t) and the fish price two months ago P(t-2).
- It is necessary to note that the fish price of two months ago can directly or indirectly affect today's fish price, possibly through last month's price P(t-1). ACF considers both direct and indirect effects between points while determining correlation.

The ACF plot can provide answers to the following questions:

- Is the observed time series **white noise/random**?
- Is an observation related to an adjacent observation, an observation twice-removed, and so on?

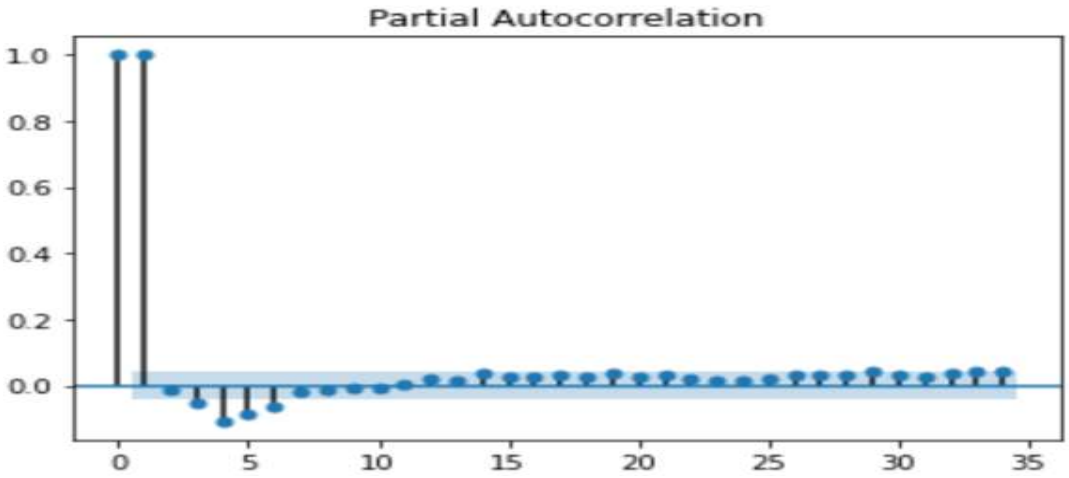
- Can the observed time series be modeled with an **MA model**? If yes, what is the order?



❖ PARTIAL AUTO CORRELATION FUNCTION (PACF)

● In contrast to ACF, PACF considers only direct effects between points while determining the correlation.

● For example, in the fish price example, PACF determines the correlation between the current month's fish price P(t) and the fish price two months ago P(t-2) by considering only P(t) and P(t-2) and ignoring P(t-1).



❖TIME SERIES FORECASTING

Time series forecasting involves making future predictions based on analysis of past data points. The following steps are generally followed during time series forecasting:

1. Analyze the time series characteristics, such as trends and seasonality, to gain a better understanding of the data.

2. Determine the best method to make the time series stationary through data analysis.

3. Record the transformation steps used to make the time series stationary and ensure that the data can be transformed back to its original scale.

4. Choose an appropriate model for time series forecasting based on the data analysis.

5. Assess the model's performance using metrics such as residual sum of squares (RSS), using the entire data set for prediction.

6. Transform the array of predictions back to the original scale to obtain the actual forecasted values.

7. Finally, conduct future forecasting to obtain forecasted values in the original scale.

❖ MODELS USED FOR TIME SERIES FORECASTING

- Autoregression (AR)
- Moving Average (MA)
- Autoregressive Moving Average (ARMA)
- Autoregressive Integrated Moving Average (ARIMA)
- Seasonal Autoregressive Integrated Moving-Average (SARIMA)
- Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors (SARIMAX)
- Vector Autoregression (VAR)

- Vector Autoregression Moving-Average (VARMA)
- Vector Autoregression Moving-Average with Exogenous Regressors (VARMAX)
- Simple Exponential Smoothing (SES)
- Holt Winter's Exponential Smoothing (HWES).

❖ APPLICATIONS OF TIME SERIES ANALYSIS

Here are few real-world applications where time series analysis is used

1. **Sales Forecasting**: Predicting future sales based on historical sales data to help businesses plan for inventory management and production schedules.

2. **Stock Market Analysis**: Analyzing historical stock prices and market trends to make investment decisions and predict future market movements.

3. **Energy Demand Forecasting**: Predicting future energy demand based on historical usage patterns to help energy providers manage their supply and distribution networks.

4. **Traffic Flow Analysis**: Analyzing historical traffic data to identify patterns and predict future traffic congestion, helping city planners and transportation authorities to optimize traffic flow.

5. **Weather Forecasting**: Analyzing historical weather data to predict future weather patterns, helping meteorologists to issue accurate weather forecasts.

6. **Epidemiological Analysis**: Analyzing historical disease outbreak data to predict future outbreaks, helping public health officials to take preventive measures.

7. **Website Traffic Analysis**: Analyzing historical website traffic data to identify patterns and predict future traffic, helping website owners to optimize their content and advertising strategies.

8. **Social Media Analysis**: Analyzing historical social media data to identify trends and predict future consumer behavior, helping businesses to tailor their marketing strategies.

9. **Financial Risk Analysis**: Analyzing historical financial data to identify risk factors and predict future financial risks, helping investors and financial institutions to make informed decisions.

❖PROJECT CONCEPT AND ANALYSIS

I have taken the foreign Tourist data of India. These data is taken from 1 January 2001 to December 2021 with the motive of predicting the number of Tourist in future using time series analysis. Data contains univariate variables such as number of Tourist per month. I perform Univariate Time Series Analysis by using Linear Regression and Arima.

❖STEPS INVOLVED IN ANALYSIS

Language:- Python

Library :- numpy , matplotlib, statsmodels, pandas, sklearn

1. **Importing the important libraries used in our analysis.**

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.tsa.stattools import adfuller
from sklearn.linear_model import LinearRegression
from mpl_toolkits import mplot3d
```

2. Import the data from google drive and read the data.
   ts=pd.read_excel('/content/drive/MyDrive/Newfolder/Indian tourist.xslx')

3. By using The head() method returns a specified number of rows, string from the top. The head() method returns the first 5 rows if a number is not specified. Note: The column names will also be returned, in addition to the specified rows .

we get the dataset has 252 entries, representing 252 months through Jan 2001 to Dec 2021.

4. Checking for null values

```
ts.info()
```

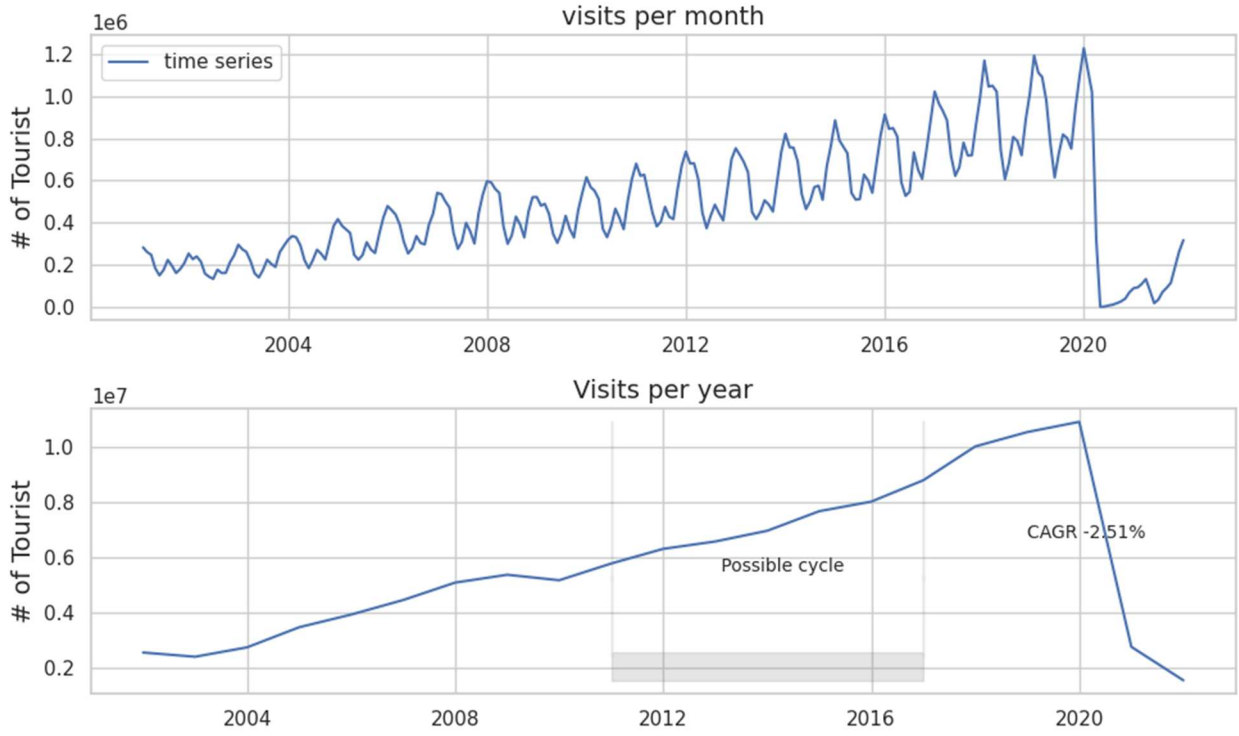
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 252 entries, 2001-01-31 to 2021-12-31
Data columns (total 1 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Tourist  252 non-null    int64
dtypes: int64(1)
```

- There are no missing values.

