**Scenarios**

Data Preprocessing

1. In the given CSV file do the preprocessing.
2. Write the steps you followed.
3. What was your thought process when you first saw the data.

Data Aggregation and Grouping

1. What all fields among them you think can be aggregated? Name them.
2. What kind of aggregation (for every column) would make sense and why?

Data Validation

1. How do you know, your preprocessing was correct?
2. How will you validate your results?
3. Do you follow any specific validation process for all questions? Explain.
4. What are the edge cases you can think of?
5. What all data integrity points you want to mention for the given scenario?

Data Visualisations

1. What all projections are possible out of the data.
2. How would be know if the data is linearly projected?
3. For all the different combinations of possible projections, what are the suitable graphical representation? (Eg: Line Chart or Bar Graph)

Output everything in Python. Also, other than visualization, insert the csv in sql and do everything in sql. Attach the code.

**SOLUTIONS**

Data Preprocessing

1. In the given CSV file do the preprocessing.
2. Write the steps you followed.
3. What was your thought process when you first saw the data.

**1. Improved Data Quality and Integrity**

* **Mean, Median, Mode:**
  + **Consistency with Data Distribution:** Mean, median, and mode are statistical measures that provide a value representative of the existing data. This can lead to more realistic and useful data than a placeholder like null or unknown, which may not reflect any actual trends or patterns.
  + **Avoid Misleading Analysis:** Using mean, median, or mode prevents the introduction of potentially misleading null values or placeholders that could distort statistical analyses, such as averages or sums.
* **Forward Fill (ffill):**
  + **Preserves Temporal Relationships:** In time series or sequential data, forward filling retains the context and patterns by filling missing values with the most recent non-null value, which preserves the sequence's temporal consistency.

**2. Enhanced Data Usability**

* **Mean, Median, Mode:**
  + **Better for Aggregation and Analysis:** When performing aggregation (e.g., calculating totals or averages), having missing values filled with statistical measures allows for more accurate and meaningful results.
  + **Avoids Data Gaps:** Filling missing values with these measures helps maintain a complete dataset, which is crucial for many machine learning models and analytical processes.
* **Forward Fill (ffill):**
  + **Maintains Continuity:** Forward filling is particularly useful in datasets where the value is expected to change gradually. It ensures that subsequent entries have realistic values, making the dataset more useful for trend analysis and forecasting.

**3. Improved Model Performance**

* **Mean, Median, Mode:**
  + **Reduces Bias:** Statistical imputation methods reduce the bias that might be introduced by missing data, allowing models to train on more complete datasets and potentially improving performance.
* **Forward Fill (ffill):**
  + **Reflects Real Trends:** In time-dependent data, forward filling reflects trends and patterns more accurately, which can enhance the performance of models that rely on sequential data, such as time series forecasting models.

**4. Increased Data Completeness**

* **Mean, Median, Mode:**
  + **Fills Gaps Effectively:** By replacing missing values with a calculated value, the dataset becomes complete, which is essential for many types of data analysis and reporting.
* **Forward Fill (ffill):**
  + **Handles Sequential Data Appropriately:** For data where values are expected to be similar over time (e.g., sensor readings), forward filling ensures that gaps are filled with the most recent valid data, maintaining the integrity of the dataset.

**5. Avoids Issues with Placeholder Values**

* **Placeholders (e.g., unknown or null):**
  + **Potential Misinterpretation:** Placeholders like unknown can be interpreted as a separate category or data point, which can mislead analysis and result in skewed findings.
  + **Handling Challenges:** Analyzing data with null or generic placeholders often requires additional handling, which can complicate the analysis or model training process.

**Summary**

Using mean, median, mode, or forward fill to handle missing values provides a more robust approach compared to simply inserting null or placeholders. These methods help in maintaining data integrity, improving usability for analysis and modeling, and ensuring that the dataset reflects real-world patterns and trends.

Here’s a summary of when to use each method:

* **Mean:** Best for numeric data where outliers are not a major concern.
* **Median:** Ideal for numeric data with outliers, as it’s more robust.
* **Mode:** Suitable for categorical data where the most common value is meaningful.
* **Forward Fill (ffill):** Excellent for time series or sequential data where recent values are likely to be relevant.

Choosing the right method depends on the nature of the data and the analysis requirements, but generally, these methods offer a more nuanced approach to handling missing values than simple placeholders

**1. Data Preprocessing Steps**

**Step 1: Import the CSV File**

Assuming you have already imported the CSV data into a table named transactions.

