DAIRY SALES DATA ANALYSIS

Kanishk Mehta

CONTENTS

1. INTRODUCTION

1.1 Overview of the Dairy Industry →

The dairy sector is a very important agricultural sector, providing fundamental products such as milk, cheese, and yogurt to millions globally. It contributes to food security, nutrition, and the economy by benefiting farmers, processors, and retailers. Through the years, technology has enhanced milk production, storage, and distribution to ensure quality and efficiency. As global demand rises, sustaining high levels of production while being sustainable is a huge challenge for the sector.

India is the world's biggest producer of dairy products with a robust system of small farmers, cooperatives, and big companies like Amul and Mother Dairy. Supportive government policies in the form of subsidies, improvement programs for cattle breeds, and quality control acts have supported the growth of the industry. Increasing demand for value-added dairy items such as flavoured milk, organic dairy, and cheese has further influenced market dynamics, dictating production and consumption patterns.

Although the dairy sector is growing, it is confronted by numerous challenges, such as price volatility, supply chain inefficiency, and climatic risks. Farmers tend to be confronted with volatile feed prices, animal disease issues, and market volatility. Consumers are also increasingly turning to healthier and plant-based options, which is a challenge for conventional dairy farmers. Sustainable agriculture practices, sophisticated data analytics, and enhanced logistics play a key role in solving these problems and creating long-term stability.

1.2 Objective of the study →

The main aim of this research is to examine different elements of the dairy sector, such as production patterns, prices, and quality assurance. Through an analysis of influential factors in dairy supply and demand, the project seeks to establish challenges and opportunities in the industry. The research also aims to offer data-driven findings that can be used by stakeholders, including farmers, firms, and policy makers, to make informed choices in order to enhance efficiency, profitability, and sustainability in dairy production and supply.

Being an integrated project, this research includes principles of

1. Python programming
2. Machine learning
3. Sampling methods
4. Inferential statistics
5. Time series analysis
6. Financial data analysis
7. Count & categorical data analysis

Through the application of these methods, the project will focus on building predictive models, financial trend analysis, and measuring categorical data in dairy production. Being an integration of the above analytical methods, the research enables a complete comprehension of the industry to accommodate data-driven decisions that maximize operational effectiveness and long-term development.

1.3 Scope of the Project →

The initiative focuses on examining some of the most important facets of the dairy business, such as patterns in production, trends in sales, and fiscal performance. With data analytics, the research tries to find meaningful information that will aid in decision-making for the stakeholders in the dairy industry.

The ambit will also include the use of machine learning algorithms for predictive modelling. Statistical methodology will be used for validating the results as well as analysing variations in data. Time series analysis will be utilized to follow changes over time, while financial analytics will provide an economic insight into dairy pricing and profitability. A balanced analysis is ensured by using this multi-angle approach.

Although its thorough methodology, the project is constrained by the information available in the dataset and does not consider external variables such as policy shifts or worldwide market fluctuations. The conclusions drawn will be valid mainly within the context of the provided dataset and thus useful for academic studies and business strategy development within specified parameters.

1.4 System Requirements →

1. Software Requirements :-

Operating System	Windows-11
Toolkit	Jupyter Notebook
Platform	Python & its libraries

2. Hardware Requirements :-

Processor	Intel CORE i5.
RAM	8 GB
SSD	128 GB
Hard Disk	1200 GB

1.5 Dataset Overview →

This dataset offers an overall picture of the dairy sector, encompassing major aspects like production, storage, pricing, and sales. It contains information on different dairy products, their quantities, selling prices, and revenues. Moreover, the dataset records information about the farms where these products are being produced, such as the number of cows and total land area, thus being helpful for analysing market trends and production efficiency.

Additionally, the dataset contains geographic and distribution attributes like customer locations and sales channels (e.g., retail, online, wholesale) that serve as the foundation for regional and market-level analysis.

With a combination of numeric and categorical variables, this dataset lends itself to several analytical purposes such as sales forecasting, demand analysis and optimization of dairy product distribution.

```
Dataset Overview:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4325 entries, 0 to 4324
Data columns (total 18 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Total Land Area (acres)     4325 non-null   float64
 1   Number of Cows              4325 non-null   int64
 2   Farm Size                   4325 non-null   object
 3   Date                        4325 non-null   object
 4   Product ID                  4325 non-null   int64
 5   Product Name                4325 non-null   object
 6   Brand                       4325 non-null   object
 7   Quantity (liters/kg)        4325 non-null   float64
 8   Price per Unit              4325 non-null   float64
 9   Total Value                 4325 non-null   float64
 10  Storage Condition           4325 non-null   object
 11  Production Date             4325 non-null   object
 12  Expiration Date             4325 non-null   object
 13  Quantity Sold (liters/kg)   4325 non-null   int64
 14  Price per Unit (sold)       4325 non-null   float64
 15  Approx. Total Revenue(INR)  4325 non-null   float64
 16  Customer Location           4325 non-null   object
 17  Sales Channel               4325 non-null   object
dtypes: float64(6), int64(3), object(9)
memory usage: 608.3+ KB
```

2. DATA CLEANING AND PREPROCESSING

Data Cleaning and Preprocessing is essential to ensure data quality and reliability for analysis. It involves handling **missing values** by filling them with appropriate statistics, removing **duplicate records** to prevent redundancy, and converting **date columns** into the correct format for accurate time-based analysis. Additionally, **outliers** are detected using statistical methods like the IQR (Interquartile Range) and removed to avoid skewed results. These steps enhance dataset consistency, improve model performance, and ensure meaningful insights. Proper preprocessing eliminates inconsistencies, making the data structured and ready for further exploration, visualization, and machine learning applications.

Python libraries used –
Pandas - Used for data cleaning tasks, including handling missing values, removing duplicates, converting date columns, and detecting/removing outliers.

Step 1 - Handling Missing Values
Missing values can occur due to various reasons such as data entry errors, system failures, or missing information in the source files. Handling missing values is crucial as they can affect the accuracy of the analysis and predictive models. In this step, we analyse missing values in the dataset and apply appropriate techniques to handle them.

```
#Handling Missing Values
print("\n HANDLING MISSING VALUES →")

#Checking for missing values
missing_values = df.isnull().sum()
total_missing = missing_values.sum()

if total_missing > 0:
    print("\nMissing Values Found:\n", missing_values[missing_values > 0])

#Handling missing values
    df.fillna({
    col: df[col].median() if df[col].dtype in ['int64', 'float64'] else df[col].mode().iloc[0]
    for col in missing_values[missing_values > 0].index
    }, inplace=True)

#Confirming missing values are handled
    print("\nAll missing values have been successfully handled.")
else:
    print("\nNo missing values exist in the dataset.")
```

Step 2 - Handling Duplicates
Duplicate records can distort analysis by over-representing certain data points. Removing duplicates ensures the dataset remains accurate and unbiased.

```
#Handling Duplicates
print("\nHANDLING DUPLICATES →")

# Checking for and removing duplicate rows
duplicates = df.duplicated().sum()
if duplicates > 0:
    df.drop_duplicates(inplace=True)
    print(f"\nRemoved {duplicates} duplicate rows.")
else:
    print("\nNo duplicate rows exist in the dataset.")
```

Step 3 - Converting Date Columns
Dates are often stored as strings, making it difficult to perform operations like sorting and filtering. Converting them into proper date formats allows for better time-based analysis.

```
#Converting Date Columns
print("\nCONVERTING DATE COLUMNS →")

#Converting date columns to datetime format
date_cols = ['Date', 'Production Date', 'Expiration Date']
for col in date_cols:
    if col in df.columns:
        df[col] = pd.to_datetime(df[col], errors='coerce')

print("\nDate columns converted to datetime format.")
```

Step 4 – Outlier Detection and Removal

Outliers are extreme values that may distort the analysis. They can result from data entry errors, sensor malfunctions, or genuine variations. Detecting and handling outliers is crucial for ensuring a reliable dataset.

```python
#Outlier Detection & Removal
print("\nOUTLIER DETECTION & REMOVAL →")

# Selecting a numerical column for outlier detection (e.g., 'Approx. Total Revenue(INR)')
column_of_interest = 'Approx. Total Revenue(INR)'

# Store original dataset size before removing outliers
original_size = df.shape[0]

# Outlier Detection using IQR Method
Q1 = df[column_of_interest].quantile(0.25)
Q3 = df[column_of_interest].quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers
df = df[(df[column_of_interest] >= lower_bound) & (df[column_of_interest] <= upper_bound)]

# Calculate the actual number of outliers removed
outliers_removed = original_size - df.shape[0]

print(f"\nTotal Outliers Removed: {outliers_removed}")
print(f"Dataset Size After Outlier Removal: {df.shape[0]} rows")
```

OUTPUT –

```
SECTION 1 - DATA CLEANING

 HANDLING MISSING VALUES →

No missing values exist in the dataset.

HANDLING DUPLICATES →

No duplicate rows exist in the dataset.

CONVERTING DATE COLUMNS →

Date columns converted to datetime format.

OUTLIER DETECTION & REMOVAL →

Total Outliers Removed: 225
Dataset Size After Outlier Removal: 4100 rows
```

3. SAMPLING TECHNIQUES

Sampling methods are crucial in data analysis to obtain meaningful conclusions from large data sets with efficiency. In this research, different sampling methods are used in the dairy dataset to select representative data. **Simple random sampling** gives an unbiased sample, **systematic sampling** samples data at regular intervals, and **stratified sampling** provides proportional representation by 'Product Name.'

These methods assist in simplifying computational complexity without compromising data integrity. By **comparing variances**, we judge each method's effectiveness and ensure the selected sample perfectly represents the whole dataset for future statistical and machine learning processing.

Python libraries used –

1. Pandas – used for implementing different sampling techniques (Simple Random, Systematic, and Stratified Sampling), saving sampled data as CSV files, and handling data operations.

2. NumPy - used for calculating the variance of sampled data to compare different sampling techniques.

Step 1: Drawing the samples –
In order to efficiently analyze the dairy dataset, we use three sampling methods: simple random sampling, systematic sampling, and stratified sampling. Simple random sampling chooses data points randomly, systematic sampling chooses data at fixed intervals, and stratified sampling provides proportional representation by 'Product Name.' These samples assist in minimizing computational effort without losing dataset diversity.

```
#SAMPLING TECHNIQUES
print("SECTION 2 - SAMPLING TECHNIQUES")

#Simple Random Sampling
simple_random_sample = df.sample(n=100, random_state=42)

#Systematic Sampling
interval = len(df) // 100
systematic_sample = df.iloc[::interval, :].head(100)

#Stratified Sampling
stratified_sample = df.groupby('Product Name', group_keys=False).apply(lambda x: x.sample(frac=0.1, random_state=42))
stratified_sample = stratified_sample.sample(n=100, random_state=42) if len(stratified_sample) > 100 else stratified_sample

# Save each sample as a CSV file
simple_random_sample.to_csv("simple_random_sample_dairy.csv", index=False)
systematic_sample.to_csv("systematic_sample_dairy.csv", index=False)
stratified_sample.to_csv("stratified_sample_dairy.csv", index=False)
```

Step 2: Finding the Best Technique –
To find the best sampling method, we contrast each technique's variance. The lower the variance, the better it represents the entire dataset. Stratified sampling tends to provide the most accurate output by maintaining category distributions. This guarantees that significant patterns are intact, which makes it a sound option for further statistical and machine learning uses.

```
#Variance Comparison
print("\nVARIANCE COMPARISON →")

def calculate_variance(sample):return np.var(sample[column_of_interest], ddof=1)

var_simple = calculate_variance(simple_random_sample)
var_systematic = calculate_variance(systematic_sample)
var_stratified = calculate_variance(stratified_sample)

print("\nVariance of Different Sampling Techniques:")
print(f"Simple Random Sampling Variance: {var_simple:.4f}")
print(f"Systematic Sampling Variance: {var_systematic:.4f}")
print(f"Stratified Sampling Variance: {var_stratified:.4f}")

#Identifying the Best Sampling Technique
print("\nBEST SAMPLING TECHNIQUE →")

best_sampling = min(var_simple, var_systematic, var_stratified)
if best_sampling == var_stratified:
    best_method = "Stratified Sampling"
elif best_sampling == var_systematic:
    best_method = "Systematic Sampling"
else:
    best_method = "Simple Random Sampling"
print(f"\nBest Sampling Technique: {best_method}")
```

OUTPUT –

```
        SECTION 2 - SAMPLING TECHNIQUES

VARIANCE COMPARISON →

Variance of Different Sampling Techniques:
Simple Random Sampling Variance: 106440096.9125
Systematic Sampling Variance: 115901283.2518
Stratified Sampling Variance: 102868643.1567

BEST SAMPLING TECHNIQUE →

Best Sampling Technique: Stratified Sampling
```

4. INFERENTIAL STATISTICS

Inferential statistics assists us in making conclusions regarding a population from a sample, allowing us to **test hypotheses** and detect meaningful differences in data. In the current research, an **ANOVA test** is used on the dairy dataset to examine differences in **Price per Unit** among various **Sales Channels**. This test assists in establishing whether or not the mean price per unit varies significantly by sales channels or if the differences are random.

Through testing statistical significance, we confirm whether the differences that we see between prices on different sales channels are significant and not random fluctuations. This is important for making price strategy decisions as well as conducting market analysis.

Python Libraries Used –

1. Pandas – For data handling, grouping by sales channels, and saving ANOVA results.

2. SciPy – For conducting the **ANOVA test** to compare the means of different sales channels.

3. NumPy – For variance calculations and sum of squares computations used in the ANOVA table.

Step 1: Defining the Test and Hypotheses –

To determine whether the price per unit differs across sales channels, we define:

Null Hypothesis (H_0): The average price per unit is the same across all sales channels.

Alternative Hypothesis (H_1): At least one sales channel has a significantly different average price per unit.

```
#INFERENTIAL STATISTICS
print("SECTION 3 - INFERENTIAL STATISTICS")

#Hypothesis Testing: ANOVA Test for Price per Unit Across Sales Channels
print("\nHYPOTHESIS TESTING: ANOVA TEST FOR PRICE PER UNIT ACROSS SALES CHANNELS →")

#Stating Hypotheses
print("\nNull Hypothesis (H0): The average price per unit is the same across all sales channels.")
print("Alternative Hypothesis (H1): At least one sales channel has a significantly different average price per unit.")
```

Step 2: Performing the ANOVA Test

We apply **one-way ANOVA** to compare the means of price per unit across different sales channels. The process involves:

1. **Grouping Data by Sales Channel** – Extracting unique sales channels and creating data groups.
2. **Applying ANOVA Test** – Using f_oneway() from SciPy to compute:
   F-statistic – Indicates the extent of variation between groups compared to within groups.
   P-value – Determines statistical significance of the variation.
3. Additionally, we construct an **ANOVA table** by computing:
   Sum of Squares (SS) – Measures variance between and within groups.
   Degrees of Freedom (df) – Helps determine variability sources.
   Mean Square (MS) – Obtained by dividing SS by df.
   F-Statistic & P-Value – Used to interpret the test results.

```python
#Extract unique sales channels
channels = df['Sales Channel'].unique()
channel_groups = [df[df['Sales Channel'] == channel]['Price per Unit'].dropna() for channel in channels]

#Perform ANOVA test
f_stat, p_value_anova = stats.f_oneway(*channel_groups)

#Degrees of freedom
df_between = len(channels) - 1
df_within = len(df) - len(channels)

#Sum of squares calculations
ss_between = sum([len(group) * (group.mean() - df['Price per Unit'].mean()) ** 2 for group in channel_groups])
ss_within = sum([sum((group - group.mean()) ** 2) for group in channel_groups])
ss_total = ss_between + ss_within

# Mean square calculations
ms_between = ss_between / df_between
ms_within = ss_within / df_within

# ANOVA Table
anova_table = pd.DataFrame({
'Source': ['Between Groups', 'Within Groups', 'Total'],
'Sum of Squares (SS)': [ss_between, ss_within, ss_total],
'Degrees of Freedom (df)': [df_between, df_within, df_between + df_within],
'Mean Square (MS)': [ms_between, ms_within, np.nan],
'F-Statistic': [f_stat, np.nan, np.nan],
'P-Value': [p_value_anova, np.nan, np.nan]
})
```

Step 3: Interpreting the Result

- If **p-value < 0.05**, we reject H_0, i.e., price per unit varies significantly across sales channels.
- If **p-value ≥ 0.05**, we fail to reject H_0, i.e., price per unit remains consistent across sales channels.

```python
# Interpretation
print(f"\nF-statistic = {f_stat:.4f}, P-value = {p_value_anova:.4f}")

if p_value_anova < 0.05:
    print("\nCONCLUSION → Since the p-value is less than 0.05, we reject the null hypothesis.")
    print("At least one sales channel has a significantly different average price per unit compared to others.")
else:
    print("\nCONCLUSION → Since the p-value is greater than 0.05, we fail to reject the null hypothesis.")
    print("There is no significant difference in the average price per unit across sales channels.")
```

OUTPUT –

```
SECTION 3 - INFERENTIAL STATISTICS

HYPOTHESIS TESTING: ANOVA TEST FOR PRICE PER UNIT ACROSS SALES CHANNELS →

Null Hypothesis (H0): The average price per unit is the same across all sales channels.
Alternative Hypothesis (H1): At least one sales channel has a significantly different average price per unit.

F-statistic = 0.0486, P-value = 0.9526

CONCLUSION → Since the p-value is greater than 0.05, we fail to reject the null hypothesis.
There is no significant difference in the average price per unit across sales channels.
```

SOURCE	SUM OF SQUARES (SS)	DEGREES OF FREEDOM (DF)	MEAN SQUARE (MS)	F-STATISTIC	P-VALUE
BETWEEN GROUPS	64.09891949	2	32.04945974	0.048574585	0.952586837
WITHIN GROUPS	2703196.28	4097	659.7989455		
TOTAL	2703260.379	4099			

5. TIME SERIES ANALYSIS

Time series analysis helps identify patterns over time and forecast future values. We analyze **monthly revenue trends** using three methods:
Least Squares Method – Fits a linear regression model.
Moving Average Method – Smooths short-term fluctuations.
Semi-Average Method – Uses two-period averages for trend estimation.

These methods help in accurate **trend detection** and **forecasting** for decision-making.

Python Libraries Used –

1. **Pandas –** For handling time-series data and resampling.
2. **NumPy –** For numerical computations.
3. **Statsmodels –** For regression analysis in the Least Squares Method.
4. **Matplotlib –** For visualizing trends.

Step 1: Data Preprocessing

- Convert the **'Date'** column to datetime format.
- Resample **daily revenue** into **monthly revenue** for trend analysis.
- Create a **time index variable** for trend calculation.

```
#TIME SERIES TREND ANALYSIS & FORECASTING
print("SECTION 4 - TIME SERIES TREND ANALYSIS & FORECASTING")

# Convert 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

# Aggregate sales monthly instead of daily
df_time = df.resample('M', on='Date')['Approx. Total Revenue(INR)'].sum().reset_index()

# Set 'Date' as index
df_time.set_index('Date', inplace=True)

# Creating time variable (X)
df_time['Time'] = np.arange(1, len(df_time) + 1)
```

Step 2: Trend Calculation Using Three Methods -

Least Squares Method: Fits a regression model on time-series data.

Moving Average Method: Computes a 3-month moving average to smooth fluctuations.

Semi-Average Method: Splits data into two halves, calculates averages, and estimates trends.

The sum of each method's trend values is computed for accuracy comparison.

```
# Least Squares Method
X = sm.add_constant(df_time['Time'])  # Adding constant for intercept
model = sm.OLS(df_time['Approx. Total Revenue(INR)'], X).fit()
df_time['Least Squares Trend'] = model.predict(X)
sum_least_squares = df_time['Least Squares Trend'].sum()

# Moving Average Method
df_time['Moving Average Trend'] = (
    df_time['Approx. Total Revenue(INR)']
    .rolling(window=3, center=True)
    .mean()
)

# Fill the first and last missing values with the original values
df_time['Moving Average Trend'].fillna(df_time['Approx. Total Revenue(INR)'], inplace=True)

sum_moving_average = df_time['Moving Average Trend'].sum()

# Semi-Average Method
mid = len(df_time) // 2
first_half = df_time.iloc[:mid]
second_half = df_time.iloc[mid:]

avg_1 = first_half['Approx. Total Revenue(INR)'].mean()
avg_2 = second_half['Approx. Total Revenue(INR)'].mean()
t1 = mid // 2
t2 = mid + (len(df_time) - mid) // 2
b = (avg_2 - avg_1) / (t2 - t1)
a = avg_1 - (b * t1)
df_time['Semi-Average Trend'] = a + b * df_time['Time']
sum_semi_average = df_time['Semi-Average Trend'].sum()
```

Step 3: Selecting the Best Trend Method –

Compare each method's total trend value with the actual revenue sum.

The method with the smallest deviation from actual revenue is selected for forecasting.

```
#Trend Analysis Results
print("\nTREND ANALYSIS RESULTS →")
print("\n 🔹 Sum of Trend Values for Each Method:")
print(f"Least Squares Method: {sum_least_squares:.2f}")
print(f"Moving Average Method: {sum_moving_average:.2f}")
print(f"Semi-Average Method: {sum_semi_average:.2f}")

print("\n 🔹 Actual Total Sum of Revenue:")
sum_actual = df_time['Approx. Total Revenue(INR)'].sum()
print(f"Actual Revenue Sum: {sum_actual:.2f}")

#Comparing Trend Accuracy
print("\nCOMPARING TREND ACCURACY WITH ACTUAL REVENUE →")
diff_least_squares = abs(sum_actual - sum_least_squares)
diff_moving_average = abs(sum_actual - sum_moving_average)
diff_semi_average = abs(sum_actual - sum_semi_average)

print("\n 🔹 Difference from Actual Revenue Sum:")
print(f"Least Squares Method: {diff_least_squares:.2f}")
print(f"Moving Average Method: {diff_moving_average:.2f}")
print(f"Semi-Average Method: {diff_semi_average:.2f}")

#Best Trend Calculation Method Selection
print("\nBEST TREND CALCULATION METHOD SELECTION →")
```

Step 4: Forecasting for Next 12 Months –

Apply the best trend estimation method to predict future revenue.

Generate revenue forecasts for the next 12 months using the selected model.

```
# Forecasting for Next 12 Months
print("\nFORECASTING FOR THE NEXT 12 MONTHS →")

forecast_time = np.arange(len(df_time) + 1, len(df_time) + 13)  # Next 12 months

if best_trend == "Least Squares Method":
    forecast_values = model.params[0] + model.params[1] * forecast_time
elif best_trend == "Moving Average Method":
    forecast_values = [df_time['Moving Average Trend'].iloc[-1]] * 12  # Repeat last known MA
elif best_trend == "Semi-Average Method":
    forecast_values = a + b * forecast_time

forecast_df = pd.DataFrame({'Forecasted Revenue (INR)': forecast_values},
   index=pd.date_range(start=df_time.index[-1] + pd.DateOffset(months=1), periods=12, freq='M'))
print(forecast_df)
```

Step 5: Visualizing & Interpreting the Trend –

Plot actual revenue vs. estimated trend using a scatter and line graph.

Trend Direction:-

Increasing: Indicates growth opportunities.

Decreasing: Suggests corrective action is needed.

Stable: Implies consistent business performance.

This analysis ensures data-driven decision-making for future financial planning.

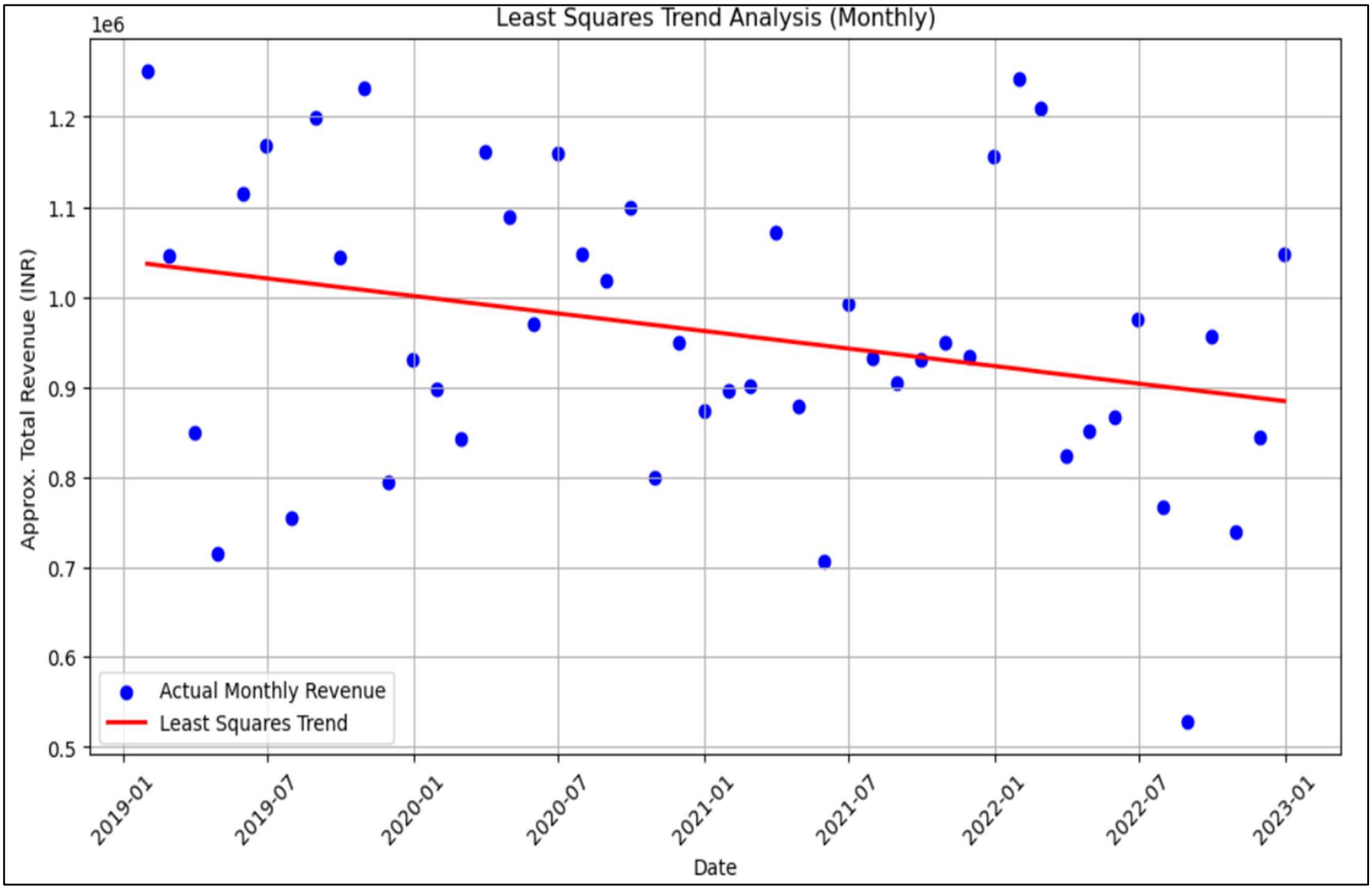
```
print("\nVISUALIZING THE TRENDS →")

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

# Scatter plot for actual monthly revenue
plt.scatter(df_time.index, df_time['Approx. Total Revenue(INR)'], label='Actual Monthly Revenue', color='blue', marker='o')

# Line plot for the least squares trend
plt.plot(df_time.index, df_time['Least Squares Trend'], label='Least Squares Trend', color='red', linestyle='-', linewidth=2)

# Formatting the plot
plt.title('Least Squares Trend Analysis (Monthly)')
plt.xlabel('Date')
plt.ylabel('Approx. Total Revenue (INR)')
plt.legend()
plt.grid()
plt.xticks(rotation=45)

plt.show()

#Interpretation of the Trend Graph
print("\nINTERPRETATION OF THE TREND GRAPH →")

# Detecting overall trend direction
if df_time['Least Squares Trend'].iloc[-1] > df_time['Least Squares Trend'].iloc[0]:
    trend_direction = "increasing"
elif df_time['Least Squares Trend'].iloc[-1] < df_time['Least Squares Trend'].iloc[0]:
    trend_direction = "decreasing"
else:
    trend_direction = "stable"

print(f"\nOverall, the revenue trend appears to be {trend_direction} over time.")
```