5. TIME SERIES VISUAL INSPECTION

- An upward trend with Compound Annual Growth Rate for 2002-2021 =- 2.51%,because of lockdown imposed these CAGR goes negative.

- Changes in the trend direction in 2012 and 2017. Possible longterm cycle.
- Clear seasonality pattern backed by a boxplot graph with visible differences in value distributions through the year.



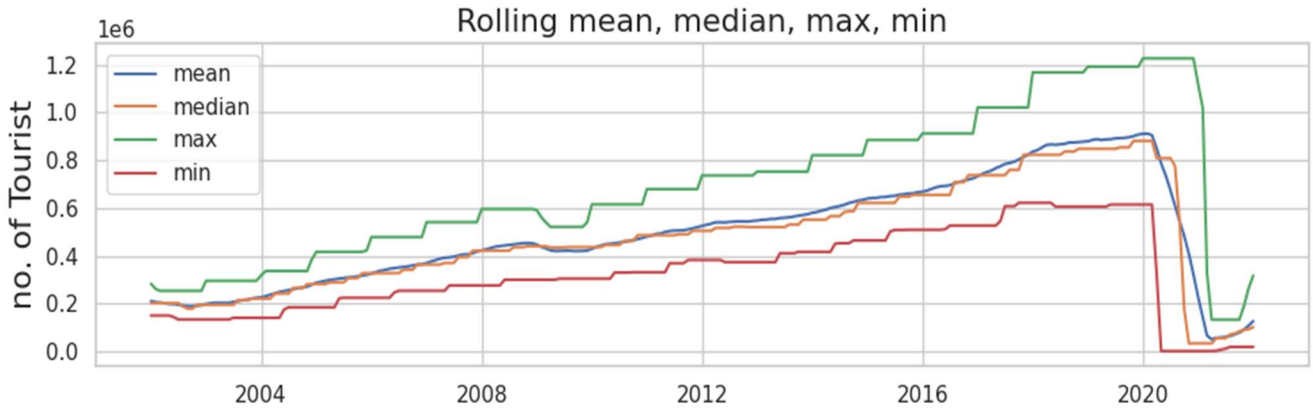
6. DESCRIPTIVE STATISTICS

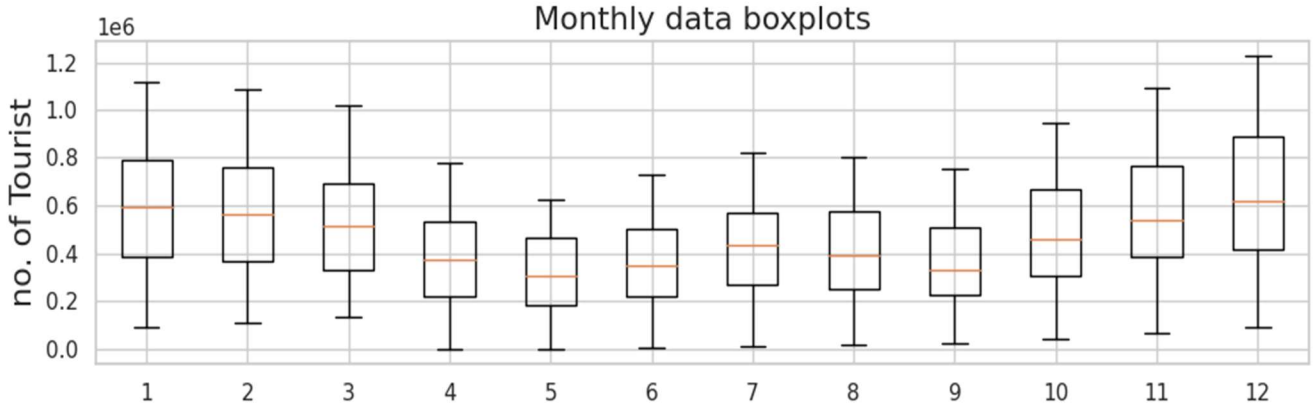
```
ts.describe().T.round(2)
```

- **Count** = 252.0
- **Mean** = 480254.76
- **Std** = 267916.85
- **Min** = 2820.0
- **25%** = 274125.25
- **50%** = 447046.0
- **75%** = 668740.25
- **Max** = 1226398.0

7. PLOTTING BASIC DESCRIPTIVE STATISTICS

```
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(10, 6))

ax[0].plot(ts["Tourist"].rolling(12).agg(
    ["mean", "median", "max", "min"]), linestyle="solid")
ax[0].set_title("Rolling mean, median, max, min", size="16")
ax[0].set_ylabel('no. of Tourist', size="16")
ax[0].legend(["mean", "median", "max", "min"])

ts['Month'] = ts.index.month
ts['Year'] = ts.index.year
ts_piv = ts.pivot(index="Month", columns="Year", values="Tourist")

month_labels = ["Jan", "Feb", "Mar", "Apr", "May",
                "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]

ax[1].boxplot(ts_piv.T)
ax[1].set_title("Monthly data boxplots", size="16")
ax[1].set_ylabel('no. of Tourist', size="16")

fig.tight_layout()
plt.show()
```





From above we can see the rolling mean ,median ,max and min value of tourist dataset. In above box and whisker plot: The left and right sides of the box are the lower and upper quartiles. The box covers the interquartile interval, where 50% of the data is found. The vertical line that split the box in two is the median.

8. THEORETICAL BACKGROUND

I decided to decompose time series straight forward, manually, instead of using tools such as Statsmodes seasonal_decompose() function. It helps to understand the logic behind decomposing and provides grater understanding of the analysis.

The underlying logic behind decomposition is the assumption that time series is the combination of trend, seasonal factor, and randomness. In this case, the relation between elements is **multiplicative**, which can be seen on the first plot as you move along the time axis; the higher the growth, the higher the variance in seasonality.

Ts= t *s* e

Following this logic the particular components of the time series are results of transformation of the above equation.

a) **S=Ts/t**
b) **e=Ts/t*s**

Where:

- Ts = timeseries
- t = trend
- s = seasonality
- e = random error

9. DECOMPOSITION AND VISUALS

```
 Identyfing trend using moving average
ts["MA12"] = ts["Tourist"].rolling(window=12).mean()

# Detrending time series
ts["Seasonality+Noise"] = ts["Tourist"] / ts["MA12"]

# Getting seasonality index
seasonality = list(ts.groupby("Month")["Seasonality+Noise"].mean())
ts["Seasonality"] = seasonality * int((ts.shape[0]/12))
```

```
# Identifying residuals
ts["Noise"] = ts["Tourist"]/(ts["MA12"]*ts["Seasonality"])
#ts["Noise"] = (ts["Tourist"]/(ts["Trend"]*ts["Seasonality"])-1)*ts["Trend"]

# Plotting results
fig, ax = plt.subplots(nrows=4, ncols=1, figsize=(15, 16), sharex=False)

ax[0].plot(ts["Tourist"])
ax[0].set_title("Visits", size="14")
ax[0].set_ylabel('no. of Tourist', size="14")

ax[1].plot(ts["MA12"], linestyle="solid",color='yellow')
ax[1].set_title("Trend", size="14")
ax[1].set_ylabel('no.of Tourist', size="14")

ax[2].plot(ts["Seasonality"],color='green')
ax[2].set_title("Seasonality ", size="14")
ax[2].set_ylabel("Seasonality", size="14")

ax[3].plot(ts["Noise"],color='red')
ax[3].set_title("Noise ", size="14")
ax[3].set_ylabel('no. of Tourist')

fig.tight_layout()
plt.show()
```

