AUTOMOBILE SALES DATA ANALYSIS

Taru Bhargava

CONTENTS

INTRODUCTION

1.1 THE AUTOMOBILE INDUSTRY

The automotive industry is a vital part of the world economy, supplying pivotal transportation solutions to businesses and individuals. It contributes immensely to economic development, job creation, and technological advancements. Progress in manufacturing, automation, and electric vehicle (EV) technology has revolutionized the industry over the decades, enhancing efficiency, safety, and sustainability. With growing consumer demands and changing regulatory regimes, sustaining production efficiency and sustainability continue to be a major challenge for the industry.

India is a large automotive market in the world with a robust manufacturer-servicer-dealer ecosystem. Large players like Maruti Suzuki, Tata Motors, Mahindra, and Hyundai own the market, along with upcoming EV players. Government policies like subsidies for electric cars and emissions norms have had a major impact on the growth and development of the industry. Increased demand for fuel-efficient, electric, and connected vehicles has further influenced manufacturing and customer choices.

Even with constant growth, the auto industry has several challenges it has to navigate, such as fluctuating raw material prices, supply chain volatility, and ecological issues. Automakers contend with increasing production costs, changing safety regulations, and shifting consumer demands. The trend toward electric and hybrid cars is also an opportunity and a challenge for conventional car manufacturers. Utilization of data analytics, predictive modelling, and financial analysis is critical in addressing these challenges and ensuring long-term industry stability.

1.2 PURPOSE OF OUR STUDY

The purpose of this study is to examine different facets of the automobile sector, including sales patterns, pricing trends, and demand in the market, through significant dataset attributes like **‘quantityordered’, ‘sales’, ‘orderdate’, ‘country’, ‘dealsize’**, etc. Through these aspects, this project hopes to determine problems and prospects of the industry. The study additionally aims to make available data-oriented findings that would be useful for stakeholders such as manufacturers, distributors, and the government to enable efficiency, profitability, and sustainability in the auto industry.

Using these methods of analysis, the project will concentrate on predictive modelling, the assessment of financial performance, and categorical data analysis in the auto industry. Using this multi-aspect approach means that market trends are fully comprehended from various angles, so that decisions made are data-informed for efficient operations and future growth.

This being an integrated study will incorporate the following concepts - **Python programming, Sampling methods, Inferential statistics, Count & categorical data analysis, Machine learning, Time series analysis & Financial data analysis.**

1.3 SCOPE OF THE PROJECT

The project is centred on the study of major dynamics of the automotive sector, such as order trends, price fluctuations, customer buying behaviour, and profitability. Through data analysis, the study seeks to reveal insightful findings that will guide stakeholders in making informed decisions.

The scope also covers the use of machine learning algorithms for predictive modelling, statistical methods for data verification, and time series analysis for monitoring changes in the market over time. Financial analytics will be utilized to analyze pricing patterns and profitability, giving a complete economic overview of the industry.

In spite of its comprehensive approach, the project is limited by the dataset at hand and does not consider external variables like changes in government policy or world economic trends. The results will mostly be relevant within the framework of the provided dataset and will be useful for scholarly research and strategic decision-making within established boundaries.

1.4 SYSTEM REQUIREMENTS

1. Operating System - Windows-10
2. Toolkit - Jupyter Notebook
3. Platform - Python & its libraries
4. Processor - Intel CORE i7

1.5 DATA DESCRIPTION

This dataset offers an in-depth look at the automobile sales industry, covering important features like order information, prices, product lines, and customer data. It involves information about various automobile products, such as product line, quantity ordered, unit price, and total revenue. The dataset also captures customer demographics, geographic locations, and transaction dates, which are useful for segmenting the market and understanding trends.

Additionally, the dataset contains attributes associated with sales performance, order status, and deal size that aid in assessing market demand, profitability, and operational efficiency. With the incorporation of both numerical and categorical variables, the dataset is capable of supporting an extensive array of analytical tasks, such as sales forecasting, demand analysis, and price optimization.

```
Dataset Summary:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2747 entries, 0 to 2746
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   ORDERNUMBER           2747 non-null   int64
 1   QUANTITYORDERED       2747 non-null   int64
 2   PRICEEACH             2747 non-null   float64
 3   ORDERLINENUMBER       2747 non-null   int64
 4   SALES                 2747 non-null   float64
 5   ORDERDATE             2747 non-null   object
 6   DAYS_SINCE_LASTORDER  2747 non-null   int64
 7   STATUS                2747 non-null   object
 8   PRODUCTLINE           2747 non-null   object
 9   ACTUAL PRICE          2747 non-null   int64
 10  PRODUCTCODE           2747 non-null   object
 11  CUSTOMERNAME          2747 non-null   object
 12  PHONE                 2747 non-null   object
 13  ADDRESSLINE1          2747 non-null   object
 14  CITY                  2747 non-null   object
 15  POSTALCODE            2747 non-null   object
 16  COUNTRY               2747 non-null   object
 17  CONTACTLASTNAME       2747 non-null   object
 18  CONTACTFIRSTNAME      2747 non-null   object
 19  DEALSIZE              2747 non-null   object
dtypes: float64(2), int64(5), object(13)
memory usage: 429.3+ KB
```

DATA CLEANING AND PREPARATION

Data cleaning is a fundamental step in preparing a dataset for meaningful analysis. This process ensures consistency, accuracy, and reliability by addressing missing values, eliminating duplicate entries, converting date columns to a proper format, and detecting/removing outliers. These refinements improve dataset quality, making it more suitable for statistical analysis, visualization, and machine learning models.

Step 1 – Removing Duplicate Entries

Duplicate rows can inflate certain patterns, leading to incorrect analysis. Using **Pandas' duplicated()** and **drop_duplicates()** functions, such redundant records are identified and removed, ensuring that each entry in the dataset represents unique information. This step maintains data authenticity and prevents over-representation of certain categories.

```
#Removing Duplicate Records
print("\nELIMINATING DUPLICATES →")

# Identify and drop duplicate rows
duplicates = df.duplicated().sum()
if duplicates > 0:
    df.drop_duplicates(inplace=True)
    print(f"\nRemoved {duplicates} duplicate entries.")
else:
    print("\nNo duplicate entries found.")
```

Step 2 – Handling Missing Data

Missing values may occur due to data entry errors, incomplete records, or unavailable information. Unhandled missing data can lead to inaccurate conclusions. Using **Pandas**, missing values are identified, and different strategies are applied based on the data type:

- **Numerical columns** are filled with their median values to maintain the overall distribution.
- **Categorical columns** are filled with their mode (most frequently occurring value) to preserve logical consistency.
  This ensures the dataset remains comprehensive without introducing biases or misleading gaps.

```
#Addressing Missing Data
print("\n DEALING WITH MISSING DATA →")

# Inspect for missing data
missing_values = df.isnull().sum()
total_missing = missing_values.sum()

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

# Fill missing data with appropriate 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)

# Confirm that missing values have been handled
    print("\nMissing values have been replaced.")
else:
    print("\nNo missing values detected in the dataset.")
```

Step 3 – Converting Date Columns

Time-related data is often stored as plain text, making it difficult to conduct chronological analysis. The pd.to_datetime() function is used to convert the **ORDERDATE** column into a standardized datetime format. This transformation allows seamless sorting, filtering, and time-based computations, making the dataset more efficient for trend analysis and forecasting.

```
#Formatting Date Columns
print("\nSTANDARDIZING DATE COLUMNS →")
df['ORDERDATE'] = pd.to_datetime(df['ORDERDATE'], format='%d-%m-%Y', errors='coerce')
print("\nDate columns have been standardized to datetime format.")
```

Step 4 – Outlier Detection and Removal

Outliers, or extreme values, can distort insights and impact model performance. These anomalies may arise due to measurement errors, one-time events, or rare instances.

Using **the Interquartile Range (IQR) method**, numerical outliers are detected and removed. The IQR technique calculates the range between the first quartile (Q1) and third quartile (Q3), setting thresholds beyond which values are considered outliers.

Any value below **Q1 - 1.5 × IQR** or above **Q3 + 1.5 × IQR** is considered an outlier and removed.
This filtering ensures that the dataset represents typical trends without being skewed by extreme variations.

```
#Outlier Detection and Removal
print("\nDETECTING AND REMOVING OUTLIERS →")

# Choose a numeric column for outlier detection (e.g., 'Approx. Total Revenue(INR)')
column_of_interest = 'SALES'

# Record the original number of rows before removing outliers
original_size = df.shape[0]

# Perform outlier detection using the IQR method
Q1 = df[column_of_interest].quantile(0.25)
Q3 = df[column_of_interest].quantile(0.75)
IQR = Q3 - Q1

# Set the thresholds for identifying outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out outliers from the dataset
df = df[(df[column_of_interest] >= lower_bound) & (df[column_of_interest] <= upper_bound)]

# Calculate how many rows were removed as outliers
outliers_removed = original_size - df.shape[0]

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

SAMPLING OF DATA

Sampling is a core technique employed in data analysis to obtain a smaller subset of data that is representative of the entire data set. This provides effective computations and credible insights without the need to process all records. In this example, three types of sampling are employed: Simple Random Sampling, Systematic Sampling, and Stratified Sampling. The efficiency of each is then compared by computing their variance.

Step 1: Simple Random Sampling

In this approach, 100 records are randomly selected from the dataset using **Pandas' .sample() function**. This ensures that every record has an equal probability of being chosen, making it a fair and unbiased selection method.

```
# 1. Sample Random Sampling
simple_random_sample = df.sample(n=100, random_state=42)
```

Step 2: Systematic Sampling

Systematic sampling selects records at fixed intervals across the dataset. The interval is determined by dividing the total dataset size by 100, ensuring an evenly distributed sample. This method is beneficial when the data is evenly spread and does not contain hidden patterns.

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

Step 3: Stratified Sampling

Stratified sampling is performed based on the **'PRODUCTLINE'** category, ensuring that each category is proportionally represented in the sample. The dataset is divided into groups (strata), and a random subset is taken from each group using **Pandas' .groupby() and .sample()** functions. If the total selected sample size exceeds 100, an additional random selection is applied to maintain consistency.

```
# 3. Stratified Sampling Based on 'PRODUCTLINE' Category
stratified_sample = df.groupby('PRODUCTLINE', 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
```

Step 4: Comparing Variance Across Sampling Techniques

To determine the effectiveness of each sampling method, variance is calculated for the **'SALES'** column in all three sampled datasets. Lower variance indicates that the sample better represents the overall dataset. **NumPy's np.var() function** is used to compute variance with **Bessel's correction (ddof=1)**, which provides an unbiased estimate.

```
# 4. Comparison of Variance Across Sampling Methods
print("\nCOMPARING VARIANCES →")

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 for Different Sampling Approaches:")
print(f"Variance for Simple Random Sampling: {var_simple:.4f}")
print(f"Variance for Systematic Sampling: {var_systematic:.4f}")
print(f"Variance for Stratified Sampling: {var_stratified:.4f}")
```

Step 5: Identifying the Most Reliable Sampling Technique

The method with the lowest variance is identified as the most optimal sampling technique. A comparison is performed, and the approach yielding the least variation is deemed the best for maintaining data consistency.

```python
# 5. Determining the Most Efficient Sampling Method
print("\nOPTIMAL 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"\nOptimal Sampling Method (Minimum Variance): {best_method}")
```

OUTPUT –

```
COMPARING VARIANCES →

Variance for Different Sampling Approaches:
Variance for Simple Random Sampling: 2470035.4851
Variance for Systematic Sampling: 2713089.3828
Variance for Stratified Sampling: 1969181.7368

OPTIMAL SAMPLING TECHNIQUE →

Optimal Sampling Method (Minimum Variance): Stratified Sampling
```

INFERENTIAL STATISTICS

Inferential statistics aid us in reaching conclusions regarding a population based on sample information. One of the most important techniques in inferential statistics is **hypothesis testing**, with which we are able to test hypotheses and assess whether patterns found are statistically significant. Here, we conduct an **ANOVA (Analysis of Variance) test** to ensure whether average sales significantly vary for various countries or not.

Step 1: Formulating the Hypotheses

To test whether sales figures significantly vary across countries, two hypotheses are established:

- **Null Hypothesis (H0):** The mean sales are the same for all countries, indicating no significant difference.
- **Alternative Hypothesis (H1):** At least one country has a significantly different mean sales value.

The dataset is grouped by the **COUNTRY** column, and sales data is extracted for each country. We use **unique()** from **pandas** to extract the unique country names. We then use a list comprehension with **pandas** filtering and **dropna()** to create separate sales groups for each country.

```
# 1. Hypothesis Testing: ANOVA for SALES Across Different COUNTRIES
print("\nHYPOTHESIS TESTING: ANOVA TEST FOR SALES ACROSS COUNTRIES →")

# Formulating the Hypotheses
print("\nNull Hypothesis (H0): The mean SALES are the same across all COUNTRY groups.")
print("Alternative Hypothesis (H1): At least one COUNTRY group has a significantly different mean SALES.")

# Extracting distinct COUNTRY categories
countries = df['COUNTRY'].unique()
country_groups = [df[df['COUNTRY'] == country]['SALES'].dropna() for country in countries]
```

Step 2: Performing the ANOVA Test

The ANOVA test is used to compare sales across different countries. It calculates:
F-statistic: Measures variance between and within country groups.
P-value: Determines if differences are significant.
We use **stats.f_oneway()** from **scipy.stats** to perform the test:

```
# Performing the ANOVA Test
f_stat, p_value_anova = stats.f_oneway(*country_groups)

# Calculating Degrees of Freedom
df_between = len(countries) - 1
df_within = len(df) - len(countries)
```

Step 3: Creating the ANOVA Table

The ANOVA table summarizes the variance in sales using:

- **Sum of Squares (SS):** calculated using list comprehensions and math operations with **pandas.**
- **Degrees of Freedom (df):** Number of independent values in calculations.
- **Mean Squares (MS):** Average variance for each category.
- **F-Statistic & P-Value:** Key metrics to assess significance.

We construct the ANOVA table using **pd.DataFrame({...})** from pandas.

```python
# Calculating Sum of Squares
ss_between = sum([len(group) * (group.mean() - df['SALES'].mean()) ** 2 for group in country_groups])
ss_within = sum([sum((group - group.mean()) ** 2) for group in country_groups])
ss_total = ss_between + ss_within

# Calculating Mean Squares
ms_between = ss_between / df_between
ms_within = ss_within / df_within

# Creating the 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 4: Interpreting the Results

- If p-value < 0.05, the null hypothesis is rejected, meaning country-based sales differences are significant.
- If p-value ≥ 0.05, the null hypothesis is not rejected, indicating no significant variation in sales across countries.

```python
# Interpretation of Results
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 smaller than 0.05, we reject the null hypothesis.")
    print("This suggests that at least one COUNTRY has a significantly different average SALES.")
else:
    print("\nCONCLUSION → Since the p-value exceeds 0.05, we fail to reject the null hypothesis.")
    print("This means there is no significant difference in the average SALES between the COUNTRY groups.")
```

OUTPUT –

ONE WAY ANOVA TABLE

Source	Sum of Squares (SS)	Degrees of Freedom (df)	Mean Square (MS)	F-Statistic	P-Value
Between Groups	47841092.45	18	2657838.47	1.072069421	0.374456409
Within Groups	6567311748	2649	2479166.383		
Total	6615152840	2667			

```
||SECTION 3 - INFERENTIAL STATISTICS||

HYPOTHESIS TESTING: ANOVA TEST FOR SALES ACROSS COUNTRIES →

Null Hypothesis (H0): The mean SALES are the same across all COUNTRY groups.
Alternative Hypothesis (H1): At least one COUNTRY group has a significantly different mean SALES.

F-statistic = 1.0721, P-value = 0.3745

CONCLUSION → Since the p-value exceeds 0.05, we fail to reject the null hypothesis.
This means there is no significant difference in the average SALES between the COUNTRY groups.
```

COUNT AND CATEGORICAL ANALYSIS

Analysis of count and categorical data aids in gaining insight into the relationship between various categorical variables in a data set. **Chi-Square Test** for Independence is one of the most important statistical techniques applied to identify if two categorical variables are related or not. In this case, we examine the relationship between **'DEALSIZE'** and **'COUNTRY'** for testing statistical dependence.

Step 1: Defining Hypotheses

To assess whether a significant relationship exists between 'COUNTRY' and 'DEALSIZE', we establish the following hypotheses:

- **Null Hypothesis (H_0):** 'COUNTRY' and 'DEALSIZE' are independent, meaning there is no relationship between them.
- **Alternative Hypothesis (H_1):** 'COUNTRY' and 'DEALSIZE' are dependent, implying a significant relationship exists.

```
# 1. Chi-Square Test for Independence: Analyzing the Relationship Between COUNTRY and DEALSIZE
print("\nCHI-SQUARE TEST FOR INDEPENDENCE: RELATIONSHIP BETWEEN COUNTRY AND DEALSIZE →")

# Define hypotheses
print("\nHYPOTHESES FOR CHI-SQUARE TEST:")
print("Null Hypothesis (H₀): 'COUNTRY' and 'DEALSIZE' are independent (no significant relationship).")
print("Alternative Hypothesis (H₁): 'COUNTRY' and 'DEALSIZE' are dependent (significantly related).")
```

Step 2: Performing the Chi-Square Test

The Chi-Square test measures the association between 'COUNTRY' and 'DEALSIZE'. It computes:

- **Chi-Square Statistic (χ^2):** Measures the discrepancy between observed and expected frequencies.
- **Degrees of Freedom (df):** Determines the number of independent values in the test.
- **P-Value:** Assesses statistical significance.

To evaluate independence, expected frequencies are calculated under the assumption that 'COUNTRY' and 'DEALSIZE' are unrelated. These expected values are then compared with the actual observed frequencies using the Chi-Square formula.

A contingency table is created to display the frequency distribution of these two categorical variables using pd.crosstab() from the **pandas** library. This table displays the frequency of each combination of 'COUNTRY' and 'DEALSIZE'.
We then apply **stats.chi2_contingency(contingency_table)** from **scipy.stats.**

```
# Creating a contingency table for 'COUNTRY' and 'DEALSIZE'
contingency_table = pd.crosstab(df['COUNTRY'], df['DEALSIZE'])


# Display the observed frequency table
print("\n 🔹 CONTINGENCY TABLE (OBSERVED FREQUENCIES):")
print(contingency_table)

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

# Convert expected frequencies into a DataFrame for clarity
expected_df = pd.DataFrame(expected, index=contingency_table.index, columns=contingency_table.columns)

# Show the expected frequency table
print("\n 🔹 EXPECTED FREQUENCIES (UNDER NULL HYPOTHESIS):")
print(expected_df)
```

Step 3: Interpretation of Results

- **If p-value < 0.05:** The null hypothesis is rejected, indicating a statistically significant relationship between ‘COUNTRY’ and ‘DEALSIZE’.
- **If p-value ≥ 0.05:** We fail to reject the null hypothesis, suggesting no significant relationship between these categorical variables.

```
# Decision-making based on significance level (α = 0.05)
alpha = 0.05
if p_value < alpha:
    print("\nCONCLUSION: The p-value is below 0.05, so we reject the null hypothesis.")
    print("   → There is evidence of a significant relationship between 'COUNTRY' and 'DEALSIZE'.")
else:
    print("\nCONCLUSION: The p-value is above 0.05, so we fail to reject the null hypothesis.")
    print("   → No significant relationship detected between 'COUNTRY' and 'DEALSIZE'.")
```

OUTPUT –

```
CHI-SQUARE TEST FOR INDEPENDENCE: RELATIONSHIP BETWEEN COUNTRY AND DEALSIZE →

📌 HYPOTHESES FOR CHI-SQUARE TEST:
🔹 Null Hypothesis (H₀): 'COUNTRY' and 'DEALSIZE' are independent (no significant relationship).
🔹 Alternative Hypothesis (H₁): 'COUNTRY' and 'DEALSIZE' are dependent (significantly related).
```

🔹 CONTINGENCY TABLE (OBSERVED FREQUENCIES):

DEALSIZE	Large	Medium	Small
COUNTRY			
Australia	0	86	92
Austria	2	29	22
Belgium	0	15	18
Canada	0	33	36
Denmark	2	31	26
Finland	4	46	41
France	8	149	144
Germany	3	30	28
Ireland	1	6	8
Italy	2	44	62
Japan	2	21	28
Norway	2	42	38
Philippines	1	14	11
Singapore	1	38	37
Spain	9	171	154
Sweden	2	31	24
Switzerland	0	22	9
UK	2	71	69
USA	32	470	399

🔹 EXPECTED FREQUENCIES (UNDER NULL HYPOTHESIS):

DEALSIZE	Large	Medium	Small
COUNTRY			
Australia	4.870315	90.000750	83.128936
Austria	1.450150	26.797976	24.751874
Belgium	0.902924	16.685532	15.411544
Canada	1.887931	34.887931	32.224138
Denmark	1.614318	29.831709	27.553973
Finland	2.489880	46.011619	42.498501
France	8.235757	152.192279	140.571964
Germany	1.669040	30.842954	28.488006
Ireland	0.410420	7.584333	7.005247
Italy	2.955022	54.607196	50.437781
Japan	1.395427	25.786732	23.817841
Norway	2.243628	41.461019	38.295352
Philippines	0.711394	13.146177	12.142429
Singapore	2.079460	38.427286	35.493253
Spain	9.138681	168.877811	155.983508
Sweden	1.559595	28.820465	26.619940
Switzerland	0.848201	15.674288	14.477511
UK	3.885307	71.798351	66.316342
USA	24.652549	455.565592	420.781859

🔹 CHI-SQUARE TEST RESULTS:
Chi-Square Statistic: 33.12
Degrees of Freedom: 36
P-Value: 0.60607

CONCLUSION: The p-value is above 0.05, so we fail to reject the null hypothesis.
→ No significant relationship detected between 'COUNTRY' and 'DEALSIZE'.

MACHINE LEARNING

Machine learning algorithms assist in making data-based predictions by discovering patterns in data, allowing businesses to make sound decisions. In this analysis, we use a **K-Nearest Neighbours (KNN)** Classifier, a popular supervised learning algorithm, to forecast **'DEALSIZE'** based on several attributes. The model learns patterns from historical data and predicts future instances based on their similarity to past data points. This method is especially valuable in business analytics, where precise classification is able to inform strategic planning and maximize operations.

Step 1: Data Preparation & Preprocessing

Before training the model, the dataset must be cleaned by performing the following steps:

- Removing Irrelevant Columns using **df.drop()** from **pandas**.
- Defining the Target Variable
- Handling Missing Values using **df.dropna()** from **pandas**

```
# Remove irrelevant columns
df = df.drop(columns=['ORDERDATE'])

# Specify the target variable
target = 'DEALSIZE'

# Eliminate rows where the target variable has missing values
df = df.dropna(subset=[target])

# Drop the 'Year-Month' column if it exists in the dataset
if 'Year-Month' in df.columns:
    df = df.drop(columns=['Year-Month'])
```

Step 2: Data Encoding and Preparation

Once the data is cleaned, categorical features must be converted into numerical format, and the dataset should be prepared for model training.

Each categorical feature is label encoded using **LabelEncoder()** from **sklearn.preprocessing**. The dataset is split into training and testing sets

from **sklearn.model_selection**, with 80-20 ratio and stratification. Feature scaling is done using **StandardScaler()** from **sklearn.preprocessing**:

```
# Identify categorical features, excluding the target variable
categorical_features = df.select_dtypes(include=['object']).columns.tolist()
if target in categorical_features:
    categorical_features.remove(target)  # Exclude the target variable from categorical features

print(f"📌 Recognized Categorical Features: {categorical_features}\n")

# Convert categorical variables into numerical form using Label Encoding
label_encoders = {}
for col in categorical_features:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le  # Save the encoder for potential future decoding

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

# Define the feature matrix and target vector
X = df.drop(columns=[target])  # Feature variables (now all numerical)
y = df[target]  # Target variable

# Split the dataset into training and testing sets (80-20 ratio)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Standardize the feature values for consistent scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 3: Choosing the Optimal k for KNN

The **Elbow Method** is used to determine the optimal number of neighbors (*k*) for the KNN model.
Several k values (1 to 20) are tested, and error rates are calculated for each.
A plot is generated to visualize the error trend, helping identify the point where adding more neighbours does not significantly reduce errors.

```
# --- Determining the Optimal k Value using the Elbow Method ---
error_rates = []
k_values = range(1, 21)  # Evaluate k values from 1 to 20

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    error_rates.append(1 - accuracy_score(y_test, y_pred))  # Compute error rate as (1 - accuracy)

# Visualizing the Elbow Curve
plt.figure(figsize=(10, 5))
sns.lineplot(x=k_values, y=error_rates, marker='o', linestyle='dashed', color='b')
plt.xlabel("Number of Neighbors (k)")
plt.ylabel("Error Rate")
plt.title("Elbow Method for Choosing Optimal k")
plt.xticks(k_values)
plt.show()

# Selecting the k value that yields the lowest error before stabilizing
optimal_k = k_values[error_rates.index(min(error_rates))]

print("\n 🔹 SELECTED PARAMETERS FOR KNN:")
print(f"   - Optimal Number of Neighbors (k) determined using the Elbow Method: {optimal_k}")
print(f"   - Distance Metric: Minkowski (Equivalent to Euclidean Distance when p=2)\n")
```

Step 4: Model Training & Evaluation

The KNN model is trained using the optimal k value identified. Predictions are made on the test dataset. Model performance is assessed using:

- **Accuracy Score**: Measures the proportion of correctly classified instances.
- **Confusion Matrix**: Evaluates true positives, false positives, true negatives, and false negatives.
- **Classification Report**: Provides precision, recall, and F1-score for each category of 'DEALSIZE'.

Predictions are made on the test set, and performance is evaluated.

```
# --- Train the KNN Classifier using the Optimal k Value ---
knn_model = KNeighborsClassifier(n_neighbors=optimal_k, metric='minkowski')
knn_model.fit(X_train_scaled, y_train)

# Make predictions on the test dataset
y_pred_knn = knn_model.predict(X_test_scaled)

# --- Evaluate the Model's Performance ---
accuracy_knn = accuracy_score(y_test, y_pred_knn)
conf_matrix_knn = confusion_matrix(y_test, y_pred_knn)
report_knn = classification_report(y_test, y_pred_knn, target_names=target_encoder.classes_)
```

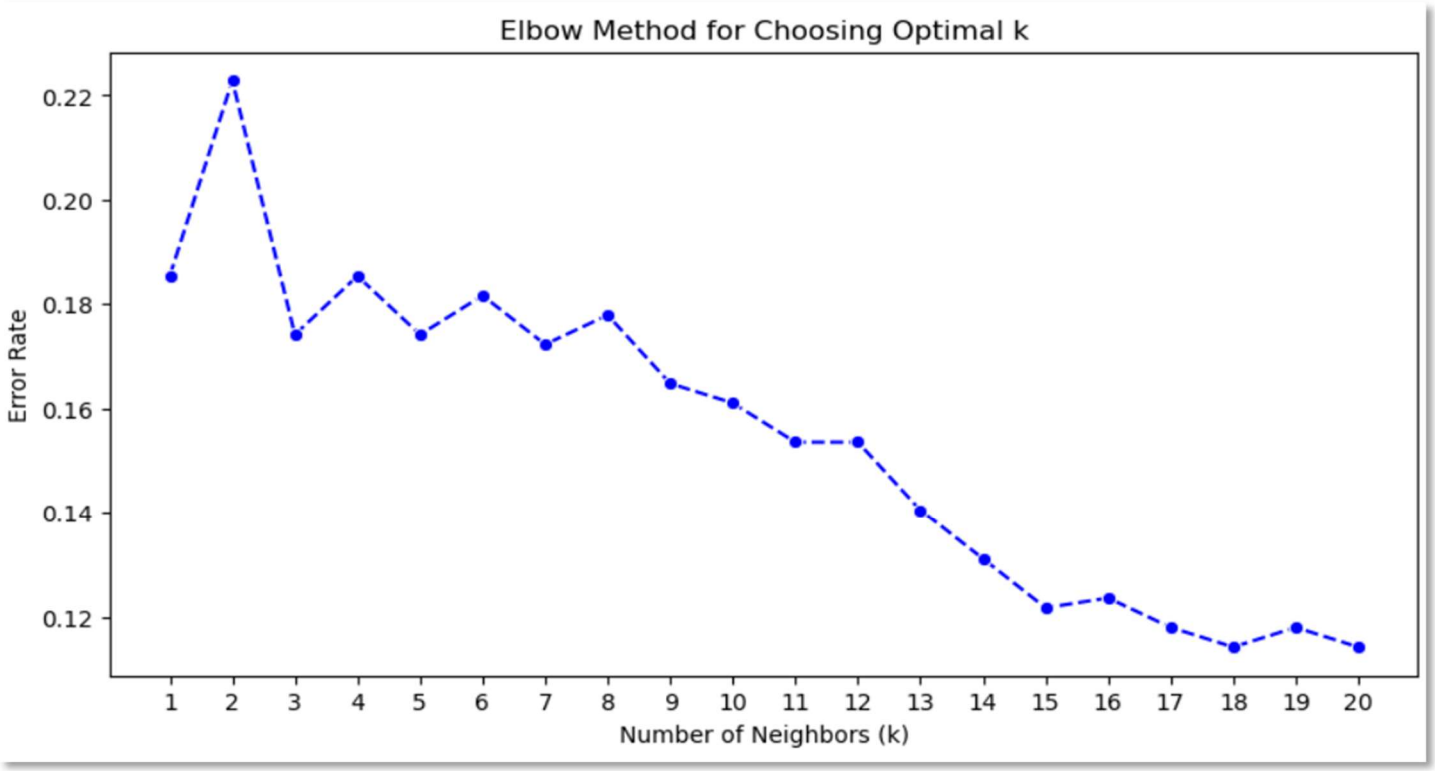
Step 5: Interpretation of Results

- If the accuracy score is high, it indicates that the model effectively predicts 'DEALSIZE'.
- A well-balanced confusion matrix suggests that misclassifications are minimal.
- The classification report helps in understanding how well each class is predicted, assisting in refining the model if necessary.

```
# --- Display Detailed Model Performance Metrics ---
print(" 🔹 **MODEL PERFORMANCE: K-Nearest Neighbors**")
print(f"Accuracy: {accuracy_knn:.4f}")
print("\nConfusion Matrix:")
print(conf_matrix_knn)
print("\nClassification Report:")
print(report_knn)

print("\nINTERPRETATION OF RESULTS:")

# Explanation of the Accuracy Score
print(f"The model achieved an accuracy of {accuracy_knn:.4f}, indicating that it correctly classified {(accuracy_knn * 100):.2f}% of the test instances.")
```

OUTPUT –

```
♦ MODEL: K-Nearest Neighbors (KNN) Classifier
♦ GOAL: Predicting 'DEALSIZE' based on available attributes

Recognized Categorical Features: ['STATUS', 'PRODUCTLINE', 'PRODUCTCODE', 'CUSTOMERNAME', 'PHONE', 'ADDRESSLINE1', 'CITY', 'POSTALCODE', 'COUNTRY', 'CONTACTLASTNAME', 'CONTACTFIRSTNAME']
```




```
🔹 SELECTED PARAMETERS FOR KNN:
   - Optimal Number of Neighbors (k) determined using the Elbow Method: 18
   - Distance Metric: Minkowski (Equivalent to Euclidean Distance when p=2)

🔹 **MODEL PERFORMANCE: K-Nearest Neighbors**
Accuracy: 0.8858

Confusion Matrix:
[[  1  14   0]
 [  0 244  26]
 [  0  21 228]]

Classification Report:
              precision    recall  f1-score   support

       Large       1.00      0.07      0.12        15
      Medium       0.87      0.90      0.89       270
       Small       0.90      0.92      0.91       249

    accuracy                           0.89       534
   macro avg       0.92      0.63      0.64       534
weighted avg       0.89      0.89      0.88       534


INTERPRETATION OF RESULTS:
The model achieved an accuracy of 0.8858, indicating that it correctly classified 88.58% of the test instances.
```

TIME SERIES ANALYSIS

Here we examine past sales patterns and project future revenue through time series analysis. Using **Least Squares**, **Moving Average**, and **Semi-Average** Trend Analysis, we ascertain trends from monthly sales figures. The most accurate technique is chosen based on precision, guaranteeing effective predictions for the next **12 months**, which informs strategic business decisions.

Step 1: Data Preparation

- **Convert 'ORDERDATE' to Date Time format** using **pd.to_datetime()** function from **pandas**.
- **Aggregate sales on a monthly basis** using **df.resample()['SALES'].sum()** from **pandas**.
- **Set 'ORDERDATE' as the index** using **set_index()** from pandas
- **Create a time variable (TIME)** using **np.arange()** from **NumPy**.

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

# Aggregate sales data on a monthly basis instead of daily
df_time = df.resample('M', on='ORDERDATE')['SALES'].sum().reset_index()

# Set 'ORDERDATE' as the index for the dataset
df_time.set_index('ORDERDATE', inplace=True)

# Generate a time variable (X)
df_time['TIME'] = np.arange(1, len(df_time) + 1)
```

Step 2: Trend Analysis Using Three Methods

2.1 Least Squares Trend Analysis (OLS Regression)

- An **Ordinary Least Squares (OLS) regression model** is fitted using **TIME** as the independent variable and **SALES** as the dependent variable using **sm.OLS(...).fit()** from **statsmodels.api**.
- The **predicted sales trend** is calculated using **.predict()** and stored and the **sum of trend values** is computed for comparison.

```
# ---------------- Least Squares Trend Analysis ----------------
X = sm.add_constant(df_time['TIME'])  # Add constant for intercept in the model
model = sm.OLS(df_time['SALES'], X).fit()  # Fit the Ordinary Least Squares (OLS) model
df_time['Least Squares Trend'] = model.predict(X)  # Calculate trend using OLS model
sum_least_squares = df_time['Least Squares Trend'].sum()  # Summing the trend values
```

2.2 Moving Average Trend Analysis

- A **3-month moving average** is calculated using **.rolling().mean()** from **pandas**.
- Missing values at the beginning or end of the series are filled with original sales values.

```
# ---------------- Moving Average Trend Analysis ----------------
df_time['Moving Average Trend'] = (
    df_time['SALES']
    .rolling(window=3, center=True)  # Apply a rolling window of 3 months
    .mean()
)

# Fill any missing values at the beginning and end of the series with the original sales values
df_time['Moving Average Trend'].fillna(df_time['SALES'], inplace=True)

sum_moving_average = df_time['Moving Average Trend'].sum()  # Summing the moving average trend values
```

2.3 Semi-Average Trend Analysis

- The dataset is split into two halves using using **iloc[]** , and average sales for each half are computed using **.mean()** from **pandas**.
- A linear trend equation is derived using the slope-intercept formula.
- The semi-average trend values are calculated and summed.

```
# ---------- Semi-Average Trend Analysis ----------
mid = len(df_time) // 2  # Split the data into two halves
first_half = df_time.iloc[:mid]
second_half = df_time.iloc[mid:]

# Calculate averages for both halves
avg_1 = first_half['SALES'].mean()
avg_2 = second_half['SALES'].mean()

# Calculate the slope and intercept for the semi-average method
t1 = mid // 2
t2 = mid + (len(df_time) - mid) // 2
b = (avg_2 - avg_1) / (t2 - t1)  # Slope
a = avg_1 - (b * t1)  # Intercept
df_time['Semi-Average Trend'] = a + b * df_time['TIME']  # Calculate the trend line
sum_semi_average = df_time['Semi-Average Trend'].sum()  # Summing the semi-average trend values
```

Step 3: Trend Analysis Results

- The total sum of trend values is displayed using **.sum()** from **pandas**
- The actual sum of revenue is compared against these computed trends.
- The accuracy of each trend estimation method is measured by calculating the absolute difference from actual revenue.

```
# ---------- 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}")
```

Step 4: Selecting the Best Trend Calculation Method

- The method with the smallest deviation from actual revenue is selected as the best model.
- The best trend estimation approach is identified among Least Squares, Moving Average, and Semi-Average methods.

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

# ---------- Comparing Accuracy of Different Trend Methods ----------
print("\nCOMPARING TREND ACCURACY WITH ACTUAL REVENUE →")
diff_least_squares = abs(sum_actual - sum_least_squares)  # Difference from actual revenue for Least Squares
diff_moving_average = abs(sum_actual - sum_moving_average)  # Difference from actual revenue for Moving Average
diff_semi_average = abs(sum_actual - sum_semi_average)  # Difference from actual revenue for Semi-Average

# Display the differences for each method
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}")

# ---------- Selecting the Best Trend Calculation Method ----------
print("\nBEST TREND CALCULATION METHOD SELECTION →")

# Choose the method with the smallest difference as the best method
best_method = min(diff_least_squares, diff_moving_average, diff_semi_average)

# Determine the best method based on the smallest difference
if best_method == diff_least_squares:
    best_trend = "Least Squares Method"
elif best_method == diff_moving_average:
    best_trend = "Moving Average Method"
else:
    best_trend = "Semi-Average Method"

print(f"\n🎯 Best Method for Trend Estimation: {best_trend}")
```

Step 5: Forecasting Future Sales

The next 12 months of sales are forecasted using the best trend method. The forecasted revenue for each month is displayed using **pd.DataFrame()** and **pd.date_range()** from **pandas**.

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

# Generate forecast for the next 12 months based on the selected best method
forecast_time = np.arange(len(df_time) + 1, len(df_time) + 13)  # Time steps for the next 12 months

if best_trend == "Least Squares Method":
    forecast_values = model.params[0] + model.params[1] * forecast_time  # Use OLS coefficients for prediction
elif best_trend == "Moving Average Method":
    forecast_values = [df_time['Moving Average Trend'].iloc[-1]] * 12  # Repeat the last known moving average
elif best_trend == "Semi-Average Method":
    forecast_values = a + b * forecast_time  # Use semi-average formula for forecasting

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

Step 6: Visualizing Sales Trends

The following trends are plotted for comparison : Actual Monthly Sales & Least Squares Trend Line, using **plt.plot()** from **matplotlib.pyplot**.

