**MARKET BASKET INSIGHTS**

****

**INTRODUCTION:**

Market basket analysis is a data-driven technique widely employed in the retail industry to uncover patterns of co-purchased products, aiding businesses in enhancing marketing strategies, optimizing inventory management, and boosting sales. By examining associations between items within customer transactions, it enables retailers to make informed decisions such as personalized product recommendations, targeted promotions, and efficient store layouts. Leveraging metrics like support, confidence, and lift, market basket analysis empowers businesses to better understand customer behaviour, ultimately leading to improved customer satisfaction and business profitability.

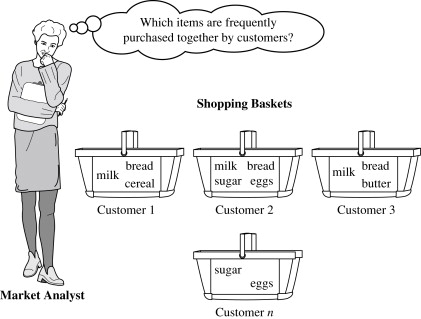
**DATA SOURCE:**

***Transaction Records***: Each transaction record represents a single purchase made by a customer. It includes information such as transaction date and time, customer ID (if applicable), and a list of items purchased during that transaction.

***Item Data***: This data source includes details about each individual item available for purchase. It may contain information such as item ID, name, category, price, and description.

***Customer Data (Optional):*** While not always necessary, customer data can be useful for segmenting and targeting specific customer groups. Customer data may include customer demographics, loyalty program information, and past purchase history.

***Store or Location Data (Optional):*** In the case of physical retail stores, information about the store's layout, aisle configurations, and location-specific data can provide additional context for analysis.



**PROGRAM:**

***MARKET BASKET INSIGHTS***

import pandas as pd

from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules

# Sample transaction data (replace with your own dataset)

data = {

'Transaction\_ID': [1, 2, 3, 4, 5],

'Items': [['A', 'B', 'C'], ['A', 'B'], ['B', 'D'], ['A', 'C'], ['A', 'B', 'D']]

}

df = pd.DataFrame(data)

# Convert the 'Items' column into a one-hot encoded DataFrame

oht = df['Items'].str.join('|').str.get\_dummies('|')

# Apply Apriori algorithm to find frequent item sets

frequent\_item\_sets = apriori(oht, min\_support=0.5, use\_colnames=True)

# Generate association rules

association\_rules\_df = association\_rules(frequent\_item\_sets, metric='lift', min\_threshold=1.0)

# Display the association rules

print("Association Rules:")

print(association\_rules\_df)

***MODEL1:LINEAR REGERESSION***

**EXPLAINATION:**

Linear regression can be employed to model the relationship between product sales and various independent variables, such as pricing, promotions, and advertising expenditure. This can help retailers optimize their pricing and promotional strategies.

**SOURCE CODE:**

from sklearn.linear\_model import LinearRegression

# Create a linear regression model

model = LinearRegression()