**Step 2: Inspect the Data**

Start by examining the table structure and a few rows of data:

sql

Copy code

SELECT \* FROM transactions LIMIT 10;

**Step 3: Handle Missing Values**

* **Forward Fill Missing IDs**

Forward fill (or ffill) for CustomerID and TransactionID can be done using window functions. Note that SQL does not have a built-in ffill, but you can achieve similar results with a window function.

sql

Copy code

-- For MySQL

SET @prev\_customer\_id := NULL;

SET @prev\_transaction\_id := NULL;

UPDATE transactions

SET CustomerID = (@prev\_customer\_id := COALESCE(CustomerID, @prev\_customer\_id)),

TransactionID = (@prev\_transaction\_id := COALESCE(TransactionID, @prev\_transaction\_id))

ORDER BY TransactionDate;

-- For PostgreSQL

WITH updated AS (

SELECT \*,

COALESCE(CustomerID, LAG(CustomerID, 1, NULL) OVER (ORDER BY TransactionDate)) AS new\_CustomerID,

COALESCE(TransactionID, LAG(TransactionID, 1, NULL) OVER (ORDER BY TransactionDate)) AS new\_TransactionID

FROM transactions

)

UPDATE transactions

SET CustomerID = updated.new\_CustomerID,

TransactionID = updated.new\_TransactionID

FROM updated

WHERE transactions.TransactionID = updated.TransactionID;

* **Handle Missing Values with Median or Mode**

For numeric columns, we’ll use median or mode. For SQL databases without direct median functions, calculate the median manually or use approximations.

* + **Calculate Median for PricePerUnit, TotalAmount, and Quantity:**

sql

Copy code

-- Calculate Median for PricePerUnit

WITH ordered AS (

SELECT PricePerUnit,

ROW\_NUMBER() OVER (ORDER BY PricePerUnit) AS rn,

COUNT(\*) OVER () AS cnt

FROM transactions

WHERE PricePerUnit IS NOT NULL

),

median\_calc AS (

SELECT AVG(PricePerUnit) AS median\_price

FROM ordered

WHERE rn IN (FLOOR((cnt + 1) / 2), CEIL((cnt + 1) / 2))

)

UPDATE transactions

SET PricePerUnit = (SELECT median\_price FROM median\_calc)

WHERE PricePerUnit IS NULL;

-- Repeat for TotalAmount

WITH ordered AS (

SELECT TotalAmount,

ROW\_NUMBER() OVER (ORDER BY TotalAmount) AS rn,

COUNT(\*) OVER () AS cnt

FROM transactions

WHERE TotalAmount IS NOT NULL

),

median\_calc AS (

SELECT AVG(TotalAmount) AS median\_amount

FROM ordered

WHERE rn IN (FLOOR((cnt + 1) / 2), CEIL((cnt + 1) / 2))

)

UPDATE transactions

SET TotalAmount = (SELECT median\_amount FROM median\_calc)

WHERE TotalAmount IS NULL;

-- Repeat for Quantity

WITH ordered AS (

SELECT Quantity,

ROW\_NUMBER() OVER (ORDER BY Quantity) AS rn,

COUNT(\*) OVER () AS cnt

FROM transactions

WHERE Quantity IS NOT NULL

),

median\_calc AS (

SELECT AVG(Quantity) AS median\_quantity

FROM ordered

WHERE rn IN (FLOOR((cnt + 1) / 2), CEIL((cnt + 1) / 2))

)

UPDATE transactions

SET Quantity = (SELECT median\_quantity FROM median\_calc)

WHERE Quantity IS NULL;

* + **Calculate Mode for DiscountApplied (assuming categorical data):**

sql

Copy code

-- Calculate Mode for DiscountApplied

WITH mode\_calc AS (

SELECT DiscountApplied,

COUNT(\*) AS freq

FROM transactions

GROUP BY DiscountApplied

ORDER BY freq DESC

LIMIT 1

)

UPDATE transactions

SET DiscountApplied = (SELECT DiscountApplied FROM mode\_calc)

WHERE DiscountApplied IS NULL;

**Step 4: Verify the Data**

After preprocessing, it’s crucial to check if there are any remaining anomalies:

sql

Copy code

-- Check for any missing values

SELECT \*

FROM transactions

WHERE CustomerID IS NULL

OR TransactionID IS NULL

OR PricePerUnit IS NULL

OR TotalAmount IS NULL

OR Quantity IS NULL

OR DiscountApplied IS NULL;

-- Check for any unexpected values

SELECT \*

FROM transactions

WHERE Quantity < 0

OR PricePerUnit < 0

OR TotalAmount < 0;

**2. Thought Process**

**Initial Thoughts:**

1. **Examine the Data Structure:**
   * Identify the columns and note which ones have missing values.
   * Assess numeric columns for missing or inconsistent values.
2. **Determine Missing Value Strategy:**
   * For categorical columns with missing values, mode is a suitable choice.
   * For numeric columns, median is often a good choice as it is robust to outliers.
3. **Plan Forward Fill for IDs:**
   * CustomerID and TransactionID should not have missing values; use forward fill to handle these.
4. **Handle Special Cases:**
   * Check and handle any negative values, as they might indicate errors or require adjustment.
5. **Verify Data Integrity:**
   * Ensure all preprocessing steps are applied correctly and validate the results.

Data Aggregation and Grouping

1. What all fields among them you think can be aggregated? Name them.
2. What kind of aggregation (for every column) would make sense and why?

**1. Fields That Can Be Aggregated**

In the dataset, the fields that can be aggregated include:

1. **Quantity**
2. **PricePerUnit**
3. **TotalAmount**
4. **TrustPointsUsed**
5. **DiscountApplied**

Fields such as TransactionID, CustomerID, TransactionDate, ProductID, and ProductCategory are identifiers or categorical variables, and while they can be used for grouping, they are not typically aggregated in the same way numeric fields are.

**2. Aggregation Types for Each Column**

Here’s a detailed look at what kind of aggregation makes sense for each field and why:

**1. Quantity**

* **Aggregation Type:** Sum, Average, Min, Max
* **Reasoning:**
  + **Sum:** To calculate the total quantity of products sold.
  + **Average:** To find the average quantity of products per transaction.
  + **Min/Max:** To determine the minimum and maximum quantities sold in any transaction.

sql

Copy code

-- Total Quantity Sold

SELECT SUM(Quantity) AS TotalQuantity

FROM transactions;

-- Average Quantity per Transaction

SELECT AVG(Quantity) AS AverageQuantity

FROM transactions;

-- Minimum and Maximum Quantity Sold

SELECT MIN(Quantity) AS MinQuantity,

MAX(Quantity) AS MaxQuantity

FROM transactions;

**2. PricePerUnit**

* **Aggregation Type:** Sum, Average, Min, Max
* **Reasoning:**
  + **Sum:** To calculate the total revenue from all transactions based on PricePerUnit (if multiplied by Quantity).
  + **Average:** To determine the average price per unit of products sold.
  + **Min/Max:** To find the lowest and highest price per unit across transactions.

sql

Copy code

-- Average Price Per Unit

SELECT AVG(PricePerUnit) AS AveragePrice

FROM transactions;

-- Minimum and Maximum Price Per Unit

SELECT MIN(PricePerUnit) AS MinPrice,

MAX(PricePerUnit) AS MaxPrice

FROM transactions;

**3. TotalAmount**

* **Aggregation Type:** Sum, Average, Min, Max
* **Reasoning:**
  + **Sum:** To get the total revenue from all transactions.
  + **Average:** To find the average total amount per transaction.
  + **Min/Max:** To determine the smallest and largest total amounts in transactions.

sql

Copy code

-- Total Revenue

SELECT SUM(TotalAmount) AS TotalRevenue

FROM transactions;

-- Average Total Amount per Transaction

SELECT AVG(TotalAmount) AS AverageTotalAmount

FROM transactions;

-- Minimum and Maximum Total Amount

SELECT MIN(TotalAmount) AS MinTotalAmount,

MAX(TotalAmount) AS MaxTotalAmount

FROM transactions;

**4. TrustPointsUsed**

* **Aggregation Type:** Sum, Average, Min, Max
* **Reasoning:**
  + **Sum:** To determine the total trust points used across all transactions.
  + **Average:** To find the average trust points used per transaction.
  + **Min/Max:** To find the minimum and maximum trust points used in any transaction.

sql

Copy code

-- Total Trust Points Used

SELECT SUM(TrustPointsUsed) AS TotalTrustPoints

FROM transactions;

-- Average Trust Points per Transaction

SELECT AVG(TrustPointsUsed) AS AverageTrustPoints

FROM transactions;

-- Minimum and Maximum Trust Points Used

SELECT MIN(TrustPointsUsed) AS MinTrustPoints,

MAX(TrustPointsUsed) AS MaxTrustPoints

FROM transactions;

**5. DiscountApplied**

* **Aggregation Type:** Mode, Average
* **Reasoning:**
  + **Mode:** To find the most common discount applied.
  + **Average:** To calculate the average discount applied per transaction (if discount values are numeric).

sql

Copy code

-- Most Common Discount Applied

WITH DiscountCounts AS (

SELECT DiscountApplied,

COUNT(\*) AS Frequency

FROM transactions

GROUP BY DiscountApplied

)

SELECT DiscountApplied

FROM DiscountCounts

ORDER BY Frequency DESC

LIMIT 1;

-- Average Discount Applied (if DiscountApplied is numeric)

SELECT AVG(CAST(DiscountApplied AS DECIMAL)) AS AverageDiscount

FROM transactions

WHERE DiscountApplied IS NOT NULL;

**Summary**

To aggregate the data effectively:

* **Quantities and Prices:** Aggregate using sum, average, min, and max to understand sales volume, pricing trends, and overall revenue.
* **Total Amount and Trust Points:** Use similar aggregations to assess revenue and loyalty points usage.
* **Discount Applied:** Analyze using mode for categorical discount values or average if discounts are numeric.