```
# ---------------- Visualizing the Trends ----------------
print("\nVISUALIZING THE TRENDS →")

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

# Plot actual sales data
plt.plot(df_time.index, df_time['SALES'], label='Actual Monthly Sales', color='purple', marker='o', linestyle='dotted', linewidth=1.5)

# Plot the Least Squares trend line
plt.plot(df_time.index, df_time['Least Squares Trend'], label='Least Squares Trend', color='black', linestyle='solid', linewidth=3)

# Add title and labels
plt.title('Sales Trend Analysis with Different Methods')
plt.xlabel('DATE')
plt.ylabel('SALES')
plt.legend()
plt.grid(True, alpha=0.5)

# Show the plot
plt.show()
```

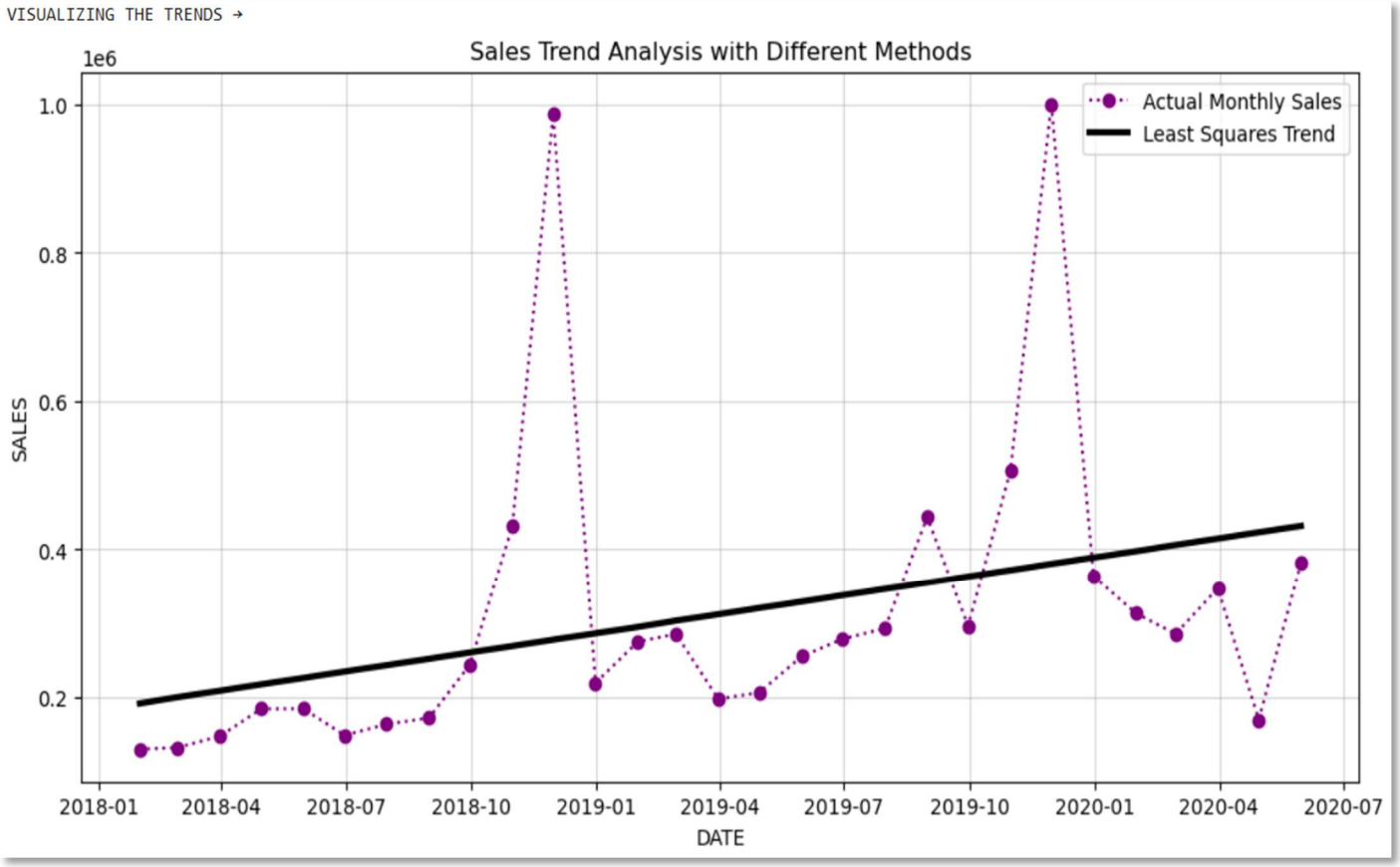
OUTPUT –

```
TREND ANALYSIS RESULTS →

 🔹  Sum of Trend Values for Each Method:
Least Squares Method: 9037308.47
Moving Average Method: 9107410.69
Semi-Average Method: 9180624.64

 🔹  Actual Total Sum of Revenue:
Actual Revenue Sum: 9037308.47

COMPARING TREND ACCURACY WITH ACTUAL REVENUE →

 🔹  Difference from Actual Revenue Sum:
Least Squares Method: 0.00
Moving Average Method: 70102.22
Semi-Average Method: 143316.17

BEST TREND CALCULATION METHOD SELECTION →

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

FORECASTING FOR THE NEXT 12 MONTHS →

	Forecasted Revenue
2020-06-30	440418.549483
2020-07-31	449004.364345
2020-08-31	457590.179207
2020-09-30	466175.994069
2020-10-31	474761.808931
2020-11-30	483347.623793
2020-12-31	491933.438655
2021-01-31	500519.253517
2021-02-28	509105.068379
2021-03-31	517690.883241
2021-04-30	526276.698103
2021-05-31	534862.512966

VISUALIZING THE TRENDS →



FINANCIAL ANALYSIS

This financial analysis looks at the trend in the **Net Profit Ratio (NPR)** over time to gauge profitability in the company. By quantifying monthly NPR and charting its direction, we establish if the business is growing, contracting, or stable. The information derived aids in optimizing pricing, cost management, and overall financial decision-making.

Step 1: Data Preparation

- Convert the ORDERDATE column to **datetime format** for accurate time-based analysis using pd.to_datetime() from pandas
- Create a **'Year-Month'** period to aggregate sales data at a monthly level **.dt.to_period("M")** from **pandas**

```
# Convert the 'ORDERDATE' column to datetime format
df["ORDERDATE"] = pd.to_datetime(df["ORDERDATE"], errors="coerce")

# Create a 'Year-Month' period for grouping data by month
df["Year-Month"] = df["ORDERDATE"].dt.to_period("M")
```

Step 2: Net Profit Ratio Calculation

- Compute **Net Profit Ratio (%)** using the formula:

$$\text{Net Profit Ratio} = \frac{\text{Sales} - (\text{Actual Price} \times \text{Quantity Ordered})}{\text{Sales}} \times 100$$

- Group data by **'Year-Month'** and calculate the **average Net Profit Ratio** for each month using **.groupby("Year-Month")** and **.mean()** from pandas.

```
# Check if necessary columns exist and then calculate the Net Profit Ratio for each entry
df["Net Profit Ratio (%)"] = ((df["SALES"] - (df["ACTUAL PRICE"] * df["QUANTITYORDERED"])) / df["SALES"]) * 100

# Group by 'Year-Month' and calculate the average Net Profit Ratio for each month
df_monthly = df.groupby("Year-Month")["Net Profit Ratio (%)"].mean().reset_index()

# Convert the 'Year-Month' back to datetime format for plotting
df_monthly["Year-Month"] = df_monthly["Year-Month"].astype(str)
df_monthly["Year-Month"] = pd.to_datetime(df_monthly["Year-Month"])
```