OUTPUT –

```
SECTION 4 - TIME SERIES TREND ANALYSIS & FORECASTING

TREND ANALYSIS RESULTS →

🔹 Sum of Trend Values for Each Method:
Least Squares Method: 46123581.30
Moving Average Method: 46259553.74
Semi-Average Method: 46044487.19

🔹 Actual Total Sum of Revenue:
Actual Revenue Sum: 46123581.30

COMPARING TREND ACCURACY WITH ACTUAL REVENUE →

🔹 Difference from Actual Revenue Sum:
Least Squares Method: 0.00
Moving Average Method: 135972.44
Semi-Average Method: 79094.11

BEST TREND CALCULATION METHOD SELECTION →

🎯 Best Method for Trend Estimation: Least Squares Method
```

FORECASTING FOR THE NEXT 12 MONTHS →

	Forecasted Revenue (INR)
2023-01-31	881552.184468
2023-02-28	878313.173885
2023-03-31	875074.163302
2023-04-30	871835.152719
2023-05-31	868596.142136
2023-06-30	865357.131553
2023-07-31	862118.120970
2023-08-31	858879.110388
2023-09-30	855640.099805
2023-10-31	852401.089222
2023-11-30	849162.078639
2023-12-31	845923.068056



INTERPRETATION OF THE TREND GRAPH →

Overall, the revenue trend appears to be decreasing over time.

6. FINANCIAL DATA ANALYSIS

The analysis of financial information is important for interpreting business performance, recognizing patterns in profit levels, and arriving at sound fiscal decisions. Observing monthly gross profit provides knowledge regarding revenue oscillations, costs, and financial health.

Here, we divide the trends in gross profit over time, determine the months of highest and lowest revenues, and analyze the general direction of financial performance. This enables companies to maximize pricing strategies, control costs effectively, and make informed decisions for long-term viability.

Python Libraries Used

1. **Pandas** – For data manipulation and aggregation.
2. **Matplotlib** – For visualizing financial trends.

Step 1: Data Preprocessing & Monthly Gross Profit Calculation

- Extract **year-month** from the date column for monthly aggregation.
- Calculate **Gross Profit** and aggregate **gross profit** for each month.

```
# FINANCIAL DATA ANALYSIS
print("SECTION 5 - FINANCIAL DATA ANALYSIS")

# Extract month and year for aggregation
df['Year-Month'] = df['Date'].dt.to_period('M')

# Calculate Gross Profit
df['Gross Profit'] = (df['Price per Unit (sold)'] - df['Price per Unit']) * df['Quantity Sold (liters/kg)']

# Aggregate Gross Profit by month
monthly_gross_profit = df.groupby('Year-Month')['Gross Profit'].sum().reset_index()

# Convert 'Year-Month' to string for saving
monthly_gross_profit['Year-Month'] = monthly_gross_profit['Year-Month'].astype(str)
```

Step 2: Trend Analysis & Interpretation

Compare **initial** and **final** gross profit to determine financial growth:

Increase → Positive financial trend.

Decrease → Declining financial trend.

Stable → Consistent performance.

Find the **highest** and **lowest** gross profit months.

Analyze peak and low-profit periods for business optimization.

```
# Determine overall trend
initial_profit = monthly_gross_profit['Gross Profit'].iloc[0]
final_profit = monthly_gross_profit['Gross Profit'].iloc[-1]

if final_profit > initial_profit:
    trend_analysis = "Overall, the gross profit has increased over time, indicating a positive financial trend."
elif final_profit < initial_profit:
    trend_analysis = "Overall, the gross profit has decreased over time, suggesting a declining financial trend."
else:
    trend_analysis = "The gross profit has remained stable without any significant increase or decrease."

# Identify months with highest and lowest profits
max_profit_month = monthly_gross_profit.loc[monthly_gross_profit['Gross Profit'].idxmax()]
min_profit_month = monthly_gross_profit.loc[monthly_gross_profit['Gross Profit'].idxmin()]
```

Step 3: Visualizing & Interpreting the Trend

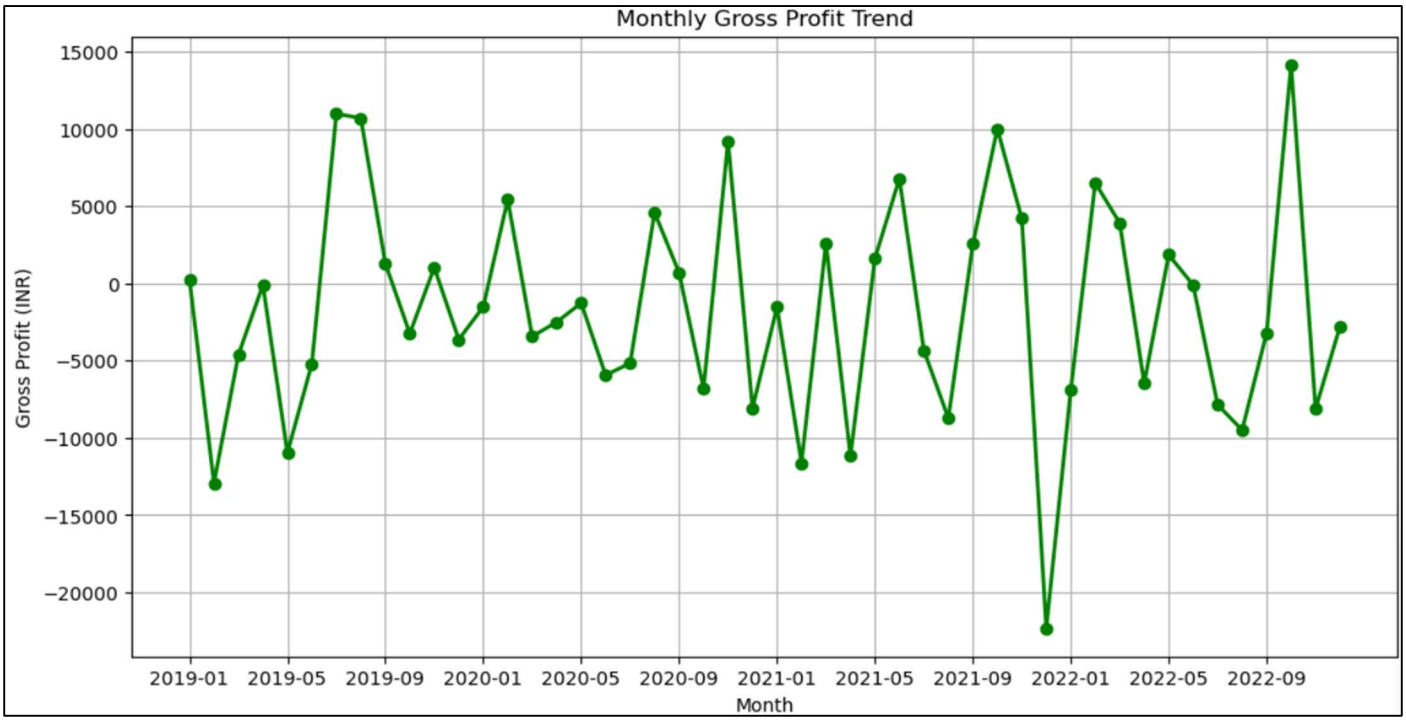
- Plot **monthly gross profit** to observe fluctuations.
- Provide actionable insights based on trend direction and variations.

