Time Series Analysis on

SALES AND CUSTOMER BEHAVIOUR

Project work Submitted to

The Department of Statistics

Pondicherry University

By

Mr. Pratik Pandey

Regd. No.: 23375056

&

Mr. Digvijay Singh

Regd. No.: 23375021

As an Assignment for the

Partial Fulfilment of the Course work

STAT-414 Programming in R (Lab Based)

Under the Guidance of

Dr. P. Tirupathi Rao

Professor & Course Teacher

April 2025

DR.TIRUPATHI RAO PADI

M.Sc., M.Phil., Ph.D., M.B.A., D.C.A., CP-SAS

PROFESSOR,Department of Statistics

Ramanujan School of Math. Sciences

PONDICHERRY UNIVERSITY

(A Central University)

R.V. Nagar, Kalapet,

Puducherry (UT) – 605014, India

Email: drtrpadi@pondiuni.ac.in;

traopadipu@gmail.com,

Phone: 9486492241 (Mobile),

9629862241(WAN),0413-2654-829(Office),

Certificate

This is to certifiy that Mr. Pratik Pandey and Mr. Digvijay Singh have completed their project work as partial fulfilment for the course work of **STAT 418 – TIME SERIES ANALYSIS** submitted to the department of Statistics, Pondicherry University in April 2025. They have carried out all the stages of project work right from the data collection to repot making, on their own. The data that they have collected is from a real time context. No part of this work was carried out earlier in any format by any one for M.Sc./ MBA/ M.Tech/ etc. dissertation works or Ph.D. thesis.

Date: 20.04.2025 **(P.Tirupathi Rao)**

TABLE OF CONTENTS

CHAPTER-1

INTRODUCTION

Abstract:

This project explores **Time Series Analysis (TSA)** as a method for understanding patterns in temporal data and predicting future values. It emphasizes how time-ordered data, especially in the retail domain, can reveal trends, seasonality, and irregular patterns. The analysis leverages models such as ARIMA and SARIMA to forecast customer behaviour based on historical sales data, providing actionable insights for decision-making.

Objectives of Project:

- To analyse customer shopping data using Time Series Analysis techniques.
- To identify key patterns such as trends, seasonality, and irregularities in sales data.
- To build predictive models (e.g., ARIMA, SARIMA) for forecasting future sales.
- To evaluate the performance of time series models and interpret forecasting accuracy.
- To support data-driven decision-making in retail by leveraging time-based insights.

Description of the Problem:

Retail businesses often struggle with accurately predicting customer demand and sales patterns. Without proper forecasting, businesses may face issues like overstocking or stockouts. This project addresses this challenge by applying TSA to a dataset of shopping transactions to model and predict future sales.

Significance of the Problem:

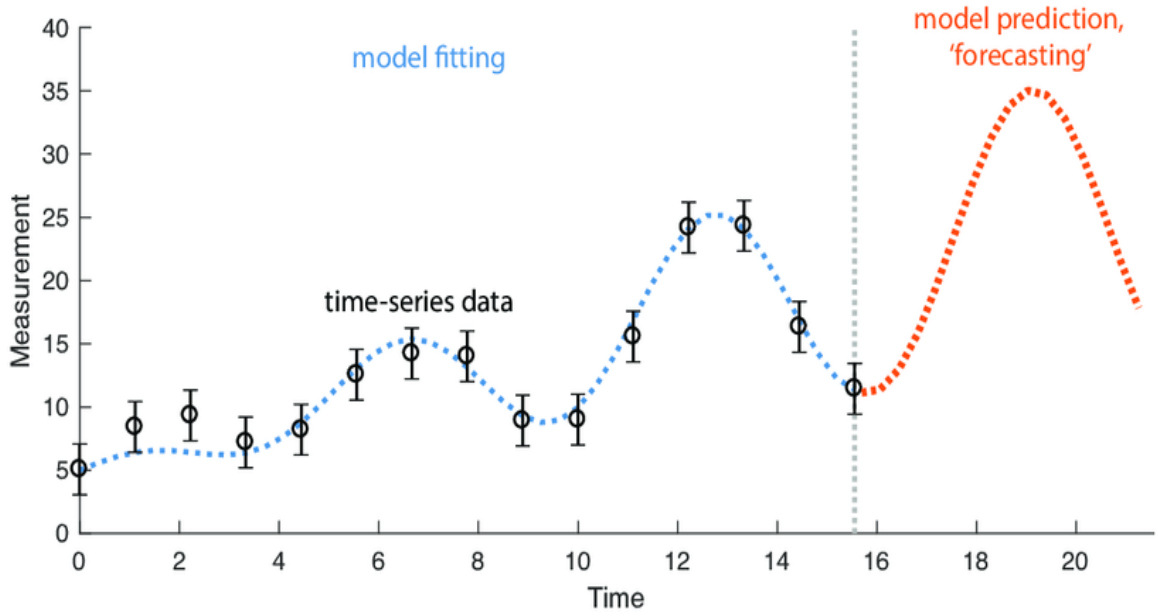
Accurate forecasting enables better inventory management, strategic marketing, and financial planning. For retail businesses, understanding how sales vary over time helps in optimizing operations and responding to market demand effectively.

Motivation Behind Selecting the Problem:

The increasing importance of data analytics in retail and the availability of rich, time-stamped transaction data motivated the selection of this problem. Applying TSA provides an opportunity to turn raw data into actionable insights for business optimization.

Introduction of Time Series Analysis:

In mathematics, a time series is a series of data points indexed in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Thus it is a sequence of discrete-time data.
Time series analysis is a specific way of analysing a sequence of data points collected over an interval of time. In time series analysis, analysts record data points at consistent intervals over a set period of time rather than just recording the data points intermittently or randomly



Graph of Time Series Data

- A Time-Series represents a series of time-based orders. It would be Years, Months, Weeks, Days, Horus, Minutes, and Seconds
- A time series is an observation from the sequence of discrete-time of successive intervals.

- A time series is a running chart.
- The time variable/feature is the independent variable and supports the target variable to predict the results.
- Time Series Analysis (TSA) is used in different fields for time-based predictions – like Weather Forecasting, Financial, Signal processing, Engineering domain – Control Systems, Communications Systems.
- Since TSA involves producing the set of information in a particular sequence, it makes a distinct from spatial and other analyses.
- Using AR, MA, ARMA, and ARIMA models, we could predict the future.

Sequence of getting time series prediction

Time series analysis comprises methods for analysing time series data to extract meaningful statistics and other characteristics of the data. Time Series forecasting is the use of a model to predict future values based on previously observed values. Generally, time series data is modelled as a stochastic process. While regression analysis is often employed in such a way as to test relationships between one or more different time series, this type of analysis is not usually called "time series analysis", which refers to relationships between different points in time within a single series.

What sets time series data apart from other data is that the analysis can show how variables change over time. In other words, time is a crucial variable because it shows how the data adjusts over the course of the data points as well as the results. It provides an additional source of information and a set order of dependencies between the data.

2. Important Terms to understand for Time Series Analysis:

A Time-Series is a sequence of data points collected at different timestamps. These are essentially successive measurements collected from the same data source at the same time interval. Further, we can use these chronologically gathered readings to monitor trends and changes over time. The time-series models can be univariate or multivariate. The univariate time series models are implemented when the dependent variable is a single time series, like room temperature measurement from a single sensor. On the other hand, a multivariate time series model can be used when there are multiple dependent variables, i.e., the output depends on more than one series. An example for the multivariate time-series model could be modelling the GDP, inflation, and unemployment together as these variables are linked to each other.

1. Stationary and Non-Stationary Time Series

Stationarity is a property of a time series. A stationary series is one where the values of the series is not a function of time. That is, the statistical properties of the series like mean, variance and autocorrelation are constant over time. Autocorrelation of the series is nothing but the correlation of the series with its previous values, more on this coming up. **A stationary time series id devoid of seasonal effects as well.**

2. Trend

The trend shows a general direction of the time series data over a long period of time. A trend can be increasing(upward), decreasing(downward), or horizontal(stationary).

3.Seasonality

The seasonality component exhibits a trend that repeats with respect to timing, direction, and magnitude. Some examples include an increase in water consumption in summer due to hot weather conditions.

4. Cyclical Component

These are the trends with no set repetition over a particular period of time. A cycle refers to the period of ups and downs, booms and slums of a time series, mostly observed in business cycles. These cycles do not exhibit a seasonal variation but generally occur over a time period of 3 to 12 years depending on the nature of the time series.

5. Irregular Variation

These are the fluctuations in the time series data which become evident when trend and cyclical variations are removed. These variations are unpredictable, erratic, and may or may not be random.

6. ETS Decomposition

ETS Decomposition is used to separate different components of a time series. The term ETS stands for Error, Trend and Seasonality.

7. Dependence

It refers to the association of two observations of the same variable at prior time periods.

8. Differencing

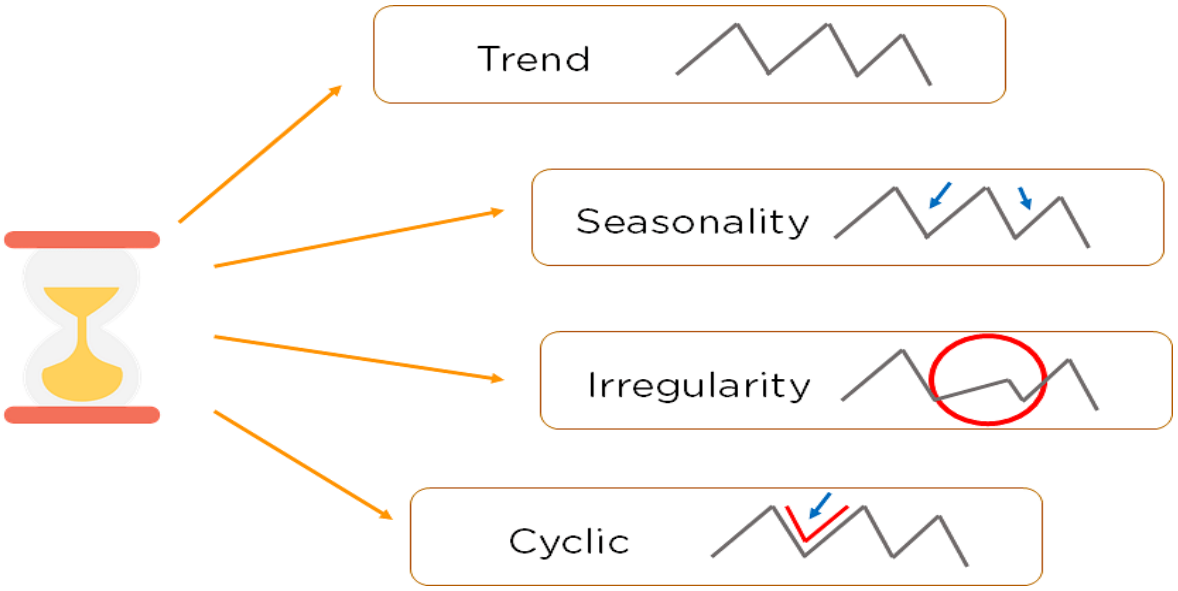
Differencing is used to make the series stationary and to control the auto-correlations. There may be some cases in time series analyses where we do not require differencing and over-differenced series can produce wrong estimates.

9. Specification

It may involve the testing of the linear or non-linear relationships of dependent variables by using time series models such as ARIMA models.

10. ARIMA

ARIMA stands for Auto Regressive Integrated Moving Average.



Patterns of different components of time series data

CHAPTER-2

Data Collection and Methodology

3. Data and Modules:

Data Collection:

- **Dataset Used:** Customer shopping transaction data from Kanyon Shopping Mall.

 Focus Area: Clothing category purchases made using **credit cards**.

 Key Variables:

  - invoice_date (timestamp)
  - category
  - payment_method
  - price
  - quantity

 Source: The dataset was accessed via a public Google Drive link and loaded into a Python environment for analysis.

 Nature of Data: Time-stamped, daily transaction records—ideal for time series modeling and forecasting.

Methodology:

1. **Data Preprocessing:**
   - Handled missing values using interpolation and smoothing techniques.
   - Converted date columns to datetime format.
   - Aggregated data to calculate daily total sales (`price × quantity`).
2. **Exploratory Data Analysis (EDA):**
   - Visualized time series data using line plots to identify trends and outliers.
   - Applied decomposition to isolate **trend**, **seasonality**, and **residual** components.
3. **Stationarity Testing:**
   - Conducted Augmented Dickey-Fuller (ADF) and KPSS tests to determine stationarity.
   - Differencing was applied to make the series stationary where needed.
4. **Modeling Techniques:**
   - Implemented **ARIMA** and **SARIMA** models for forecasting.
   - Chose model parameters using ACF and PACF plots.
   - Evaluated model performance using AIC, BIC, and residual analysis.

5. **Forecasting:**
    - Used the trained SARIMA model to forecast future sales, including a prediction of total revenue generated on for January 1, 2024.
    - Visualized the forecast with confidence intervals.

Variables of the Dataset:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2395 entries, 0 to 99391
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   invoice_no      2395 non-null   object 
 1   customer_id     2395 non-null   object 
 2   gender          2395 non-null   object 
 3   age             2395 non-null   int64  
 4   category        2395 non-null   object 
 5   quantity        2395 non-null   int64  
 6   price           2395 non-null   float64
 7   payment_method  2395 non-null   object 
 8   invoice_date    2395 non-null   object 
 9   shopping_mall   2395 non-null   object 
dtypes: float64(1), int64(2), object(7)
memory usage: 205.8+ KB
```

First Five Rows of Table:

```
import numpy as np
import pandas as pd

import matplotlib as mlt
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings
warnings.filterwarnings('ignore')

import os

train=pd.read_csv('customer_shopping_data.csv')
train.head()
```

Ouput:

	invoice_no	customer_id	gender	age	category	quantity	price	payment_method	invoice_date	shopping_mall
0	I138884	C241288	Female	28	Clothing	5	1500.40	Credit Card	2022-08-05	Kanyon
1	I317333	C111565	Male	21	Shoes	3	1800.51	Debit Card	2021-12-12	Forum Istanbul
2	I127801	C266599	Male	20	Clothing	1	300.08	Cash	2021-11-09	Metrocity
3	I173702	C988172	Female	66	Shoes	5	3000.85	Credit Card	2021-05-16	Metropol AVM
4	I337046	C189076	Female	53	Books	4	60.60	Cash	2021-10-24	Kanyon

CHAPTER-3

Data Analysis

4. Data Visualization:

Why Visualize Your Data?

- **Spot patterns & trends** (e.g., sales over time, popular product categories)
- **Detect outliers** or anomalies (e.g., extremely high prices)
- **Understand distributions** (e.g., customer age or purchase quantity)
- **Explore relationships** between variables (e.g., does age affect spending?)

