1. Question 1:A retail store wants to identify customers who make frequent purchases. Given the dataset below, write a Python program to: Group customers by their IDs. Calculate the total purchase amount per customer. Identify the top 3 customers with the highest purchase amounts. Dataset: data = {'Customer_ID': [101, 102, 103, 101, 104, 102, 101, 105, 102, 103], 'Purchase_Amount': [200, 150, 180, 220, 300, 200, 100, 400, 250, 300]} Expected Output: Total Purchases per Customer: Customer_ID Purchase_Amount 0 101 520 1 102 600 2 103 480 3 104 300 4 105 400 Top 3 Frequent Customers: Customer_ID Purchase_Amount 1 102 600 0 101 520 2 103 480

In [1]:

```
import pandas as pd

# Dataset
data = {
    'Customer_ID': [101, 102, 103, 101, 104, 102, 101, 105, 102, 103],
    'Purchase_Amount': [200, 150, 180, 220, 300, 200, 100, 400, 250, 300]
}

# Step 1: Create DataFrame
df = pd.DataFrame(data)

# Step 2: Group customers by ID and calculate total purchase amount
total_purchases = df.groupby('Customer_ID', as_index=False)['Purchase_Amount'].sum()

print("Total Purchases per Customer:")
print(total_purchases)

# Step 3: Identify Top 3 customers with highest purchase amounts
top3_customers = total_purchases.sort_values(by='Purchase_Amount', ascending=False).head(3)

print("\nTop 3 Frequent Customers:")
print(top3_customers)
```

```
Total Purchases per Customer:
   Customer_ID  Purchase_Amount
0          101              520
1          102              600
2          103              480
3          104              300
4          105              400

Top 3 Frequent Customers:
   Customer_ID  Purchase_Amount
1          102              600
0          101              520
2          103              480
```

#2. Predicting House Prices with Linear Regression A real estate company wants to predict house prices based on square footage. Write a Python program to: Train a Linear Regression model. Predict house prices for given test data. Dataset: data = {'Square_Feet': [1500, 2000, 2500, 3000, 3500], 'Price': [300000, 400000, 500000, 600000, 700000]} Test Data: [[1800], [2800]] Expected Output: Predicted Prices: [360000. 560000.]

In [2]:

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Step 1: Create the dataset
data = {
    'Square_Feet': [1500, 2000, 2500, 3000, 3500],
    'Price': [300000, 400000, 500000, 600000, 700000]
}

# step 2: Convert the dictionary into a DataFrame
df = pd.DataFrame(data)

# Step 3: Define features (X) and target (y)
X = df[['Square_Feet']]   # must be 2D for sklearn
y = df['Price']
```

```
# Step 4: Create and train the Linear Regression model
model = LinearRegression()
model.fit(X, y)

# Step 5: Predict prices for the given test data
test_data = [[1800], [2800]]
predicted_prices = model.predict(test_data)

# Step 6: Print predicted prices
print("Predicted Prices:")
print(predicted_prices)
```

```
Predicted Prices:
[360000. 560000.]
C:\Anaconda3\envs\dineshML\Lib\site-packages\sklearn\base.py:465: UserWarning: X does not have
valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

#3: Identifying Frequent Labels in a Dataset A company wants to identify the top 3 most common categories in a dataset. Given the dataset below, write a Python program to: Group the data by Category. Count the total occurrences of each category. Identify the top 3 most frequent categories. Dataset: data = {'Category': ['A', 'B', 'C', 'A', 'D', 'B', 'A', 'E', 'B', 'C', 'C', 'A'], 'Value': [10, 15, 20, 30, 25, 18, 22, 40, 35, 50, 45, 15]} Expected Output: Total Occurrences per Category: Category Count 0 A 4 1 B 3 2 C 3 3 D 1 4 E 1 Top 3 Frequent Categories: Category Count 0 A 4 1 B 3 2 C 3

In [3]:

```
import pandas as pd

# Step 1: Create the dataset
data = {
    'Category': ['A', 'B', 'C', 'A', 'D', 'B', 'A', 'E', 'B', 'C', 'C', 'A'],
    'Value': [10, 15, 20, 30, 25, 18, 22, 40, 35, 50, 45, 15]
}

# Step 2: Convert the dictionary into a DataFrame
df = pd.DataFrame(data)

# Step 3: Group by 'Category' and count occurrences
category_counts = df.groupby('Category', as_index=False).size()

# Step 4: Rename the column for clarity
category_counts.rename(columns={'size': 'Count'}, inplace=True)

# Step 5: Print total occurrences per category
print("Total Occurrences per Category:")
print(category_counts)

# Step 6: Sort by 'Count' in descending order and select top 3
top3_categories = category_counts.sort_values(by='Count', ascending=False).head(3)

# Step 7: Print top 3 frequent categories
print("\nTop 3 Frequent Categories:")
print(top3_categories)
```

```
Total Occurrences per Category:
  Category  Count
0        A      4
1        B      3
2        C      3
3        D      1
4        E      1

Top 3 Frequent Categories:
  Category  Count
0        A      4
1        B      3
2        C      3
```