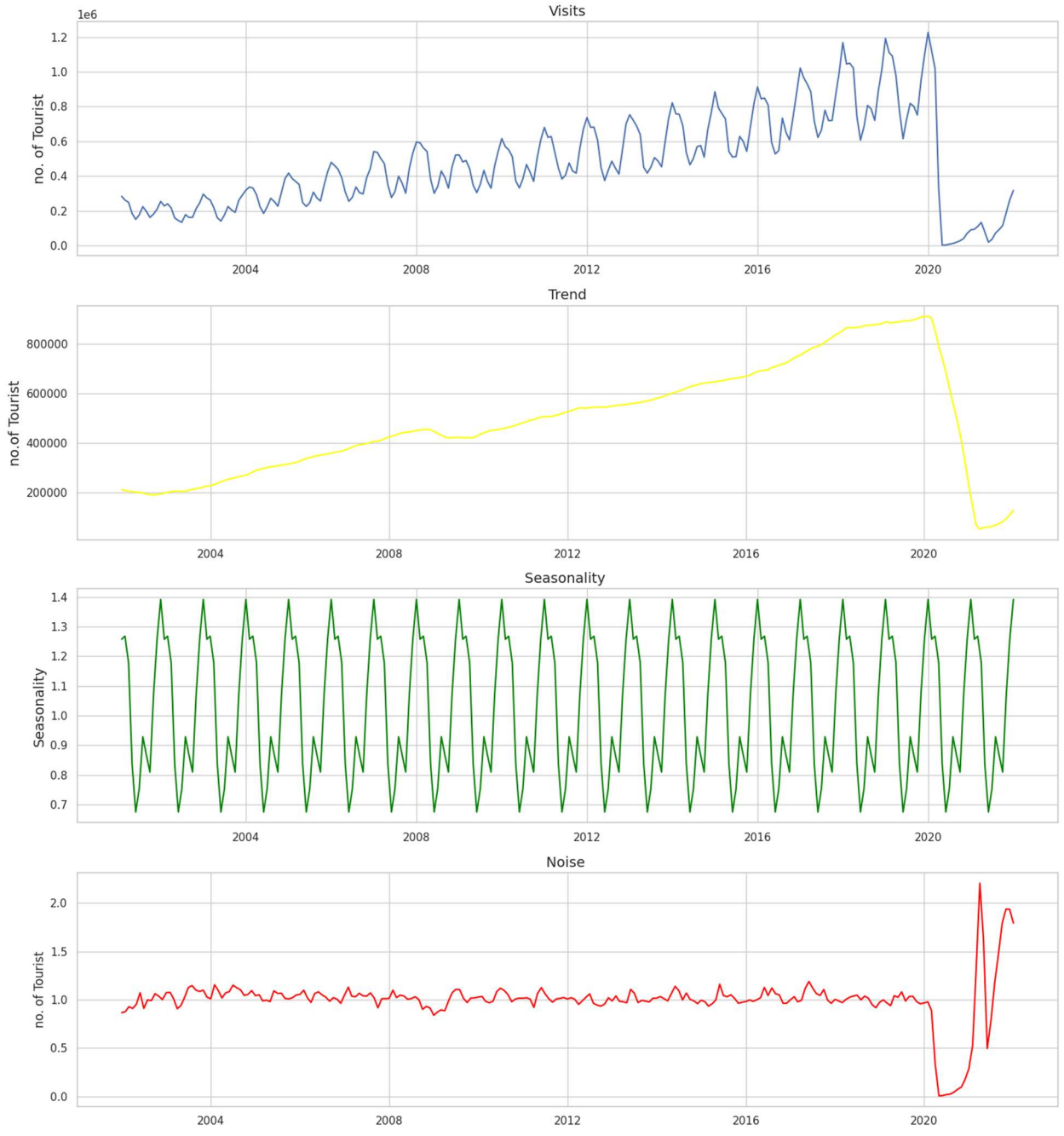
performing time series decomposition

There are basically 4 components of time series.

1. **Trend**
2. **Seasonality**

3. **Cyclical**

4. **Irregularity**



```
10  ts_piv
```

We have now converted the unaligned time series into aligned, equally spaced time series using binning-and-averaging. We can see below that despite our efforts, nearly 90% of the values in our new time series are NaN, so there will be major challenges trying to make use of the converted data with tools that cannot cope well with NAs.

Normal procedures like mean imputation or interpolation won't work well here due to the prevalance of missing values. However, some algorithms like LSTM may learn to ignore padded values and data gaps and still manage to extract valuable information from the time series. It is even possible to build multiple time series collections with different bins to look for features at different time scales.

```
ts_piv
```

Year	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	...	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Month																					
1	283750	228150	274215	337345	385977	459489	535631	591337	481308	568719	...	681002	720321	757786	790854	844533	964109	1045027	1111040	1119250	94662
2	262306	241133	262692	331697	369844	439090	501692	561393	489787	552152	...	681193	688569	755678	761007	848782	931025	1049259	1090516	1018440	110312
3	248965	216839	218473	293185	352094	391009	472494	541478	442062	512152	...	606456	639530	690441	729154	809107	885936	1021539	978236	328304	133768
4	185338	159789	160941	223884	248416	309208	350550	384203	347544	371956	...	447581	450580	535321	541551	592004	717899	745033	774651	2820	78718
5	151098	144571	141508	185502	225394	255008	277017	300840	305183	332087	...	374476	417453	465043	509869	527466	622408	606513	615136	3764	19765
6	176716	134566	176324	223122	246970	278370	310364	340159	352353	384642	...	433390	451223	502028	512341	546972	663470	683935	726446	8590	36070
7	224432	178231	225359	272456	307870	337332	399866	429456	432900	466715	...	485808	506427	568871	628323	733834	779309	806493	818125	12655	72501
8	196517	162594	204940	253301	273856	304387	358446	391423	369707	422173	...	445632	486338	575750	599478	652111	719129	785993	800837	19761	92728
9	162326	163089	191339	226773	257184	297891	301892	330874	330707	369821	...	411562	453561	509142	542600	608177	719964	719894	751513	28167	115661
10	181605	213267	260569	307447	347757	391399	444564	452566	458849	507093	...	556488	598095	668398	683286	741770	866976	890223	945017	41494	191415
11	209685	245661	290583	385238	423837	442413	532428	521247	541524	608178	...	701185	733923	765497	815947	878280	997738	1012569	1092440	70977	263867
12	254544	296474	319271	417527	479411	541571	596560	521990	615775	680004	...	752972	821581	885144	912723	1021375	1167840	1191498	1226398	90544	317647

12 rows × 21 columns

12.SEASONALITY INSPECTION

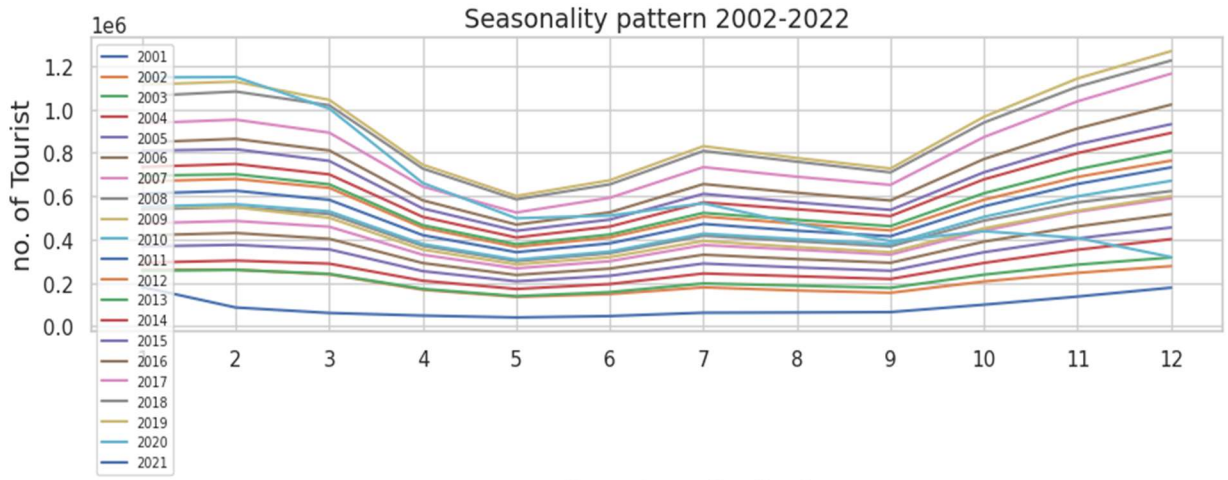
A deeper dive into seasonality confirms a steady pattern existence. The twenty years sample visual inspection shows that in every single year, the pattern had almost identical shape.

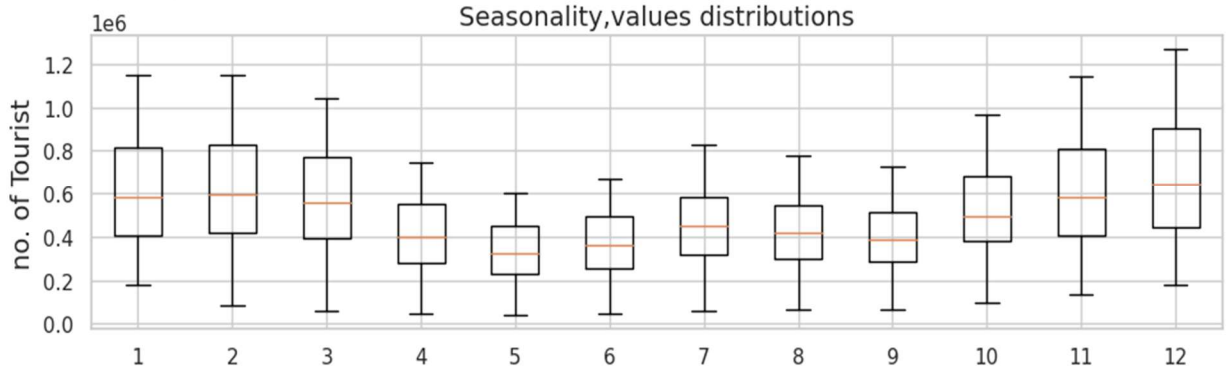
Valuable insights concern the distributions of the values from the per month breakdown shown on a boxplot. The data skews to the right, toward higher positive values. The first half of the year might be less predictable and could be prone to outliers.

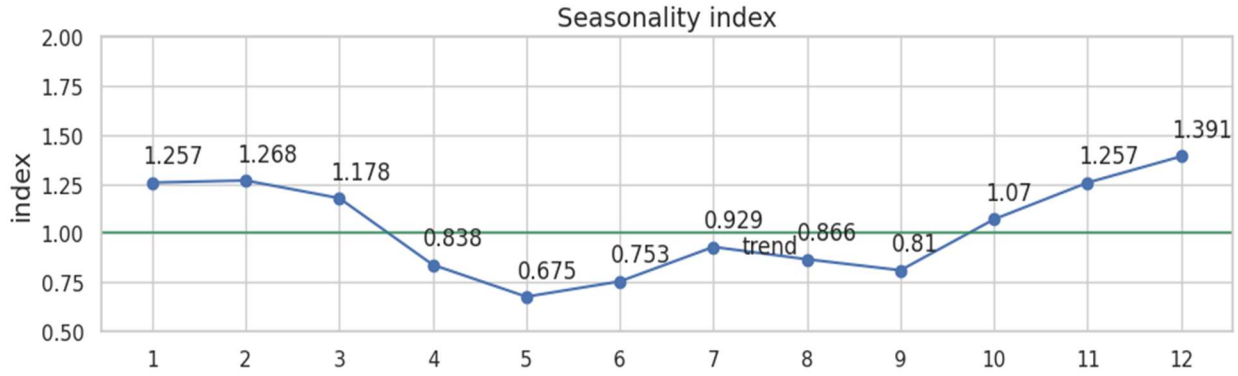
We can see the highest seasonality for December month ,with seasonality index max=1.391 in dec. and min for may=0.675.

```
#deeper dive into seasonality
ts["Seasonality_Tourist"] = (ts["Seasonality"] * ts["MA12"])
ts_piv = ts.pivot(index='Month', columns="Year", values="Seasonality_Tourist")


fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(10, 10), sharex=False)


ax[0].plot(ts_piv)
ax[0].set_title("Seasonality pattern 2002-2022", size="14")
ax[0].set_ylabel('no. of Tourist', size="14")
ax[0].set_xticks(ts_piv.index)
ax[0].legend(ts_piv.columns,loc='upper left',fontsize=7)


ax[1].boxplot(ts_piv.iloc[:, 1:].T)
ax[1].set_title("Seasonality,values distributions", size="14")
ax[1].set_ylabel('no. of Tourist', size="14")
ax[1].set_xticks(ts_piv.index)
```

```
sindex = pd.Series(data=ts.groupby("Month")[
                "Seasonality"].mean(), name="Seasonality_Index").round(3)

ax[2].plot(sindex, marker="o")
ax[2].axhline(1.0, linestyle="solid", color="seagreen", alpha=0.75)
ax[2].annotate(text="trend", xy=(7.3, 0.9))
ax[2].set_title("Seasonality index", size="14")
ax[2].set_ylabel('index', size="14")
ax[2].set_ylim(bottom=0.5, top=2.0)
ax[2].set_xticks(sindex.index)

for x, y in zip(sindex.index, sindex):
    ax[2].annotate(text=sindex.iloc[x-1], xy=(x-0.1, y+0.1))

plt.tight_layout()
plt.show()
```


Seasonality pattern 2002-2022
1e6
no. of Tourist
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021


Seasonality,values distributions
1e6
no. of Tourist


Seasonality index
index
1.257
1.268
1.178
0.838
0.675
0.753
0.929
trend
0.866
0.81
1.07
1.257
1.391

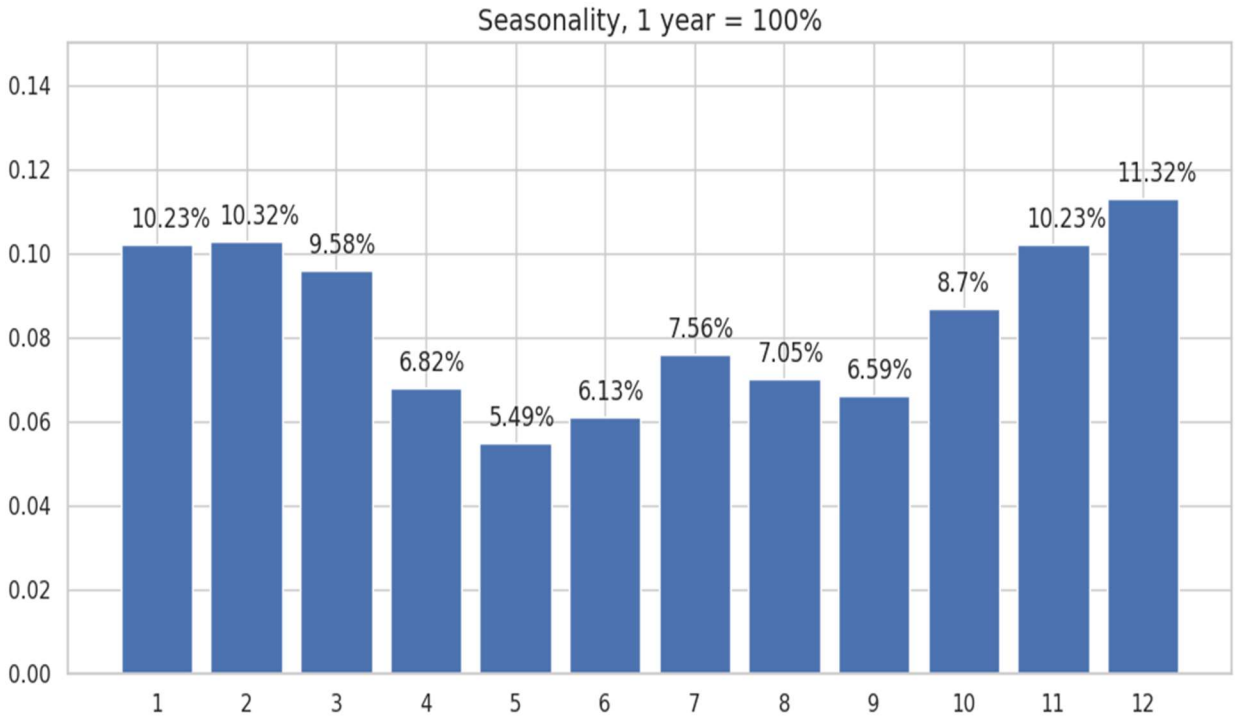
13.SEASONALITY NUMBERS FOR BUDGETING

From a business application perspective, instead of seasonality index, more useful for budgeting and forecasting is a monthly contribution to the annual result breakdown.

```
x = [i for i in range(1, 13)]

y = [(i/sindex.sum()).round(3) for i in sindex]
s = [str(((i/sindex.sum())*100).round(2))+"%" for i in sindex]


plt.figure(figsize=(10, 5))
plt.bar(x, y)
plt.ylim(top=0.15)
plt.title("Seasonality, 1 year = 100%", size=14)

for s, x, y in zip(s, x, y):
    plt.annotate(s, xy=(x, y), xytext=(x-0.3, y+0.004), size=12)


plt.xticks(sindex.index)


plt.tight_layout()
plt.show()
```

here we can see that we get maximum budget in the month of dec . That means maximum number of tourist arrives in the month of December and as more visitors arrive in india the revenue is also increases. We can see that

from month of nov,dec,jan,feb and march gives more revenue as compars to other months.



14. LINEAR TREND

A linear trend combined with the seasonality, is useful in forecasting. It might also helps recognizing cyclicality. Having univariate data, a simple linear regression formula applies.

- Yi =ß◦+ß1*xi+ei
  where
- y=dependent variable
- ß◦ = Y **intercept**
- ß1 = **slope** coefficient

- xi = independent variable
- ei = random error

Linear Regression model

- The linear trend formula that applies for the time series dataset is as follows:

We get ,

- Slope: 1987.9658425782848
- Intercept: 233187.00975651114
- **Yi=1987.965 + 233187.009 *Xi**

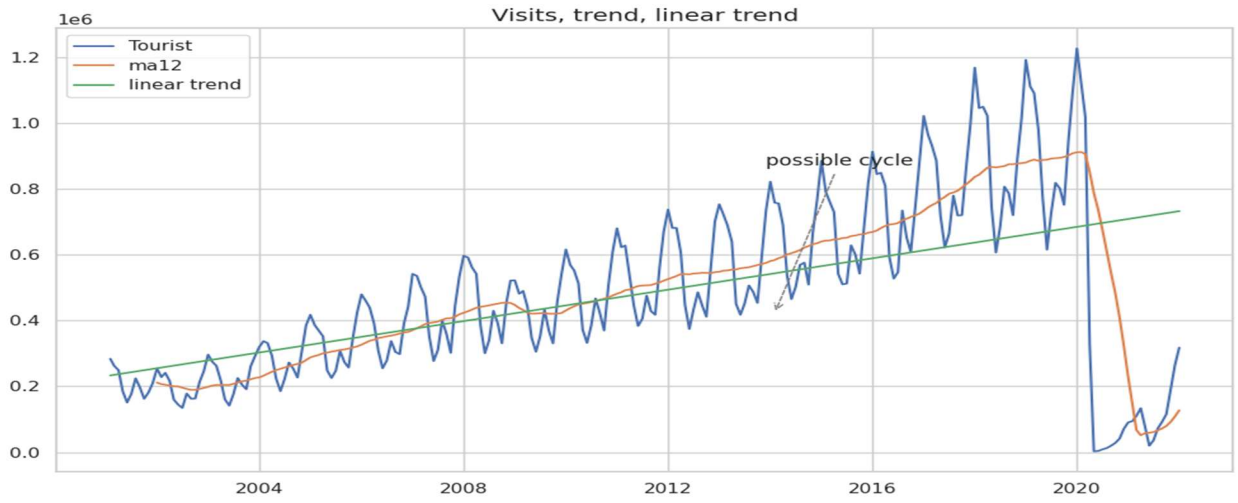
```
# preparing data for linear regression
y = ts["MA12"].reset_index(drop=True)[12:].values.reshape(-1, 1)
X = ts["MA12"].reset_index(drop=True)[12:].index.values.reshape(-1, 1)
print("y shape :", y.shape)
print("X shape :", X.shape)
```

```
y shape : (240, 1)
X shape : (240, 1)
```

```
lr = LinearRegression()

model = lr.fit(X, y)

slope = lr.coef_[0, 0]

intercept = lr.intercept_[0]


# linear regression results
```

```
y_pred = [x * slope + intercept for x in range(len(ts))]

ts["LR"] = y_pred


print("Slope: ", slope)

print("Intercept: ", intercept)
```