These aggregation methods provide a comprehensive overview of transaction patterns, customer behavior, and sales performance  
  
  
  
Data Validation

1. How do you know, your preprocessing was correct?
2. How will you validate your results?
3. Do you follow any specific validation process for all questions? Explain.
4. What are the edge cases you can think of?
5. What all data integrity points you want to mention for the given scenario?

**1. How to Know if Preprocessing Was Correct**

**Validation Checks:**

* **Consistency of Aggregated Data:**
  + Ensure that the aggregated values (like totals and averages) are consistent with expectations based on the data trends and business rules.
  + Verify if the computed aggregates match with pre-imputation data to confirm that imputation did not distort the results.
* **Non-null Values:**
  + Confirm that all critical fields (e.g., CustomerID, TransactionID, ProductID) are populated. Fields that should not have null values should be checked to ensure they were filled correctly during preprocessing.
* **Correct Aggregations:**
  + Check if the methods used for filling missing values (e.g., forward fill) and statistical measures (mean, median, mode) were applied correctly.
  + Ensure that the calculations align with the expected outcomes. For example, the total revenue should be a sum of TotalAmount across all transactions.
* **Logical Data Ranges:**
  + Validate that numeric fields like Quantity, PricePerUnit, and TotalAmount fall within plausible ranges. Negative values or zero values in these fields should be scrutinized, as they might indicate errors.

**Example SQL Queries for Validation:**

sql

Copy code

-- Check for Remaining NULLs in Critical Columns

SELECT COUNT(\*) AS NullCount

FROM transactions

WHERE CustomerID IS NULL

OR TransactionID IS NULL

OR ProductID IS NULL;

-- Validate Aggregated Data

SELECT SUM(Quantity) AS TotalQuantity,

AVG(Quantity) AS AverageQuantity

FROM transactions;

-- Check Data Range for Numerical Columns

SELECT MIN(Quantity) AS MinQuantity,

MAX(Quantity) AS MaxQuantity,

MIN(PricePerUnit) AS MinPrice,

MAX(PricePerUnit) AS MaxPrice,

MIN(TotalAmount) AS MinTotalAmount,

MAX(TotalAmount) AS MaxTotalAmount

FROM transactions;

-- Validate Forward Fill (if applied)

SELECT COUNT(\*) AS MissingCustomerIDCount

FROM transactions

WHERE CustomerID IS NULL;

**2. How to Validate Your Results**

**Steps for Validation:**

* **Spot Checks:**
  + Manually review a sample of records before and after preprocessing to confirm that missing values have been appropriately handled.
* **Cross-Validation:**
  + Compare aggregate statistics (such as totals and averages) with expected values or historical data. This can help ensure that preprocessing hasn’t skewed the results.
* **Consistency Checks:**
  + Ensure that forward filling and other imputation methods haven’t introduced inconsistencies. For example, check if filling methods have logically applied values based on adjacent records.
* **Compare Summary Statistics:**
  + Check summary statistics (mean, median) before and after preprocessing to confirm they are within expected ranges. This helps validate that preprocessing steps like imputation and forward filling have not distorted the data.

**Example SQL Queries for Validation:**

sql

Copy code

-- Compare Total Revenue Before and After Imputation

SELECT SUM(TotalAmount) AS TotalRevenueBefore

FROM transactions\_before;

SELECT SUM(TotalAmount) AS TotalRevenueAfter

FROM transactions\_after;

-- Compare Average Quantity Before and After Imputation

SELECT AVG(Quantity) AS AverageQuantityBefore

FROM transactions\_before;

SELECT AVG(Quantity) AS AverageQuantityAfter

FROM transactions\_after;

**3. Specific Validation Process**

**General Validation Process:**

1. **Data Integrity Check:**
   * Ensure that critical fields are non-null and unique where necessary. For example, TransactionID should be unique across the dataset.
2. **Logical Consistency:**
   * Verify that the imputed values make sense in the context of the data. For example, if CustomerID was filled forward, check that the imputed CustomerID aligns with the expected sequence.
3. **Statistical Validation:**
   * Confirm that the statistical measures (mean, median) calculated after preprocessing align with expected trends. This ensures that imputation or filling hasn’t introduced biases.
4. **Cross-Validation:**
   * Compare results against benchmarks or manually verified samples to confirm that the preprocessing steps did not introduce errors.

**Example SQL Queries for Statistical Validation:**

sql

Copy code

-- Example: Verify Average Trust Points Used

SELECT AVG(TrustPointsUsed) AS AverageTrustPoints

FROM transactions;

-- Verify Mode of Discount Applied

WITH DiscountCounts AS (

SELECT DiscountApplied,

COUNT(\*) AS Frequency

FROM transactions

GROUP BY DiscountApplied

)

SELECT DiscountApplied

FROM DiscountCounts

ORDER BY Frequency DESC

LIMIT 1;

**4. Edge Cases to Consider**

**Possible Edge Cases:**

* **Negative Values:**
  + Values like Quantity, PricePerUnit, or TotalAmount being negative due to errors in data entry or preprocessing. Such values should be validated and corrected.
* **Inconsistent Forward Fill:**
  + Situations where forward filling has introduced inconsistencies, such as filling CustomerID from one transaction to a logically different product or category.
* **Zero Values:**
  + Zero values in fields like Quantity or TotalAmount, which might indicate missing or erroneous data entries.
* **Extreme Values:**
  + Extremely high or low values that might result from errors or outliers. Such values should be checked to ensure they are plausible.

**Example SQL Queries to Check Edge Cases:**

sql

Copy code

-- Check for Negative Values

SELECT COUNT(\*) AS NegativeValuesCount

FROM transactions

WHERE Quantity < 0

OR PricePerUnit < 0

OR TotalAmount < 0;

-- Check for Zero Values in Quantity or TotalAmount

SELECT COUNT(\*) AS ZeroValuesCount

FROM transactions

WHERE Quantity = 0

OR TotalAmount = 0;

-- Check for Extreme Values

SELECT MIN(Quantity) AS MinQuantity,

MAX(Quantity) AS MaxQuantity,

MIN(PricePerUnit) AS MinPrice,

MAX(PricePerUnit) AS MaxPrice

FROM transactions;

**5. Data Integrity Points**

**Key Data Integrity Considerations:**

1. **Uniqueness of Identifiers:**
   * Ensure that fields like TransactionID and CustomerID are unique and non-null. Duplicate or missing IDs can lead to data integrity issues.
2. **Accuracy of Aggregated Data:**
   * Confirm that aggregated values like totals and averages accurately reflect the intended calculations and do not include errors from preprocessing.
3. **Consistency of Forward Filling:**
   * Verify that forward filling has been applied logically and consistently, and that it aligns with the sequence and context of the data.
4. **Validation of Numeric Ranges:**
   * Check that numeric fields fall within plausible ranges and that imputation or filling has not introduced unrealistic values.
5. **Handling of Categorical Data:**
   * Ensure categorical values are correctly imputed and that they align with valid categories. For example, DiscountApplied should only contain known discount types or values.

By rigorously applying these validation techniques, you can ensure that the preprocessing was done correctly and that the data is reliable for subsequent analysis or modeling

**VISUALISATIONS**

To effectively analyze and visualize the data provided, we'll address the following tasks:

1. **Possible Projections from the Data**
2. **Determining Linearity in Data**
3. **Suitable Graphical Representations for Each Projection**

**1. Possible Projections from the Data**

To gain insights from the dataset, consider the following projections:

**1.1 Time-Based Projections**

* **Number of Transactions Over Time:** Visualize how transactions trend over days or weeks.
* **Total Amount Spent Over Time:** Track spending patterns over time.

**1.2 Categorical Projections**

* **Product Category Analysis:** Examine quantities and total amounts spent across different product categories.
* **Payment Method Distribution:** Analyze the distribution of various payment methods used.

**1.3 Numerical Projections**

* **Quantity vs. Total Amount:** Analyze the relationship between quantity purchased and the total amount spent.
* **Price Per Unit Analysis:** Investigate the variation in price per unit across different products.

**1.4 Customer-Specific Projections**

* **Customer Spending Patterns:** Evaluate the total spending by each customer.
* **Trust Points Usage:** Analyze how trust points are used by different customers.

**1.5 Discount Analysis**

* **Impact of Discounts on Total Amount:** Assess how the application of discounts affects the total amount spent.

**2. Determining Linearity in Data**

To determine if relationships between variables are linear:

**2.1 Scatter Plots**

* **Plot Relationships:** Create scatter plots for pairs of numerical variables (e.g., Quantity vs. TotalAmount) to visually inspect for linear relationships.