#4. Predicting Missing Values Using Mean Imputation A dataset contains missing values in the Age column. Write a Python program to: Replace missing values with the mean of the column. Display the updated DataFrame. Dataset: data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'], 'Age': [25, 30, None, 35, None]} Expected Output: Original Data: Name Age 0 Alice 25.0 1 Bob 30.0 2 Charlie NaN 3 David 35.0 4 Eve NaN Data after Imputation: Name Age 0 Alice 25.0 1 Bob 30.0 2 Charlie 30.0 3 David 35.0 4 Eve 30.0

In [4]:

```python
import pandas as pd

# Step 1: Create the dataset
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Age': [25, 30, None, 35, None]
}

# Step 2: Convert to DataFrame
df = pd.DataFrame(data)

# Step 3: Print original data
print("Original Data:")
print(df)

# Step 4: Calculate mean of 'Age' column (ignores NaN automatically)
mean_age = df['Age'].mean()

# Step 5: Safely fill missing values (no future warnings)
df['Age'] = df['Age'].fillna(mean_age)

# Step 6: Print updated DataFrame
print("\nData after Imputation:")
print(df)
```

```
Original Data:
      Name   Age
0    Alice  25.0
1      Bob  30.0
2  Charlie   NaN
3    David  35.0
4      Eve   NaN

Data after Imputation:
      Name   Age
0    Alice  25.0
1      Bob  30.0
2  Charlie  30.0
3    David  35.0
4      Eve  30.0
```

5: Implementing a Simple Linear Regression Model You are given a dataset with Experience (years) and Salary ( ). $Write a Python program to: Train a Linear Regression model. Predict the salary for an individual with 6 years of experience. Dataset$: $import pandas as pd data = 'Experience' : [1, 2, 3, 4, 5], 'Salary' : [30000, 35000, 40000, 45000, 50000] df = pd$. $DataFrame(data) Expected Output (Example) : Predicted Salary for 6 years of experience :$

55000 data = {'word_count': [100, 150, 200, 120, 180, 220], 'is_spam': ['ham', 'spam', 'spam', 'ham', 'spam', 'spam']} df = pd.DataFrame(data)
Expected Output (Example): Prediction for email with 200 words: Spam

In [5]:

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Dataset
data = {'Experience': [1, 2, 3, 4, 5],
        'Salary': [30000, 35000, 40000, 45000, 50000]}
df = pd.DataFrame(data)

# Step 1: Prepare features (X) and target (y)
X = df[['Experience']]   # 2D array for sklearn
y = df['Salary']

# Step 2: Train the Linear Regression model
model = LinearRegression()
model.fit(X, y)

# Step 3: Predict salary for 6 years of experience
predicted_salary = model.predict([[6]])

# Step 4: Print prediction
print(f"Predicted Salary for 6 years of experience: ${int(predicted_salary[0])}")
```

```
Predicted Salary for 6 years of experience: $55000
C:\Anaconda3\envs\dineshML\Lib\site-packages\sklearn\base.py:465: UserWarning: X does not have
valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

In [6]:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression

# Dataset
data = {'word_count': [100, 150, 200, 120, 180, 220],
        'is_spam': ['ham', 'spam', 'spam', 'ham', 'spam', 'spam']}
df = pd.DataFrame(data)

# Step 1: Prepare features (X) and target (y)
X = df[['word_count']]    # feature
y = df['is_spam']         # labels

# Step 2: Train the Logistic Regression model
spam_model = LogisticRegression()
spam_model.fit(X, y)

# Step 3: Predict for a new email with 200 words
prediction = spam_model.predict([[200]])

# Step 4: Print prediction
print(f"Prediction for email with 200 words: {prediction[0].capitalize()}")
```

```
Prediction for email with 200 words: Spam
C:\Anaconda3\envs\dineshML\Lib\site-packages\sklearn\base.py:465: UserWarning: X does not have
valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```

In [ ]: