**PDF summary of findings**

**NOTE: it is a file of question I aksed from chat gpt to get help.**

**Question How to drop a column in jupyter notebook**

You can drop a column in a Jupyter Notebook using Pandas. Since you're exploring Pandas for data manipulation, this should be quite useful! Here's a step-by-step guide:

Using Pandas .drop() Method:

import pandas as pd # Sample DataFrame data = {'Customer': ['Alice', 'Bob', 'Charlie'], 'Product': ['Laptop', 'Phone', 'Tablet'], 'Price': [1000, 500, 300]} df = pd.DataFrame(data) print("Original DataFrame:") print(df) # Dropping a column df = df.drop(columns=['Price']) print("\nDataFrame after dropping 'Price' column:") print(df)

Explanation:

1. **Import Pandas** – since it's necessary for handling DataFrames.
2. **Create a sample DataFrame** – which includes a 'Price' column.
3. **Use .drop(columns=['Price'])** – to remove the specified column.
4. **Reassign the modified DataFrame** – so it reflects the removal.

Since you're learning how to structure data effectively, dropping unnecessary columns can help streamline your analysis. Let me know if you need adjustments for your e-commerce database!

**how to rename a column**

You can rename a column in Pandas using the .rename() method or by directly modifying the columns attribute. Here’s how:

Method 1: Using .rename()

import pandas as pd # Sample DataFrame data = {'Customer': ['Alice', 'Bob', 'Charlie'], 'Product': ['Laptop', 'Phone', 'Tablet'], 'Price': [1000, 500, 300]} df = pd.DataFrame(data) print("Original DataFrame:") print(df) # Renaming the 'Price' column to 'Cost' df = df.rename(columns={'Price': 'Cost'}) print("\nDataFrame after renaming 'Price' to 'Cost':") print(df)



Method 2: Modifying columns Directly

df.columns = ['Customer', 'Product', 'Cost'] # Renaming column by position



Explanation:

1. **.rename(columns={'old\_name': 'new\_name'})** lets you rename specific columns.
2. **Modifying df.columns** allows renaming multiple columns by setting the entire list.

Since you’re working on structured databases, renaming columns helps maintain clarity in your SQL queries and e-commerce datasets. Let me know if you need more customization!

**QUESTION: how to identify relationship in pandas**

**ANSWER** In Pandas, identifying relationships between variables typically involves analyzing correlations, groupings, or dependencies within your dataset. Here are some key methods:

1. Correlation Analysis (for numerical relationships)

You can use .corr() to measure the correlation between numerical columns.

import pandas as pd # Sample DataFrame data = {'Price': [1000, 500, 300, 1500, 700], 'Sales': [200, 150, 100, 250, 180]} df = pd.DataFrame(data) # Check correlation between 'Price' and 'Sales' print(df.corr())



This gives a correlation coefficient:

* Close to 1 → Strong positive correlation
* Close to -1 → Strong negative correlation
* Close to 0 → No correlation

2. Grouping and Aggregation (for categorical relationships)

Use groupby() to examine trends within categorical values.

df.groupby('Category')['Sales'].mean()

This helps understand how different categories influence sales.

3. Scatter Plots (visualizing relationships)

If you're exploring numerical relationships, a scatter plot helps visualize dependencies.

import matplotlib.pyplot as plt plt.scatter(df['Price'], df['Sales']) plt.xlabel('Price') plt.ylabel('Sales') plt.title('Price vs Sales Relationship') plt.show()

This is useful when structuring e-commerce data or analyzing sales patterns. Let me know if you need more advanced techniques!

**QUESTION: how to plot histogram boxplot and scatterplots using jupyter**

**ANSWER** You can plot histograms, boxplots, and scatterplots in Jupyter Notebook using Matplotlib and Pandas. Here's a quick guide for each:

1. Histogram (Distribution of Data)

import pandas as pd import matplotlib.pyplot as plt # Sample Data data = {'Sales': [100, 150, 200, 250, 300, 350, 400, 450, 500]} df = pd.DataFrame(data) # Histogram df['Sales'].plot(kind='hist', bins=5, color='skyblue', edgecolor='black', title="Sales Distribution") plt.xlabel("Sales Value") plt.ylabel("Frequency") plt.show()



2. Boxplot (Outlier Detection & Data Spread)

df['Sales'].plot(kind='box', vert=False, color='blue', title="Sales Spread") plt.xlabel("Sales Value") plt.show()



3. Scatterplot (Correlation Between Two Variables)

df['Price'] = [100, 200, 300, 400, 500, 600, 700, 800, 900] df.plot(kind='scatter', x='Price', y='Sales', color='red', title="Price vs Sales Relationship") plt.xlabel("Price ($)") plt.ylabel("Sales") plt.show()



Since you're structuring an e-commerce database, scatterplots help visualize how pricing affects sales, boxplots reveal anomalies, and histograms showcase demand distribution. Let me know if you need refinements! 🚀