**2.2 Correlation Analysis**

* **Compute Correlations:** Calculate Pearson correlation coefficients to quantify the strength and direction of linear relationships between variables.

**2.3 Regression Analysis**

* **Fit Linear Models:** Use linear regression to model relationships. Check R-squared and residual plots to assess the goodness of fit.

**2.4 Residual Analysis**

* **Plot Residuals:** After fitting a linear regression model, plot residuals to see if they are randomly scattered. Non-random patterns suggest deviations from linearity.

**3. Suitable Graphical Representations**

For each projection, choose the appropriate visualization to convey the data effectively:

**3.1 Time-Based Projections**

* **Line Chart:** Ideal for visualizing trends over time.
* **Heatmap:** Useful for showing density or intensity of transactions over time (e.g., month vs. day).

**Python Code Example:**

python

Copy code

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the data

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

# Convert TransactionDate to datetime

data['TransactionDate'] = pd.to\_datetime(data['TransactionDate'], errors='coerce')

# Number of Transactions Over Time

daily\_transactions = data.groupby(data['TransactionDate'].dt.date).size()

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

daily\_transactions.plot(kind='line')

plt.title('Number of Transactions Over Time')

plt.xlabel('Date')

plt.ylabel('Number of Transactions')

plt.grid(True)

plt.show()

# Heatmap: Transactions by Month and Day

data['Month'] = data['TransactionDate'].dt.month

data['Day'] = data['TransactionDate'].dt.day

heatmap\_data = data.groupby(['Month', 'Day']).size().unstack(fill\_value=0)

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

sns.heatmap(heatmap\_data, cmap='YlGnBu', annot=True)

plt.title('Heatmap of Transactions by Month and Day')

plt.xlabel('Day')

plt.ylabel('Month')

plt.show()

**3.2 Categorical Projections**

* **Bar Graph:** Effective for comparing quantities or amounts across different categories.
* **Pie Chart:** Useful for showing the proportion of different payment methods or categories.

**Python Code Example:**

python

Copy code

# Bar Graph: Total Amount by Product Category

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

sns.barplot(x='ProductCategory', y='TotalAmount', data=data, estimator=sum)

plt.title('Total Amount by Product Category')

plt.xlabel('Product Category')

plt.ylabel('Total Amount')

plt.xticks(rotation=45)

plt.show()

# Pie Chart: Payment Method Distribution

payment\_method\_counts = data['PaymentMethod'].value\_counts()

plt.figure(figsize=(10, 8))

plt.pie(payment\_method\_counts, labels=payment\_method\_counts.index, autopct='%1.1f%%', startangle=140)

plt.title('Payment Method Distribution')

plt.show()

**3.3 Numerical Projections**

* **Scatter Plot:** Ideal for examining relationships between two numerical variables.
* **Box Plot:** Useful for showing the distribution of a numerical variable by category.

**Python Code Example:**

python

Copy code

# Scatter Plot: Quantity vs. Total Amount

plt.figure(figsize=(10, 6))

sns.scatterplot(x='Quantity', y='TotalAmount', data=data)

plt.title('Scatter Plot: Quantity vs. Total Amount')

plt.xlabel('Quantity')

plt.ylabel('Total Amount')

plt.show()

# Box Plot: Price Per Unit by Product Category

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

sns.boxplot(x='ProductCategory', y='PricePerUnit', data=data)

plt.title('Price Per Unit by Product Category')

plt.xlabel('Product Category')

plt.ylabel('Price Per Unit')

plt.xticks(rotation=45)

plt.show()

**3.4 Customer-Specific Projections**

* **Bar Graph:** Useful for visualizing total spending by each customer.
* **Histogram:** Ideal for showing the distribution of trust points usage.

**Python Code Example:**

python

Copy code

# Bar Graph: Total Spending by Customer

customer\_spending = data.groupby('CustomerID')['TotalAmount'].sum()

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

customer\_spending.plot(kind='bar')

plt.title('Total Spending by Customer')

plt.xlabel('Customer ID')

plt.ylabel('Total Amount')

plt.show()

# Histogram: Trust Points Used by Customers

plt.figure(figsize=(10, 6))

sns.histplot(data['TrustPointsUsed'].dropna(), bins=10, kde=True)

plt.title('Distribution of Trust Points Used')

plt.xlabel('Trust Points Used')

plt.ylabel('Frequency')

plt.show()

**3.5 Discount Analysis**

* **Box Plot:** Compare distributions of total amounts with and without discounts.
* **Bar Graph:** Show average spending with different levels of discount applied.