This analysis aids in strategic decision-making and improving **financial stability.**

```python
# Plot the monthly gross profit
plt.figure(figsize=(12, 6))
plt.plot(monthly_gross_profit['Year-Month'], monthly_gross_profit['Gross Profit'], marker='o', linestyle='-', color='green', linewidth=2)
plt.xlabel('Month')
plt.ylabel('Gross Profit (INR)')
plt.title('Monthly Gross Profit Trend')

# Adjust x-axis labels to reduce congestion
plt.xticks(ticks=range(0, len(monthly_gross_profit), 4), labels=monthly_gross_profit['Year-Month'][::4])

plt.grid(True)
plt.show()


# Interpretation of the Gross Profit Trend
print("\nINTERPRETATION OF THE GROSS PROFIT TREND →")
print(trend_analysis)
print(f"\n🔹 The highest gross profit was recorded in {max_profit_month['Year-Month']} with a value of INR {max_profit_month['Gross Profit']:.2f}.")
print(f"🔹 The lowest gross profit was recorded in {min_profit_month['Year-Month']} with a value of INR {min_profit_month['Gross Profit']:.2f}.")
```

OUTPUT –

Year-Month	Gross Profit
2019-01	255.84
2019-02	-12925.8
2019-03	-4614.32
2019-04	-88.85
2019-05	-10963.3
2019-06	-5202.32
2019-07	10989.73
2019-08	10709.85
2019-09	1302.22
2019-10	-3243.97
2019-11	1035.2
2019-12	-3627.04
2020-01	-1516.05
2020-02	5428.09
2020-03	-3426.76
2020-04	-2502.34
2020-05	-1264.47
2020-06	-5937.64
2020-07	-5162.81
2020-08	4593.99
2020-09	698.24
2020-10	-6810.84
2020-11	9203.52
2020-12	-8082.71
2021-01	-1530.64
2021-02	-11624.8
2021-03	2614.01
2021-04	-11172.9
2021-05	1618.07
2021-06	6794.24
2021-07	-4331.07
2021-08	-8732.21
2021-09	2547.82
2021-10	10020.62
2021-11	4200.18
2021-12	-22328.2
2022-01	-6911.1
2022-02	6516.35
2022-03	3912.69
2022-04	-6470.6
2022-05	1856.36
2022-06	-99.96
2022-07	-7851.5
2022-08	-9494.03
2022-09	-3208.15
2022-10	14174.63
2022-11	-8120.15
2022-12	-2769.19




```
INTERPRETATION OF THE GROSS PROFIT TREND →
Overall, the gross profit has decreased over time, suggesting a declining financial trend.

🔹 The highest gross profit was recorded in 2022-10 with a value of INR 14174.63.
🔹 The lowest gross profit was recorded in 2021-12 with a value of INR -22328.17.
```

7. COUNT AND CATEGORICAL ANALYSIS

Count and categorical data analysis is useful in grasping relationships among categorical variables like brands, locations of customers, and product preference. It yields valuable insights regarding customer behaviour, market segmentation, and buying patterns.

We apply the Chi-Square Test for Independence in this analysis to identify if there is a significant relationship between Customer Location and Brand. With an analysis of a contingency table, we establish whether the distribution of customers by location depends on the brand they buy.

Python Libraries Used

1. **pandas** – For data manipulation and creating contingency tables
2. **scipy.stats** – For performing the **Chi-Square Test for Independence**
3. **numpy** – For handling numerical operations

Step 1: Constructing the Contingency Table -

- Extract the categorical variables: **'Brand'** and **'Customer Location'**
- Create a **contingency table** using pd.crosstab() to show frequency distribution

```
# COUNT AND CATEGORICAL DATA
print("SECTION 5 - COUNT AND CATEGORICAL DATA")x

# 1. Chi-Square Test for Independence: Relationship Between Brand and Customer Location
print("\nCHI-SQUARE TEST FOR INDEPENDENCE: RELATIONSHIP BETWEEN BRAND AND CUSTOMER LOCATION →")

# Selecting categorical variables: 'Brand' and 'Customer Location'
contingency_table = pd.crosstab(df['Brand'], df['Customer Location'])

contingency_table.to_csv("contingency_table_Dairysales.csv", index=False)
```

Step 2: Defining the hypotheses -
Display the formulated hypotheses:

- **H_0 (Null Hypothesis)**: No significant relationship between 'Brand' and 'Customer Location'
- **H_1 (Alternative Hypothesis)**: A significant relationship exists between 'Brand' and 'Customer Location'

```
# Display Hypotheses
print("\n📌 HYPOTHESES FOR CHI-SQUARE TEST:")
print(" 🔹 Null Hypothesis (H₀): There is no significant relationship between 'Brand' and 'Customer Location'.")
print(" 🔹 Alternative Hypothesis (H₁): There is a significant relationship between 'Brand' and 'Customer Location'.")
```

Step 3: Performing the Chi-Square Test -

Use stats.chi2_contingency() to compute:

Chi-Square Statistic

Degrees of Freedom (dof)

P-value

Expected Frequencies

```
# Perform Chi-Square test
chi2_stat, p_value, dof, expected = stats.chi2_contingency(contingency_table)

# Convert expected values to DataFrame for better readability
expected_df = pd.DataFrame(expected, index=contingency_table.index, columns=contingency_table.columns)

# Display expected frequencies
expected_df.to_csv("expected_df_Dairysales.csv", index=False)

# Display Chi-Square test results
print("\n🔹 CHI-SQUARE TEST RESULTS:")
print(f"Chi-Square Statistic: {chi2_stat:.2f}")
print(f"Degrees of Freedom: {dof}")
print(f"P-Value: {p_value:.5f}")
```