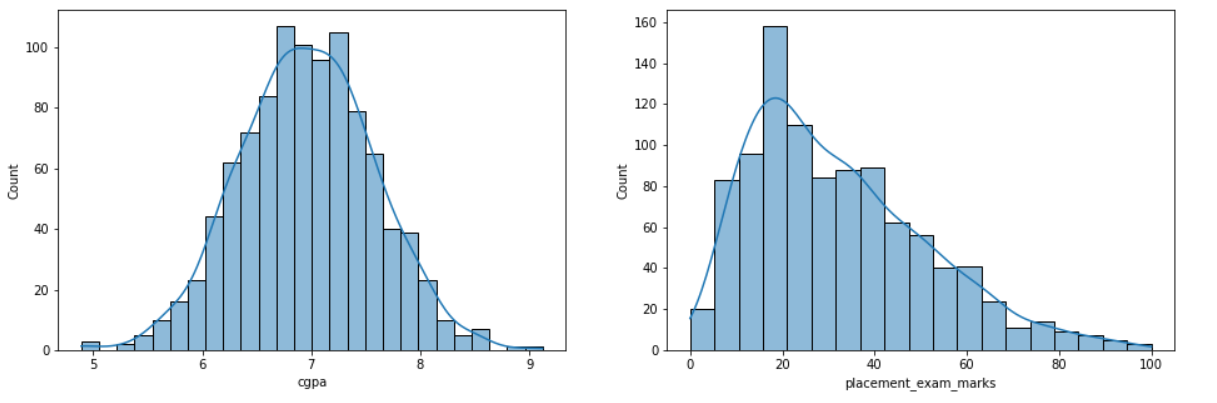
# Fit the model to your data

model.fit(X, y)

# Predict sales based on pricing and promotion

prediction = model.predict(new\_data)

**OUTPUT:**

****

***MODEL2:LOGISTIC REGRESSION***

**EXPLAINATION:**

Logistic regression can be used to predict the likelihood of a customer purchasing a specific product or a group of products based on their historical transaction data. It can help in understanding the factors that influence the purchase decision, such as customer demographics, previous purchases, and seasonality.

**SOURCE CODE:**

from sklearn.linear\_model import LogisticRegression

# Create a logistic regression model

model = LogisticRegression()

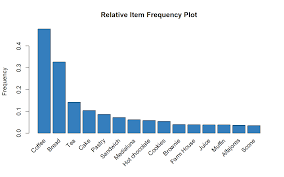
# Fit the model to your data

model.fit(X, y)

# Predict the likelihood of purchase for a new customer

prediction = model.predict(new\_customer\_data)

**OUTPUT:**

****

***MODEL3:NEGATIVE BINOMIAL REGRESSION***

**EXPLAINATION:**

Negative binomial regression is an extension of Poisson regression and is suitable for over dispersed count data. It can provide more accurate predictions when there is variability in purchase behaviour.

**SOURCE CODE:**

import statsmodels.api as sm

# Create a negative binomial regression model

model = sm.GLM(y, X, family=sm.families.NegativeBinomial())

# Fit the model to your data

results = model.fit()

# Predict purchase frequency for new data

prediction = results.predict(new\_data)

***MODEL4:RANDOM FOREST REGRESSION***

**EXPLAINATION:**

Random forest regression is an ensemble machine learning technique that can capture complex relationships between product sales and various predictors. It can handle both numerical and categorical variables, making it suitable for analysing market basket data with diverse attributes.

**SOURCE CODE:**

from sklearn.ensemble import RandomForestRegressor

# Create a random forest regression model

model = RandomForestRegressor()

# Fit the model to your data

model.fit(X, y)

# Predict sales based on features

prediction = model.predict(new\_data)

***MODEL5:TIME SERIES ANALYSIS***

**EXPLAINATION:**

Time series analysis techniques, such as ARIMA (Auto Regressive Integrated Moving Average) or Prophet, can be applied to analyse historical sales data and forecast future sales for individual products or product categories. This is particularly important for inventory management and demand forecasting.

**SOURCE CODE:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as sm

# Load your time series data (replace 'your\_data.csv' with your data file)

data = pd.read\_csv('your\_data.csv')

# Convert the date column to a datetime object

data['Date'] = pd.to\_datetime(data['Date'])

# Set the date column as the index

data.set\_index('Date', inplace=True)

# Visualize the time series

plt.figure(figsize=(12, 6))

plt.plot(data.index, data['Value'], label='Time Series Data')

plt.xlabel('Date')

plt.ylabel('Value')

plt.title('Time Series Data Visualization')

plt.legend()

plt.show()

# Check for missing values

missing\_values = data.isnull().sum()

print('Missing Values:')

print(missing\_values)

# Fill missing values if necessary (e.g., forward fill or interpolate)

# Decompose the time series into trend, seasonal, and residual components

decomposition = sm.tsa.seasonal\_decompose(data['Value'], model='additive')

trend = decomposition.trend

seasonal = decomposition.seasonal

residual = decomposition.resid

# Plot the decomposed components

plt.figure(figsize=(12, 8))

plt.subplot(411)

plt.plot(data['Value'], label='Original')

plt.legend()

plt.subplot(412)

plt.plot(trend, label='Trend')

plt.legend()

plt.subplot(413)

plt.plot(seasonal, label='Seasonal')

plt.legend()

plt.subplot(414)

plt.plot(residual, label='Residuals')

plt.legend()

plt.tight\_layout()

plt.show()

