Project Title	**Supermart Grocery Sales - Retail Analytics Dataset**
Tools	**Python, ML**
Domain	**Data Analyst & Data Scientist**
Project Difficulties Level	**intermediate**

Dataset: The dataset is available at the given link.

Click here to download data set

About Dataset: This fictional dataset is created to help data analysts practice exploratory data analysis and data visualization. It contains information on customer orders placed through a grocery delivery application, with the assumption that all customers are based in Tamil Nadu, India.

Step 1: Import Libraries: Import necessary Python libraries for data handling, visualization, and evaluation metrics.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

Step 2: Load Dataset: Load the grocery sales dataset from a CSV file into a pandas DataFrame.

```
df = pd.read_csv('dataset.csv')
```

Step 3: View Data Structure: Display data types, non-null counts, and memory usage of the dataset.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Order ID      9994 non-null   object 
 1   Customer Name  9994 non-null   object 
 2   Category      9994 non-null   object 
 3   Sub Category   9994 non-null   object 
 4   City          9994 non-null   object 
 5   Order Date    9994 non-null   object 
 6   Region        9994 non-null   object 
 7   Sales         9994 non-null   int64  
 8   Discount      9994 non-null   float64
 9   Profit        9994 non-null   float64
 10  State         9994 non-null   object 
dtypes: float64(2), int64(1), object(8)
memory usage: 859.0+ KB
```

Step 4: Convert Date Column: Convert the 'Order Date' column to datetime format while handling errors.

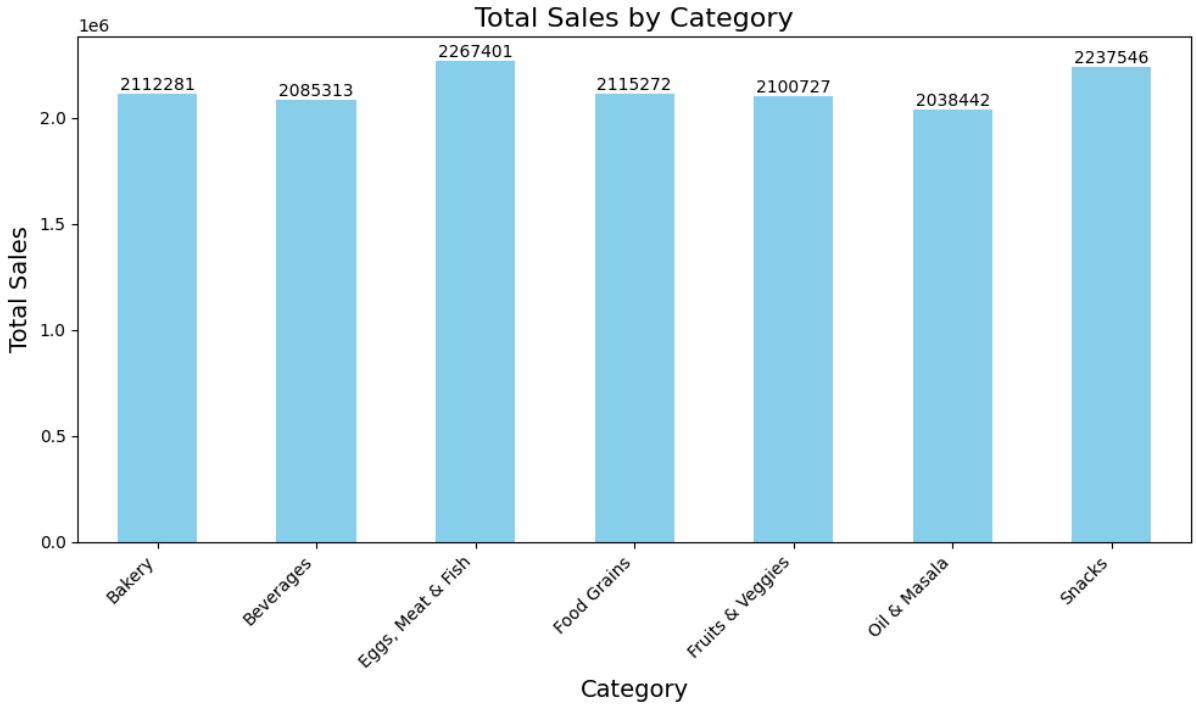
```
df['Order Date'] = pd.to_datetime(df['Order Date'], errors='coerce')
```

Step 5: Extract Date Features: Extract month number, month name, and year from the order date.

```
df['month_no'] = df['Order Date'].dt.month
df['Month'] = df['Order Date'].dt.strftime('%B')
df['year'] = df['Order Date'].dt.year
```

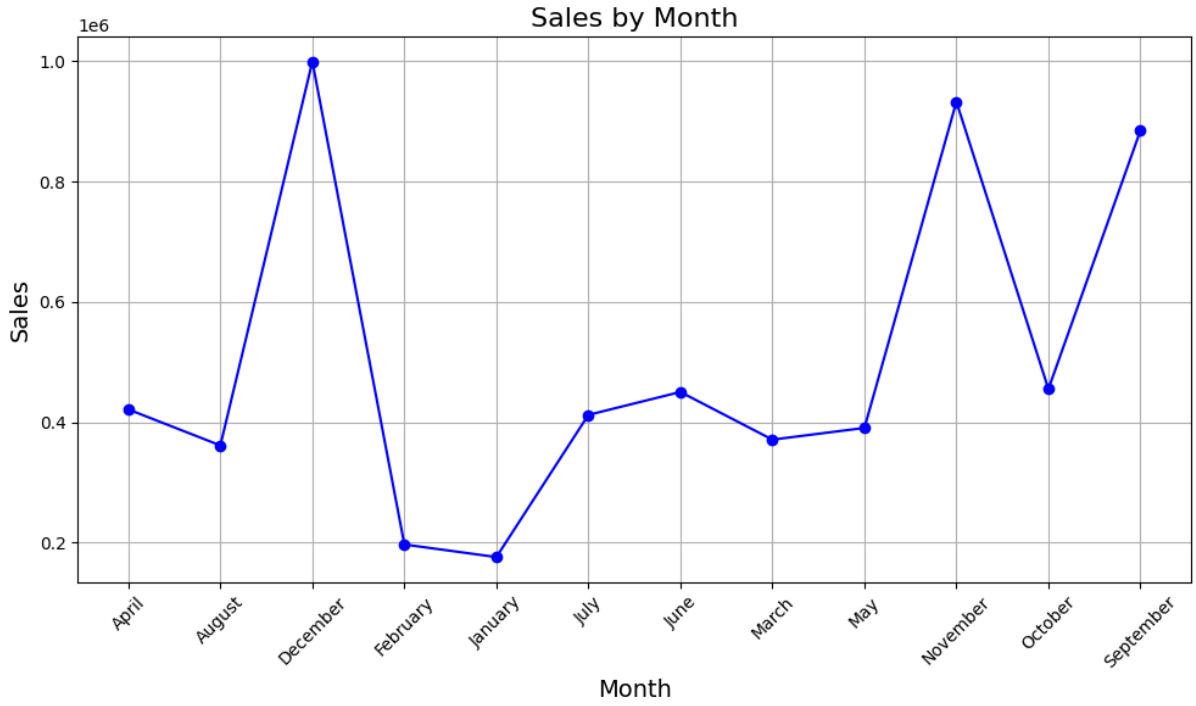
Step 6: Total Sales by Category: Group and visualize total sales per category using a bar chart.

```
sales_category = df.groupby("Category")["Sales"].sum()
plt.figure(figsize=(10, 6))
bars = sales_category.plot(kind='bar', color='skyblue')
plt.title('Total Sales by Category', fontsize=16)
plt.xlabel('Category', fontsize=14)
plt.ylabel('Total Sales', fontsize=14)
plt.xticks(rotation=45, ha='right')
for bar in bars.patches:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 2), ha='center', va='bottom')
plt.tight_layout()
plt.show()
```



Step 7: Sales by Month: Group and plot total sales for each month using a line chart.

```
monthly_sales = df.groupby('Month')['Sales'].sum().reset_index()

monthly_sales_sorted = monthly_sales.sort_values(by='Month')

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

plt.plot(monthly_sales_sorted['Month'], monthly_sales_sorted['Sales'], marker='o',
color='b')

plt.title('Sales by Month', fontsize=16)

plt.xlabel('Month', fontsize=14)

plt.ylabel('Sales', fontsize=14)

plt.xticks(rotation=45)

plt.grid(True)

plt.tight_layout()

plt.show()
```



Step 8: Sales by Year: Display yearly sales distribution using a pie chart.

yearly_sales = df.groupby("year")["Sales"].sum()

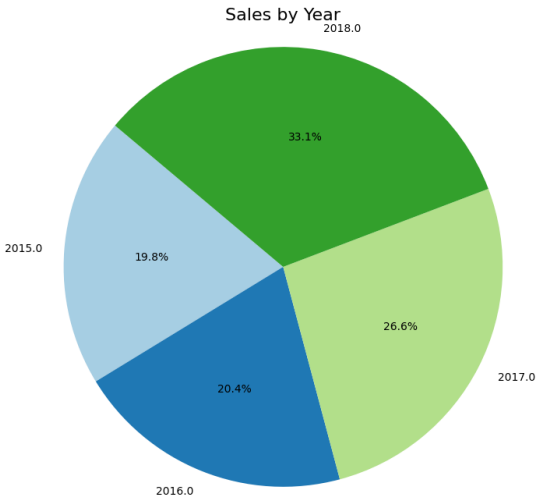
plt.figure(figsize=(8, 8))

plt.pie(yearly_sales, labels=yearly_sales.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired.colors)

plt.title('Sales by Year', fontsize=16)

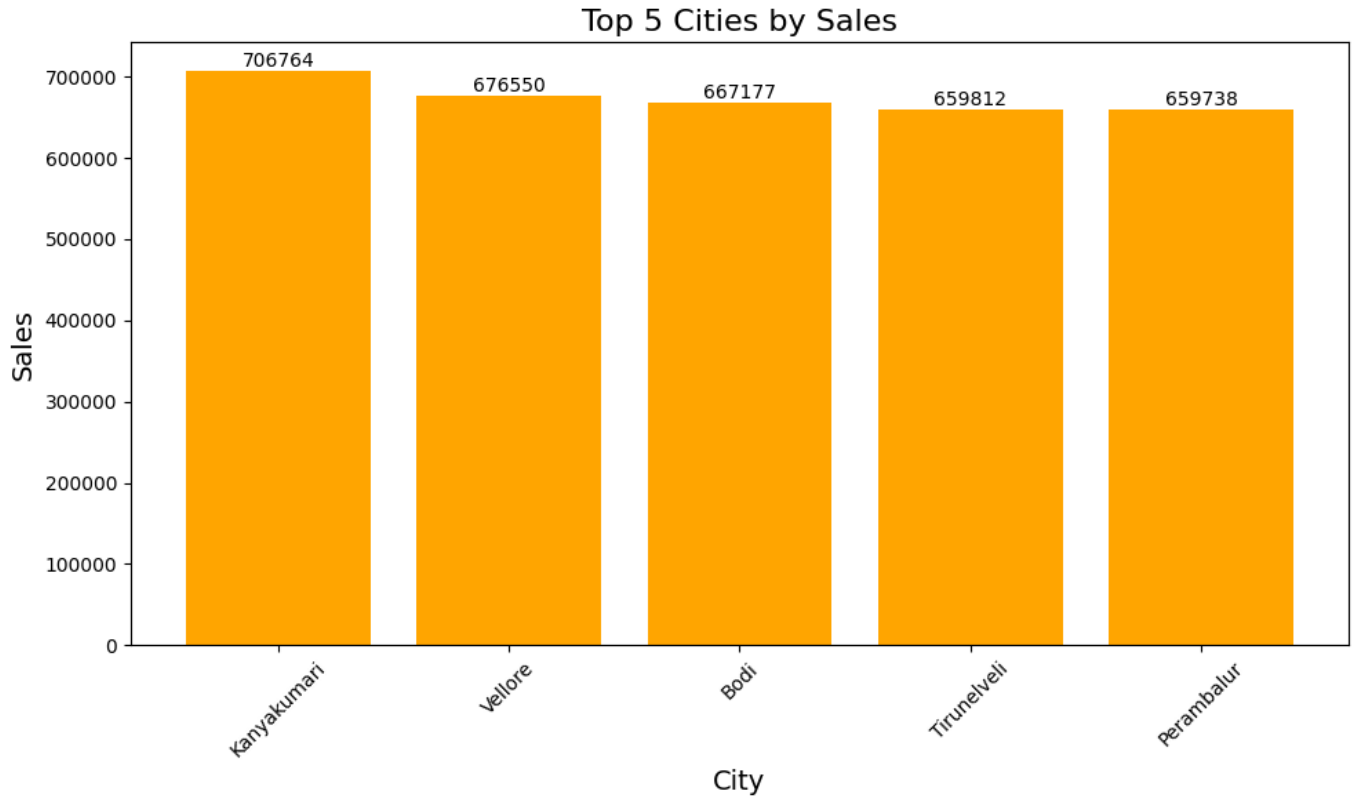
plt.axis('equal')

plt.show()



Step 9: Top 5 Cities by Sales: Identify and visualize the top 5 cities with the highest total sales.

```
top_cities = df.groupby('City')["Sales"].sum().sort_values(ascending=False).head(5)
plt.figure(figsize=(10, 6))
bars = plt.bar(top_cities.index, top_cities.values, color='orange')
plt.xlabel('City', fontsize=14)
plt.ylabel('Sales', fontsize=14)
plt.title('Top 5 Cities by Sales', fontsize=16)
plt.xticks(rotation=45)
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 2), ha='center', va='bottom')
plt.tight_layout()
plt.show()
```



Step 10: Simulate Predictions and Evaluate Performance: Simulate sales predictions by adding random noise, evaluate model accuracy using MSE, MAE, and R^2 metrics, and print the results.

```
y_true = df['Sales'][:100]

y_pred = y_true + np.random.normal(0, 100, size=len(y_true))

mse = mean_squared_error(y_true, y_pred)

mae = mean_absolute_error(y_true, y_pred)

r2 = r2_score(y_true, y_pred)

print(f"Mean Squared Error: {mse}")

print(f"Mean Absolute Error: {mae}")

print(f"R-squared Score: {r2}")
```

```
Mean Squared Error: 8678.300772701677

Mean Absolute Error: 73.04073482757248

R-squared Score: 0.971398259074341
```

Conclusion:

This project provided a comprehensive analysis of grocery sales data using various data visualization techniques. We explored sales trends across different product categories, months, years, and top-performing cities. The visualizations helped identify which categories and cities contributed most to sales. Additionally, we simulated predictions and evaluated model performance using standard metrics like MSE, MAE, and R-squared. Overall, the project demonstrated how exploratory data analysis can uncover valuable business insights and support data-driven decision-making.