Step 4: Interpreting the Results -

Compare **p-value** with the significance level (**0.05**):

- **If p-value < 0.05** → Reject H_0 → Significant relationship exists
- **If p-value ≥ 0.05** → Fail to reject H_0 → No significant relationship

```
# Display Chi-Square test results
print("\n🔹 CHI-SQUARE TEST RESULTS:")
print(f"Chi-Square Statistic: {chi2_stat:.2f}")
print(f"Degrees of Freedom: {dof}")
print(f"P-Value: {p_value:.5f}")

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("\nCONCLUSION: The p-value is less than 0.05, so we reject the null hypothesis.")
    print("   → There is a significant relationship between 'Brand' and 'Customer Location'.")
else:
    print("\nCONCLUSION: The p-value is greater than 0.05, so we fail to reject the null hypothesis.")
    print("   → There is no significant relationship between 'Brand' and 'Customer Location'.")
```

OUTPUT –

CONTINGENCY TABLE

Bihar	Chandigarh	Delhi	Gujarat	Haryana	Jharkhand	Karnataka	Kerala	Madhya Pradesh	Maharashtra	Rajasthan	Tamil Nadu	Telangana	Uttar Pradesh	West Bengal
65	115	119	61	57	56	63	63	56	61	52	61	42	61	61
6	2	10	1	6	4	4	4	7	6	6	6	9	11	3
14	31	22	7	11	14	16	10	14	13	12	12	14	12	10
5	11	8	5	8	7	3	10	5	9	5	7	3	7	4
60	110	108	58	56	48	68	48	64	58	54	56	53	53	67
17	20	24	10	9	12	11	17	16	15	9	14	11	6	10
8	17	8	9	6	4	9	2	3	2	4	8	4	9	5
6	15	9	2	6	5	4	9	2	3	3	6	4	3	12
41	64	87	41	23	45	29	49	35	43	31	35	52	39	33
32	69	72	33	32	33	38	32	32	40	36	41	45	45	40
6	9	14	7	3	3	6	4	2	8	8	8	5	10	4

EXPECTED VALUES TABLE

Bihar	Chandigarh	Delhi	Gujarat	Haryana	Jharkhand	Karnataka	Kerala	Madhya Pradesh	Maharashtra	Rajasthan	Tamil Nadu	Telangana	Uttar Pradesh	West Bengal
62.97	112.14	116.50	56.67	52.56	55.95	60.79	60.06	57.16	62.49	53.28	61.52	58.61	62.00	60.31
5.39	9.60	9.97	4.85	4.50	4.79	5.20	5.14	4.89	5.35	4.56	5.27	5.02	5.31	5.16
13.44	23.94	24.87	12.10	11.22	11.94	12.98	12.82	12.20	13.34	11.38	13.13	12.51	13.24	12.88
6.15	10.95	11.38	5.54	5.13	5.47	5.94	5.87	5.58	6.10	5.20	6.01	5.73	6.06	5.89
60.94	108.52	112.74	54.85	50.86	54.14	58.83	58.13	55.32	60.47	51.57	59.54	56.72	60.00	58.36
12.75	22.70	23.58	11.47	10.64	11.32	12.31	12.16	11.57	12.65	10.79	12.45	11.86	12.55	12.21
6.21	11.07	11.50	5.59	5.19	5.52	6.00	5.93	5.64	6.17	5.26	6.07	5.78	6.12	5.95
5.64	10.05	10.44	5.08	4.71	5.01	5.45	5.38	5.12	5.60	4.78	5.51	5.25	5.56	5.41
41.03	73.06	75.90	36.93	34.24	36.45	39.61	39.14	37.24	40.71	34.72	40.08	38.19	40.40	39.29
39.32	70.01	72.74	35.39	32.81	34.93	37.96	37.50	35.69	39.01	33.27	38.41	36.60	38.71	37.65
6.15	10.95	11.38	5.54	5.13	5.47	5.94	5.87	5.58	6.10	5.20	6.01	5.73	6.06	5.89

```
🔹 CHI-SQUARE TEST RESULTS:
Chi-Square Statistic: 146.40
Degrees of Freedom: 140
P-Value: 0.33844

CONCLUSION: The p-value is greater than 0.05, so we fail to reject the null hypothesis.
   → There is no significant relationship between 'Brand' and 'Customer Location'.
```

8. MACHINE LEARNING

Machine learning plays a significant role in predictive analytics and allows organizations to make evidence-based decisions. Here, we apply a Decision Tree Classification model to predict Sales Channel based on multiple features. Decision Trees have broad applications in classification problems due to their inherent clear interpretability and ability to handle both continuous and discrete features well.

Our objective is to train a Decision Tree model that is able to effectively classify channels of sales, interpret feature importance, and deliver actionable insights.

Python Libraries Used

1. **pandas** – For data handling and preprocessing
2. **numpy** – For numerical computations
3. **scikit learn –** For implementation of decision tree
4. **matplotlib & seaborn** – For visualizing feature importance

Step 1: Data Preparation

- Identify categorical and numerical features
- Perform Label Encoding for categorical variables
- Define 'Sales Channel' as the target variable

```
# MACHINE LEARNING
print("SECTION 6 - MACHINE LEARNING")
print("\nDECISION TREE CLASSIFICATION: PREDICTING SALES CHANNEL →")

# Define target variable
target = 'Sales Channel'

# Update categorical features list after dropping datetime columns
categorical_features = df.select_dtypes(include=['object']).columns.tolist()
print(f"🔄 Updated categorical features: {categorical_features}")

# Identify numerical columns
numerical_features = df.select_dtypes(include=['number']).columns.tolist()

# Select only valid features
feature_columns = categorical_features + numerical_features
print(f"✅ Selected features: {feature_columns}")

# Label Encoding for categorical variables
label_encoders = {}
for col in categorical_features:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le  # Save encoder for future decoding

# Encode target variable
target_encoder = LabelEncoder()
df[target] = target_encoder.fit_transform(df[target])

# Select features and target
X = df.drop(columns=[target])
y = df[target]
```

Step 2: Model Training & Prediction

- Split data into training (80%) and testing (20%) using train_test_split()
- Train a Decision Tree Classifier using the Entropy criterion
- Fit the model on the training data
- Predict 'Sales Channel' on the test set

```
# Split into training & testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Train Decision Tree Classifiers
entropy_tree = DecisionTreeClassifier(criterion='entropy', random_state=42)
entropy_tree.fit(X_train, y_train)

# Predictions
y_pred_entropy = entropy_tree.predict(X_test)
```