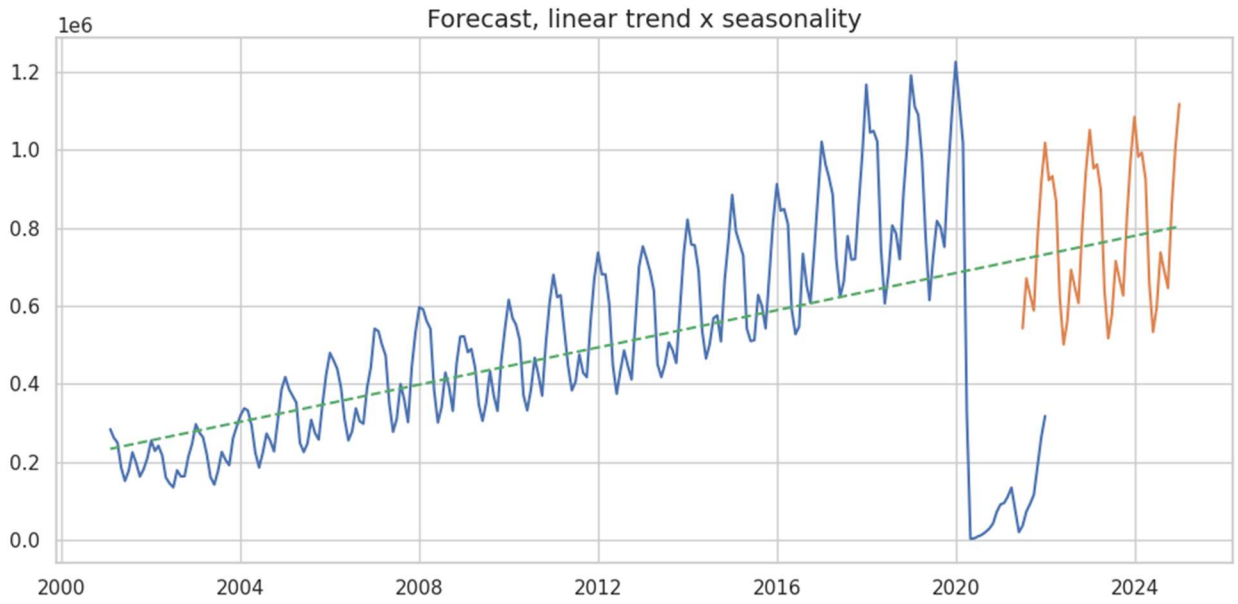
```
# plotting lr results
fig = plt.figure(figsize=(10, 6))

plt.plot(ts[["Tourist", "MA12", "LR"]])
plt.legend(["Tourist", "ma12", "linear trend"])
plt.title("Visits, trend, linear trend", size=14)
plt.annotate("possible cycle",
             xy=(pd.Timestamp('2014-01-31'), 423000),
             xytext=(pd.Timestamp('2013-11-30'), 870000),
             arrowprops=dict(arrowstyle='->', ls="dashed", lw=1, color="gray"))

plt.tight_layout()
plt.show()
```



14. Time series forecasting

Forecast based on linear trend

```
trend = [x * slope + intercept for x in range(288)]
season = seasonality*24
y_pred = [a*b for a, b in zip(trend, season)]
labels = pd.date_range(start="2001-01-31", end="2024-12-31", freq="M")

ts_fcast = pd.DataFrame({"Trend": trend, "Forecast": y_pred, "Index":
labels}).set_index("Index", drop=True)
ts_fcast["Tourist"] = ts["Tourist"]

fig = plt.figure(figsize=(10, 5))

plt.plot(ts_fcast["Tourist"])
plt.plot(ts_fcast["Forecast"][245:])
plt.plot(ts_fcast["Trend"], linestyle="dashed")
plt.title("Forecast, linear trend x seasonality", size=14)

plt.tight_layout()
plt.show()
```



15. R2 metric

Now finding coefficient of determination.

R squared (R2) is **a regression error metric that justifies the performance of the model.** It represents the value of how much the independent variables are able to describe the value for the response/target variable.

Hence our model **gives 81% accuracy ..**

The formula I am using to calculate R-squared is:

R^2=SSE /SST

Where:

- **SSE = sum of squared error**

- **SST = sum of squared total**

More precisely:

$R^2 = \sum(Y - Y^{\wedge})^2 \ / \ \sum(Y - \bar{Y})^2$

Where:

- **Y = actual value**
- **Y^ = predicted value of Y**
- **$\bar{Y}$ = mean of Y values**

```
# calculating R2
vs_mean = ts_fcast["Tourist"].mean()
SSE = sum((ts_fcast["Tourist"][:252] - ts_fcast["Forecast"][:252])**2)
SSW = sum((ts_fcast["Tourist"][:252] - ts_fcast["Tourist"][:252].transform(func=lambda
x:  x - vs_mean))**2)
R2 = 1-SSE/SSW
print("R2:", round(R2, 3))
```

```
R2: 0.81
```

Hence these model is good for prediction with 81% of accuracy.

16. Data Types of Time Series

Let's discuss the time series' data types and their influence. While discussing TS data-types, there are two major types.

- Stationary
- Non- Stationary

To know my data is stationary or not ,I am using adifuller tests for stationarity.

he integrated order d=n with the p-value less than 0.05 is the right one.

```
# adfuller tests for stationarity
print("TS d=0 P-value: ", adfuller(ts["Tourist"])[1].round(3))
print("TS d=1 P-value: ",
    adfuller(ts["Tourist"].diff(periods=1).dropna())[1].round(3))
print("TS d=2 P-value: ",
    adfuller(ts["Tourist"].diff(periods=1).diff(periods=1).dropna())[1].round(3))
```

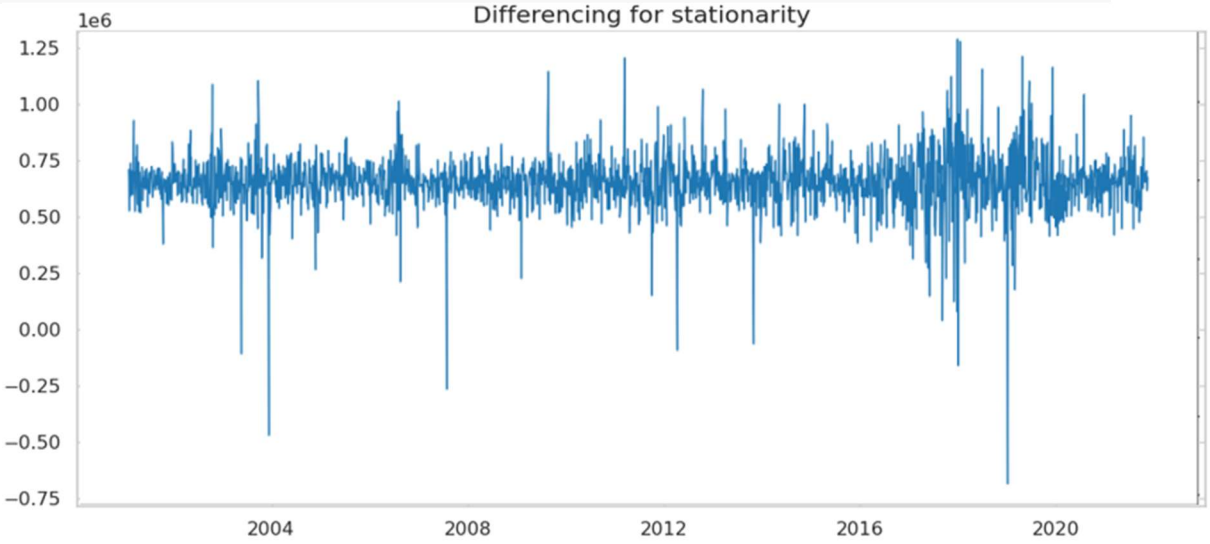
```
TS d=0 P-value:  0.049
TS d=1 P-value:  0.008
TS d=2 P-value:  0.0
```

- After applying Argumented Dicky Fullers Test
- We get p values :
- TS d=0 P-value: 0.049
- TS d=1 P-value: 0.008
- TS d=2 P-value: 0.0
- The integrated order d=n with the p-value less than 0.05 is the right one.

  Hence all the observed p values are less that 0.05 hence our data set is stationary.

