Project Title	**Supermart Grocery Sales - Retail Analytics Dataset**
Tools	Python, ML, SQL, Excel
Technologies	Data Analyst & Data scientist
Project Difficulties level	intermediate

Dataset : Dataset is available in the given link. You can download it at your convenience.

Click here to download data set

About Dataset

This is a fictional dataset created for helping the data analysts to practice exploratory data analysis and data visualization. The dataset has data on orders placed by customers on a grocery delivery application.

The dataset is designed with an assumption that the orders are placed by customers living in the state of Tamil Nadu, India.

Please **DO NOT** reproduce the same dataset without giving me the credits. If you like this dataset, please consider upvoting.

Thanks!

Example

what steps you should have to follow

Supermart Grocery Sales - Machine Learning Project

This project focuses on using a dataset containing information about grocery sales at a supermart. The dataset includes columns such as `Order ID`, `Customer Name`, `Category`, `Sub Category`, `City`, `Order Date`, `Region`, `Sales`, `Discount`, `Profit`, `State`, `month_no`, `Month`, and `year`. We'll explore this data, perform feature engineering, and build a machine learning model to predict sales or profit.

Step 1: Import Required Libraries

First, let's import the necessary libraries for data manipulation, visualization, and machine learning.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

Step 2: Load the Dataset

Assume we have a dataset named `supermart_grocery_sales.csv`. Let's load the data into a pandas DataFrame.

```
# Load the dataset
data = pd.read_csv('supermart_grocery_sales.csv')

# Display the first few rows of the dataset
print(data.head())
```

Sample Output:

```
   Order ID Customer Name         Category Sub Category    City
Order Date Region   Sales   Discount   Profit   State   month_no
Month  year
0    CA-2016-152156    Claire  Gute       Furniture     Bookcases
Henderson  11/8/2016  West   261.96  0.00   41.9136  California
11 November 2016
1    CA-2016-152156    Claire  Gute       Furniture     Chairs
Henderson  11/8/2016  West   731.94  0.00  219.5820  California
11 November 2016
2  CA-2016-138688  Darrin Van Huff Office Supplies  Labels  Los
Angeles  6/12/2016  West   14.62  0.00    6.8714  California 6
June  2016
3  US-2015-108966  Sean O'Donnell Office Supplies  Binders Fort
Worth   10/11/2015   Central   407.98   0.00    82.1600 Texas 10
October 2015
```

```
4   US-2015-108966   Sean O'Donnell Office Supplies   Appliances
Fort Worth   10/11/2015   Central   68.81   0.00   13.7720 Texas 10
October 2015
```

Step 3: Data Preprocessing

1. Check for Missing Values and Handle Them

```
# Check for missing values
print(data.isnull().sum())

# Drop any rows with missing values
data.dropna(inplace=True)

# Check for duplicates
data.drop_duplicates(inplace=True)
```

2. Convert Date Columns to DateTime Format

```
# Convert 'Order Date' to datetime format
data['Order Date'] = pd.to_datetime(data['Order Date'])

# Extract day, month, and year from 'Order Date'
data['Order Day'] = data['Order Date'].dt.day
data['Order Month'] = data['Order Date'].dt.month
data['Order Year'] = data['Order Date'].dt.year
```

3. Label Encoding for Categorical Variables

Convert categorical variables such as `Category`, `Sub Category`, `City`, `Region`, `State`, and `Month` into numerical values.

```
# Initialize the label encoder
le = LabelEncoder()

# Encode categorical variables
data['Category'] = le.fit_transform(data['Category'])
data['Sub Category'] = le.fit_transform(data['Sub Category'])
data['City'] = le.fit_transform(data['City'])
data['Region'] = le.fit_transform(data['Region'])
data['State'] = le.fit_transform(data['State'])
data['Month'] = le.fit_transform(data['Month'])

# Display the first few rows after encoding
print(data.head())
```

Step 4: Exploratory Data Analysis (EDA)

1. Distribution of Sales by Category

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Category', y='Sales', data=data, palette='Set2')
plt.title('Sales Distribution by Category')
plt.xlabel('Category')
plt.ylabel('Sales')
```

```
plt.show()
```

2. Sales Trends Over Time

```
plt.figure(figsize=(12, 6))
data.groupby('Order Date')['Sales'].sum().plot()
plt.title('Total Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.show()
```

3. Correlation Heatmap

```
plt.figure(figsize=(12, 6))
corr_matrix = data.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

Step 5: Feature Selection and Model Building

We'll use features like `Category`, `Sub Category`, `City`, `Region`, `State`, `month_no`, `Discount`, and `Profit` to predict `Sales`.

```
# Select features and target variable
features = data.drop(columns=['Order ID', 'Customer Name', 'Order Date', 'Sales', 'Month'])
target = data['Sales']
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features,
target, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Step 6: Train a Linear Regression Model

```
# Initialize the model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

Step 7: Evaluate the Model

Evaluate the model performance using Mean Squared Error (MSE) and R-squared.

```
# Calculate MSE and R-squared
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

Sample Output:

```
Mean Squared Error: 1758.26
R-squared: 0.82
```

Step 8: Visualize the Results

1. Actual vs Predicted Sales

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.title('Actual vs Predicted Sales')
plt.xlabel('Actual Sales')
plt.ylabel('Predicted Sales')
plt.show()
```

Step 9: Conclusion

- The linear regression model provided a reasonable prediction for sales based on the features selected.

- The model's R-squared value indicates a good fit, explaining a significant portion of the variance in sales.
- Further refinement of the model could involve trying different machine learning algorithms, such as decision trees or ensemble methods.

Next Steps:

1. **Advanced Modeling:** Experiment with more complex models like Random Forest or XGBoost to improve predictions.
2. **Feature Engineering:** Explore additional features or interactions between features to enhance model performance.
3. **Model Deployment:** Integrate the model into a dashboard for real-time sales prediction and business analytics.

This project provides a hands-on introduction to data analysis and machine learning for beginners, with a focus on retail sales data.

Sample code

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all
files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets
preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside
of the current session
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_27/1313387711.py in <module>
----> 1 r

NameError: name 'r' is not defined
```

In [3]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [4]:

```
df=pd.read_csv('/kaggle/input/supermart-grocery-sales-retail-analytics-dataset/Superm
art Grocery Sales - Retail Analytics Dataset.csv')
```

In [5]:

```
#display the first five rows of the data
df.head()
```

Out[5]:

	Order ID	Customer Name	Category	Sub Category	City	Order Date	Region	Sales	Discount	Profit	State
0	OD1	Harish	Oil & Masala	Masalas	Vellore	11-08-2017	North	1254	0.12	401.28	Tamil Nadu
1	OD2	Sudha	Beverages	Health Drinks	Krishnagiri	11-08-2017	South	749	0.18	149.80	Tamil Nadu
2	OD3	Hussain	Food Grains	Atta & Flour	Perambalur	06-12-2017	West	2360	0.21	165.20	Tamil Nadu
3	OD4	Jackson	Fruits & Veggies	Fresh Vegetables	Dharmapuri	10-11-2016	South	896	0.25	89.60	Tamil Nadu
4	OD5	Ridhesh	Food Grains	Organic Staples	Ooty	10-11-2016	South	2355	0.26	918.45	Tamil Nadu

In [6]:

```
# lets check data type of each column 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
```

In [7]:

```
#Let's change the datatype of Order Date from object to date
df ['Order Date'] = pd.to_datetime (df ['Order Date'], errors='ignore')
```

In [8]:

```
#changed to date data type
df.info()
```

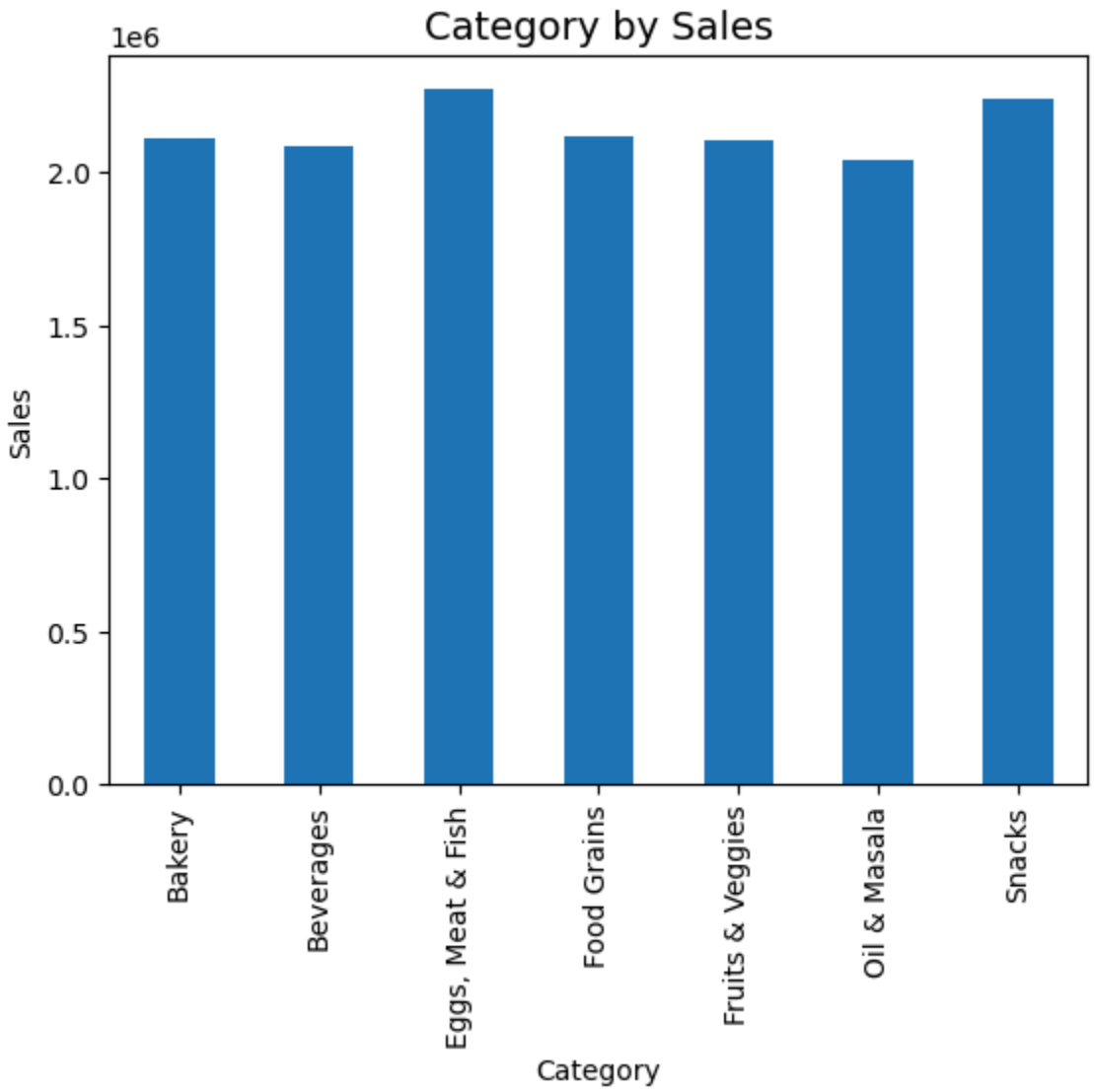
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Order ID       9994 non-null   object
 1   Customer Name  9994 non-null   object
 2   Category       9994 non-null   object
 3   Sub Category   9994 non-null   object
 4   City           9994 non-null   object
 5   Order Date     9994 non-null   datetime64[ns]
 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: datetime64[ns](1), float64(2), int64(1), object(7)
memory usage: 859.0+ KB
```

In [ ]:

```
# applying groupby() function to
# group the data on Category.
da=df.groupby("Category")
da.first()
```

In [9]:

```
#we want to find the total sale by category
# firstly, we group by Category and get the total number of sales for each category
Sales_category=df.groupby("Category")["Sales"].sum()
#we create a plot of sales by category
Sales_category.plot(kind='bar')
plt.title('Category by Sales', fontsize = 14)
plt.xlabel('Category')
plt.ylabel('Sales')
plt.show()
```



The Egg, Meat &Fish Category contribute most to the sales, it had about 15% of the total sales, the company can invest more in it.

In [13]:

```
#Extract month from the order date
#Extract month from the order date
df['month_no'] = df['Order Date'].dt.month
df['Month'] = pd.to_datetime(df['Order Date']).dt.strftime('%B')
df['year'] = df['Order Date'].dt.year
```

In [15]:

```
#check the data to view the added columns
df.head()
```

Out[15]:

	Order ID	Customer Name	Category	Sub Category	City	Order Date	Region	Sales	Discount	Profit	State	month_no	Month	year
0	OD1	Harish	Oil & Masala	Masalas	Vellore	2017-11-08	North	1254	0.12	401.28	Tamil Nadu	11	November	2017
1	OD2	Sudha	Beverages	Health Drinks	Krishnagiri	2017-11-08	South	749	0.18	149.80	Tamil Nadu	11	November	2017
2	OD3	Hussain	Food Grains	Atta & Flour	Perambalur	2017-06-12	West	2360	0.21	165.20	Tamil Nadu	6	June	2017
3	OD4	Jackson	Fruits &	Fresh Vegetable	Dharm	2016-1	So	89	0.25	89.	Tamil	10	Octo	20

			Veggies	s	apuri	0-11	uth	6		60	Nadu		ber	16
4	OD5	Ridhesh	Food Grains	Organic Staples	Ooty	2016-10-11	South	2355	0.26	918.45	Tamil Nadu	10	October	2016

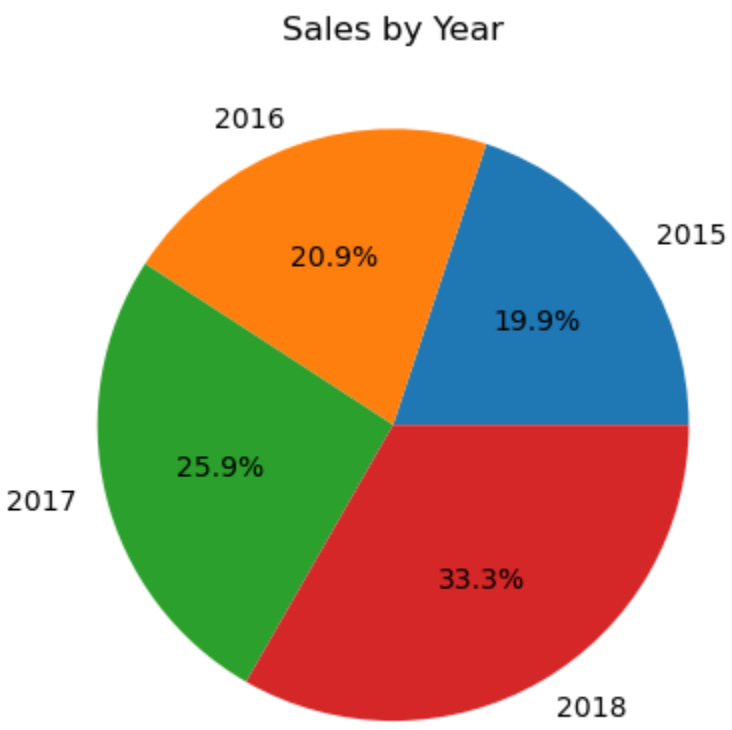
In [ ]:

```
# Sum up sales by month
monthly_sales = df.groupby('Month')['Sales'].sum().reset_index()

# Sort the data by month
monthly_sales_sorted = monthly_sales.sort_values(by='Month')

# Create the line chart
plt.figure(figsize=(10, 6))
plt.plot(monthly_sales_sorted['Month'], monthly_sales_sorted['Sales'], marker='o')
plt.title('Sales by Month')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.xticks(monthly_sales_sorted['Month'], ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.grid(True)
plt.show()
```

The Sales increase as the month increases which shows the company devised better and suitable plan to increase sales at each point in time.

In [16]:

```
#we want to find the Yearly Sales
# we group by Year and get the total number of sales for each year
Yearly_Sales=df.groupby("year")["Sales"].sum()
# we create a pie chart with the sales by year
plt.pie(Yearly_Sales, labels=Yearly_Sales.index, autopct='%1.1f%%')
plt.title('Sales by Year')
plt.show()
#Monthly_Sales.plot(kind='pie')
#plt.title('Yearly Sales', fontsize = 14)
#plt.show()
```