Step 3: Model Evaluation

- Compute accuracy score
- Generate a Confusion Matrix and Classification Report (precision, recall, F1-score)

```
# Model Performance
accuracy_entropy = accuracy_score(y_test, y_pred_entropy)

# Confusion Matrix & Classification Report
conf_matrix_entropy = confusion_matrix(y_test, y_pred_entropy)

target_labels = list(map(str, target_encoder.classes_))
report_entropy = classification_report(y_test, y_pred_entropy, target_names=target_labels)
```

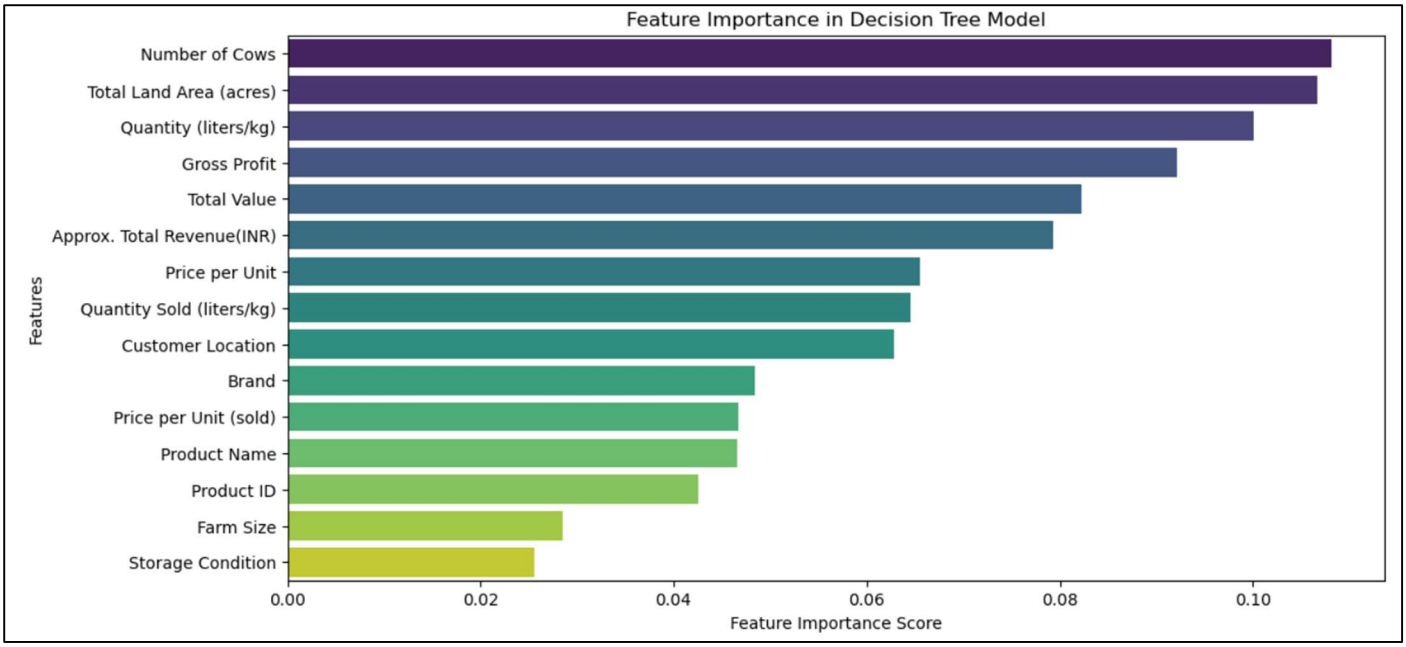
Step 4: Feature Importance & Interpretation

- Extract feature importance scores from the Decision Tree model
- Visualize feature importance using a bar plot
- Identify key features influencing sales channel prediction for better decision-making

```
# Displaying Results
print("\nDECISION TREE USING ENTROPY:")
print(f"Accuracy: {accuracy_entropy:.4f}")
print("Confusion Matrix:")
print(conf_matrix_entropy)
print("\nClassification Report:")
print(report_entropy)

# Display Feature Importance Graph
feature_importances = pd.Series(entropy_tree.feature_importances_, index=X.columns)
sorted_features = feature_importances.sort_values(ascending=False)

plt.figure(figsize=(12, 6))
sns.barplot(x=sorted_features, y=sorted_features.index, palette="viridis")
plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Feature Importance in Decision Tree Model")
plt.show()

# Print Interpretation Below the Graph
print("\nInterpretation of Feature Importance Graph:")
print("Features at the top have the highest importance, meaning they strongly influence the sales channel prediction.")
print("Features at the bottom contribute the least and may have minimal impact on the decision-making process.")
```

OUTPUT –

```
DECISION TREE CLASSIFICATION: PREDICTING SALES CHANNEL →
🔴 Dropping datetime columns: ['Date', 'Production Date', 'Expiration Date']
🔄 Updated categorical features: ['Farm Size', 'Product Name', 'Brand', 'Storage Condition', 'Customer Location', 'Sales Channel']
✅ Selected features: ['Farm Size', 'Product Name', 'Brand', 'Storage Condition', 'Customer Location', 'Sales Channel', 'Total Land Area (acres)', 'Number of Cows', 'Product ID', 'Quantit
y (liters/kg)', 'Price per Unit', 'Total Value', 'Quantity Sold (liters/kg)', 'Price per Unit (sold)', 'Approx. Total Revenue(INR)', 'Gross Profit']

🔹 DECISION TREE USING ENTROPY:
✅ Accuracy: 0.3183
📊 Confusion Matrix:
[[ 81  75 106]
 [ 94  95  90]
 [ 98  96  85]]

📋 Classification Report:
              precision    recall  f1-score   support

           0       0.30      0.31      0.30       262
           1       0.36      0.34      0.35       279
           2       0.30      0.30      0.30       279

    accuracy                           0.32       820
   macro avg       0.32      0.32      0.32       820
weighted avg       0.32      0.32      0.32       820
```



📌 Interpretation of Feature Importance Graph:

1. Features at the top have the highest importance, meaning they strongly influence the sales channel prediction.
2. Features at the bottom contribute the least and may have minimal impact on the decision-making process.

9. CONCLUSION & REFERENCES

This project integrates the dairy sector with data analytics, involving Python programming, machine learning, statistical methods, time series analysis, financial analysis, and categorical data analysis. The research uncovers significant trends in production, sales, and profitability, giving stakeholders practical insights.

Time series analysis identifies changing market trends, while machine learning improves predictive modelling for sales and production predictions. Financial analysis analyses profitability trends, and categorical data analysis informs customer preference. These insights enable informed decision-making for dairy producers, companies, and policymakers.

Although effective, the research only reflects the range of the dataset and does not factor in factors outside the system, such as policy changes or changes in global markets. However, it establishes the worth of data-driven practices in maximizing supply chain efficiency and price. In the future, researchers can add more to the dataset to maximize predictability so that decision-making is further maximized in the dairy industry.

This dairy dataset was sourced from Kaggle, and the link is provided below.

https://www.kaggle.com/datasets/suraj520/dairy-goods-sales-dataset