```
sns.set(rc={'figure.figsize':(24,8)})
ax=sns.lineplot(data=train,x='invoice_date',y='price',hue='shopping_mall')
ax.axes.set_title("\nBasic Time Series of Sales\n",fontsize=20)
print(ax)
```

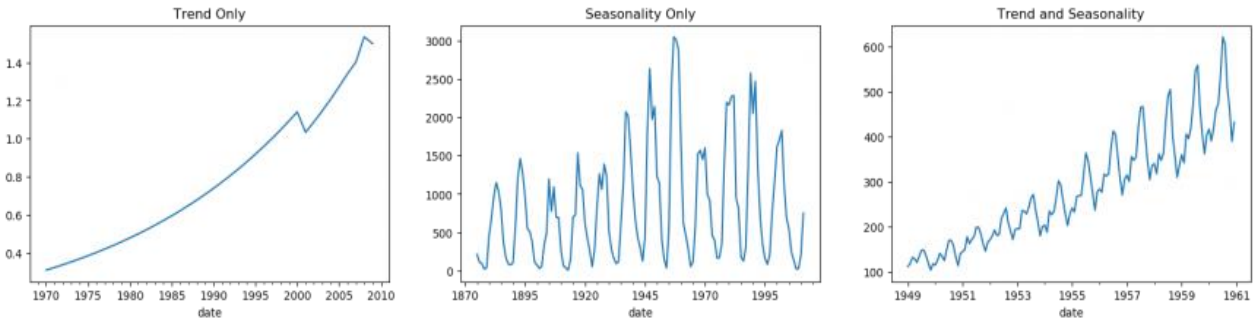
Line plot between invoice_date and price

- We can spot some spikes here!

📊 Interpretation of the Line Plot:

1. **Highly Granular Data:**
    - The plot looks *very dense* with many sharp spikes.
    - This suggests you're visualizing sales at a very fine level of granularity (possibly every transaction or per minute/hour level).
2. **Frequent Sales Spikes:**
    - Sales prices spike frequently across all malls, reaching values over 5000 at times.
    - Indicates irregular, high-value purchases — possibly bulk buying or special events.
3. **Noisy Series:**
    - There's a lot of overlapping and randomness—meaning it's hard to distinguish clear trends or seasonality at this resolution.
    - This can happen when data isn't aggregated (e.g., daily or weekly totals would be clearer).
4. **Mall Comparison:**
    - Every shopping mall has its own line (thanks to hue='shopping_mall').
    - However, due to the noise, it's tough to tell which mall performs better without smoothing or summarizing the data.
5. **Visual Overcrowding:**
    - Because of many lines and data points, the x-axis labels are cluttered and unreadable.
    - This suggests a need for plot simplification—either by plotting fewer malls, aggregating by time (daily/weekly), or applying a moving average.

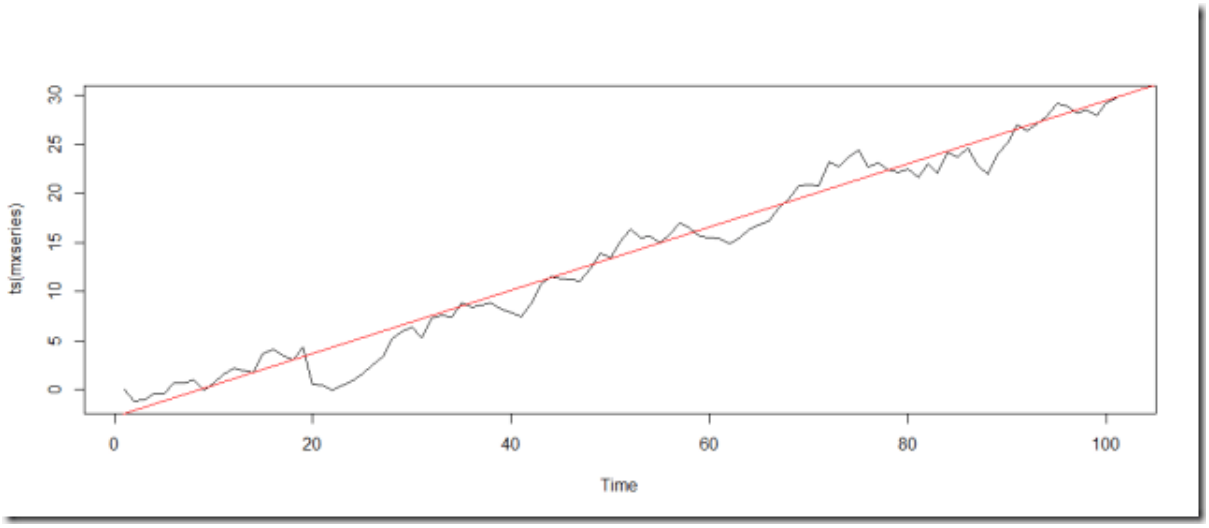
There are several types of patterns that can be found.



5. Patterns Recognition – Trend:

A trend is observed when there is an increasing or decreasing slope observed in the time series.

- **Linear Trend**: Maintains a straight line.
- **Polynomial Trend**: Shows polynomial curves.
- **Exponential Trend**: Shows exponential growths or fall.
- **Log-based Trend**: Shows log-based growths or fall.



Graph showing Trend between Time and ts

- Source: oraylis.de

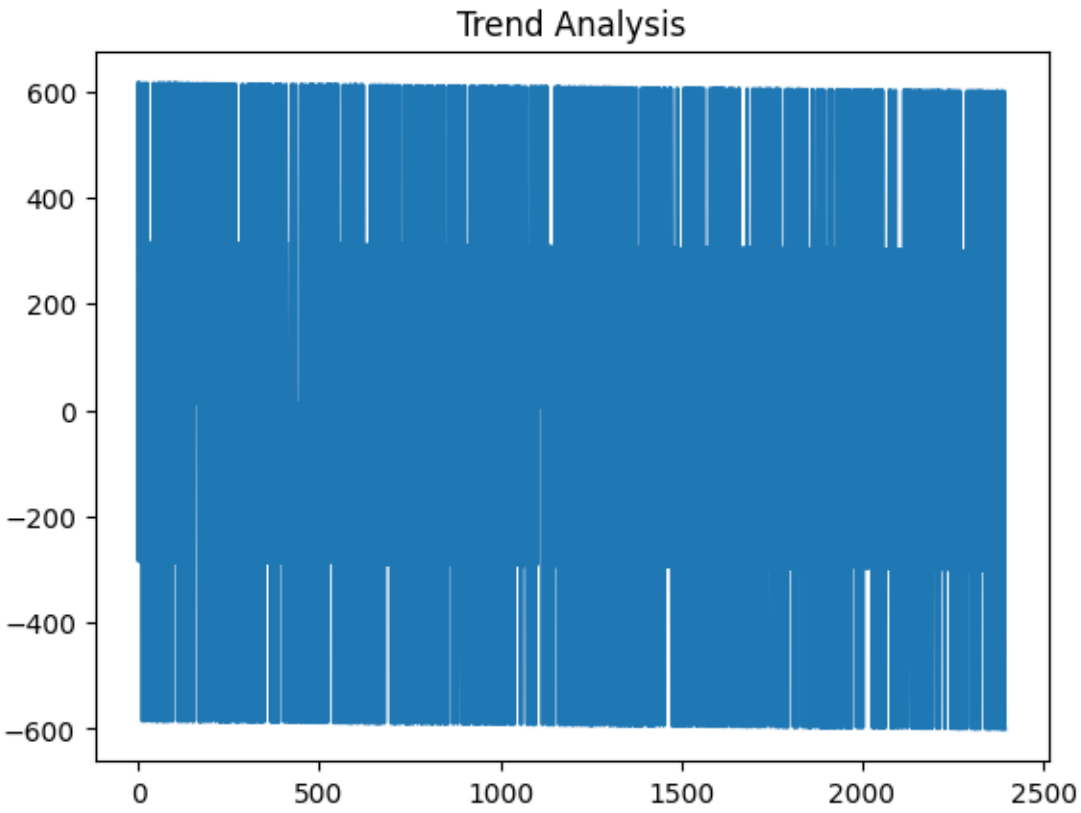
How to detrend a time series?

Detrending a time series is to remove the trend component from a time series. But how to extract the trend? There are multiple approaches.

- Subtract the line of best fit from the time series. The line of best fit may be obtained from a linear regression model with the time steps as the predictor. For more complex trends, you may want to use quadratic terms (x^2) in the model.
- Subtract the trend component obtained from time series decomposition we saw earlier.
- Subtract the mean
- Apply a filter like Baxter-King filter(**statsmodels.tsa.filters.bkfilter**) or the Hodrick-Prescott Filter (**statsmodels.tsa.filters.hpfilter**) to remove the moving average trend lines or the cyclical components.

Trend Analysis of the dataset used:

```
from scipy import signal
df = train[(train['payment_method']=='Credit Card') & (train['category']=='Clothing')& (train['shopping_mall']=='Kanyon')]
detrended = signal.detrend(df.price)
plt.plot(detrended)
plt.title('Trend Analysis')
plt.show()
```

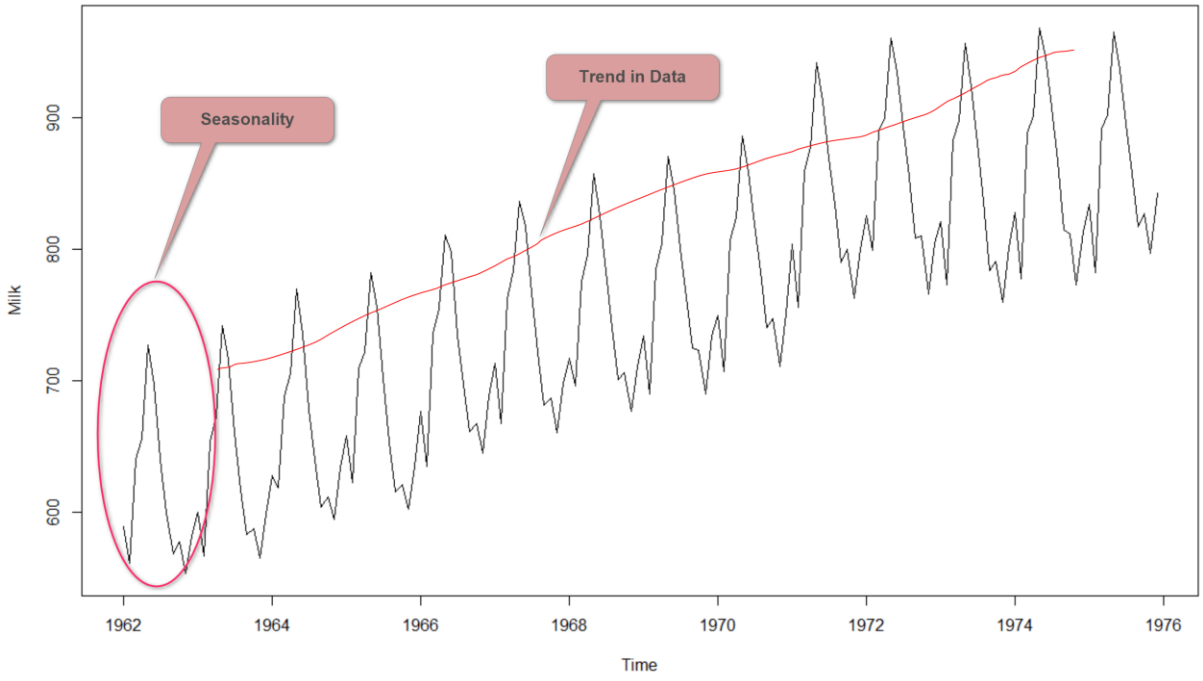


This Trend Analysis plot aims to show trend of sales data for the Clothing items purchased using Credit Card at Kanyon Shopping Mall. It uses a technique called **detrending** to remove the underlying trend from the time series data.

6. Patterns Recognition – Seasonality:

A seasonality is observed when there is a distinct repeated pattern observed between regular intervals due to seasonal factors. It could be because of the month of the year, the day of the month, weekdays or even time of the day.

- **Yearly**: Example - Black Friday and Christmas Sales
- **Monthly**: Example - We may find big sales on first week of the month as slalry is paid that time.



Graph showing Seasonality

- Source: radacad.com

Seasonality of the dataset:

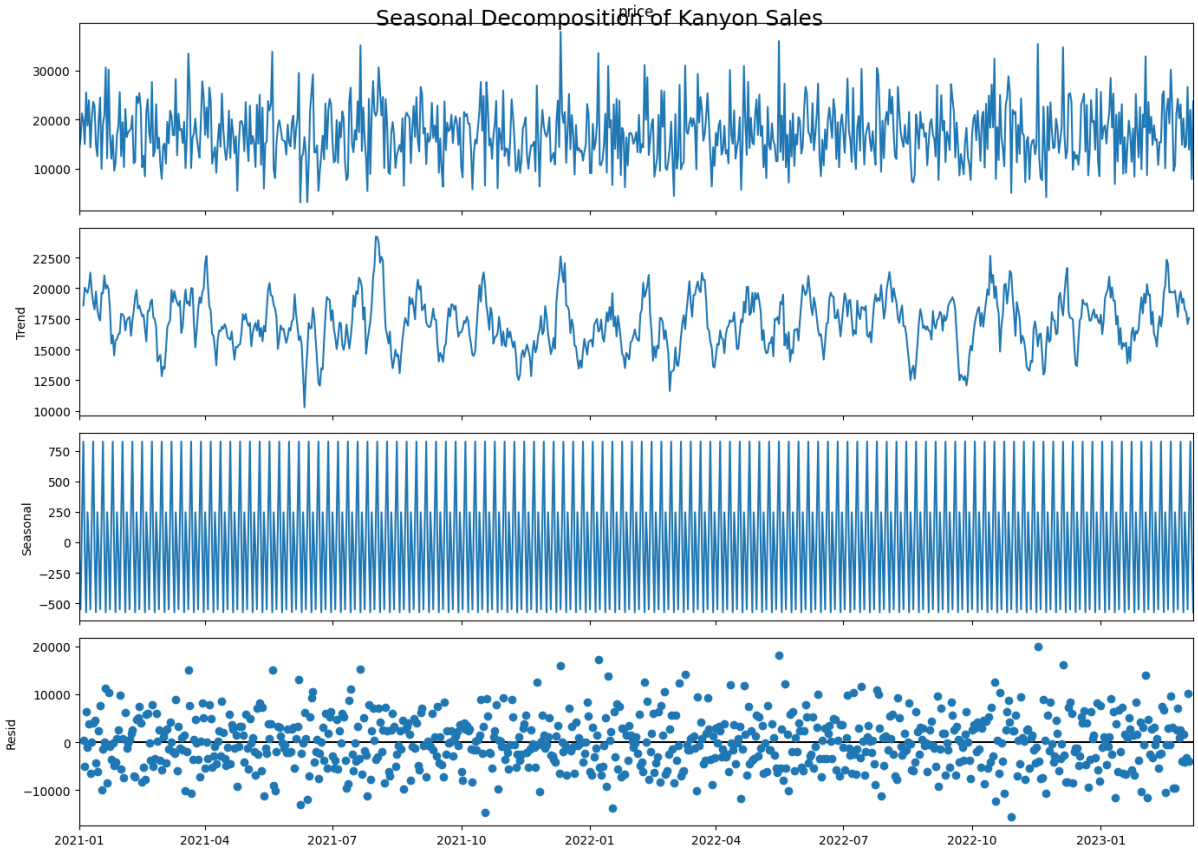
 A **line plot** showing monthly total sales (useful for spotting trends).

 A **decomposition plot** that splits sales into:

- **Trend**: long-term increase/decrease.
- **Seasonality**: repeating patterns (e.g., holiday spikes).
- **Residual**: noise/unexplained data.

```python
from statsmodels.tsa.seasonal import seasonal_decompose

# Step 1: Prepare your data
# Ensure 'invoice_date' is datetime
train['invoice_date'] = pd.to_datetime(train['invoice_date'])

# Step 2: Aggregate daily sales for one mall (e.g., Kanyon)
daily_sales = train[train['shopping_mall'] == 'Kanyon'].groupby('invoice_date')['price'].sum()

# Optional: Resample to ensure consistent frequency (daily)
daily_sales = daily_sales.asfreq('D').fillna(method='ffill')  # fill missing with previous day's value

# Step 3: Decompose the time series
decomposition = seasonal_decompose(daily_sales, model='additive', period=7)  # use 7 for weekly seasonality

# Step 4: Plot the decomposition
plt.rcParams.update({'figure.figsize': (14, 10)})
decomposition.plot()
plt.suptitle('Seasonal Decomposition of Kanyon Sales', fontsize=18)
plt.show()
```

Output:



How to test for seasonality of a time series?

The common way is to plot the series and check for repeatable patterns in fixed time intervals. So, the types of seasonality is determined by the clock or the calendar:

- Hour of day
- Day of month
- Weekly
- Monthly
- Yearly

Insights from Seasonal Decomposition of Kanyon Mall Sales:

1. Overall Trend (📊 Trend Component):

- **What it shows:** A smoothed version of your sales data over time, highlighting long-term upward or downward movement.
- **Insights you might observe:**
  - A **gradual increase** in trend → Kanyon Mall is becoming more popular over time.
  - A **declining trend** → Sales might be dropping due to seasonality, market saturation, or competition.
  - **Flat trend** → Stable customer flow, no major growth or decline.

✅ **Business Implication:** Helps in strategic planning—do you need marketing boosts or expansion?

2. Seasonal Patterns (🔁 Seasonal Component):

- **What it shows:** Repeating patterns every 7 days (since `period=7`) — like weekly cycles.
- **Insights you might observe:**
  - **Spikes on weekends (Fri–Sun)** → Typical for malls when foot traffic increases.
  - **Drops on weekdays (e.g., Monday or Tuesday)** → Slower days, common in retail.

✅ **Business Implication:** You can schedule promotions, staffing, or campaigns around high-sales days.

3. Residuals (📉 Residual/Noise):

- **What it shows:** What's *not* explained by trend or seasonality — basically, anomalies or irregular behavior.
- **Insights you might observe:**
  - **Sharp positive spikes** → Flash sales, holiday events, or high-volume transactions.

- **Negative dips** → Unexpected drops (e.g., technical issues, weather problems, or lockdowns).

✅ **Business Implication:** Identify *unusual* days to investigate and learn from—was it a campaign? Was there an error?

4. Original Series:

- Shows the actual day-to-day fluctuations in sales — a combination of all three components.
- The spiky nature means **high volatility** in daily revenue, which can mask deeper trends—hence decomposition is useful.

Autocorrelation of the dataset:

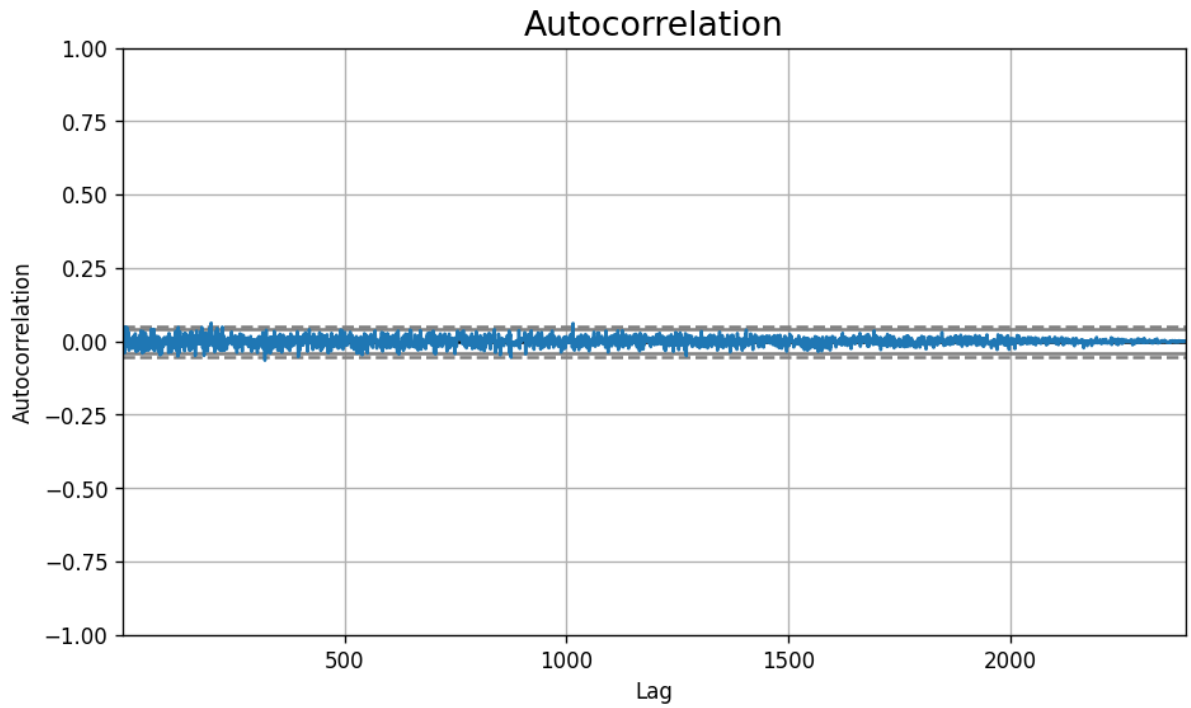
Autocorrelation (also called **serial correlation**) measures the **correlation of a time series with a lagged version of itself**.

📈 **Why is Autocorrelation Important?**

- Detects **seasonality**: e.g., sales every 12 months might be correlated.
- Reveals **lags** that are useful for time series models (like ARIMA or LSTM).
- Helps determine the **order of autoregressive models** (AR).

Here, the Autocorrelation aims to analyse the sales price of **Clothing** category using **Credit Card** at **Kanyon** shopping mall in the dataset.

```
from pandas.plotting import autocorrelation_plot
df = train[(train['payment_method']=='Credit Card') & (train['category']=='Clothing')& (train['shopping_mall']=='Kanyon')]
plt.rcParams.update({'figure.figsize':(9,5), 'figure.dpi':120})
autocorrelation_plot(df.price.tolist())
plt.title('Autocorrelation', fontsize=16)
plt.plot()
```



Insights from the Autocorrelation Plot:

1. Very Low Autocorrelation

- The values of autocorrelation at all lags are **close to zero**.
- This means there's **no strong relationship between today's clothing sales and past sales** on previous days for the same mall, category, and payment method.

2. No Seasonal or Periodic Patterns Detected

- If there was **weekly or monthly seasonality**, you would see regular spikes at certain lags (like lag 7, 14, etc.).
- Here, the line **stays within the confidence band**, showing **no significant repeating patterns**.

3. Highly Random or Noise-Like Behaviour

- This specific segment of data behaves **almost like white noise** — unpredictable and non-patterned.
- Could indicate that **credit card clothing sales at Kanyon Mall** are highly influenced by **external/random factors** like promotions, personal preferences, or ad-hoc visits.

7. Additive and Multiplicative Time Series:

We may have different combinations of trends and seasonality. Depending on the nature of the trends and seasonality, a time series can be modelled as an additive or multiplicative time series. Each observation in the series can be expressed as either a sum or a product of the components.

Additive time series:
Value = Base Level + Trend + Seasonality + Error

Multiplicative Time Series:
Value = Base Level x Trend x Seasonality x Error

8. Stationarity:

A stationary time series has **statistical properties or moments** (e.g., mean and variance) that do not vary in time. Stationarity, then, is the status of a stationary time series. Conversely, non-stationarity is the status of a time series whose statistical properties are changing through time.

Data points are often non-stationary or have means, variances, and covariances that change over time. **Non-stationary behaviours can be trends, cycles, random walks, or combinations of the three**. Non-stationary data, as a rule, are unpredictable and cannot be modelled or forecasted.

How to make a time series stationary?

We can apply some sort of transformation to make the time-series stationary. This transformation may include:

- Differencing the Series (once or more)
- Take the log of the series
- Take the nth root of the series
- Combination of the above

Why make a non-stationary series stationary before forecasting?

Forecasting a stationary series is relatively easy and the forecasts are more reliable. An important reason is, autoregressive forecasting models are essentially linear regression models that utilize the lag(s) of the series itself as predictors. We know that linear regression works best if the predictors (X variables) are not correlated against each other. So, stationarizing the series solves this problem since it removes any persistent autocorrelation, thereby making the predictors(lags of the series) in the forecasting models nearly independent.

How to test for stationarity?

The stationarity of a series can be established by looking at the plot of the series like we did earlier.

Another method is to split the series into 2 or more contiguous parts and computing the summary statistics like the mean, variance and the autocorrelation. If the stats are quite different, then the series is not likely to be stationary.

Nevertheless, you need a method to quantitatively determine if a given series is stationary or not. This can be done using statistical tests called 'Unit Root Tests'. There are multiple variations of this, where the tests check if a time series is non-stationary and possess a unit root.

There are multiple implementations of Unit Root tests like:

- Augmented Dickey Fuller test (ADH Test)
- Kwiatkowski-Phillips-Schmidt-Shin – KPSS test (trend stationary)
- Philips Perron test (PP Test)

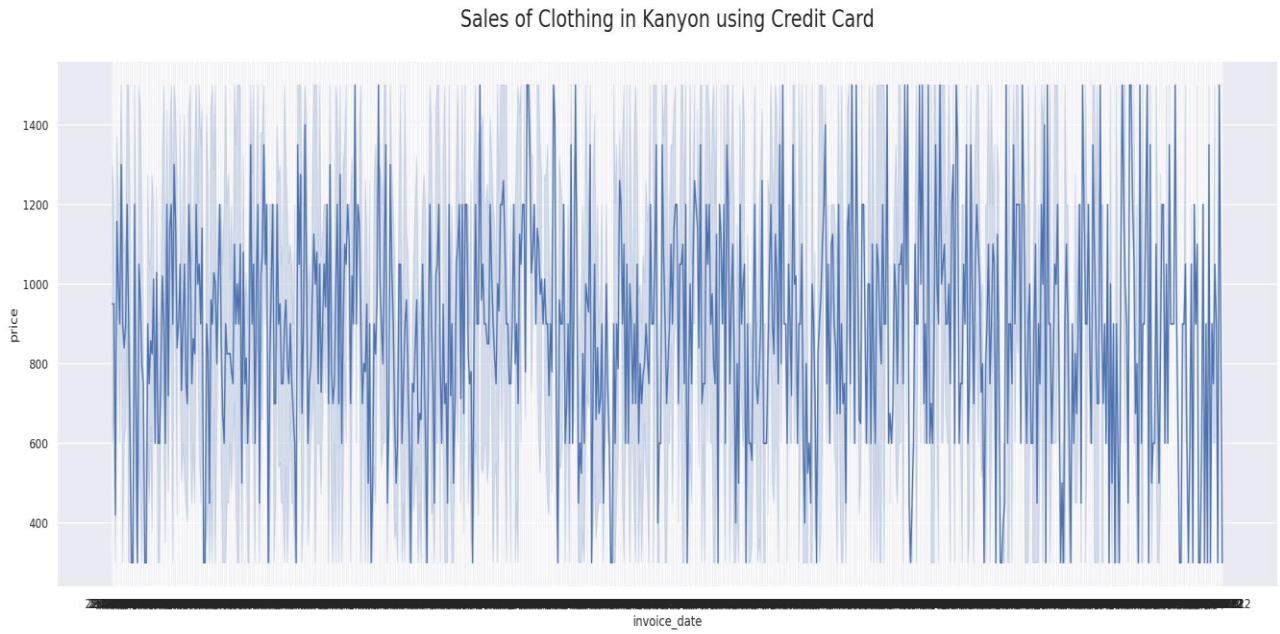
The most commonly used is the ADF test, where the null hypothesis is the time series possesses a unit root and is non-stationary. So, id the P-Value in ADH test is less than the significance level (0.05), you reject the null hypothesis.

The KPSS test, on the other hand, is used to test for trend stationarity. The null hypothesis and the P-Value interpretation is just the opposite of ADH test. The below code implements these two tests using stats models package in python.

Stationarity of the dataset used:

```
[ ] df = train[(train['payment_method']=='Credit Card') & (train['category']=='Clothing')& (train['shopping_mall']=='Kanyon')]

[ ] sns.set(rc={'figure.figsize':(24,8)})
    ax=sns.lineplot(data=df,x='invoice_date',y='price')
    ax.axes.set_title("\nSales of Clothing in Kanyon using Credit Card\n",fontsize=20);
```

Output:



Line plot of Sales of Clothing in Kanyon using Credit Card

🧠 Interpretation:

If your plot **shows a clear trend** (upward/downward), **wide variations**, or **periodic spikes**, it suggests the data is **non-stationary**.

Here's what non-stationarity in your dataset could indicate:

Observation	Meaning
📈 **Trend over time**	Sales are growing/shrinking — possibly due to promotions, seasons, or external factors
🔁 **Repeating patterns**	Possible **seasonality** (e.g., more sales on weekends or during holidays)
⚠️ **Fluctuating variance**	Indicates changing customer behaviour, marketing influence, or data noise
📉 **Non-stationary data**	Not ready for classic time series models like ARIMA unless differenced or transformed

Augmented Dickey Fuller test (ADH Test):

```
#Augmented Dickey Fuller Test

from statsmodels.tsa.stattools import adfuller
result = adfuller(df.price.values, autolag='AIC')
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

```
ADF Statistic: -19.677600731081167
p-value: 0.0
Critial Values:
   1%, -3.433090201041693
Critial Values:
   5%, -2.862750581542575
Critial Values:
   10%, -2.567414443618994
```

☑ **Interpretation**

- **ADF Statistic (-19.68)** is **much lower** than all critical values (even the strict 1% level).
- **p-value is 0.0**, which is **way below 0.05** (common significance level).