# Perform a Dickey-Fuller test for stationarity

from statsmodels.tsa.stattools import adfuller

result = adfuller(data['Value'])

print('ADF Statistic:', result[0])

print('p-value:', result[1])

print('Critical Values:')

for key, value in result[4].items():

print(f'{key}: {value}')

# If the time series is non-stationary, apply differencing to make it stationary

data['Differenced'] = data['Value'].diff().dropna()

# Perform the Dickey-Fuller test on the differenced series

# Fit a time series model (e.g., ARIMA)

model = sm.tsa.ARIMA(data['Value'], order=(1, 1, 1))

results = model.fit()

# Print model summary

print(results.summary())

# Forecast future values

forecast\_steps = 10

forecast, stderr, conf\_int = results.forecast(steps=forecast\_steps)

# Plot the original time series and forecasted values

plt.figure(figsize=(12, 6))

plt.plot(data.index, data['Value'], label='Original Data')

plt.plot(pd.date\_range(start=data.index[-1], periods=forecast\_steps + 1, closed='right'), [data['Value'].iloc[-1]] + list(forecast), label='Forecast', linestyle='--')

plt.fill\_between(pd.date\_range(start=data.index[-1], periods=forecast\_steps + 1, closed='right'), [data['Value'].iloc[-1] - stderr] + list(forecast - stderr), [data['Value'].iloc[-1] + stderr] + list(forecast + stderr), color='gray', alpha=0.2, label='Confidence Interval')

plt.xlabel('Date')

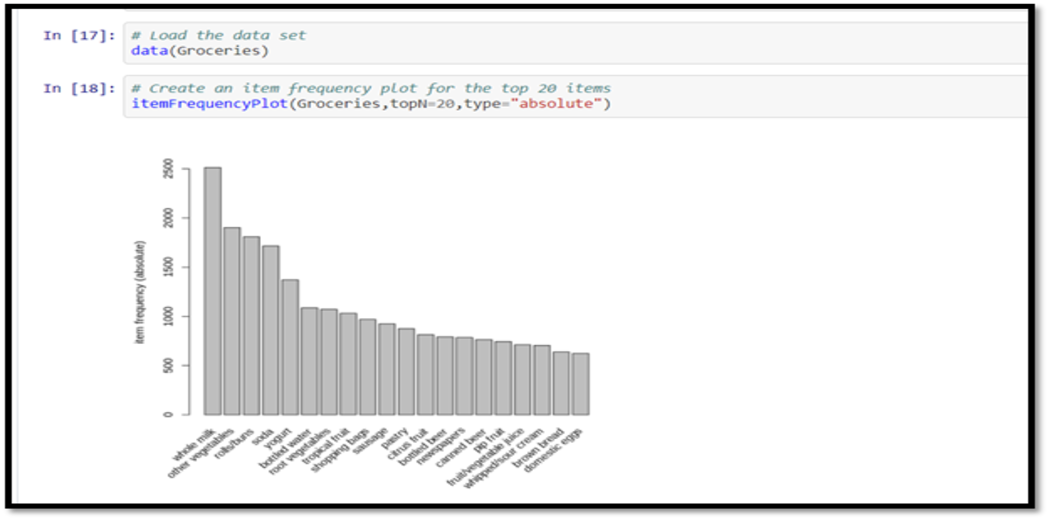
plt.ylabel('Value')

plt.title('Time Series Forecasting')

plt.legend()

plt.show()

**OUTPUT:**

****

**CONCLUSION:**

Market basket insights reveal product affinities, aid in customer segmentation, optimize inventory, inform pricing and promotions, enable personalization, and highlight seasonal trends. Businesses use this data to enhance operations, drive sales, and improve the customer experience.