17. Differencing for stationarity

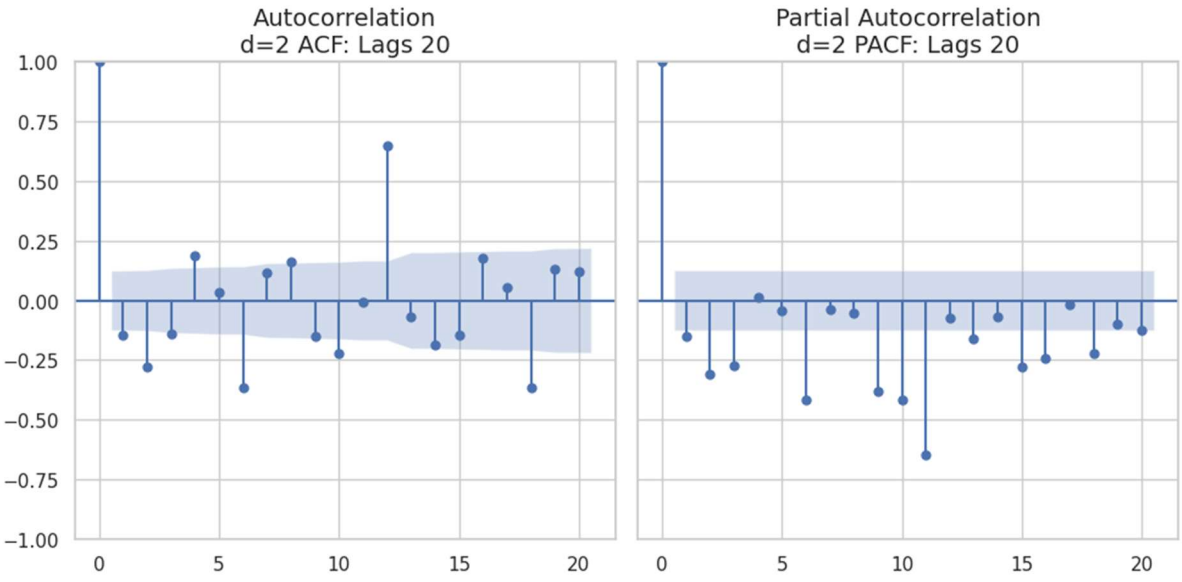
```
# Plotting the results
fig = plt.figure(figsize=(10, 5))

plt.plot(ts["Tourist"])
plt.plot(ts["Tourist"].diff(periods=1))
plt.plot(ts["Tourist"].diff(periods=1).diff(periods=1))
plt.legend(["diff d=0", "diff d=1", "diff d=2"])
plt.title("Differencing for stationarity", size=14)

fig.tight_layout()
plt.show()
```



18. Auto Corelation, Partial Autocorrelation

Surprisingly, the initial series are stationary; the ADFuller test rejected the null hypothesis that data is stationary. Actually, we can see this on the plot itself -- we do not have a visible trend, so the mean is constant

and the variance is pretty much stable. The only thing left is seasonality, which we have to deal with prior to modeling. To do so, let's take the "seasonal difference", which means a simple subtraction of the series from itself with a lag that equals the seasonal period.
as we see that, in ACF and PACF plot there is a same pattern after every yearly (doubt in taking yearly because of seasonality) hence take shift=12



19. ADIFULLER TEST

Surprisingly, the initial series are stationary; the ADIFuller test rejected the null hypothesis that data is stationary. Actually, we can see this on the plot itself -- we do not have a visible trend, so the mean is constant and the variance is pretty much stable. The only thing left is seasonality, which we have to deal with prior to modeling. To do so, let's take the "seasonal difference", which means a simple subtraction

of the series from itself with a lag that equals the seasonal period.

as we see that, in ACF and PACF plot there is a same pattern after every yearly (doubt in taking yearly because of seasonality) hence take shift=12

Choosing p, d, q elements of the model

- p: Trend autoregression order. Periods taken for autoregressive model.
- d: Trend difference order. Integrated order difference.
- q: Trend moving average order. Number of periods in moving average model.

```
training_data=ts["Tourist"][0:217]
testing_data=ts["Tourist"][217:]
arima= ARIMA(training_data,order=(p,d,q))

arima=ARIMA(training_data,order=(3,2,4))
model=arima.fit()
```

Doing predictions..

```
pred=model.forecast(steps=len(testing_data))
np.sqrt(mean_squared_error(testing_data,pred))
```

946735.2417862881

Lets do model Tuning or Hyperparameter Tuning..

```
#### now define hyper-para=meters
p_values=range(0,5)
d_values=range(0,3)
q_values=range(0,6)

 orders=[]
for p in p_values:
  for d in d_values:
    for q in q_values:
      order=(p,d,q)
      orders.append(order)

## arima= ARIMA(training_data,order=(p,d,q))
for i in orders:
  arima=ARIMA(training_data,order=i)
  model=arima.fit()
  pred=model.forecast(steps=len(testing_data))
  print(f"order {i} has root mean square error {np.sqrt(mean_squared_error(testing_data,pred))}")
```

```
order (0, 0, 0) has root mean square error 428458.2157935046
order (0, 0, 1) has root mean square error 422720.14624627924
order (0, 0, 2) has root mean square error 413893.5018919342
order (0, 0, 3) has root mean square error 412036.3005973348
order (0, 0, 4) has root mean square error 411290.88745022396
order (0, 1, 0) has root mean square error 822997.3065735999
order (0, 1, 1) has root mean square error 770300.1736941846
order (0, 1, 2) has root mean square error 772509.0295712878
order (0, 1, 3) has root mean square error 658855.9896105939
order (0, 1, 4) has root mean square error 635363.0440565964
order (0, 2, 0) has root mean square error 941198.0709543554
order (0, 2, 1) has root mean square error 925059.1356739198
order (0, 2, 2) has root mean square error 857324.7716658432
order (0, 2, 3) has root mean square error 860215.5960605178
order (0, 2, 4) has root mean square error 823736.4576369963
order (1, 0, 0) has root mean square error 496592.536219413
order (1, 0, 1) has root mean square error 402713.52631058876
order (1, 0, 2) has root mean square error 404320.46666463185
order (1, 0, 3) has root mean square error 634853.8818454912
order (1, 0, 4) has root mean square error 607914.3869939089
```

```
order (1, 1, 0) has root mean square error 793966.2979757133
order (1, 1, 1) has root mean square error 770027.6827025992
order (1, 1, 2) has root mean square error 645265.0377046806
order (1, 1, 3) has root mean square error 647690.1721191104
order (1, 1, 4) has root mean square error 645666.4975957738
order (1, 2, 0) has root mean square error 355928.19693509984
order (1, 2, 1) has root mean square error 881752.6430281538
order (1, 2, 2) has root mean square error 858969.3460262754
order (1, 2, 3) has root mean square error 850680.3197374386
order (1, 2, 4) has root mean square error 850800.6657025845
order (2, 0, 0) has root mean square error 390971.5905760113
order (2, 0, 1) has root mean square error 403846.28902816016
order (2, 0, 2) has root mean square error 405704.96882272995
order (2, 0, 3) has root mean square error 623147.7890163731
order (2, 0, 4) has root mean square error 622920.8629847779
order (2, 1, 0) has root mean square error 766752.6340653833
order (2, 1, 1) has root mean square error 764446.5411022048
order (2, 1, 2) has root mean square error 865819.5376100339
order (2, 1, 3) has root mean square error 648296.9997574047
order (2, 1, 4) has root mean square error 645073.310942836
order (2, 2, 0) has root mean square error 827524.0754755664
order (2, 2, 1) has root mean square error 850745.2290023979
order (2, 2, 2) has root mean square error 315237.0861637732
order (2, 2, 3) has root mean square error 972712.3813853418
order (2, 2, 4) has root mean square error 874046.843459116
order (3, 0, 0) has root mean square error 433745.14980831003
order (3, 0, 1) has root mean square error 399036.57073608774
order (3, 0, 2) has root mean square error 511286.6667162471
order (3, 0, 3) has root mean square error 624398.0083145781
order (3, 0, 4) has root mean square error 906957.7588180767
order (3, 1, 0) has root mean square error 764989.7355743544
order (3, 1, 1) has root mean square error 766053.4381881303
order (3, 1, 2) has root mean square error 639251.0262205546
order (3, 1, 3) has root mean square error 680960.7953138989
order (3, 1, 4) has root mean square error 670443.0047643122
order (3, 2, 0) has root mean square error 1627816.4198364168
order (3, 2, 1) has root mean square error 1627771.62981413
order (3, 2, 2) has root mean square error 2190149.670240063
order (3, 2, 3) has root mean square error 1012760.2797352785
order (3, 2, 4) has root mean square error 946735.2417862881
order (4, 0, 0) has root mean square error 425197.68072931643
order (4, 0, 1) has root mean square error 423851.93434184045
order (4, 0, 2) has root mean square error 416084.22246798855
order (4, 0, 3) has root mean square error 494734.74346292793
order (4, 0, 4) has root mean square error 463890.1000557968
order (4, 1, 0) has root mean square error 771103.0969557075
order (4, 1, 1) has root mean square error 777569.3625238076
order (4, 1, 2) has root mean square error 637597.5743636686
order (4, 1, 3) has root mean square error 650709.218291444
order (4, 1, 4) has root mean square error 641932.0563000275
order (4, 2, 0) has root mean square error 1627806.5147256686
order (4, 2, 1) has root mean square error 858381.3480484536
order (4, 2, 2) has root mean square error 831320.8969723012
order (4, 2, 3) has root mean square error 857744.9167560285
```

```
order (4, 2, 4) has root mean square error 961692.7661998807
```

order (2, 2, 2) has min root mean square error 315237.0861637732

Best seasonal parameters p,d,q =(2, 2, 2)

```
model = ARIMA(training_data, order=(2, 2, 2))
arima_fit = model.fit()
print(arima_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Tourist   No. Observations:                  217
Model:                 ARIMA(2, 2, 2)   Log Likelihood               -2735.005
Date:                Sat, 13 May 2023   AIC                           5480.011
Time:                        18:29:22   BIC                           5496.864
Sample:                    01-31-2001   HQIC                          5486.820
                         - 01-31-2019
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.8891      0.044     20.345      0.000       0.803       0.975
ar.L2         -0.7743      0.049    -15.709      0.000      -0.871      -0.678
ma.L1         -1.2622      0.023    -54.817      0.000      -1.307      -1.217
ma.L2          0.9513      0.024     39.643      0.000       0.904       0.998
sigma2      6.758e+09   5.41e-13   1.25e+22      0.000    6.76e+09    6.76e+09
===================================================================================
Ljung-Box (L1) (Q):                   0.12   Jarque-Bera (JB):                 2.70
Prob(Q):                              0.73   Prob(JB):                         0.26
Heteroskedasticity (H):               5.63   Skew:                             0.00
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.55
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 2.24e+37. Standard errors may be unstable.
```

