Supermarket Grocery Project

1. Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Load datasets

In [2]:
```
df = pd.read_csv("E:\Data Analyst Project\Supermart Grocery Sales - Retail Analytic
```

In [3]:
```
df.head()
```

Out[3]:

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

In [4]:
```
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
```

3. Data Preprocessing

Handling Missing Values

In [5]:
```
print(df.isnull().sum())
```

```
Order ID         0
Customer Name    0
Category         0
Sub Category     0
City             0
Order Date       0
Region           0
Sales            0
Discount         0
Profit           0
State            0
dtype: int64
```

In [6]:
```
df.dropna(inplace = True)
```

In [7]:
```
df.drop_duplicates(inplace = True)
```

Convert Date Columns to Date Time Format

In [8]:
```
df['Order Date'] = pd.to_datetime(df['Order Date'], format='mixed', errors='coerce'
```

In [9]:
```
df['Day'] = df['Order Date'].dt.day
df['Month'] = df['Order Date'].dt.month
df['Year'] = df['Order Date'].dt.year
```

In [10]:
```
df.head()
```

Out[10]:

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

Label Encoding for Categorical Variables

In [11]:

```
from sklearn.preprocessing import LabelEncoder,StandardScaler
le = LabelEncoder()
df['State'] = le.fit_transform(df['State'])
df['Sub Category'] = le.fit_transform(df['Sub Category'])
df['City'] = le.fit_transform(df['City'])
df['Region'] = le.fit_transform(df['Region'])
```

In [12]:

```
df.head()
```

Out[12]:

	Order ID	Customer Name	Category	Sub Category	City	Order Date	Region	Sales	Discount	Profit	State	Da
0	OD1	Harish	Oil & Masala	14	21	2017-11-08	2	1254	0.12	401.28	0	
1	OD2	Sudha	Beverages	13	8	2017-11-08	3	749	0.18	149.80	0	
2	OD3	Hussain	Food Grains	0	13	2017-06-12	4	2360	0.21	165.20	0	1
3	OD4	Jackson	Fruits & Veggies	12	4	2016-10-11	3	896	0.25	89.60	0	1
4	OD5	Ridhesh	Food Grains	18	12	2016-10-11	3	2355	0.26	918.45	0	1

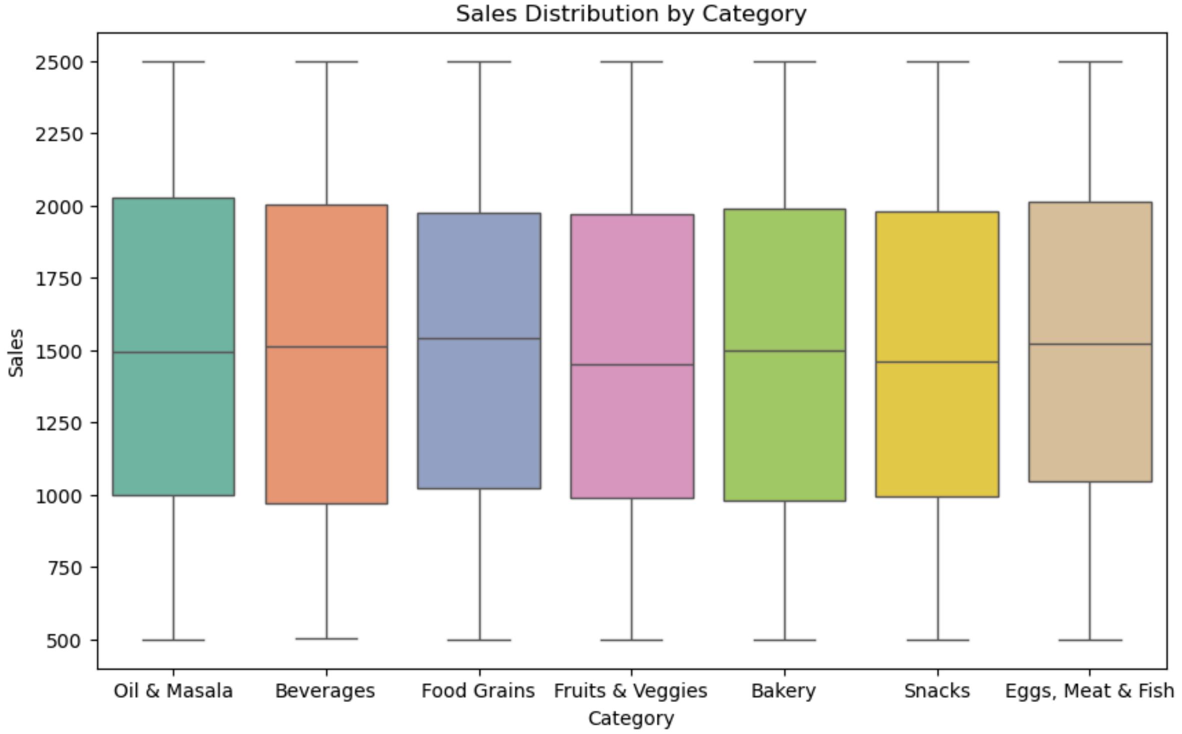
4. Exploratory Data Analysis

Sales By Category

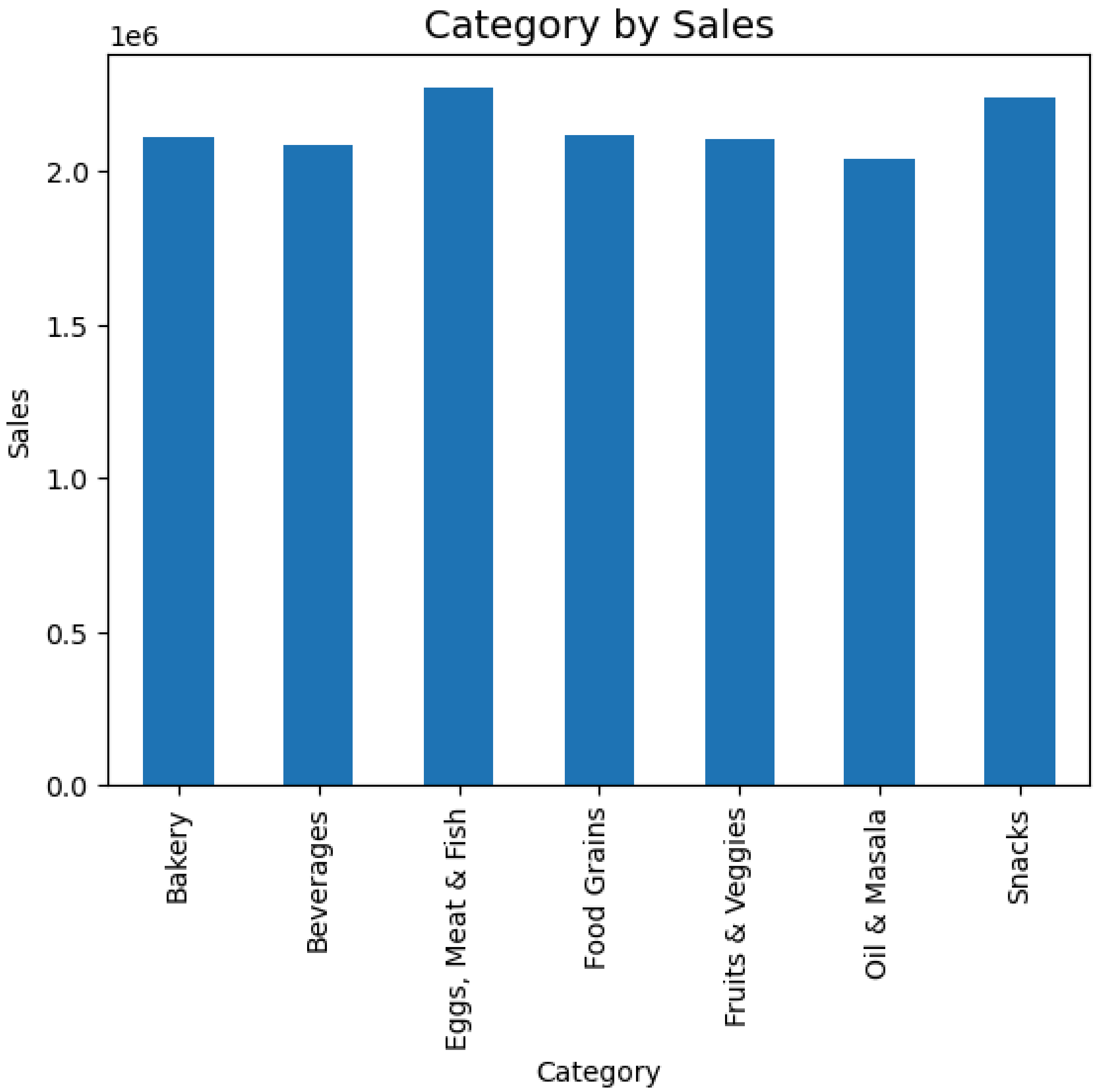
In [13]:

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

```
C:\Users\USER\AppData\Local\Temp\ipykernel_10152\3764011399.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.
14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='Category', y='Sales', data= df, palette='Set2')
```

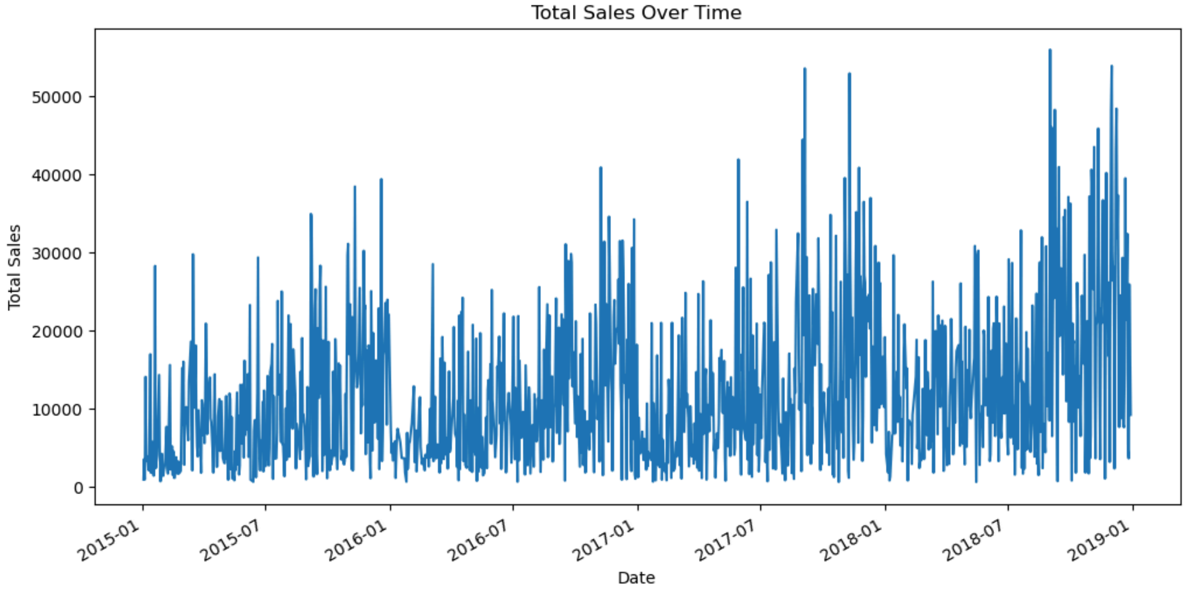


In [14]:
```
Sales_category=df.groupby("Category")["Sales"].sum()

Sales_category.plot(kind='bar')
plt.title('Category by Sales', fontsize = 14)
plt.xlabel('Category')
plt.ylabel('Sales')
plt.show()
```



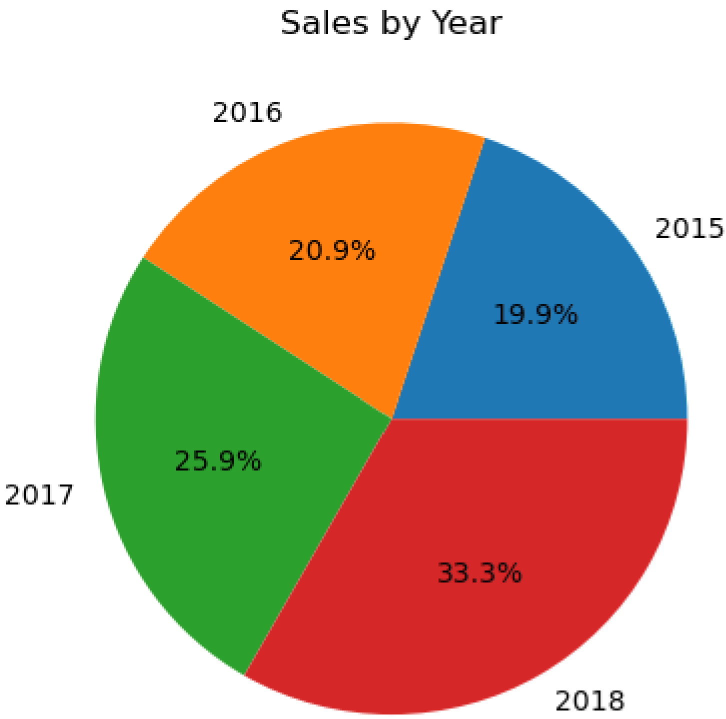
Sales Trends Over Time

In [15]:
```
plt.figure(figsize=(12, 6))
df.groupby('Order Date')['Sales'].sum().plot()
plt.title('Total Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.show()
```



Sales By Year

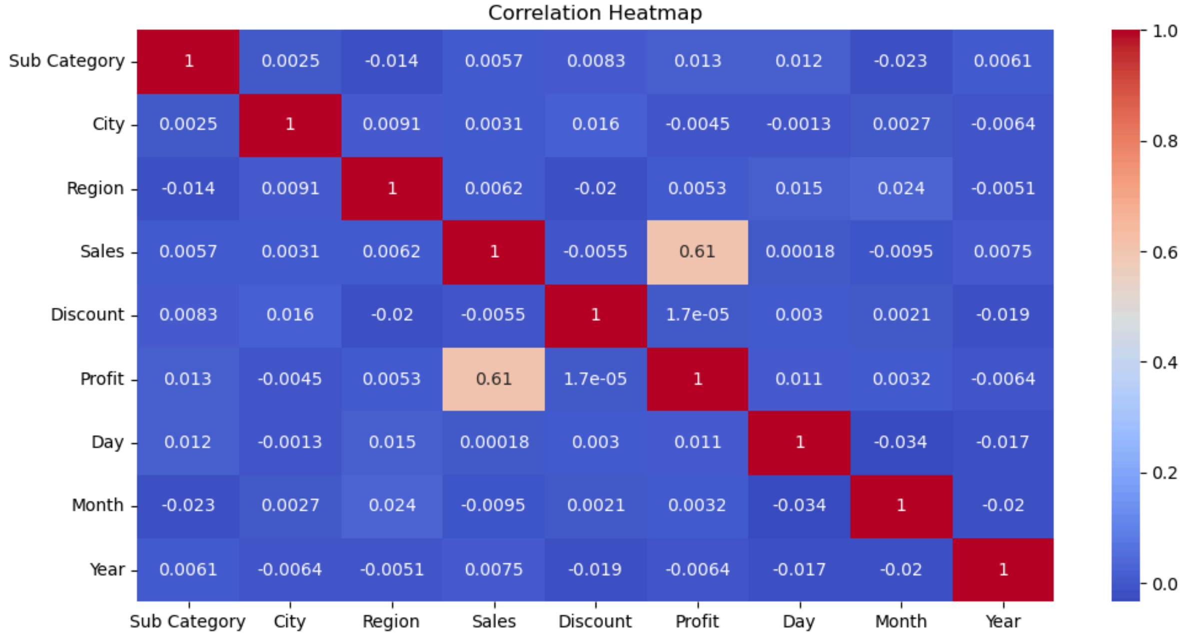
In [16]:
```
Yearly_Sales=df.groupby("Year")["Sales"].sum()

plt.pie(Yearly_Sales, labels=Yearly_Sales.index,
autopct='%1.1f%%')
plt.title('Sales by Year')
plt.show()
```



Correlation HeatMap

In [17]:
```
plt.figure(figsize=(12, 6))

# Select only numeric columns, but drop 'State'
numeric_cols = df.select_dtypes(include=['number']).drop(columns=['State'], errors=
```

```
corr_matrix = numeric_cols.corr()

sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



5.Feature Selection And Model Building

In [18]:
```
features = df.drop(columns = ['Order ID','Customer Name','Order Date','Sales','Mont
target = df['Sales']
```

In [19]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(features,target,test_size = 0.2, r
```

In [20]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Example: target is 'Sales'
X = df.drop(columns=['Sales'])   # Features
y = df['Sales']                  # Target

# Drop datetime columns
X = X.drop(columns=X.select_dtypes(include=['datetime64[ns]']).columns)

# Convert categorical columns to numeric
X = pd.get_dummies(X, drop_first=True)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Scaling numeric features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

6. Model Training And Evaluation

Linear Regression Model

In [21]:
```
# Model training
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
```

In [22]:
```
# Model Evaluation
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)

print(f"R² Score: {r2:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
```

```
R² Score: 0.2741
Root Mean Squared Error (RMSE): 489.29
Mean Absolute Error (MAE): 414.89
```

Random Forest Model

In [23]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error

import numpy as np

rf_model = RandomForestRegressor(n_estimators = 100, random_state = 42)
rf_model.fit(X_train,y_train)
rf_pred = rf_model.predict(X_test)
```

In [24]:
```
rf_r2 = r2_score(y_test,rf_pred)
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_pred))
rf_mae = mean_absolute_error(y_test,rf_pred)

print("Random Forest Performance: ")
print(f"R² Score: {rf_r2:.4f}")
print(f"RMSE: {rf_rmse:.2f}")
print(f"MAE: {rf_mae:.2f}")
```

```
Random Forest Performance:
R² Score: 0.3233
RMSE: 472.42
MAE: 376.04
```

XGBoost Regressor

In [25]:
```
from xgboost import XGBRegressor
xgb_model = XGBRegressor(n_estimators= 100, learning_rate = 0.1,random_state = 42)
xgb_model.fit (X_train,y_train)
```

Out[25]:

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds
=None,
             enable_categorical=False, eval_metric=None, feature_types
=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=
None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
```

In [27]:

```
xgb_pred = xgb_model.predict(X_test)
```

In [28]:

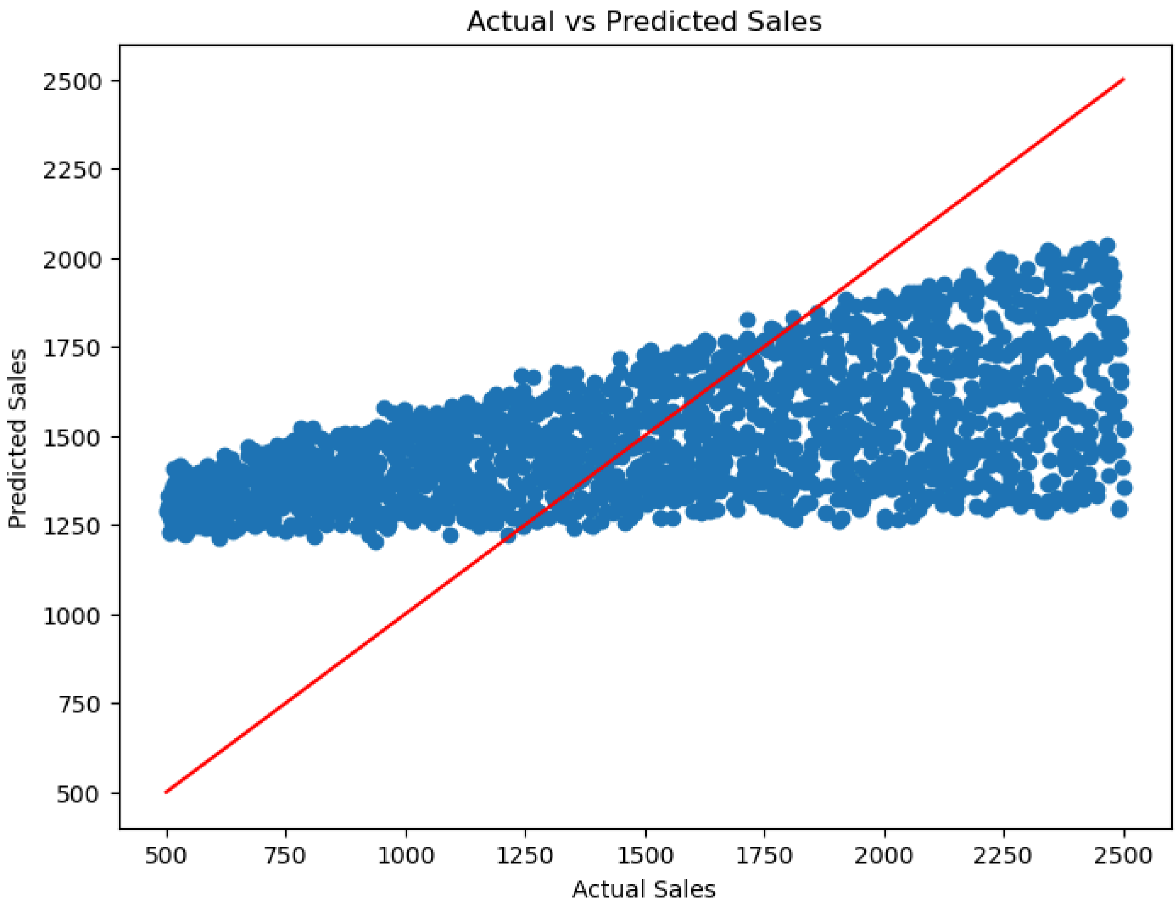
```
# Model Evaluation
xgb_r2 = r2_score(y_test, xgb_pred)
xgb_rmse = np.sqrt(mean_squared_error(y_test, xgb_pred))
xgb_mae = mean_absolute_error(y_test, xgb_pred)

print("\nXGBoost Performance:")
print(f"R² Score: {xgb_r2:.4f}")
print(f"RMSE: {xgb_rmse:.2f}")
print(f"MAE: {xgb_mae:.2f}")
```

```
XGBoost Performance:
R² Score: 0.3568
RMSE: 460.60
MAE: 377.11
```

7.Visualize Results

In [29]:

```
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()
```



8. Conclusions

1. Compare R-Squared Scores

- The model with the highest R^2 explains the most variance in target variable (Sales or Profit).
- Typically, Random Forest and XGBoost outperform Linear Regression when relationships are non-linear and there are interactions between variables.

2. Compare RMSE And MAE

- Lower RMSE and MAE mean better prediction accuracy.
- RMSE penalizes large errors more, while MAE shows the average error magnitude.
- The model with the lowest RMSE & MAE is generally preferred.

3. Handling Non - Linearity And Complex Pattern

- Linear Regression assumes a straight-line relationship between features and the target, which is often too simple for retail datasets.
- Random Forest handles non-linear patterns and feature interactions automatically.
- XGBoost also handles complex relationships and can be tuned more precisely.

4. Overfitting Consideration

- Random Forest can overfit if the number of trees is too high or depth is uncontrolled.
- XGBoost has regularization parameters to control overfitting, making it more robust when tuned.

5. Speed And Resource Use

- Linear Regression is fastest and requires least memory.
- Random Forest is slower but still manageable for medium datasets.
- XGBoost is usually the slowest but can achieve the best accuracy after tuning.

Best Model Selection

- Therefore, XGBoost is the overall the best Model as it has highest R-Squared which best at explaining variance.
- It has lowest RMSE and MAE as it gives most accurate predictions