Data Science Answers

1. What is data wrangling? Describe the processes involved in cleaning, transforming, merging, and reshaping datasets during data wrangling.

Data wrangling (or data munging) is the process of cleaning, transforming, and organizing raw data into a usable format. It's an essential step in data analysis.

Processes involved:

- **Cleaning:** Removing duplicates, correcting errors, handling missing values.
- **Transforming:** Changing data formats, normalizing values, parsing columns.
- **Merging:** Combining multiple datasets based on common keys.
- **Reshaping:** Pivoting or melting datasets to fit analysis needs.

2. Explain the methods for combining and merging datasets. How is merging on index different from concatenating datasets? Provide Python suitable examples.

```python
import pandas as pd

df1 = pd.DataFrame({'ID': [1, 2], 'Name': ['Alice', 'Bob']})
df2 = pd.DataFrame({'ID': [1, 2], 'Score': [90, 85]})

# Merge on key
merged = pd.merge(df1, df2, on='ID')

# Merge on index
df1.set_index('ID', inplace=True)
df2.set_index('ID', inplace=True)
merged_index = pd.merge(df1, df2, left_index=True, right_index=True)

# Concatenation
concat_axis0 = pd.concat([df1, df2], axis=0)
concat_axis1 = pd.concat([df1, df2], axis=1)
```

Merging vs Concatenating:

- **Merging:** Combines rows based on a key/index (like SQL joins).
- **Concatenating:** Stacks DataFrames either vertically (rows) or horizontally (columns) without matching keys.

3. What are missing data? Describe the different techniques to handle missing data during data cleaning and preparation.

Missing data refers to absent values in a dataset.

Techniques:

- **Detection:** `df.isnull()`, `df.info()`
- **Removal:** `df.dropna()`
- **Imputation:** `df.fillna(method='ffill')`, `df.fillna(value)`
- **Custom logic:** Replace with mean, median, etc.

4. Discuss the methods to detect and handle outliers, noise, and anomalies in large datasets. Why is handling anomalies important in data preparation?

- **Detection:** Box plots, Z-score, IQR
- **Handling:**
  - **Remove:** Drop anomalies
  - **Cap:** Use limits
  - **Transform:** Log or scale

Why it's important? Anomalies distort statistical models, skew results, and mislead analyses.

5. import pandas as pd
import numpy as np
data = {
'Customer_ID': ['C100', 'C101', 'C102', 'C103', 'C104'],
'Name': ['john doe', 'Alice_Green', 'BOB SMITH', 'diya patel', 'Eva_Lee'],
'Age': [25, 33, 28, 45, 22],
'Purchase_Amount': [120, 250, 300, 50, 5000],
'City': ['New York', 'los angeles', 'Chicago', 'chicago', 'NEW YORK'],
'Rating': ['4.5', '3.2', '4.0', '2.5', '4.8']
}
df = pd.DataFrame(data)
Write python code for the following

- Convert all names to Title Case
- Find mean and median 'Purchase_Amount'
- Number of unique cities

Convert the 'Age' column into 3 bins: "Young (18-25)", "Adult (26-40)", "Senior (41+)"

```
import pandas as pd
import numpy as np

data = {
    'Customer_ID': ['C100', 'C101', 'C102', 'C103', 'C104'],
    'Name': ['john doe', 'Alice_Green', 'BOB SMITH', 'diya patel', 'Eva_Lee'],
    'Age': [25, 33, 28, 45, 22],
    'Purchase_Amount': [120, 250, 300, 50, 5000],
    'City': ['New York', 'los angeles', 'Chicago', 'chicago', 'NEW YORK'],
    'Rating': ['4.5', '3.2', '4.0', '2.5', '4.8']
}
df = pd.DataFrame(data)
```

```python
# Title case names
df['Name'] = df['Name'].str.replace('_', ' ').str.title()

# Mean & Median
mean_purchase = df['Purchase_Amount'].mean()
median_purchase = df['Purchase_Amount'].median()

# Unique cities
unique_cities = df['City'].str.lower().nunique()

# Age bins
df['Age_Group'] = pd.cut(df['Age'],
                         bins=[17, 25, 40, 100],
                         labels=['Young (18-25)', 'Adult (26-40)', 'Senior (41+)'])

print(df)
```