**Python Code Example:**

python

Copy code

# Box Plot: Total Amount with and without Discounts

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

data['DiscountApplied'] = data['DiscountApplied'].fillna(0)

sns.boxplot(x='DiscountApplied', y='TotalAmount', data=data)

plt.title('Total Amount by Discount Applied')

plt.xlabel('Discount Applied')

plt.ylabel('Total Amount')

plt.show()

# Bar Graph: Average Spending by Discount Level

average\_spending = data.groupby('DiscountApplied')['TotalAmount'].mean()

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

average\_spending.plot(kind='bar')

plt.title('Average Spending by Discount Level')

plt.xlabel('Discount Applied')

plt.ylabel('Average Spending')

plt.show()

**Summary**

1. **Projections:** Identify and visualize time-based, categorical, numerical, customer-specific, and discount-related projections to uncover insights.
2. **Linearity:** Assess linear relationships using scatter plots, correlation analysis, regression models, and residual analysis.
3. **Graphical Representations:** Select appropriate charts (line charts, bar graphs, pie charts, scatter plots, box plots) based on the type of projection to effectively communicate the data insights.

This approach ensures that you leverage the dataset's full potential, making data-driven decisions easier and more informed.

MORE DETAILS – VISUALISATION

**1. Possible Projections and Suitable Graphical Representations**

**1.1 Categorical Data Projections**

**Objective:** Understand distributions and relationships for categorical data like product categories and payment methods.

**Why:** Categorical data often represents different groups or categories. Analyzing these can reveal trends and distributions that are useful for business insights.

**Visualizations:**

* **Bar Graph:** Useful for comparing total amounts or quantities across different categories.
* **Pie Chart:** Ideal for showing the proportion of each category in a dataset.

**Code and Explanation:**

python

Copy code

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the data

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

# Convert 'TotalAmount' to numeric to handle potential non-numeric values

data['TotalAmount'] = pd.to\_numeric(data['TotalAmount'], errors='coerce')

# Aggregate total spending by ProductCategory

category\_totals = data.groupby('ProductCategory')['TotalAmount'].sum()

# Bar Graph: Total Amount by Product Category

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

sns.barplot(x=category\_totals.index, y=category\_totals.values, ci=None)

plt.title('Total Amount by Product Category')

plt.xlabel('Product Category')

plt.ylabel('Total Amount')

plt.xticks(rotation=45)

plt.grid(True)

plt.show()

# Pie Chart: Payment Method Distribution

payment\_method\_counts = data['PaymentMethod'].value\_counts()

plt.figure(figsize=(10, 8))

plt.pie(payment\_method\_counts, labels=payment\_method\_counts.index, autopct='%1.1f%%', startangle=140)

plt.title('Payment Method Distribution')

plt.show()

**Reasoning:**

* **Bar Graph:** Aggregates data to compare how much is spent on each product category, providing a clear visual representation of spending distribution.
* **Pie Chart:** Shows the proportion of each payment method, allowing for an understanding of how different payment methods are utilized.

**1.2 Numerical Data Projections**

**Objective:** Explore relationships and distributions within numerical data, such as quantity and total amount.

**Why:** Numerical data can show trends and correlations that are valuable for deeper analysis.

**Visualizations:**

* **Scatter Plot:** To examine the relationship between two numerical variables.
* **Box Plot:** To show the distribution and detect outliers within numerical data grouped by categories.

**Code and Explanation:**

python

Copy code

# Scatter Plot: Quantity vs. Total Amount

plt.figure(figsize=(10, 6))

sns.scatterplot(x='Quantity', y='TotalAmount', data=data)

plt.title('Scatter Plot: Quantity vs. Total Amount')

plt.xlabel('Quantity')

plt.ylabel('Total Amount')

plt.grid(True)

plt.show()

# Box Plot: Price Per Unit by Product Category

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

sns.boxplot(x='ProductCategory', y='PricePerUnit', data=data)

plt.title('Price Per Unit by Product Category')

plt.xlabel('Product Category')

plt.ylabel('Price Per Unit')

plt.xticks(rotation=45)

plt.grid(True)

plt.show()

**Reasoning:**

* **Scatter Plot:** Helps identify potential correlations between Quantity and TotalAmount. A pattern might indicate a relationship between these variables.
* **Box Plot:** Provides a summary of the distribution of PricePerUnit across different product categories, highlighting variations and outliers.

**1.3 Customer-Specific Data Projections**

**Objective:** Analyze spending behavior and trust points usage for individual customers.

**Why:** Understanding customer behavior helps in targeting and personalizing strategies.

**Visualizations:**

* **Bar Graph:** To show total spending by each customer.
* **Histogram:** To display the distribution of trust points used.

**Code and Explanation:**

python

Copy code

# Bar Graph: Total Spending by Customer

customer\_spending = data.groupby('CustomerID')['TotalAmount'].sum().sort\_values(ascending=False)

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

customer\_spending.plot(kind='bar')

plt.title('Total Spending by Customer')

plt.xlabel('Customer ID')

plt.ylabel('Total Amount')

plt.grid(True)

plt.show()

# Histogram: Trust Points Used by Customers

plt.figure(figsize=(10, 6))

sns.histplot(data['TrustPointsUsed'].dropna(), bins=10, kde=True)

plt.title('Distribution of Trust Points Used')

plt.xlabel('Trust Points Used')

plt.ylabel('Frequency')

plt.show()

**Reasoning:**

* **Bar Graph:** Allows for comparison of spending across different customers, identifying high-value customers.
* **Histogram:** Reveals how trust points are distributed across transactions, which can provide insights into how frequently trust points are used.

**1.4 Discount Analysis**

**Objective:** Analyze the impact of discounts on total spending.

**Why:** Discounts can affect customer purchasing behavior and overall revenue.

**Visualizations:**

* **Box Plot:** To compare total spending across different discount levels.
* **Bar Graph:** To show the average spending by discount level.

**Code and Explanation:**

python

Copy code

# Fill NaN values in 'DiscountApplied' with 0 for analysis

data['DiscountApplied'] = data['DiscountApplied'].fillna(0)

# Box Plot: Total Amount by Discount Applied

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

sns.boxplot(x='DiscountApplied', y='TotalAmount', data=data)

plt.title('Total Amount by Discount Applied')

plt.xlabel('Discount Applied')

plt.ylabel('Total Amount')

plt.grid(True)

plt.show()

# Bar Graph: Average Spending by Discount Level

average\_spending = data.groupby('DiscountApplied')['TotalAmount'].mean()

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

average\_spending.plot(kind='bar')

plt.title('Average Spending by Discount Level')

plt.xlabel('Discount Applied')

plt.ylabel('Average Spending')

plt.grid(True)

plt.show()

**Reasoning:**

* **Box Plot:** Provides insights into the variability of spending with different discounts, including outliers.
* **Bar Graph:** Shows the average spending for each discount level, helping to assess the effectiveness of discounts on average spend.

**2. Determining Linearity in Data**

**Objective:** Check if there is a linear relationship between numerical variables.

**Why:** Understanding linear relationships helps in building predictive models and identifying trends.

**Visualizations:**

* **Scatter Plot:** To visualize the relationship between two numerical variables.
* **Correlation Analysis:** To quantify the strength and direction of a linear relationship.
* **Linear Fit:** To assess how well a linear model describes the relationship.

**Code and Explanation:**

python

Copy code

import numpy as np

from scipy.stats import pearsonr

from sklearn.linear\_model import LinearRegression

# Scatter Plot: Quantity vs. Total Amount

plt.figure(figsize=(10, 6))

sns.scatterplot(x='Quantity', y='TotalAmount', data=data)

plt.title('Scatter Plot: Quantity vs. Total Amount')

plt.xlabel('Quantity')

plt.ylabel('Total Amount')

plt.grid(True)

plt.show()

# Correlation Analysis: Pearson Correlation Coefficient

quantity = data['Quantity'].dropna()

total\_amount = data['TotalAmount'].dropna()

corr, \_ = pearsonr(quantity, total\_amount)

print(f'Pearson Correlation between Quantity and Total Amount: {corr:.2f}')

# Fit a Linear Model (Optional for further analysis)

X = data[['Quantity']].dropna()

y = data['TotalAmount'].dropna()

model = LinearRegression().fit(X, y)

predictions = model.predict(X)

# Plot Linear Fit

plt.figure(figsize=(10, 6))

sns.scatterplot(x='Quantity', y='TotalAmount', data=data)

plt.plot(X, predictions, color='red', linestyle='--')

plt.title('Quantity vs. Total Amount with Linear Fit')

plt.xlabel('Quantity')

plt.ylabel('Total Amount')

plt.grid(True)

plt.show()

**Reasoning:**

* **Scatter Plot:** Provides a visual representation to identify if the relationship between Quantity and TotalAmount looks linear.
* **Correlation Analysis:** Measures the strength of the linear relationship using Pearson's correlation coefficient, which ranges from -1 (perfect negative) to 1 (perfect positive).
* **Linear Fit:** Helps in understanding how well a linear model explains the relationship between the variables.

This comprehensive approach will enable you to perform a thorough analysis and visualize various aspects of your data effectively.