Step 3: Trend Visualization

- Convert **'Year-Month'** to datetime format for plotting.
- Generate a **line plot** to display the Monthly Net Profit Ratio trend over time using pandas

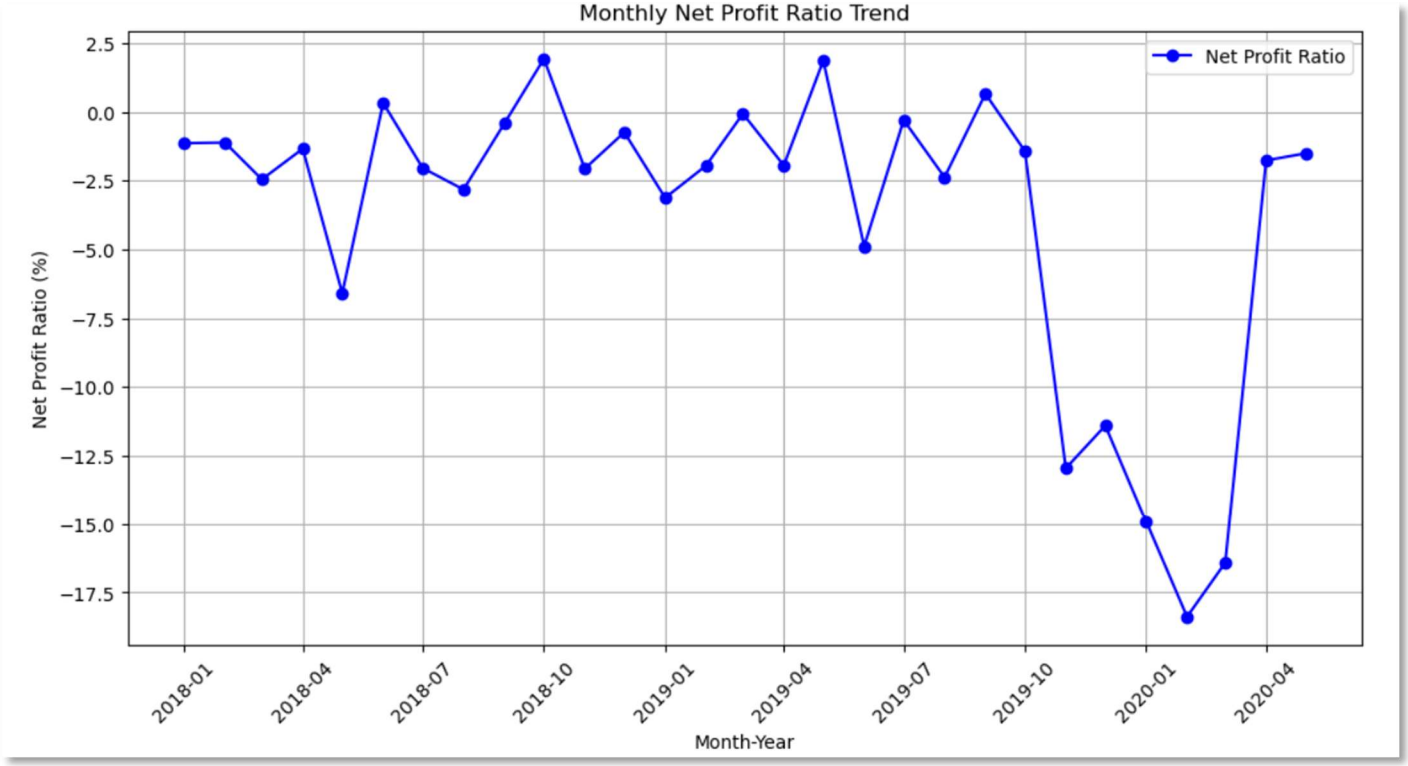
```
# Plot the trend of Monthly Net Profit Ratio
plt.figure(figsize=(12, 6))
plt.plot(df_monthly["Year-Month"], df_monthly["Net Profit Ratio (%)"], marker="o", linestyle="-", color="blue", label="Net Profit Ratio")

plt.title("Monthly Net Profit Ratio Trend")
plt.xlabel("Month-Year")
plt.ylabel("Net Profit Ratio (%)")
plt.xticks(rotation=45)
plt.grid()
plt.legend()

# Display the plot
plt.show()
```

Step 4: Interpretation of Results

- Compare the **first and last month's Net Profit Ratio** to determine if the trend is **increasing, decreasing, or stable**.
- Provide insights on the **financial health** and profitability trends of the business.

```
# ---------------- Interpretation of the Net Profit Ratio Trend ----------------
print("\nINTERPRETATION OF THE NET PROFIT RATIO TREND →")

# Analyze the trend direction based on the first and last month's ratios
if df_monthly['Net Profit Ratio (%)'].iloc[-1] > df_monthly['Net Profit Ratio (%)'].iloc[0]:
    trend_direction = "increasing"
elif df_monthly['Net Profit Ratio (%)'].iloc[-1] < df_monthly['Net Profit Ratio (%)'].iloc[0]
    trend_direction = "decreasing"
else:
    trend_direction = "stable"

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

OUTPUT –




```
INTERPRETATION OF THE NET PROFIT RATIO TREND →

Overall, the net profit ratio trend appears to be decreasing over time.
```

CONCLUSION & REFERENCES

This project integrates the automotive industry with contemporary data analysis using Python coding, machine learning algorithms, sampling, inferential statistics, time series forecasting, financial analysis and categorical data analysis.

In sampling, we successfully compared the efficiency of each sampling technique and declared stratified sampling as the best technique. After applying the ANOVA test, we found out that there was no significant difference in average sales of various countries. Similarly, after the chi square test, no significant relation could be identified between countries and deal size. Machine learning was implemented to forecast 'dealsize' based on several attributes. and facilitated identifying the effects of major features on it. Time series analysis was employed to monitor the sales pattern over time and overall market trends using the method of least squares. Financial analysis gave a complete picture of NPR and concluded that it has a decreasing trend with time.

The findings provide auto makers, dealers, and stakeholders with actionable information to make informed decisions on inventory management, pricing, marketing, and sales planning. Although the project provides meaningful conclusions within the confines of the dataset, it does not include external factors like government policies, world economic conditions, or supply chain interruptions. Increasing the dataset in subsequent work may further enhance model precision and enable more in-depth analysis, improving strategic planning and competitiveness.

This dataset was sourced from Kaggle.
Link to the dataset – https://www.kaggle.com/datasets/ddosad/auto-sales-data