1. How do you install NumPy?

```
pip install numpy
```

2. Load the dataset and preview the first few rows to understand its structure.

```
import numpy as np
data = np.genfromtxt('/mnt/data/your_dataset.csv', delimiter=',', dtype=None, encoding='utf-8',
        skip_header=1)
print(data[:5])  # Preview first 5 rows
```

3. Determine the number of rows and columns in the dataset.

```
print(data.shape)
```

4. Calculate the minimum, maximum, mean, and standard deviation for numerical columns.

```
item_weights = data[:, 1].astype(float)
print("Min:", np.min(item_weights))
print("Max:", np.max(item_weights))
print("Mean:", np.mean(item_weights))
print("Standard Deviation:", np.std(item_weights))
```

5. Count the number of unique values in a categorical column like Item_Type.

```
unique_item_types = np.unique(data[:, 4])
print(f"Unique Item Types: {len(unique_item_types)}")
```

6. Filter and display all rows where Item_Fat_Content is Low Fat.

```
low_fat_items = data[data[:, 2] == 'Low Fat']
print(low_fat_items[:5])  # Display first 5 rows
```

7. Extract only the Item_MRP and Item_Outlet_Sales columns.

```
item_mrp_sales = data[:, [5, 11]].astype(float)
print(item_mrp_sales[:5])
```

8. Find the difference between the maximum and minimum item weights.

```
item_weight_range = np.max(item_weights) - np.min(item_weights)
print(f"Range of Item Weights: {item_weight_range}")
```

9. Count how many items are sold in each Outlet_Type.

```
outlet_types, counts = np.unique(data[:, 10], return_counts=True)
```

```
print(f"Items per Outlet Type: {dict(zip(outlet_types, counts))}")
```

10. Identify the items with the highest and lowest Item_Outlet_Sales.

```
max_sales_index = np.argmax(data[:, 11].astype(float))
min_sales_index = np.argmin(data[:, 11].astype(float))
print(f"Item with Max Sales: {data[max_sales_index]}")
print(f"Item with Min Sales: {data[min_sales_index]}")
```

11. Check if any column has missing values.

```
missing_values = np.isnan(data.astype(str))
print(f"Missing values in each column: {np.sum(missing_values, axis=0)}")
```

12. Calculate the total sales amount in the dataset.

```
total_sales = np.sum(data[:, 11].astype(float))
print(f"Total Sales: {total_sales}")
```

13. Count the number of rows with missing values in the Item_Weight column.

```
missing_weights = np.sum(data[:, 1] == '')
```

```
print(f"Rows with Missing Item_Weight: {missing_weights}")
```

14. Find the items with the highest Item_MRP value.

```
max_mrp = np.max(data[:, 5].astype(float))
items_with_max_mrp = data[data[:, 5].astype(float) == max_mrp]
print(items_with_max_mrp)
```

15. Calculate the average sales amount for each outlet.

```
outlet_ids = data[:, 6]
sales_per_outlet = np.array([np.mean(data[outlet_ids == outlet, 11].astype(float)) for outlet in np.unique(outlet_ids)])
print(sales_per_outlet)
```

16. Identify the top 5 outlets with the highest total sales.

```
outlet_sales = np.array([np.sum(data[outlet_ids == outlet, 11].astype(float)) for outlet in np.unique(outlet_ids)])
top_5_outlets_indices = np.argsort(outlet_sales)[-5:]
top_5_outlets = np.unique(outlet_ids)[top_5_outlets_indices]
```

```
print("Top 5 Outlets by Sales:", top_5_outlets)
```

17. Analyze how sales have grown over the years.

```
years = data[:, 7].astype(int)
sales_by_year = np.array([np.sum(data[years == year, 11].astype(float)) for year in np.unique(years)])
print("Sales Growth Over Years:", sales_by_year)
```

18. Identify outliers in Item_MRP.

```
item_mrp = data[:, 5].astype(float)
z_scores = (item_mrp np.mean(item_mrp)) / np.std(item_mrp)
outliers = data[np.abs(z_scores) > 3]
print("Outliers in Item MRP:", outliers)
```

19. Calculate total sales based on Outlet_Location_Type.

```
location_types = data[:, 9]
sales_by_location = np.array([np.sum(data[location_types == location, 11].astype(float)) for location in
        np.unique(location_types)])
print("Sales by Outlet Location Type:", sales_by_location)
```

20. Calculate the average Item_Visibility for each Item_Type.

```
item_types = data[:, 4]
item_visibility = data[:, 3].astype(float)
avg_visibility = np.array([np.mean(item_visibility[item_types == item_type]) for item_type in
        np.unique(item_types)])
print("Average Item Visibility per Item Type:", avg_visibility)
```

21. Calculate the percentage contribution of each item to the total sales.

```
item_ids = data[:, 0]
item_sales = np.array([np.sum(data[item_ids == item, 11].astype(float)) for item in
        np.unique(item_ids)])
total_sales = np.sum(item_sales)
percentage_contribution = (item_sales / total_sales) * 100
print("Percentage Contribution to Total Sales:", percentage_contribution)
```

22. Analyze the total sales by different outlet types.

```
outlet_types = data[:, 10]
```

```
sales_by_outlet_type = np.array([np.sum(data[outlet_types == outlet_type, 11].astype(float)) for
          outlet_type in np.unique(outlet_types)])
print("Sales by Outlet Type:", sales_by_outlet_type)
```

23. Identify the item type with the highest total sales.

```
    item_types = data[:, 4]
    sales_by_item_type = np.array([np.sum(data[item_types == item_type, 11].astype(float)) for
          item_type in np.unique(item_types)])
    most_popular_item_type = np.unique(item_types)[np.argmax(sales_by_item_type)]
    print("Most Popular Item Type by Sales:", most_popular_item_type)
```

24. Analyze sales trends over months to identify any seasonal patterns.

```
dates = np.array([date.split('-')[1] for date in data[:, 7]])  # Extract month
sales_by_month = np.array([np.sum(data[dates == month, 11].astype(float)) for month in
          np.unique(dates)])
print("Sales by Month (Seasonality Analysis):", sales_by_month)
```

25. Identify items with the highest profit margin if cost data is available.

```
item_cost = np.random.uniform(50, 200, size=len(data))  # Assuming random cost data
item_sales = data[:, 11].astype(float)
profit_margin = item_sales item_cost
```

26. Calculate the average Item_Outlet_Sales for each Outlet_Size.

```
outlet_sizes = np.unique(data[:, 8])
avg_sales_per_size = np.array([np.mean(data[data[:, 8] == size, 11].astype(float)) for size in
        outlet_sizes])
print(f"Average Sales per Outlet Size: {dict(zip(outlet_sizes, avg_sales_per_size))}")
```

27. Explore how Item_Visibility affects Item_Outlet_Sales.

```
import matplotlib.pyplot as plt
item_visibility = data[:, 3].astype(float)
item_sales = data[:, 11].astype(float)
plt.scatter(item_visibility, item_sales)
plt.xlabel('Item Visibility')
plt.ylabel('Item Outlet Sales')
plt.title('Relationship between Item Visibility and Sales')
plt.show()
```

28. Sum up the total sales for each Item_Type.

```
item_types = np.unique(data[:, 4])
total_sales_by_type = np.array([np.sum(data[data[:, 4] == item_type, 11].astype(float)) for item_type in item_types])
print(f"Total Sales by Item Type: {dict(zip(item_types, total_sales_by_type))}")
```

29. Calculate the median Item_MRP for each Outlet_Type.

```
outlet_types = np.unique(data[:, 10])
median_mrp_by_outlet = np.array([np.median(data[data[:, 10] == outlet_type, 5].astype(float)) for outlet_type in outlet_types])
print(f"Median Item MRP by Outlet Type: {dict(zip(outlet_types, median_mrp_by_outlet))}")
```

30. Find the most frequently occurring Item_Fat_Content category.

```
fat_content, counts = np.unique(data[:, 2], return_counts=True)
most_common_fat_content = fat_content[np.argmax(counts)]
print(f"Most Common Item Fat Content: {most_common_fat_content}")
```