6. import pandas as pd

data = {

'Emp_ID': ['E001', 'E002', 'E003', 'E004', 'E005'],

'Full_Name': ['raj kumar', 'SITA SHARMA', 'vijay_verma', 'ANITA JAIN', 'rahul_singh'],

'Salary': [50000, 72000, 65000, 48000, 150000],

'Department': ['HR', 'Finance', 'IT', 'it', 'HR'],

'Experience_Years': [2, 8, 5, 1, 12],

'Performance_Score': ['3.8', '4.5', '3.2', '4.0', '4.9']

}

df = pd.DataFrame(data)

- Convert all names in the Full_Name column to Title Case (e.g., "raj kumar" → "Raj Kumar").
- Calculate the mean and standard deviation of the Salary column.
- Find the number of unique departments.
- Categorize the Experience_Years into the following bins:
  "Junior (0–3)",
  "Mid-level (4–8)",
  "Senior (9+)".

```
data = {
    'Emp_ID': ['E001', 'E002', 'E003', 'E004', 'E005'],
    'Full_Name': ['raj kumar', 'SITA SHARMA', 'vijay_verma', 'ANITA JAIN', 'rahul_singh'],
    'Salary': [50000, 72000, 65000, 48000, 150000],
    'Department': ['HR', 'Finance', 'IT', 'it', 'HR'],
    'Experience_Years': [2, 8, 5, 1, 12],
    'Performance_Score': ['3.8', '4.5', '3.2', '4.0', '4.9']
}
df = pd.DataFrame(data)

# Name formatting
df['Full_Name'] = df['Full_Name'].str.replace('_', ' ').str.title()

# Salary stats
mean_salary = df['Salary'].mean()
std_salary = df['Salary'].std()

# Unique departments
unique_departments = df['Department'].str.lower().nunique()

# Experience bins
df['Experience_Level'] = pd.cut(df['Experience_Years'],
                                bins=[-1, 3, 8, float('inf')],
                                labels=['Junior (0-3)', 'Mid-level (4-8)', 'Senior (9+)'])

print(df)
```

7, You are given two datasets about students' academic records. Write Python code using **pandas** to perform the following tasks:

```
import pandas as pd

# DataFrame 1
data1 = {
    'Student_ID': ['S001', 'S002', 'S003', 'S004'],
    'Name': ['john_doe', 'ALICE SMITH', 'bob_jones', 'Clara Brown'],
    'Math_Score': [85, 78, 92, 67]
}
df1 = pd.DataFrame(data1)
df1.set_index('Student_ID', inplace=True)

# DataFrame 2
data2 = {
    'Student_ID': ['S001', 'S002', 'S003', 'S004'],
    'Science_Score': [88, 74, 90, 72]
}
df2 = pd.DataFrame(data2)
df2.set_index('Student_ID', inplace=True)
```

Tasks:

1. Merge the two dataframes **on the Student_ID index**.
2. Convert all names in the **Name** column to **Title Case** and replace underscores _ with spaces.
3. Suppose there is noise in the **Math_Score** column (e.g., scores less than 0 or greater than 100 are invalid). Write code to **detect and correct** such anomalies by capping values within the 0–100 range.
4. Create a **pivot table** showing the average **Math_Score** and **Science_Score**.

```python
# Merging
merged_df = df1.join(df2)

# Name cleanup
merged_df['Name'] = merged_df['Name'].str.replace('_', ' ').str.title()

# Noise handling
merged_df['Math_Score'] = merged_df['Math_Score'].clip(lower=0, upper=100)

# Pivot table
pivot = merged_df[['Math_Score', 'Science_Score']].mean().to_frame().T
```

8. import pandas as pd

```python
df1 = pd.DataFrame({
    'ID': [1, 2, 3],
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Dept': ['HR', 'Tech', 'Finance']
})
df2 = pd.DataFrame({
    'ID': [2, 3, 4],
    'Salary': [50000, 60000, 45000],
    'Join_Date': ['2020-01-15', '2019-05-20', '2021-11-10']
})
```

- Concatenate df1 and df2 (axis = 0 and axis = 1)
- Merge df1 and df2 (inner, left, right and outer)
  - Join df1 and df2

```python
# Concatenate
concat_axis0 = pd.concat([df1, df2], axis=0)
concat_axis1 = pd.concat([df1, df2], axis=1)

# Merge types
inner = pd.merge(df1, df2, on='ID', how='inner')
left = pd.merge(df1, df2, on='ID', how='left')
right = pd.merge(df1, df2, on='ID', how='right')
outer = pd.merge(df1, df2, on='ID', how='outer')

# Join
df1.set_index('ID', inplace=True)
df2.set_index('ID', inplace=True)
joined = df1.join(df2)
```

9. You are given two DataFrames representing employee and project details. Write Python code to perform the following operations:

```python
import pandas as pd

df_a = pd.DataFrame({
    'Emp_ID': [101, 102, 103],
    'Name': ['Raj', 'Anita', 'Vikram'],
    'Project': ['Alpha', 'Beta', 'Gamma']
})

df_b = pd.DataFrame({
    'Emp_ID': [102, 103, 104],
    'Hours_Worked': [160, 150, 170],
    'Month': ['Jan', 'Jan', 'Jan']
})
```

Tasks:

1. Concatenate df_a and df_b using axis=0 and axis=1.
2. Merge df_a and df_b on Emp_ID using:

    - inner join
    - left join
    - right join
    - outer join
3. Use the join() function to combine the two DataFrames. Before joining, set Emp_ID as the index for both.

```python
# Concatenate
concat0 = pd.concat([df_a, df_b], axis=0)
concat1 = pd.concat([df_a, df_b], axis=1)

# Merge joins
inner_merge = pd.merge(df_a, df_b, on='Emp_ID', how='inner')
left_merge = pd.merge(df_a, df_b, on='Emp_ID', how='left')
right_merge = pd.merge(df_a, df_b, on='Emp_ID', how='right')
outer_merge = pd.merge(df_a, df_b, on='Emp_ID', how='outer')

# Join
df_a.set_index('Emp_ID', inplace=True)
df_b.set_index('Emp_ID', inplace=True)
joined_df = df_a.join(df_b)
```

“Super Thambi Unit-2 Mudinchu”

Unit Three

1.Explain the various customization options available in Matplotlib to control axes, ticks, labels, legends, and annotations. Give suitable examples.

✅ Axes Customization:

```python
plt.axis('equal')           # Equal aspect ratio
plt.xlim(0, 100)            # Set X-axis limits
plt.ylim(0, 100)            # Set Y-axis limits
```

✅ Ticks:

```python
plt.xticks([0, 50, 100], ['Low', 'Mid', 'High'])  # Custom tick locations and labels
plt.tick_params(axis='both', direction='in', length=6, color='red')  # Style
```

✅ Labels and Title:

```python
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Sample Plot')
```

✅ **Legends:**

```python
plt.plot(x, y, label='Line 1')
plt.legend(loc='upper right', title='Legend')
```

✅ **Annotations:**

```python
plt.annotate('Peak', xy=(x_val, y_val), xytext=(x_val+1, y_val+10),
             arrowprops=dict(facecolor='black', shrink=0.05))
```

2.Discuss the differences between various plot types in Matplotlib and Seaborn like line plots, bar charts, histograms, and box plots with use-cases.

Plot Type	Description	Best Use-case	Seaborn Advantage
Line Plot	Connects points via lines	Trend over time (e.g. stock prices)	Auto-styling and grouping
Bar Chart	Compares quantities	Compare categories (e.g. sales by region)	Better color themes and group support
Histogram	Distribution of a variable	Frequency of marks, age	Smoother aesthetics
Box Plot	Shows distribution and outliers	Data spread comparison	Built-in grouping, styling

3. You are given a dataset representing student performance. Using **Matplotlib** and **Seaborn**, draw different types of plots to analyze the data:

```
import pandas as pd

data = {
    'Student': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Maths': [78, 85, 60, 90, 95],
    'Science': [88, 75, 70, 85, 92],
    'English': [82, 80, 65, 78, 88]
```

}
df = pd.DataFrame(data)

Tasks:

1. Plot a **line chart** showing marks in Maths, Science, and English for each student.
2. Create a **bar chart** comparing Maths marks for all students.
3. Draw a **scatter plot** of Science vs. Maths scores.
4. Plot a **box plot** for marks in all three subjects using Seaborn.
5. Create a **pair plot** of the dataset using Seaborn.
6. Customize one of the plots by changing line styles, colors, and adding legends, titles, and grid.
7. Save one of the plots as a PNG image.

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = {
    'Student': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Maths': [78, 85, 60, 90, 95],
    'Science': [88, 75, 70, 85, 92],
    'English': [82, 80, 65, 78, 88]
}
df = pd.DataFrame(data)
```

1. Line Chart

python   Copy   Edit

```python
plt.figure(figsize=(8, 5))
plt.plot(df['Student'], df['Maths'], label='Maths', marker='o')
plt.plot(df['Student'], df['Science'], label='Science', marker='s')
plt.plot(df['Student'], df['English'], label='English', marker='^')
plt.title('Subject-wise Scores')
plt.xlabel('Students')
plt.ylabel('Marks')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

🔹 2. Bar Chart

python

```
plt.figure(figsize=(6, 4))
sns.barplot(x='Student', y='Maths', data=df, palette='cool')
plt.title('Maths Scores Comparison')
plt.tight_layout()
plt.show()
```

🔹 3. Scatter Plot

python

```
plt.figure(figsize=(6, 4))
plt.scatter(df['Maths'], df['Science'], color='green', edgecolor='black')
plt.title('Science vs Maths')
plt.xlabel('Maths')
plt.ylabel('Science')
plt.grid(True)
plt.tight_layout()
plt.show()
```

🔹 4. Box Plot (Seaborn)

python

```
plt.figure(figsize=(6, 4))
sns.boxplot(data=df[['Maths', 'Science', 'English']], palette='Set2')
plt.title('Subject Score Distribution')
plt.tight_layout()
plt.show()
```

🔹 5. Pair Plot

python

```
sns.pairplot(df[['Maths', 'Science', 'English']])
plt.suptitle('Pair Plot of Subject Scores', y=1.02)
plt.show()
```

🔹 **6. Customized Plot**

python

Copy Edit

```
plt.figure(figsize=(8, 5))
plt.plot(df['Student'], df['Maths'], linestyle='--', color='blue', marker='o', label='Maths')
plt.plot(df['Student'], df['Science'], linestyle='-.', color='green', marker='s', label='Science')
plt.title('Customized Subject Line Chart', fontsize=14)
plt.xlabel('Student')
plt.ylabel('Marks')
plt.grid(True, linestyle=':', color='gray')
plt.legend(title='Subjects')
plt.tight_layout()
plt.show()
```

🔹 **7. Save Plot as PNG**

python

Copy Edit

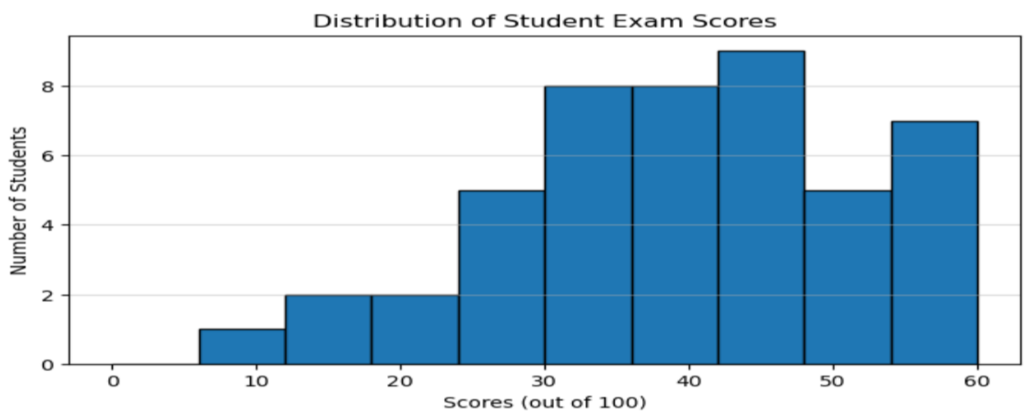
```
plt.figure(figsize=(6, 4))
sns.barplot(x='Student', y='English', data=df, color='purple')
plt.title('English Scores')
plt.tight_layout()
plt.savefig('english_scores.png')  # Saves the plot as a PNG image
```

4. *import seaborn as sns*
import pandas as pd
import numpy as np
np.random.seed(42)
data = pd.DataFrame({
'Age': np.random.randint(20, 70, 100),
'Income': np.random.normal(50000, 15000, 100).astype(int),
*'Spending': np.random.normal(0.8, 0.2, 100) * np.random.normal(50000, 15000, 100).astype(int),*
'Savings': np.random.uniform(1000, 20000, 100).astype(int)
})

- Write python code to create the Pair plot
- If Savings vs. Age scatter plot had a negative slope, suggest solution.

If Age vs. Spending scatter plot shows no clear pattern. Interpret the relationship.

5.



- If the passing score is 30, how many students failed?
- Estimate the total number of students represented in the histogram.

What type of distribution does this histogram represent?

✅ 1. How many students failed (score < 30)?

Looking at the bins to the **left of 30**, we estimate the number of students in each relevant bar:

- 0–10: ~1 student
- 10–20: ~2 students
- 20–30: ~5 students

Total failed students ≈ 1 + 2 + 5 = 8 students

6.

✅ 2. Estimate total number of students:

Add the heights of all the bars (frequencies):

- 0–10: ~1
- 10–20: ~2
- 20–30: ~5
- 30–40: ~8
- 40–50: ~9
- 50–60: ~7
- (other bins are 2 and 4 approx)

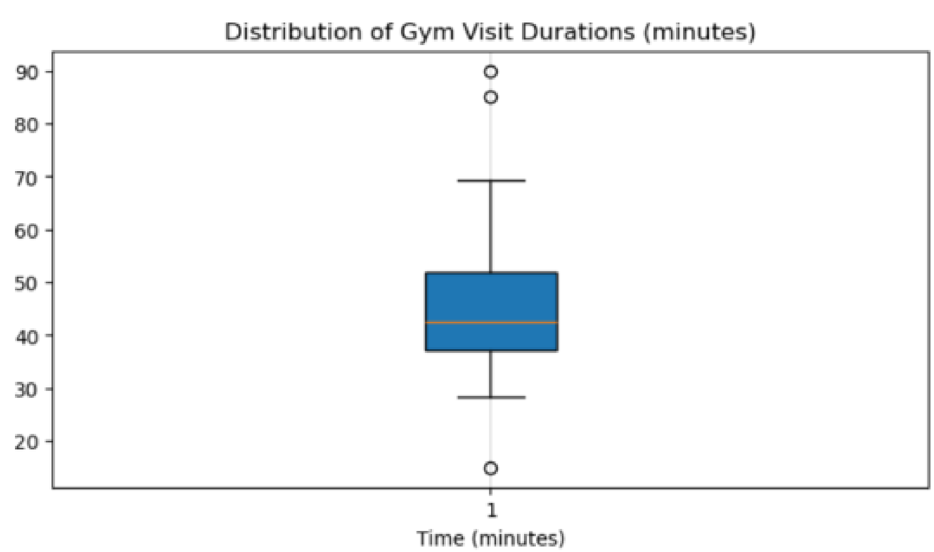
Estimated total ≈ 1 + 2 + 5 + 8 + 9 + 7 + 6 + 4 = ~42 students

✅ 3. What type of distribution is this?

This appears to be a **right-skewed distribution** (also called **positively skewed**), because:

- Most students scored between 30–60.
- There's a longer tail on the **left side**, representing a smaller number of low scorers.

6.



- What is the median gym visit duration?
- How many outliers are shown in the plot?
- 50% of members stayed between Q1 and Q3. Ste True or False.
- If a visit lasts 100 minutes, would it be an outlier? Justify.

✅ 1. What is the median gym visit duration?

The **median** is represented by the line inside the box.

- From the plot, the median appears to be **around 40 minutes.**

✅ 2. How many outliers are shown in the plot?

Outliers are marked as **individual dots** outside the whiskers.

- There are **3 outliers above** the upper whisker (at ~70, ~85, and ~90 minutes).
- There is **1 outlier below** the lower whisker (at ~15 minutes).

Total outliers = 4

✅ 3. "50% of members stayed between Q1 and Q3." – True or False?

True.
By definition, the box in a box plot represents the **interquartile range (IQR)** — from **Q1 (25th percentile)** to **Q3 (75th percentile)** — which contains the **middle 50%** of the data.

✅ 4. If a visit lasts 100 minutes, would it be an outlier? Justify.

Yes, **100 minutes would be an outlier.**

- Outliers are typically defined as values that fall **below Q1 – 1.5×IQR** or **above Q3 + 1.5×IQR.**
- Since values like 85 and 90 are already outliers in the plot, 100 minutes would lie **even farther outside the whiskers,** and thus is **clearly an outlier.**

7. You are given the sales data for a store over 6 months:

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']

sales = [15000, 18000, 22000, 21000, 25000, 27000]

"Plot a line graph to show the store's sales trend over the first six months of the year. Use months as the X-axis and sales (in dollars) as the Y-axis."

"Create a bar chart representing the sales for each month from January to June. Which month had the highest and lowest sales?"

"Using the given sales data, draw a graph and analyze the change in sales between consecutive months. Highlight any months where sales dropped."

```
import matplotlib.pyplot as plt

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
sales = [15000, 18000, 22000, 21000, 25000, 27000]

# 1. Line graph
plt.figure(figsize=(8, 4))
plt.plot(months, sales, marker='o', linestyle='-', color='blue')
plt.title("Monthly Sales Trend")
plt.xlabel("Month")
plt.ylabel("Sales (USD)")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
# 2. Bar chart
plt.figure(figsize=(8, 4))
plt.bar(months, sales, color='orange')
plt.title("Monthly Sales (Bar Chart)")
plt.xlabel("Month")
plt.ylabel("Sales (USD)")
plt.tight_layout()
plt.show()

# Identifying highest and lowest
max_month = months[sales.index(max(sales))]
min_month = months[sales.index(min(sales))]
print(f"Highest Sales: {max_month} (${max(sales)})")
print(f"Lowest Sales: {min_month} (${min(sales)})")
```

```python
# Identifying highest and lowest
max_month = months[sales.index(max(sales))]
min_month = months[sales.index(min(sales))]
print(f"Highest Sales: {max_month} (${max(sales)})")
print(f"Lowest Sales: {min_month} (${min(sales)})")

# 3. Change in sales and drop analysis
plt.figure(figsize=(8, 4))
sales_change = [sales[i] - sales[i-1] for i in range(1, len(sales))]
drop_months = [months[i] for i in range(1, len(sales)) if sales_change[i-1] < 0]

plt.plot(months[1:], sales_change, marker='s', linestyle='--', color='red')
plt.axhline(0, color='black', linestyle='dashed')
plt.title("Sales Change Between Months")
plt.xlabel("Month")
plt.ylabel("Change in Sales (USD)")
plt.grid(True)
plt.tight_layout()
plt.show()

print("Months with sales drop:", ", ".join(drop_months))
```

8. Create two subplots: the first showing a line plot of sales, the second showing the same data as a bar plot. Ensure each subplot has its own title and labeled axes.

```python
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Line Plot
axes[0].plot(months, sales, marker='o', color='green')
axes[0].set_title("Sales Line Plot")
axes[0].set_xlabel("Month")
axes[0].set_ylabel("Sales (USD)")

# Bar Plot
axes[1].bar(months, sales, color='purple')
axes[1].set_title("Sales Bar Plot")
axes[1].set_xlabel("Month")
axes[1].set_ylabel("Sales (USD)")

plt.tight_layout()
plt.show()
```

9. Use Seaborn to plot a boxplot showing the distribution of sales data. Label axes and add a title.

```python
import seaborn as sns
import pandas as pd

sales_df = pd.DataFrame({'Sales': sales})

plt.figure(figsize=(6, 4))
sns.boxplot(data=sales_df, x='Sales', color='skyblue')
plt.title("Distribution of Monthly Sales")
plt.xlabel("Sales (USD)")
plt.tight_layout()
plt.show()
```

10. Create a pair plot using a custom dataset with multiple sales-related features (e.g., monthly sales, customer count, profit). Analyze trends

```
# Custom dataset
data = {
    'Month': months,
    'Sales': sales,
    'Customer_Count': [200, 240, 280, 260, 310, 330],
    'Profit': [3000, 4000, 5500, 5200, 6000, 7000]
}
df = pd.DataFrame(data)

# Pairplot
sns.pairplot(df.drop(columns='Month'))
plt.suptitle("Sales Data Pair Plot", y=1.02)
plt.tight_layout()
plt.show()
```

🔍 **Insights:**

- **Highest sales:** June ($27,000)
- **Lowest sales:** January ($15,000)
- **Sales dropped** in **April** compared to March.
- Pair plot can reveal linear relationships (e.g., Sales vs. Profit or Sales vs. Customers).

Thambi good night Sweet dreams