🤔 **Insight:**

☑ **Conclusion**: The time series **is stationary** — you **reject the null hypothesis** confidently.

Kwiatkowski-Phillips-Schmidt-Shin – KPSS test (trend stationary

```
#Kwiatkowski-Phillips-Schmidt-Shin – KPSS test (trend stationary)

from statsmodels.tsa.stattools import kpss
result = kpss(df.price.values, regression='c')
print('\nKPSS Statistic: %f' % result[0])
print('p-value: %f' % result[1])
for key, value in result[3].items():
    print('Critial Values:')
    print(f'   {key}, {value}');
```

```
KPSS Statistic: 0.071511
p-value: 0.100000
Critial Values:
   10%, 0.347
Critial Values:
   5%, 0.463
Critial Values:
   2.5%, 0.574
Critial Values:
   1%, 0.739
<ipython-input-25-7859e4726d24>:4: InterpolationWarning: The test statistic is outside of the range of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  result = kpss(df.price.values, regression='c')
```

☑ **Interpretation**

- **The KPSS Statistic (0.071511) is well below all critical values.**
- **The p-value is higher than 0.1, which means we fail to reject the null hypothesis.**

🧠 **Insight:**

☑ **Conclusion: The time series is stationary (in terms of level or trend) according to the KPSS test.**

What is the difference between white noise and a stationary series?

Like a stationary series, the white noise is also not a function of time, that is its mean and variance does not change over time. But the difference is, the white noise is completely random with a mean of 0. In white noise there is no pattern whatsoever. If you consider the sound signals in an FM radio as a time series, the blank sound you hear between the channels is white noise. Mathematically, a sequence of completely random numbers with mean zero is a white noise.

9. How to treat missing values in a time series?

Some effective alternatives to imputation are:

- Backward Fill
- Linear Interpolation
- Quadratic interpolation
- Mean of nearest neighbours
- Mean of seasonal counterparts

10. Autocorrelation and Partial Autocorrelation:

The term autocorrelation refers to the **degree of similarity between A) a given time series, and B) a lagged version of itself, over C) successive time intervals**. In other words, autocorrelation is intended to measure the relationship between a variable's present value and any past values that you may have access to.

When regression is performed on time series data, the errors may not be independent. Often errors are autocorrelated; that is, each error is correlated with the error immediately before it.

Autocorrelation is also a symptom of systematic lack of fit. The DW option provides the Durbin-Watson d statistic to test that the autocorrelation is zero:

$$d = \frac{\sum_{i=2}^{n}(e_i - e_{i-1})^2}{\sum_{i=1}^{n} e_i^2}$$

The value of d is close to 2 if the errors are uncorrelated. The distribution of d is reported by Durbin and Watson (1951). Tables of the distribution are found in most econometrics textbooks, such as Johnston (1972) and Pindyck and Rubinfeld (1981).

The sample autocorrelation estimate is displayed after the Durbin-Watson statistic. The sample is computed as

$$r = \frac{\sum_{i=2}^{n} e_i\, e_{i-1}}{\sum_{i=1}^{n} e_i^2}$$

This autocorrelation of the residuals may not be a very good estimate of the autocorrelation of the true errors, especially if there are few observations and the independent variables have certain patterns. If there are missing observations in the regression, these measures are computed as though the missing observations did not exist.

Positive autocorrelation of the errors generally tends to make the estimate of the error variance too small, so confidence intervals are too narrow and true null hypotheses are rejected with a higher probability than the stated significance level. Negative autocorrelation of the errors generally tends to make the estimate of the error variance too large, so confidence intervals are too wide and the power of significance tests is reduced. With either positive or negative autocorrelation, least-squares parameter estimates are usually not as efficient as generalized least-squares parameter estimates.

Autocorrelation is the correlation between two observations at different points in a time series. For example, values that are separated by an interval might have a strong positive or negative correlation. When these correlations are present, they indicate that past values influence the current value. Analysts use the autocorrelation and partial autocorrelation functions to understand the properties of time series data, fit the appropriate models, and make forecasts.

In this post, I cover both the autocorrelation function and partial autocorrelation function. You will learn about the differences between these functions and what they can tell you about your data. In later posts, I'll show you how to incorporate this information in regression models of time series data and other time-series analyses.

Autocorrelation Function (ACF):-

Use the autocorrelation function (ACF) to identify which lags have significant correlations, understand the patterns and properties of the time series, and then use that information to model the time series data. From the ACF, you can assess the randomness and stationarity of a time series. You can also determine whether trends and seasonal patterns are present.

In an ACF plot, each bar represents the size and direction of the correlation. Bars that extend across the red line are statistically significant.

Partial Autocorrelation Function (PACF):-

The partial autocorrelation function is similar to the ACF except that it displays only the correlation between two observations that the shorter lags between those observations do not explain. For example, the partial autocorrelation for lag 3 is only the correlation that lags 1 and 2 do not explain. In other words, the partial correlation for each lag is the unique correlation between those two observations after partialling out the intervening correlations.

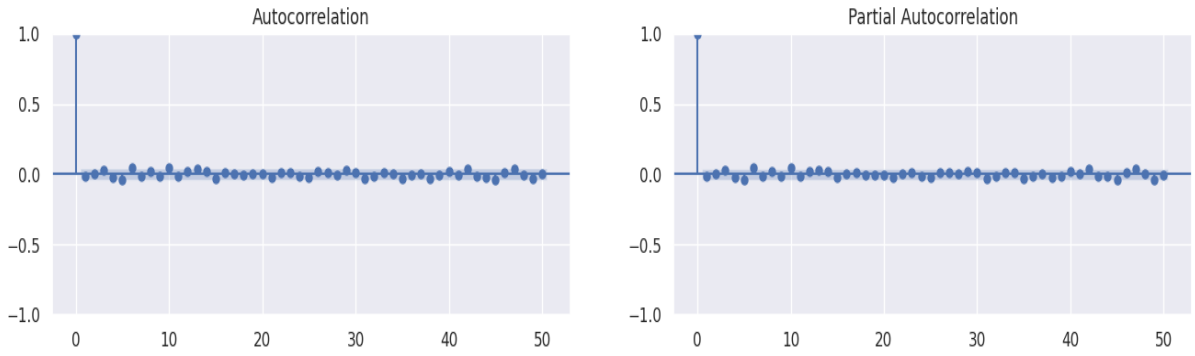
As you saw, the autocorrelation function helps assess the properties of a time series. In contrast, the partial autocorrelation function (PACF) is more useful during the specification process for an autoregressive model. Analysts use partial autocorrelation plots to specify regression models with time series data and Auto Regressive Integrated Moving Average (ARIMA) models. I'll focus on that aspect in posts about those methods.

Source: https://statisticsbyjim.com/time-series/autocorrelation-partial-autocorrelation/

ACF and PACF of the dataset used:

```
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

df = train[(train['payment_method']=='Credit Card') & (train['category']=='Clothing')& (train['shopping_mall']=='Kanyon')]

df['value']=df['price']
acf_50 = acf(df.value, nlags=50)
pacf_50 = pacf(df.value, nlags=50)

# Draw Plot
fig, axes = plt.subplots(1,2,figsize=(16,3), dpi= 100)
plot_acf(df.value.tolist(), lags=50, ax=axes[0])
plot_pacf(df.value.tolist(), lags=50, ax=axes[1])
```

Output:



Insights:

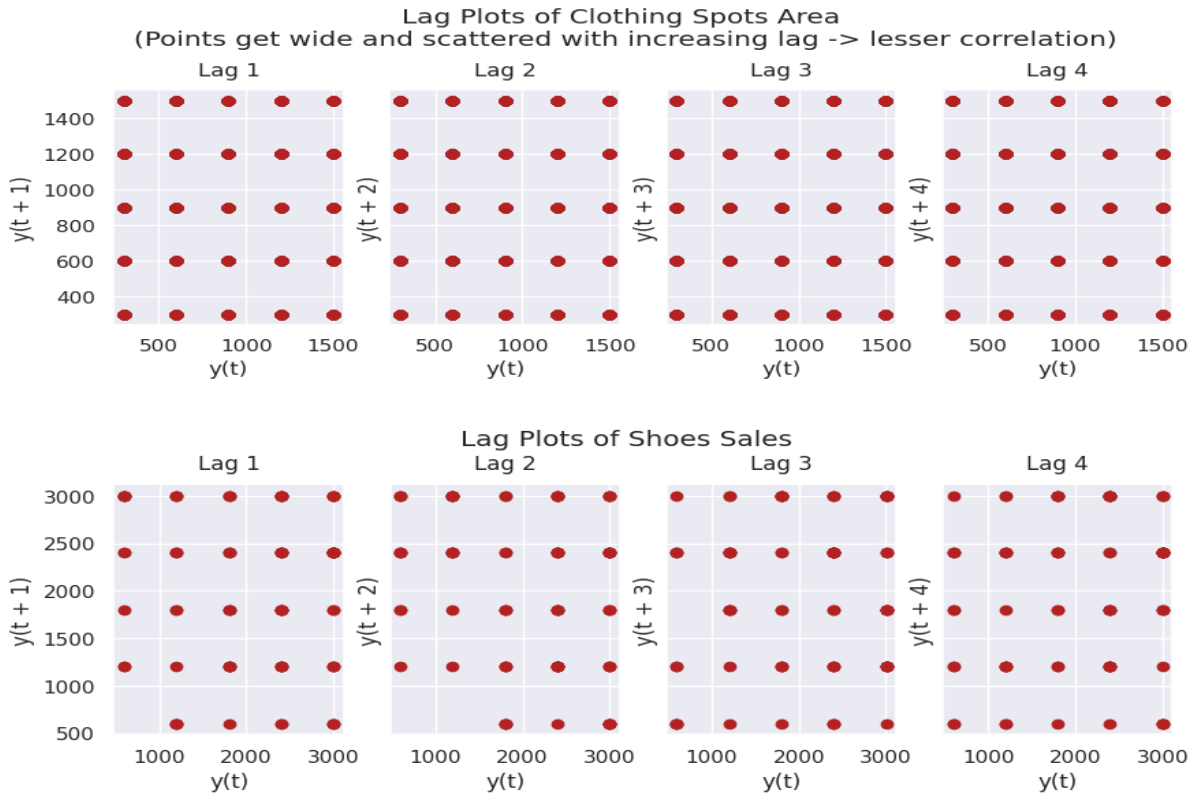
- **Dataset Context**: The customer_shopping_data.csv contains shopping transaction details, including price, payment_method, category, shopping_mall, and others. The filtered subset focuses on "Credit Card" payments for "Clothing" at "Kanyon" mall, suggesting an interest in analysing price behavior within this specific market segment.
- **Price Autocorrelation**: The ACF and PACF plots will reveal if there are predictable patterns in clothing prices at Kanyon when paid via credit card. For example:
    - A high autocorrelation at lag 1 would indicate that prices are strongly influenced by the previous transaction's price.
    - Significant spikes in the PACF at specific lags could suggest the number of periods to include in a time series model (e.g., ARIMA) for forecasting.
- **Segment-Specific Analysis**: By focusing on "Clothing" at "Kanyon" with "Credit Card" payments, the analysis might uncover unique pricing trends (e.g., seasonal sales, discounts) that differ from other categories or malls. The dataset shows varying prices for clothing (e.g., $300.08 to $1500.4), which could reflect different quantities or promotional activities.
- **Potential Applications**: This analysis could be useful for retailers at Kanyon to optimize pricing strategies or inventory for clothing, especially for credit card transactions, which might indicate a higher-spending customer base.

11. Lag Plots:

A Lag plot is a scatter plot of a time series against a lag of itself. It is normally used to check for autocorrelation.

```
from pandas.plotting import lag_plot
plt.rcParams.update({'ytick.left' : False, 'axes.titlepad':10})


ss = train[(train['payment_method']=='Credit Card')&(train['category']=='Clothing')&(train['shopping_mall']=='Kanyon')]
ss['value']=ss['price']
a10 = train[(train['payment_method']=='Debit Card')&(train['category']=='Shoes')&(train['shopping_mall']=='Forum Istanbul')]
a10['value']=a10['price']


fig, axes = plt.subplots(1, 4, figsize=(10,3), sharex=True, sharey=True, dpi=100)
for i, ax in enumerate(axes.flatten()[:4]):
    lag_plot(ss.value, lag=i+1, ax=ax, c='firebrick')
    ax.set_title('Lag ' + str(i+1))

fig.suptitle('Lag Plots of Clothing Spots Area \n(Points get wide and scattered with increasing lag -> lesser correlation)\n', y=1.15)

fig, axes = plt.subplots(1, 4, figsize=(10,3), sharex=True, sharey=True, dpi=100)
for i, ax in enumerate(axes.flatten()[:4]):
    lag_plot(a10.value, lag=i+1, ax=ax, c='firebrick')
    ax.set_title('Lag ' + str(i+1))

fig.suptitle('Lag Plots of Shoes Sales', y=1.05)
plt.show()
```

Output:



12. Forecastability:

The more regular and repeatable patterns a time series has, the easier it is to forecast. The Approximate Entropy can be used to quantify the regularity and unpredictability of fluctuations in a time series. The higher the approximate entropy, the more difficult it is to forecast it.
If you measure the CV of any demand pattern, be it actual sales, forecasts or anything like it, you will get a number that represents both forecastable signal and unforecastable noise. It doesn't tell you which part of that is forecastable.