20. AKAIKE INFORMATION CRITERION

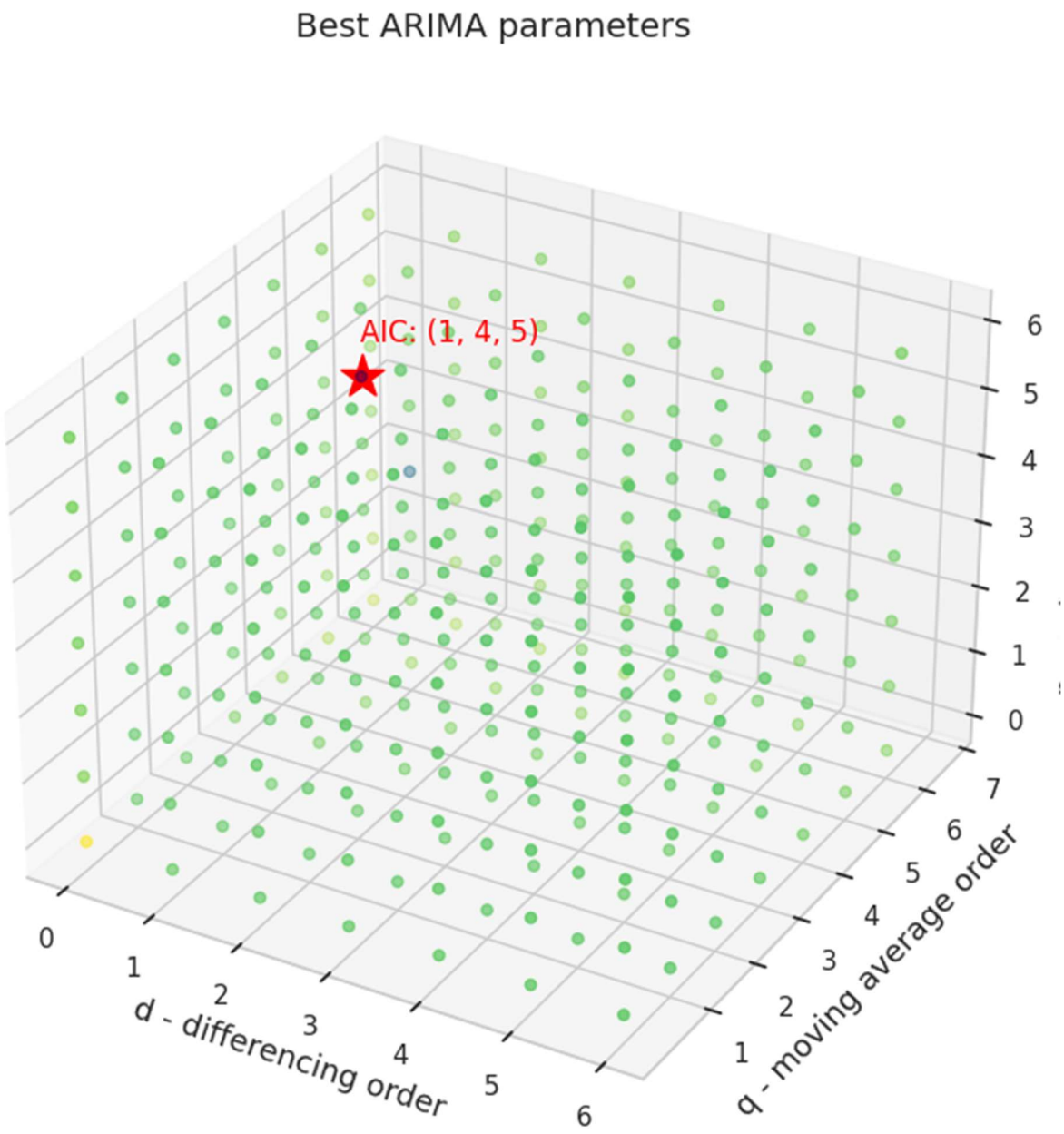
To determine the best ARIMA parameter values, you typically use model evaluation techniques such as evaluating the AIC (Akaike Information Criterion) or BIC (Bayesian Information Criterion) values. These criteria measure the goodness of fit of the ARIMA model while also considering model complexity.

A lower AIC or BIC value indicates a better fit. Therefore, you can iterate over different combinations of ARIMA parameters (p, d, q) and choose the parameter values that result in the lowest AIC or BIC value.

Here's an example of how you can find the best ARIMA parameter values based on the lowest AIC:

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

p = d = q = range(0, 7)
pdq = list(itertools.product(p, d, q))

params = []
aic = []

for param in pdq:
    try:
        model = ARIMA(ts["Tourist"], order=param,
freq="M")
        arima_fit = model.fit()
        params.append(param)
        aic.append(arima_fit.aic)
    except:
        continue

if aic:
    order = params[aic.index(min(aic))]
    p = order
else:
    # Handle the case when no models were
successfully fitted
    p = None
```

```
p = [p for p, d, q in params]
d = [d for p, d, q in params]
q = [q for p, d, q in params]

fig = plt.figure(figsize=(9, 7))

ax = plt.axes(projection='3d')
ax.scatter(p, d, q, c=aic, cmap="viridis")
ax.scatter(*order, c="red", s=350, marker="*")

a, b, c = order
c = c* 1.1
ax.text(a,b,c, f"AIC: {order}", c="red")


ax.set_title("Best ARIMA parameters", size=14)
ax.set_xlabel("d - differencing order", size=13)
ax.set_ylabel("q - moving average order", size=13)
ax.set_zlabel("p - autoregression order", size=13)
ax.set_yticks([1,2,3,4,5,6,7])

plt.tight_layout()
plt.show()
```



The provided code generates a 3D scatter plot visualization of the best ARIMA parameters. The plot includes the autoregression order (p) on the z-axis, the differencing order (d) on the x-axis, and the moving average order (q) on the y-axis. Each point in the scatter plot represents a combination of ARIMA parameters, and its color is determined by the corresponding AIC (Akaike Information Criterion) value.

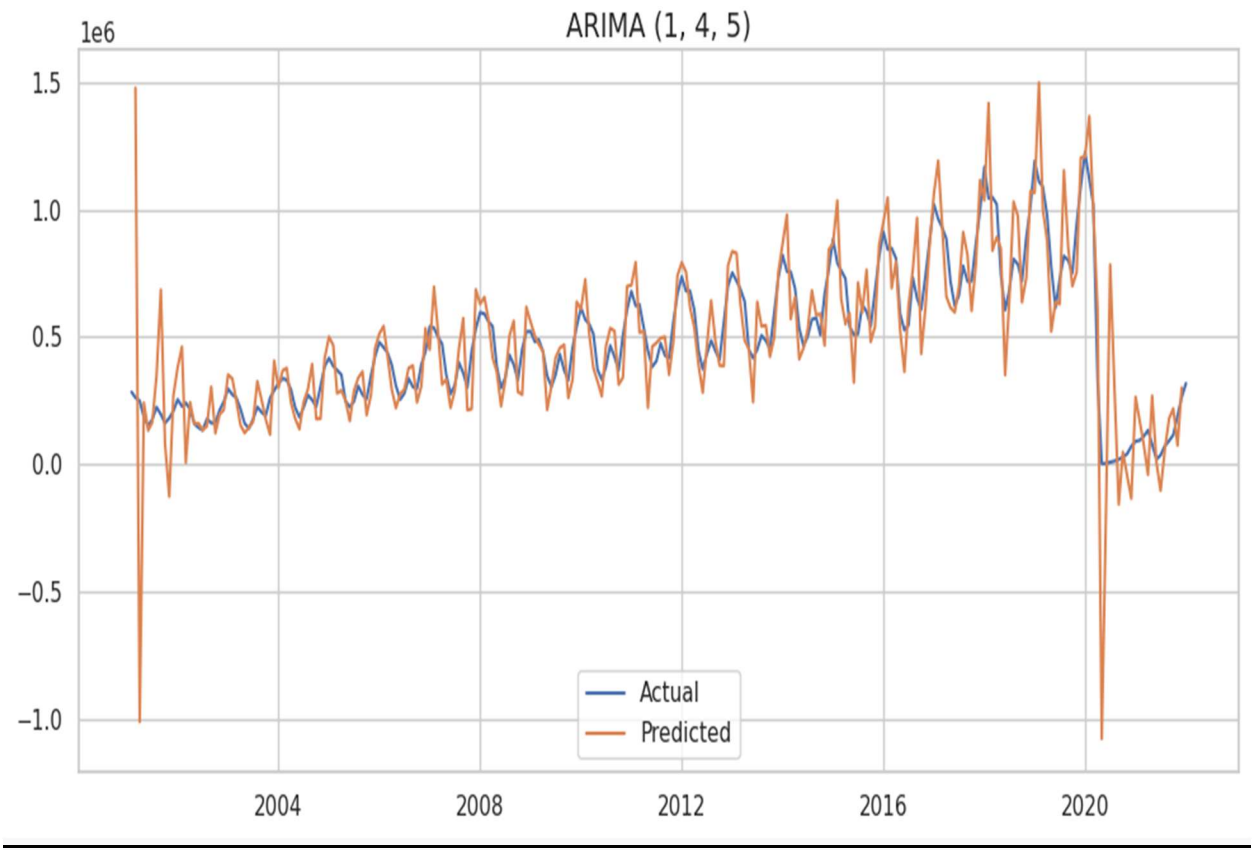
Additionally, the code marks a specific point in the plot with a red star marker, representing the best ARIMA parameter values. The AIC value associated with the best parameters is displayed as a text annotation next to the marked point.

When you run this code, it should display the 3D scatter plot with the best parameter point highlighted and the AIC value annotated.

Hence we get best paremeter for our model at AIC:(1,4,5).

20. Visualize the actual values and the predicted values generated by an ARIMA model.

```
import matplotlib.pyplot as plt

# Your previous code snippet here

# Create the figure and axes
fig, ax = plt.subplots(figsize=(10, 5))

# Get the predicted values
predictions = arima_fit.predict(start=1, end=250)

# Plot the actual values and the predictions
ax.plot(ts["Tourist"], label="Actual")
ax.plot(predictions, label="Predicted")

# Set the title of the plot
ax.set_title("ARIMA {}".format(order), size=14)

# Add a legend
ax.legend()

# Adjust the layout and display the plot
fig.tight_layout()
plt.show()
```

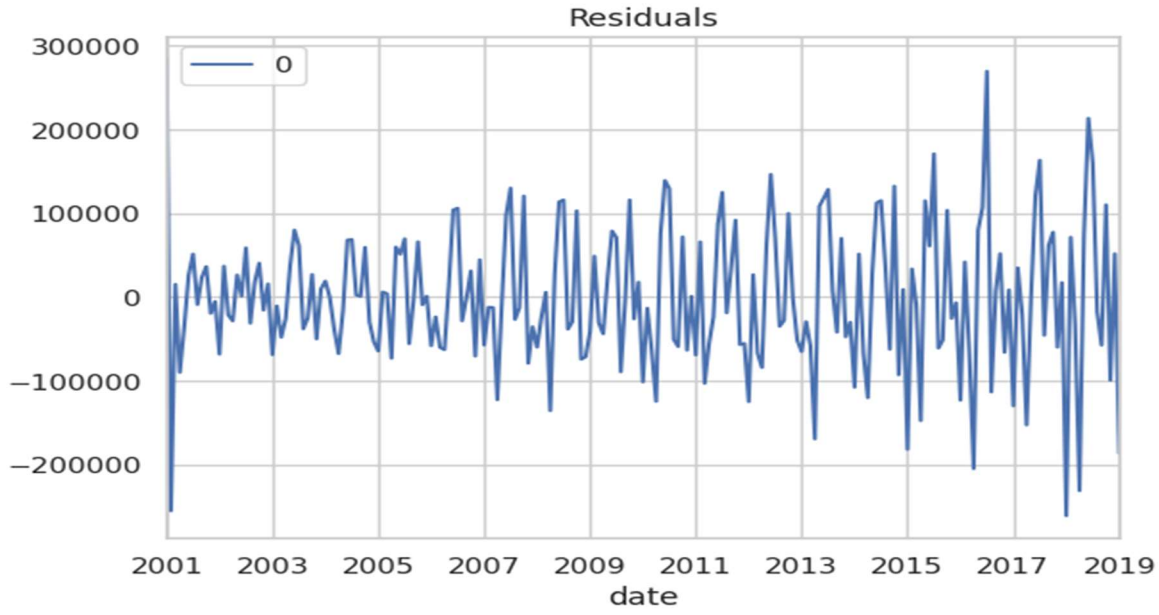


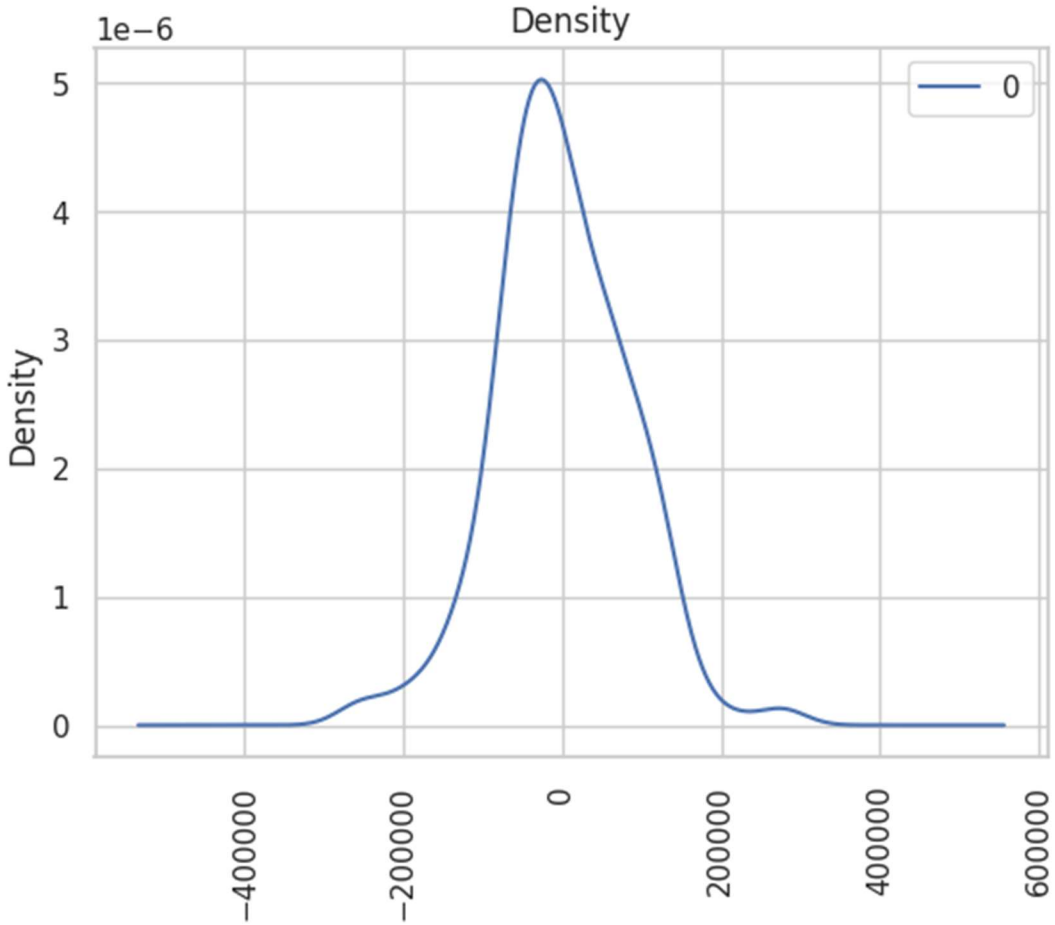
From above graph you can see our actual values are almost close to our predicted values.

you can visually compare the actual tourist values with the predicted values generated by the ARIMA model, gaining insights into the model's performance and the accuracy of its predictions.

21. RESIDUALS

```
residuals = pd.DataFrame(arima_fit.resid)
residuals.plot(title="Residuals")
residuals.plot(kind='kde', title='Density')
plt.xticks(rotation=90)
plt.show()
```


Residuals
0
300000
200000
100000
0
−100000
−200000
2001
2003
2005
2007
2009
2011
2013
2015
2017
2019
date


Density
1e−6
0
5
4
3
2
1
0
Density
−400000
−200000
0
200000
400000
600000

By examining these plots, you can gain insights into the performance and assumptions of the ARIMA model. Ideally, the residuals should exhibit no obvious patterns, be centered around zero, and follow a normal distribution. Any significant patterns or deviations in the plots may indicate areas for improvement or violations of the model assumptions.

==========
==========

CONCLUSION

- Based on the provided list of root mean square errors (RMSE) for different ARIMA models, the order (2, 2, 2) achieved the lowest RMSE value of 315,237.0861637732. Therefore, we can conclude that the ARIMA model with seasonal parameters p=2, d=2, and q=2 is the best fit for the given time series data.

To further solidify the conclusion, you can proceed with the following steps:

1. **Fit the ARIMA model:**

Create an ARIMA model with the order `(3,2,4)` using the training data.

2. **Evaluate the model:**

   Use the testing data to assess the model's performance. Calculate additional evaluation metrics such as mean absolute error (MAE) or mean percentage error (MPE) to further gauge the accuracy of the forecasts.

3. **Compare with other models:**

   Compare the performance of the chosen ARIMA model with other models that were tested. Consider factors such as simplicity, computational efficiency, and interpretability in addition to the RMSE. We get RMSE= **`946735.2417862881`**

4. **Examine model summary:**

   Generate a summary of the ARIMA model's parameters, coefficients, and statistical measures using the fitted model. This summary provides insights into the significance and

impact of each parameter on the model's performance. We get best Arima model at (2, 2, 2)

5. **Visualize the results:**
   Plot the actual values of the time series along with the predicted values obtained from the ARIMA model. This visualization helps in assessing the model's ability to capture patterns, trends, and irregularities in the data.
6. **Validate the conclusions:** If the ARIMA model with order (2, 2, 2) consistently outperforms other models across multiple evaluation metrics and validation techniques, it provides stronger evidence to support the conclusion that this is the best ARIMA model for the given time series data.

7.Remember that the conclusion should take into account both quantitative evaluation metrics and qualitative analysis, considering the specific characteristics and context of the data.

8.Then check iterates over different combinations of ARIMA parameters, fits the models, and calculates the AIC. It keeps track of the best parameter values and the lowest AIC encountered. Finally, it outputs the best parameter values and the corresponding AIC. We get best parameter AIC:(1,4,5).

9.Remember that this approach assumes that the lowest AIC value indicates the best model fit. Depending on your specific context and requirements, you may also consider other evaluation metrics or conduct additional validation before selecting the final ARIMA parameter values.

==========