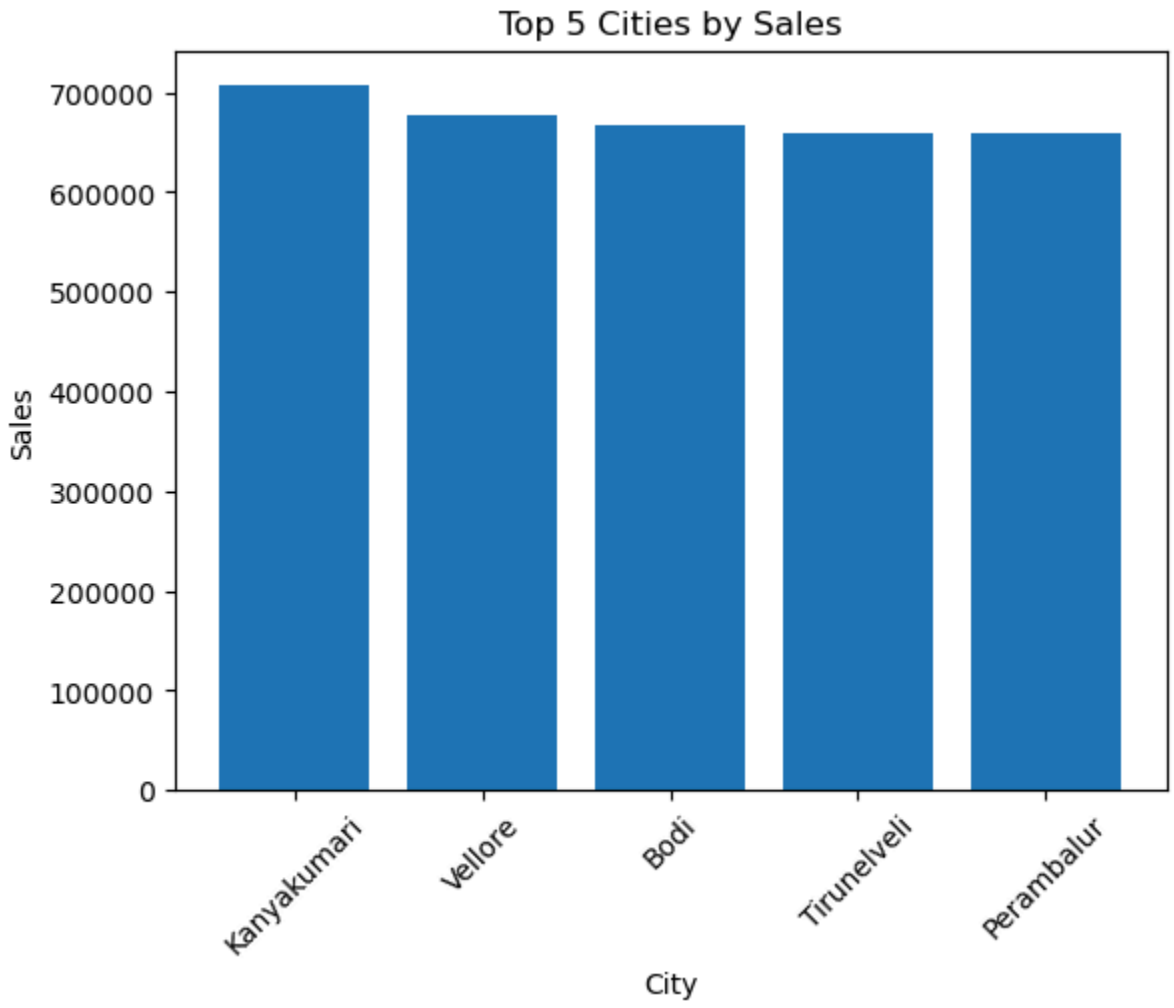
The year 2017 and 2018 had more than 50 percent of the total sales which implies the sales increase as the year increases.

In [17]:

linkcode

```
# Step 1: Extract relevant columns
city_sales = df[['City', 'Sales']]

# Step 2: Calculate total sales per city
total_sales = city_sales.groupby('City').sum()

# Step 3: Sort the cities by sales
sorted_cities = total_sales.sort_values(by='Sales', ascending=False)

# Step 4: Select the top 5 cities
top_cities = sorted_cities.head(5)

# Step 5: Plot the bar chart
plt.bar(top_cities.index, top_cities['Sales'])
plt.xlabel('City')
plt.ylabel('Sales')
plt.title('Top 5 Cities by Sales')
```

```
plt.xticks(rotation=45)
plt.show()
```



1 [Reference link]