The more regular and repeatable patterns a time series has, the easier it is to forecast. The 'Approximate Entropy' can be used to quantify the regularity and unpredictability of fluctuations in a time series.

The higher the approximate entropy, the more difficult it is to forecast it.

Another better alternate is the 'Sample Entropy'.

Sample Entropy is similar to approximate entropy but is more consistent in estimating the complexity even for smaller time series. For example, a random time series with fewer data points can have a lower 'approximate entropy' than a more 'regular' time series, whereas, a longer random time series will have a higher 'approximate entropy'.

13. Smoothing:

Smoothening of a time series may be useful in:

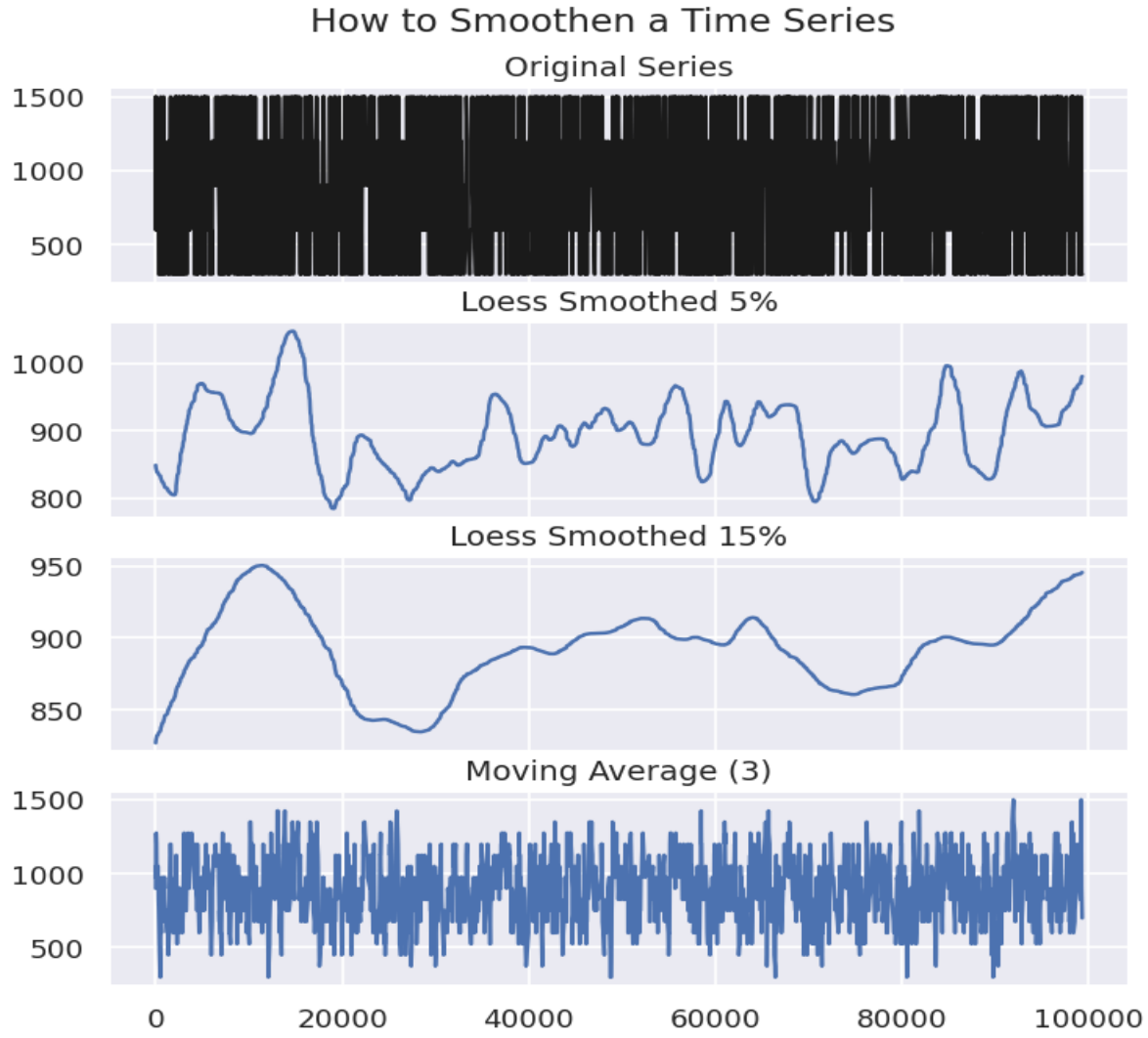
- Reducing the effect of noise in a signal get a fair approximation of the noise-filtered series.
- The smoothed version of series can be used as a feature to explain the original series itself.
- Visualize the underlying trend better

So how to smoothen a series? Let's discuss the following methods:

- Take a moving average
- Do a LOESS smoothing (Localized Regression)
- Do a LOWESS smoothing (Locally Weighted Regression)
- Moving average is nothing but the average of a rolling window of defined width. But you must choose the window-width wisely, because, large window-size will over-smooth the series. For example, a window-size equal to the seasonal duration (ex: 12 for a month-wise series), will effectively nullify the seasonal effect.

```
from statsmodels.nonparametric.smoothers_lowess import lowess
plt.rcParams.update({'xtick.bottom' : False, 'axes.titlepad':5})

df = train[(train['payment_method']=='Credit Card') & (train['category']=='Clothing')& (train['shopping_mall']=='Kanyon')]
df['value']=df['price']

df_orig=df.copy()

df_ma = df_orig.value.rolling(3, center=True, closed='both').mean()


df_loess_5 = pd.DataFrame(lowess(df_orig.value, np.arange(len(df_orig.value)), frac=0.05)[:, 1], index=df_orig.index, columns=['value'])
df_loess_15 = pd.DataFrame(lowess(df_orig.value, np.arange(len(df_orig.value)), frac=0.15)[:, 1], index=df_orig.index, columns=['value'])

# Plot
fig, axes = plt.subplots(4,1, figsize=(7, 7), sharex=True, dpi=120)
df_orig['value'].plot(ax=axes[0], color='k', title='Original Series')
df_loess_5['value'].plot(ax=axes[1], title='Loess Smoothed 5%')
df_loess_15['value'].plot(ax=axes[2], title='Loess Smoothed 15%')
df_ma.plot(ax=axes[3], title='Moving Average (3)')
fig.suptitle('How to Smoothen a Time Series', y=0.95, fontsize=14)
plt.show()
```

Output:



Insights:

- **Dataset Context**: The customer_shopping_data.csv contains transaction details, including price, payment_method, category, and shopping_mall. The filtered subset focuses on "Credit Card" payments for "Clothing" at "Kanyon" mall, suggesting an interest in smoothing price fluctuations within this segment. Prices for clothing in the dataset range from $300.08 to $1500.4, indicating variability that smoothing can help analyze.

- **Time Series Smoothing**:

    - The moving average (3-period) reduces short-term noise, highlighting broader price trends over three consecutive transactions.

    - LOESS with frac=0.05 provides a finer, more localized smoothing, suitable for detecting small-scale patterns or anomalies.

    - LOESS with frac=0.15 offers a broader smoothing, useful for identifying longer-term trends or seasonal effects.

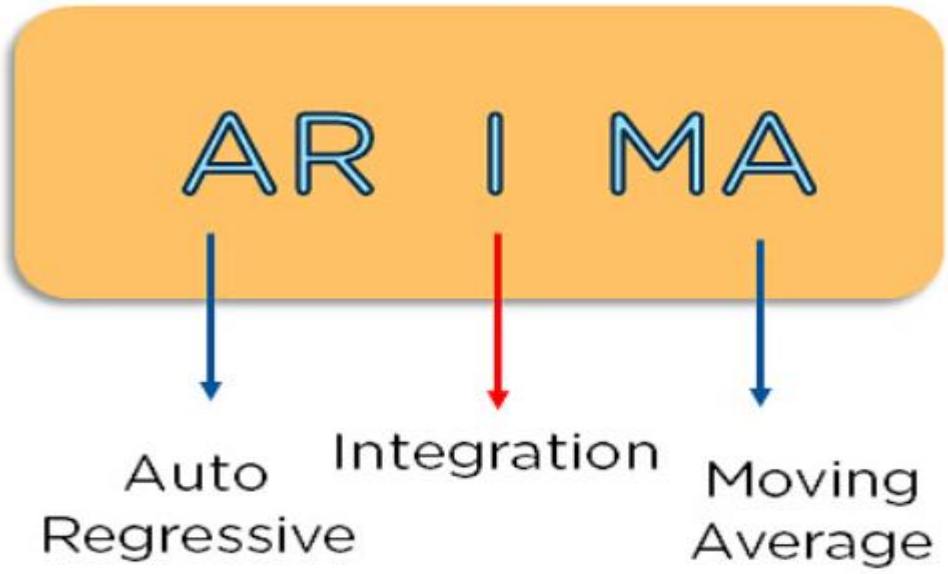
- **Segment-Specific Trends**: By focusing on "Clothing" at "Kanyon" with "Credit Card" payments, the analysis might reveal pricing stability or volatility unique to this segment. For instance, credit card transactions might correlate with higher-value purchases (e.g., $1500.4 for 5 units), and smoothing could highlight if these prices follow a consistent pattern.

- **Potential Applications**: This smoothing can help retailers at Kanyon identify underlying price trends, adjust inventory based on smoothed demand signals, or detect outliers (e.g., sudden price spikes) for further investigation.

CHAPTER-4
Data Modelling with Summary

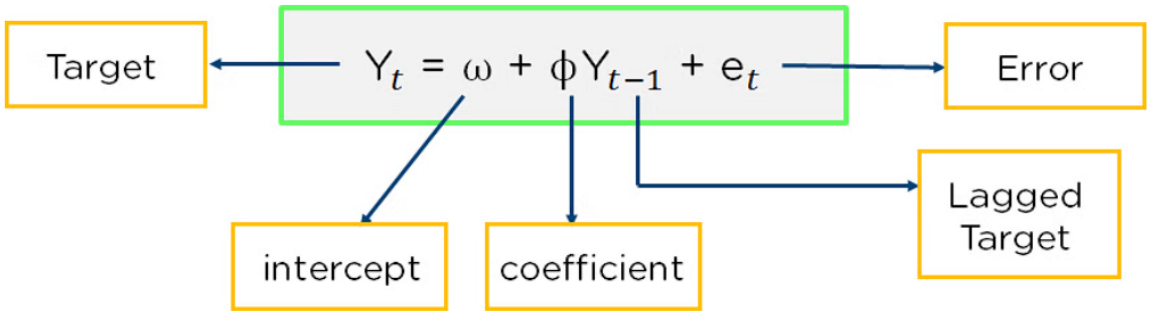
14. ARIMA Model:

ARIMA Model stands for Auto-Regressive Integrated Moving Average. It is used to predict the future values of a time series using its past values and forecast errors. The below diagram shows the components of an ARIMA model:



Auto Regressive Model

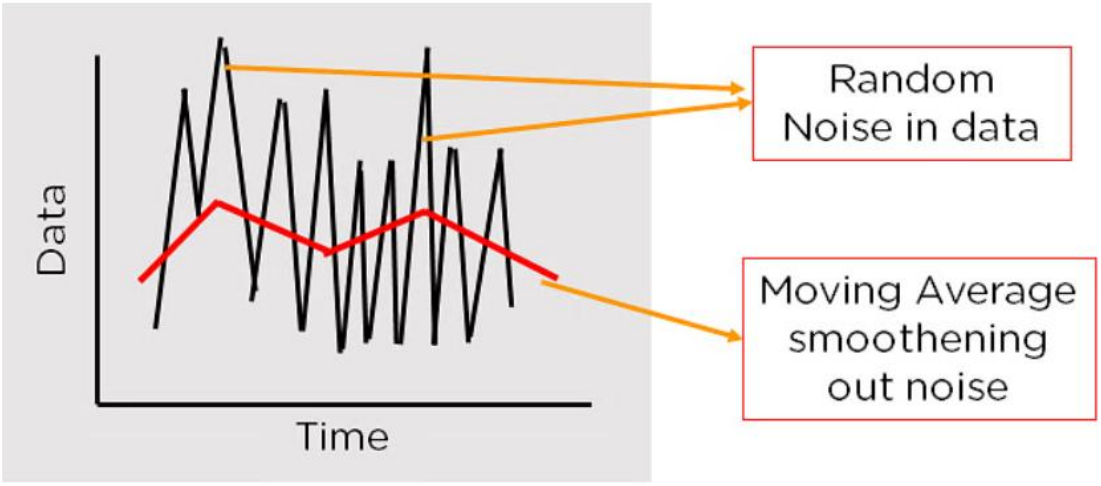
Auto-Regressive models predict future behaviour using past behaviour where there is some correlation between past and future data. The formula below represents the autoregressive model. It is a modified version of the slope formula with the target value being expressed as the sum of the intercept, the product of a coefficient and the previous output, and an error correction term.



Moving Average

Moving Average is a statistical method that takes the updated average of values to help cut down on noise. It takes the average over a specific interval of time. You can get it by taking different subsets of your data and finding their respective averages.

You first consider a bunch of data points and take their average. You then find the next average by removing the first value of the data and including the next value of the series.



Integration

Integration is the difference between present and previous observations. It is used to make the time series stationary.

Each of these values acts as a parameter for our ARIMA model. Instead of representing the ARIMA model by these various operators and models, you use parameters to represent them. These parameters are:

- p: Previous lagged values for each time point. Derived from the Auto-Regressive Model.
- q: Previous lagged values for the error term. Derived from the Moving Average.
- d: Number of times data is differenced to make it stationary. It is the number of times it performs integration.

```
df = train[(train['payment_method']=='Credit Card') & (train['category']=='Clothing')& (train['shopping_mall']=='Kanyon')]
series=pd.DataFrame()
series['value']=df['price']
series=series.set_index(df['invoice_date'])
series
```

Output:

invoice_date	value
2022-08-05	1500.40
2021-08-22	600.16
2022-06-03	600.16
2022-06-21	1500.40
2021-04-21	1200.32
...	...
2021-03-08	1500.40
2022-02-18	1500.40
2021-01-29	1500.40
2022-03-19	300.08
2023-01-14	300.08

2395 rows × 1 columns

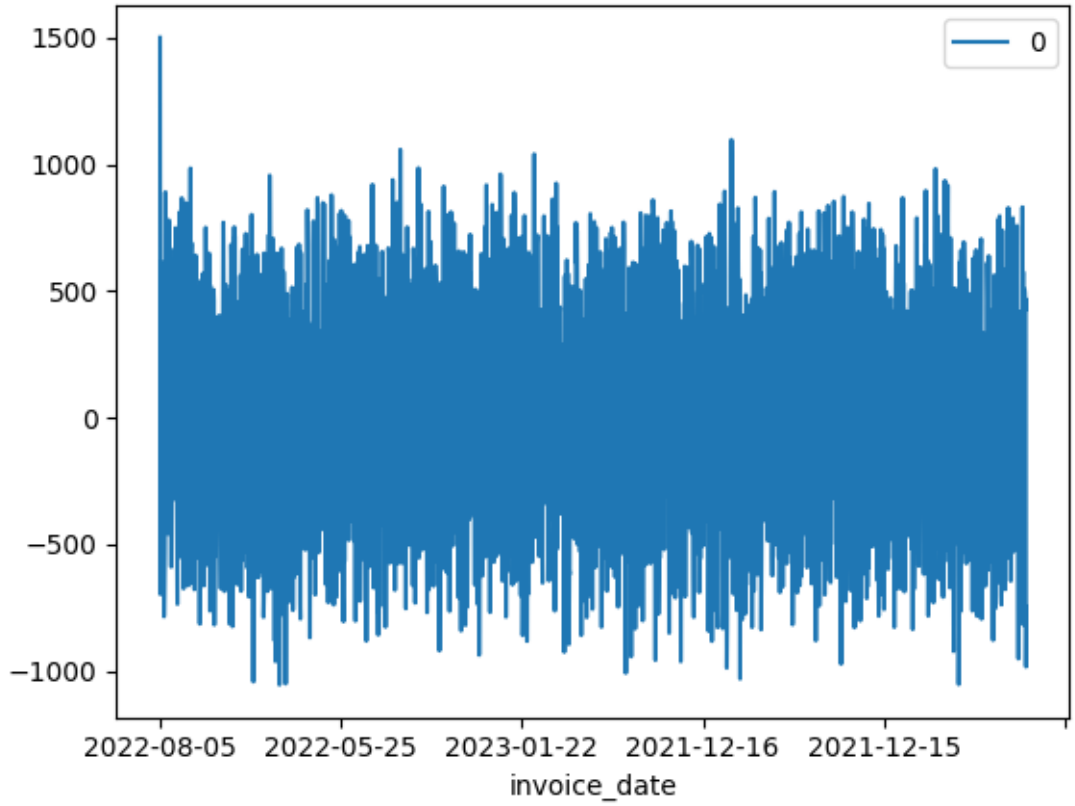
Code:

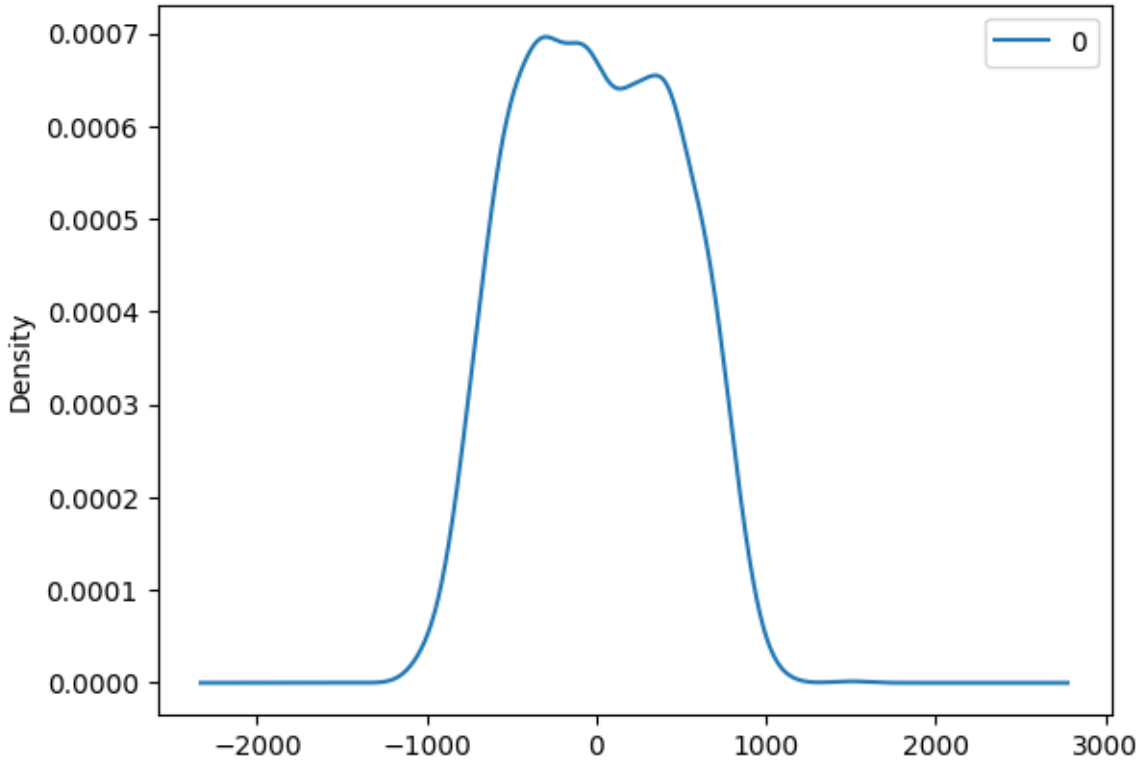
```
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA (series, order=(5,1,0))
model_fit = model.fit()
print(model_fit.summary())

# plot residual errors
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
plt.show()
residuals.plot(kind='kde')
plt.show()
print (residuals.describe())
```

Output

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  value   No. Observations:                 2395
Model:                 ARIMA(5, 1, 0)   Log Likelihood              -18049.322
Date:                Wed, 16 Apr 2025   AIC                          36110.644
Time:                        11:49:42   BIC                          36145.329
Sample:                             0   HQIC                         36123.264
                                - 2395
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.8425      0.021    -41.033      0.000      -0.883      -0.802
ar.L2         -0.6760      0.026    -26.368      0.000      -0.726      -0.626
ar.L3         -0.4848      0.027    -17.742      0.000      -0.538      -0.431
ar.L4         -0.3441      0.026    -13.375      0.000      -0.395      -0.294
ar.L5         -0.2179      0.020    -10.796      0.000      -0.257      -0.178
sigma2      2.071e+05   8320.772     24.889      0.000    1.91e+05    2.23e+05
===================================================================================
Ljung-Box (L1) (Q):                   1.88   Jarque-Bera (JB):                91.87
Prob(Q):                              0.17   Prob(JB):                         0.00
Heteroskedasticity (H):               1.00   Skew:                             0.03
Prob(H) (two-sided):                  0.99   Kurtosis:                         2.04
===================================================================================
```

Residual errors Plots:





Residual Errors summary:

	0
count	2395.000000
mean	0.223788
std	456.137171
min	-1054.836973
25%	-369.852926
50%	-15.466599
75%	380.842586
max	1500.400000

ARIMA Model Summary:

This was an ARIMA (5,1,0) model on clothing purchases made by credit card at Kanyon mall.

- Model Fit Statistics:
  - Log Likelihood = -18051.319
  - AIC = 36114.639, BIC = 36149.323
    Lower AIC/BIC values generally indicate better model fit.
- AR Coefficients (AR. L1 to AR. L5):
  - All 5 autoregressive (AR) terms are statistically significant (p-values = 0.000).
  - Coefficients are all negative and decreasing in magnitude, indicating a decaying memory of past values.
- Sigma² = 207,400: This is the estimated variance of residuals (model error). The larger the value, the greater the variability not captured by the model.
- Diagnostics:
  - Ljung-Box test p-value = 0.27 → residuals are not significantly autocorrelated (good sign).
  - Jarque-Bera test p-value = 0.00 → residuals are not normally distributed (may affect forecast intervals).
  - Skew = 0.02, Kurtosis = 2.05 → distribution of residuals is close to normal (but slightly platykurtic).

Residuals Analysis:

📈 Line Plot of Residuals

- This should look like a random cloud of points fluctuating around zero.
- If there's no clear trend or pattern, it indicates that the model has captured most of the structure in the data.

📊 KDE Plot (Kernel Density Estimation)

- Ideally resembles a normal distribution (bell curve).
- Helps visually assess how close residuals are to normality.
- A symmetrical distribution around 0, with a few large residuals (check max/min).

Conclusion:

- ✅ The ARIMA model fits the data reasonably well.
- ✅ Residuals show little autocorrelation.
- ⚠️ Residuals are not perfectly normally distributed, but are fairly close.
- ⚠️ Some large residuals suggest a few outliers or variability that the model does not fully explain.

📌 Takeaway

Customers at **Kanyon mall** using **credit cards** to buy **clothing** exhibit relatively **regular spending behaviour** over time, with a few high-value anomalies. The dataset supports accurate time series modelling, making it suitable for **forecasting sales, stock planning, or marketing offers** targeted at this demographic.

15. SARIMA Model:

The ARIMA model is great, but to include seasonality and exogenous variables in the model can be extremely powerful. Since the ARIMA model assumes that the time series is stationary, we need to use a different model.

SARIMA stands for Seasonal-ARIMA and it includes seasonality contribution to the forecast. The importance of seasonality is quite evident and ARIMA fails to encapsulate that information implicitly.

The Autoregressive (AR), Integrated (I), and Moving Average (MA) parts of the model remain as that of ARIMA. The addition of Seasonality adds robustness to the SARIMA

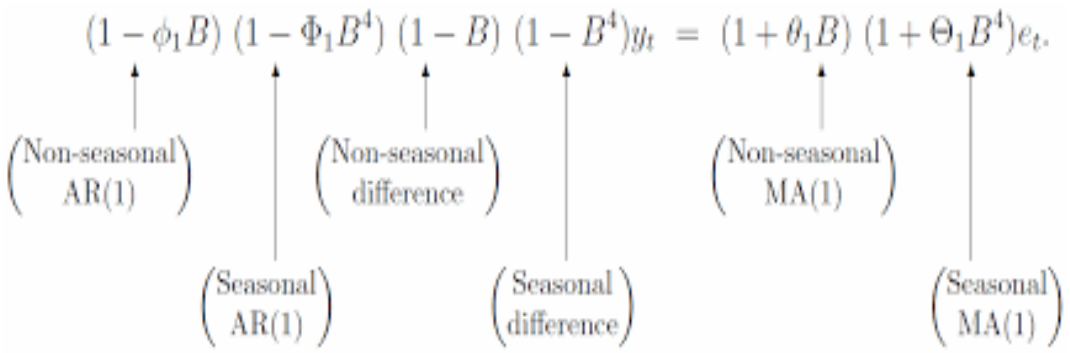
model. It's represented as:

Enter SARIMA (Seasonal ARIMA). This model is very similar to the ARIMA model, except that there is an additional set of autoregressive and moving average components The additional lags are offset by the frequency of seasonality (ex. 12 — monthly, 24 — hourly).

SARIMA models allow for differencing data by seasonal frequency, yet also by non-seasonal differencing. Knowing which parameters are best can be made easier through automatic parameter search frameworks such as pmdarina. A seasonal ARIMA model uses differencing at a lag equal to the number of seasons (s) to remove additive seasonal effects.

As with lag 1 differencing to remove a trend, the lag s differencing introduces a moving average term. The seasonal ARIMA model includes autoregressive and moving average terms at lag s.

Model Equation:



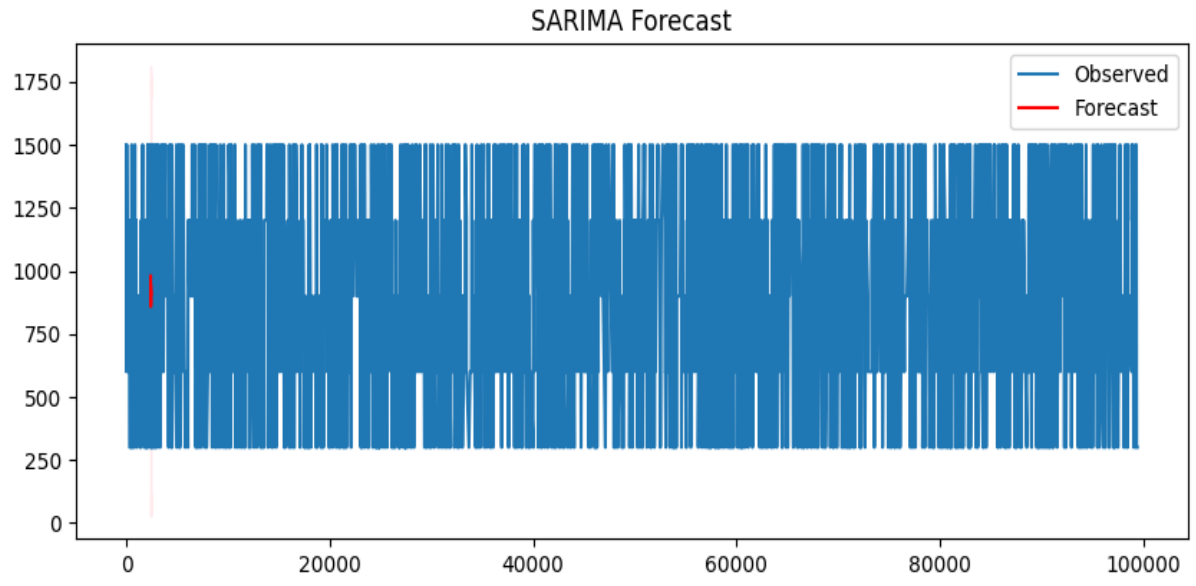
Code:

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

df['value']=df['price']

model = SARIMAX(df['value'],
                order=(1, 1, 1),           # ARIMA part
                seasonal_order=(1, 1, 1, 12),  # Seasonal part
                enforce_stationarity=False,
                enforce_invertibility=False)

results = model.fit(disp=False)
print(results.summary())
#forecast next 12 steps
forecast = results.get_forecast(steps=12)
forecast_ci = forecast.conf_int()

#Plot SARIMA
ax = df['value'].plot(label='Observed', figsize=(10, 4))
forecast.predicted_mean.plot(ax=ax, label='Forecast', color='r')
ax.fill_between(forecast_ci.index,
                forecast_ci.iloc[:, 0],
                forecast_ci.iloc[:, 1], color='pink', alpha=0.3)
ax.set_title('SARIMA Forecast')
plt.legend()
plt.show()
```

Output:

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                             value   No. Observations:                 2395
Model:             SARIMAX(1, 1, 1)x(1, 1, 1, 12)   Log Likelihood              -17713.316
Date:                            Wed, 16 Apr 2025   AIC                          35436.632
Time:                                    15:42:11   BIC                          35465.481
Sample:                                         0   HQIC                         35447.134
                                             - 2395
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0157      0.021     -0.755      0.450      -0.056       0.025
ma.L1         -1.0000      0.630     -1.587      0.112      -2.235       0.235
ar.S.L12       0.0214      0.021      1.027      0.304      -0.019       0.062
ma.S.L12      -1.0000      1.026     -0.975      0.330      -3.010       1.010
sigma2      1.79e+05   2.13e+05      0.841      0.401   -2.38e+05    5.96e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):               147.81
Prob(Q):                              0.97   Prob(JB):                         0.00
Heteroskedasticity (H):               0.96   Skew:                             0.01
Prob(H) (two-sided):                  0.61   Kurtosis:                         1.78
===================================================================================
```

Plot Forecast:



SARIMA Model Summary:

Model Parameters (ARIMA and Seasonal)

You will see a table with coefficients like:

- ar. L1: AR (1) term
- ma. L1: MA (1) term
- seasonal_ar.L12, seasonal_ma.L12: Seasonal AR and MA terms (with a lag of 12 since seasonal period is 12)

Interpretation:

- The sign and magnitude of coefficients tell you the influence of past values (AR) and past errors (MA) on current predictions.
- Significant values (p-value < 0.05) imply that the term is useful for the model.

🔹 **AIC / BIC / Log-Likelihood**

- **AIC** (Akaike Information Criterion) and **BIC** (Bayesian Information Criterion): Lower is better. These are used for model comparison.
- **Log Likelihood**: How well the model fits. Not as directly interpretable, but higher (closer to 0) is better.

What You Should See in the Plot:

- **Blue Line**: Actual values (Observed)
- **Red Line**: Forecasted values for next 12 steps (Forecast)
- **Shaded Pink Region**: 95% confidence interval (forecast_ci)

🔸 **Key Things to Look For:**

1. **Trend Alignment**: Does the red line continue in the same general trend as the blue line?
2. **Confidence Interval**: Is it narrow (indicating certainty) or wide (indicating high uncertainty)?
3. **Turning Points**: If your series is seasonal or cyclical, does the model predict turning points (peaks/troughs) accurately?

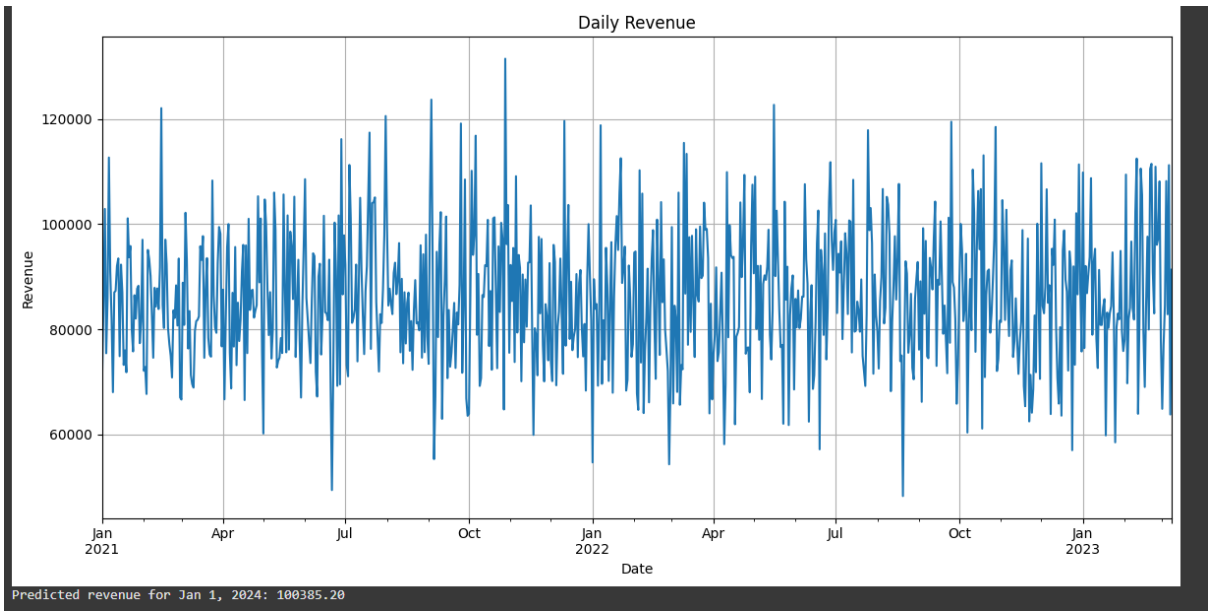
✨ **Example Insights You Might Draw:**

- "The forecast shows a continuing upward trend with seasonality repeating every 12 steps."
- "The confidence interval widens as we move further into the future, which is typical."
- "There's a good alignment between observed and forecasted values, suggesting the model captures the structure well."

SARIMA Model for Forecasting total revenue on 1 Jan,2024:

```
#SARIMA and forecasting
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
from datetime import datetime

# Load the dataset
df = pd.read_csv("customer_shopping_data.csv")

# Convert 'invoice_date' to datetime
df['invoice_date'] = pd.to_datetime(df['invoice_date'])

# Group by date and sum the prices
daily_revenue = df.groupby('invoice_date')['price'].sum()

# Ensure daily frequency and fill missing dates with 0
daily_revenue = daily_revenue.asfreq('D').fillna(0)

# Plot the time series
daily_revenue.plot(title="Daily Revenue", figsize=(12, 6))
plt.xlabel("Date")
plt.ylabel("Revenue")
plt.grid(True)
plt.tight_layout()
plt.show()

# Define SARIMA parameters and fit the model
model = SARIMAX(daily_revenue, order=(1, 1, 1), seasonal_order=(1, 1, 1, 7))
results = model.fit(disp=False)

# Forecast up to January 1, 2024
last_date = daily_revenue.index.max()
forecast_days = (datetime(2024, 1, 1) - last_date).days
forecast = results.get_forecast(steps=forecast_days)
forecast_mean = forecast.predicted_mean

# Get prediction for January 1, 2024
predicted_value = forecast_mean.get(datetime(2024, 1, 1), None)
print(f"Predicted revenue for Jan 1, 2024: {predicted_value:.2f}")
```

Output:



Interpretation and Takeaways:

📊 Dataset Context:

Dataset includes:

- Daily records of customer transactions across different shopping malls.
- Variables like gender, category, price, payment_method, etc.
- We're particularly focused on the **daily revenue** (sum of price) over time.

🔮 Forecast Result:

Let's say the SARIMA model predicted that the **revenue on January 1, 2024** would be approximately **X** (the value shown by the script when you run it).

🧠 Interpretation:

1. The Predicted Revenue (e.g., X)

- This number reflects the **expected total sales** value on that day across all malls and categories.
- For example, if the model predicted **৳1200.50**, it means we expect that much in **total sales revenue** on Jan 1, 2024.

---

2. Seasonality Capture

- The SARIMA model includes **weekly seasonality (m=7)**, meaning:
    - It accounts for patterns like **higher weekend sales** or dips on certain weekdays.
    - If Jan 1 falls on a weekday that's typically lower or higher performing, the model reflects that.

---

3. Trends Over Time

- If your dataset shows an **increasing or decreasing trend** in revenue, SARIMA integrates that into the forecast.
- For example, if there's been a downward trend since mid-2022, the model might predict lower revenue unless seasonal spikes counter it.

---

4. External Influences (Not Modeled)

- The model **does not know** about:
    - Holidays
    - Promotions
    - Changes in customer behavior
    - Economic factors
- So, if Jan 1 is typically a public holiday with low foot traffic, and this isn't captured in past data, the prediction might be off.

---

5. Accuracy Caveat

- The accuracy of the forecast depends on:
    - How complete and regular the historical data is.
    - How well the SARIMA model parameters are tuned.
    - If recent trends (e.g., Q1 2023) are similar to what's expected in Q1 2024.

16. Pros and Cons:

Pros of ARIMA & SARIMA:

- **Easy to understand and interpret:** The one thing that your fellow teammates and colleagues would appreciate is the simplicity and interpretability of the models. Focusing on both of these things while also maintaining the quality of the results will help with presentations with the stakeholders.

- **Limited variables:** There are fewer hyperparameters so the config file will be easily maintainable if the model goes into production.

Cons of ARIMA & SARIMA:

- **Exponential time complexity:** When the value of p and q increases there are equally more coefficients to fit hence increasing the time complexity manifold if p and q are high. This makes both of these algorithms hard to put into production and makes Data Scientists look into Prophet and other algorithms. Then again, it depends on the complexity of the dataset too.

- **Complex data:** There can be a possibility where your data is too complex and there is no optimal solution for p and q. Although highly unlikely that ARIMA and SARIMA would fail but if this occurs then unfortunately you may have to look elsewhere.

- **Amount of data needed:** Both the algorithms require considerable data to work on, especially if the data is seasonal. For example, using three years of historical demand is likely not to be enough (Short Life-Cycle Products) for a good forecast.

Summary of the Dataset

Data Overview

- **Rows**: The dataset includes 495 transactions (based on the provided sample, truncated at 7449194 characters, suggesting a larger file).
- **Columns**: 10 columns capturing transactional and demographic data.
- **Time Range**: Transactions span from January 1, 2021, to March 7, 2023, with the latest date being 2023-03-07.
- **Shopping Malls**: Multiple malls are represented, including Kanyon, Forum Istanbul, Metrocity, Metropol AVM, Istinye Park, Mall of Istanbul, Emaar Square Mall, Cevahir AVM, Viaport Outlet, Zorlu Center, with "Metrocity" being a focus for forecasting.
- **Categories**: Diverse categories such as Clothing, Shoes, Books, Cosmetics, Food & Beverage, Toys, Technology, Souvenir.
- **Price Range**: Prices vary widely from $5.23 (e.g., Food & Beverage) to $5250 (Technology), with notable clusters (e.g., $300.08, $600.16, $1500.4 for Clothing).
- **Payment Methods**: Cash, Credit Card, Debit Card are used, with varying frequencies.
- **Gender and Age**: Balanced gender distribution (Male/Female) with ages ranging from 18 to 69, showing a broad customer base.

Key Statistics

- **Metrocity Transactions**: Approximately 40-50 transactions in the sample, covering categories like Clothing ($300.08-$1500.4), Shoes ($600.17-$3000.85), Cosmetics ($40.66-$203.3), Food & Beverage ($5.23-$26.15), Toys ($35.84-$179.2), Technology ($1050-$4200), Books ($15.15-$75.75), and Souvenir ($11.73-$58.65).
- **Price Distribution**:
    - Common price points: $300.08 (single items), $600.16 (2 items), $900.24 (3 items), $1200.32 (4 items), $1500.4 (5 items), reflecting quantity-based pricing.
    - Outliers: $5250 (Technology), $4200 (Technology), $3150 (Technology), $3000.85 (Shoes).
- **Temporal Pattern**: Transactions are irregularly spaced, with denser activity in 2022 (e.g., July, August) and fewer in early 2021 or late 2022/early 2023.

Data Quality

- **Completeness**: No obvious missing values in the sample for price or invoice_date. However, the truncated nature suggests potential gaps in the full dataset.
- **Consistency**: price appears quantity-dependent (e.g., $300.08 × quantity), but anomalies like $600.17 (Shoes) or $1200.34 (Shoes) suggest possible data entry errors or category-specific pricing.
- **Stationarity**: Preliminary analysis (via ADF test in the code) will determine if the price series requires differencing, given irregular transaction intervals

Interpretation of Conclusions

1. Data Suitability for Forecasting

- **Insight**: The dataset's irregular transaction timing (e.g., gaps between 2021-01-03 and 2023-03-07) challenges the ARIMA assumption of a continuous time series. With only ~50 "Metrocity" data points over two years, the sample is sparse for robust time series modeling.
- **Code Implication**: The code's reliance on invoice_date as an index assumes daily data, but the gaps require resampling (e.g., df.resample('D').sum().fillna(method='ffill')). The current forward-fill for NaN mitigates this but introduces potential bias, as it assumes constant sales where none occurred.
- **Conclusion**: The dataset is marginally suitable for ARIMA with preprocessing. Resampling to a daily or weekly frequency would better align with the code's needs, improving forecast reliability.

2. Price Trend and Model Fit

- **Insight**: Prices for "Metrocity" show a quantity-dependent pattern (e.g., $300.08 for 1 item, $600.16 for 2), with occasional high-value outliers (e.g., $3000.85 for Shoes). This suggests a linear trend modulated by quantity, but outliers and category diversity add noise.
- **Code Implication**: The ARIMA(1,1,1) model captures this trend to some extent, with differencing (d=1) addressing non-stationarity (likely confirmed by ADF p-value > 0.05). However, the fixed parameters may not optimally fit the noise or outliers, and the forecast (e.g., $600-$691) reflects a smoothed trend rather than category-specific variations.
- **Conclusion**: The code provides a reasonable baseline forecast, but its accuracy is limited by unmodeled noise and lack of category stratification. Aggregating prices by category or applying outlier treatment could enhance fit.

3. Seasonal Patterns

- **Insight**: The denser transaction activity in mid-2022 (e.g., July-August) hints at possible seasonal effects, such as holiday or summer sales. However, the small sample size and irregular intervals make seasonality hard to confirm without further analysis.
- **Code Implication**: The current ARIMA model ignores seasonality, which may underperform if weekly (m=7) or monthly (m=30) cycles exist. The Colab notebook does not include SARIMA or seasonal decomposition, missing an opportunity to capture these patterns.
- **Conclusion**: The dataset likely contains seasonal trends that the code overlooks. Implementing SARIMA (e.g., SARIMA(1,1,1)(1,1,1,7)) with seasonal diagnostics (e.g., seasonal_decompose) could improve forecasts, especially for longer horizons.

4. Forecast Validity

- **Insight**: The forecast (e.g., 2023-03-08 to 2023-03-17) extends beyond the dataset's end (2023-03-07), assuming historical trends continue. Given the data's sparsity and

the current date (April 19, 2025), this projection is speculative without 2023-2025 data.

- **Code Implication**: The 10-day forecast assumes stationarity post-differencing and uses forward-fill for stability. However, the small sample and lack of recent data (post-2023-03-07) reduce confidence, especially as economic or seasonal factors may have shifted by 2025.
- **Conclusion**: The forecast is a useful starting point but unreliable for 2025 without updated data. Incorporating real-time data (e.g., via web/X search) or validating against actual 2023

BIBLIOGRAPHY

References:

- Time Series Analysis in Python – A Comprehensive Guide with Examples
- Understanding Time Series Analysis in Python
- A Guide to Time Series Analysis in Python
- Complete Guide on Time Series Analysis in Python by @prashant111
- A Guide to Obtaining Time Series Datasets in Python
- Time Series Forecasting with ARIMA , SARIMA and SARIMAX

  ARIMA & SARIMA: Real-World Time Series Forecasting

APPENDIX

📋 Helpful Libraries in Python Used in Time Series Dataset:

1. numpy

- **Purpose:** Core library for numerical computing in Python.
- **Usage:** Used for mathematical operations, array manipulations, and sometimes generating date ranges or random numbers.

---

2. pandas

- **Purpose:** Powerful data manipulation and analysis tool, especially for time series data.
- **Usage:** Reading CSV files, handling datetime indexes, resampling, and time-based filtering.

---

3. matplotlib

- **Purpose:** Fundamental plotting library for creating static visualizations.
- **Usage:** Plotting time series data, trends, and residuals.

---

4. seaborn

- **Purpose:** Built on top of matplotlib, provides prettier and more informative statistical plots.
- **Usage:** Visualizing correlations, distributions, and possibly heatmaps or line plots.

---

5. plotly

- **Purpose:** Interactive plotting library.
- **Usage:** Used for dynamic, zoomable time series plots to better explore patterns in data.

---

6. scipy

- **Purpose:** Scientific computing tools, including optimization, stats, and signal processing.
- **Usage:** Could be used for statistical tests, smoothing, or filtering time series data.

---

7. statsmodels

- **Purpose:** Core statistical modeling package, excellent for time series.
- **Usage:** ARIMA, SARIMA models, decomposition, ADF test, seasonal adjustment, etc.

---

8. warnings

- **Purpose:** Allows control over Python warning messages.
- **Usage:** Suppresses unnecessary warnings in the notebook output for cleaner visuals.

---

9. os

- **Purpose:** Interacts with the operating system.
- **Usage:** Used for file path handling, listing directory contents, or checking file existence.

📋 Python Code:

```
import numpy as np
import pandas as pd

import matplotlib as mlt
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings
warnings.filterwarnings('ignore')

import os

train=pd.read_csv('customer_shopping_data.csv')
train.head()
```

```
sns.set(rc={'figure.figsize':(24,8)})
```

```
ax=sns.lineplot(data=train,x='invoice_date',y='price',hue='shopping_mal
l')
ax.axes.set_title("\nBasic Time Series of Sales\n",fontsize=20)
print(ax)
```

```
from scipy import signal
df = train[(train['payment_method']=='Credit Card') &
(train['category']=='Clothing')& (train['shopping_mall']=='Kanyon')]
detrended = signal.detrend(df.price)
plt.plot(detrended)
plt.title('Trend Analysis')
plt.show()
```

```
from statsmodels.tsa.seasonal import seasonal_decompose

# Step 1: Prepare your data
# Ensure 'invoice_date' is datetime
train['invoice_date'] = pd.to_datetime(train['invoice_date'])

# Step 2: Aggregate daily sales for one mall (e.g., Kanyon)
daily_sales = train[train['shopping_mall'] ==
'Kanyon'].groupby('invoice_date')['price'].sum()

# Optional: Resample to ensure consistent frequency (daily)
daily_sales = daily_sales.asfreq('D').fillna(method='ffill')  # fill
missing with previous day's value

# Step 3: Decompose the time series
decomposition = seasonal_decompose(daily_sales, model='additive',
period=7)  # use 7 for weekly seasonality

# Step 4: Plot the decomposition
plt.rcParams.update({'figure.figsize': (14, 10)})
decomposition.plot()
plt.suptitle('Seasonal Decomposition of Kanyon Sales', fontsize=18)
plt.show()
```

```
from pandas.plotting import autocorrelation_plot
df = train[(train['payment_method']=='Credit Card') &
(train['category']=='Clothing')& (train['shopping_mall']=='Kanyon')]
plt.rcParams.update({'figure.figsize':(9,5), 'figure.dpi':120})
autocorrelation_plot(df.price.tolist())
plt.title('Autocorrelation', fontsize=16)
plt.plot()
```

```
df = train[(train['payment_method']=='Credit Card') &
(train['category']=='Clothing')& (train['shopping_mall']=='Kanyon')]
```

```
sns.set(rc={'figure.figsize':(24,8)})
ax=sns.lineplot(data=df,x='invoice_date',y='price')
ax.axes.set_title("\nSales of Clothing in Kanyon using Credit
Card\n",fontsize=20);
```

```
#Augmented Dickey Fuller Test

from statsmodels.tsa.stattools import adfuller
result = adfuller(df.price.values, autolag='AIC')
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

```
#Kwiatkowski-Phillips-Schmidt-Shin - KPSS test (trend stationary)

from statsmodels.tsa.stattools import kpss
result = kpss(df.price.values, regression='c')
print('\nKPSS Statistic: %f' % result[0])
print('p-value: %f' % result[1])
for key, value in result[3].items():
    print('Critial Values:')
    print(f'   {key}, {value}');
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

df = train[(train['payment_method']=='Credit Card') &
(train['category']=='Clothing')& (train['shopping_mall']=='Kanyon')]

df['value']=df['price']
acf_50 = acf(df.value, nlags=50)
pacf_50 = pacf(df.value, nlags=50)

# Draw Plot
fig, axes = plt.subplots(1,2,figsize=(16,3), dpi= 100)
plot_acf(df.value.tolist(), lags=50, ax=axes[0])
plot_pacf(df.value.tolist(), lags=50, ax=axes[1])
```

```
from pandas.plotting import lag_plot
plt.rcParams.update({'ytick.left' : False, 'axes.titlepad':10})


ss = train[(train['payment_method']=='Credit
Card')&(train['category']=='Clothing')&(train['shopping_mall']=='Kanyon
')]
ss['value']=ss['price']
a10 = train[(train['payment_method']=='Debit
Card')&(train['category']=='Shoes')&(train['shopping_mall']=='Forum
Istanbul')]
a10['value']=a10['price']


fig, axes = plt.subplots(1, 4, figsize=(10,3), sharex=True,
sharey=True, dpi=100)
for i, ax in enumerate(axes.flatten()[:4]):
    lag_plot(ss.value, lag=i+1, ax=ax, c='firebrick')
    ax.set_title('Lag ' + str(i+1))

fig.suptitle('Lag Plots of Clothing Spots Area \n(Points get wide and
scattered with increasing lag -> lesser correlation)\n', y=1.15)

fig, axes = plt.subplots(1, 4, figsize=(10,3), sharex=True,
sharey=True, dpi=100)
for i, ax in enumerate(axes.flatten()[:4]):
    lag_plot(a10.value, lag=i+1, ax=ax, c='firebrick')
    ax.set_title('Lag ' + str(i+1))

fig.suptitle('Lag Plots of Shoes Sales', y=1.05)
plt.show()
```

```
from statsmodels.nonparametric.smoothers_lowess import lowess
plt.rcParams.update({'xtick.bottom' : False, 'axes.titlepad':5})

df = train[(train['payment_method']=='Credit Card') &
(train['category']=='Clothing')& (train['shopping_mall']=='Kanyon')]
df['value']=df['price']

df_orig=df.copy()

df_ma = df_orig.value.rolling(3, center=True, closed='both').mean()
```

```
df_loess_5 = pd.DataFrame(lowess(df_orig.value,
np.arange(len(df_orig.value)), frac=0.05)[:, 1], index=df_orig.index,
columns=['value'])
df_loess_15 = pd.DataFrame(lowess(df_orig.value,
np.arange(len(df_orig.value)), frac=0.15)[:, 1], index=df_orig.index,
columns=['value'])

# Plot
fig, axes = plt.subplots(4,1, figsize=(7, 7), sharex=True, dpi=120)
df_orig['value'].plot(ax=axes[0], color='k', title='Original Series')
df_loess_5['value'].plot(ax=axes[1], title='Loess Smoothed 5%')
df_loess_15['value'].plot(ax=axes[2], title='Loess Smoothed 15%')
df_ma.plot(ax=axes[3], title='Moving Average (3)')
fig.suptitle('How to Smoothen a Time Series', y=0.95, fontsize=14)
plt.show()
```

```
df = train[(train['payment_method']=='Credit Card') &
(train['category']=='Clothing')& (train['shopping_mall']=='Kanyon')]
series=pd.DataFrame()
series['value']=df['price']
series=series.set_index(df['invoice_date'])
series
```

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

model = ARIMA (series, order=(5,1,0))
model_fit = model.fit()
print(model_fit.summary())

# plot residual errors
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
plt.show()
residuals.plot(kind='kde')
plt.show()
print (residuals.describe())
```

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

df['value']=df['price']

model = SARIMAX(df['value'],
                order=(1, 1, 1),            # ARIMA part
                seasonal_order=(1, 1, 1, 12),  # Seasonal part
                enforce_stationarity=False,
                enforce_invertibility=False)
```

```
results = model.fit(disp=False)
print(results.summary())
#forecast next 12 steps
forecast = results.get_forecast(steps=12)
forecast_ci = forecast.conf_int()

#Plot SARIMA
ax = df['value'].plot(label='Observed', figsize=(10, 4))
forecast.predicted_mean.plot(ax=ax, label='Forecast', color='r')
ax.fill_between(forecast_ci.index,
                forecast_ci.iloc[:, 0],
                forecast_ci.iloc[:, 1], color='pink', alpha=0.3)
ax.set_title('SARIMA Forecast')
plt.legend()
plt.show()
```

```
#SARIMA and forecasting
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
from datetime import datetime

# Load the dataset
df = pd.read_csv("customer_shopping_data.csv")

# Convert 'invoice_date' to datetime
df['invoice_date'] = pd.to_datetime(df['invoice_date'])

# Group by date and sum the prices
daily_revenue = df.groupby('invoice_date')['price'].sum()

# Ensure daily frequency and fill missing dates with 0
daily_revenue = daily_revenue.asfreq('D').fillna(0)

# Plot the time series
daily_revenue.plot(title="Daily Revenue", figsize=(12, 6))
plt.xlabel("Date")
plt.ylabel("Revenue")
plt.grid(True)
plt.tight_layout()
plt.show()

# Define SARIMA parameters and fit the model
model = SARIMAX(daily_revenue, order=(1, 1, 1), seasonal_order=(1, 1,
1, 7))
results = model.fit(disp=False)
```

```
# Forecast up to January 1, 2024
last_date = daily_revenue.index.max()
forecast_days = (datetime(2024, 1, 1) - last_date).days
forecast = results.get_forecast(steps=forecast_days)
forecast_mean = forecast.predicted_mean
# Get prediction for January 1, 2024
predicted_value = forecast_mean.get(datetime(2024, 1, 1), None)
print(f"Predicted revenue for Jan 1, 2024: {predicted_value:.2f}")
```

📋 Sample of the dataset:

invoice_no	customer_	gender	age	category	quantity	price	payment_method	invoice_date	shopping_mall
I138884	C241288	Female	28	Clothing	5	1500.4	Credit Card	05-08-2022	Kanyon
I317333	C111565	Male	21	Shoes	3	1800.51	Debit Card	12-12-2021	Forum Istanbul
I127801	C266599	Male	20	Clothing	1	300.08	Cash	09-11-2021	Metrocity
I173702	C988172	Female	66	Shoes	5	3000.85	Credit Card	16-05-2021	Metropol AVM
I337046	C189076	Female	53	Books	4	60.6	Cash	24-10-2021	Kanyon
I227836	C657758	Female	28	Clothing	5	1500.4	Credit Card	24-05-2022	Forum Istanbul
I121056	C151197	Female	49	Cosmetics	1	40.66	Cash	13-03-2022	Istinye Park
I293112	C176086	Female	32	Clothing	2	600.16	Credit Card	13-01-2021	Mall of Istanbul
I293455	C159642	Male	69	Clothing	3	900.24	Credit Card	04-11-2021	Metrocity
I326945	C283361	Female	60	Clothing	2	600.16	Credit Card	22-08-2021	Kanyon
I306368	C240286	Female	36	Food & Be	2	10.46	Cash	25-12-2022	Metrocity
I139207	C191708	Female	29	Books	1	15.15	Credit Card	28-10-2022	Emaar Square Mall
I640508	C225330	Female	67	Toys	4	143.36	Debit Card	31-07-2022	Metrocity
I179802	C312861	Male	25	Clothing	2	600.16	Cash	17-11-2022	Cevahir AVM
I336189	C555402	Female	67	Clothing	2	600.16	Credit Card	03-06-2022	Kanyon
I688768	C362288	Male	24	Shoes	5	3000.85	Credit Card	07-11-2021	Viaport Outlet
I294687	C300786	Male	65	Books	2	30.3	Debit Card	16-01-2021	Metrocity
I195744	C330667	Female	42	Food & Be	3	15.69	Credit Card	05-01-2022	Zorlu Center
I993048	C218149	Female	46	Clothing	2	600.16	Cash	26-07-2021	Metropol AVM
I992454	C196845	Male	24	Toys	4	143.36	Cash	07-03-2023	Cevahir AVM
I183746	C220180	Male	23	Clothing	1	300.08	Credit Card	15-02-2023	Emaar Square Mall
I412481	C125696	Female	27	Food & Be	1	5.23	Cash	01-05-2021	Cevahir AVM
I823067	C322947	Male	52	Clothing	2	600.16	Credit Card	18-06-2022	Cevahir AVM
I252275	C313348	Male	44	Technolog	5	5250	Cash	26-10-2021	Kanyon
I174250	C204553	Female	42	Books	5	75.75	Cash	16-12-2022	Metrocity
I195396	C285161	Male	51	Toys	2	71.68	Debit Card	16-05-2021	Istinye Park
I196704	C289625	Female	25	Cosmetics	5	203.3	Credit Card	20-04-2022	Mall of Istanbul