Detailed Project Report: Walmart E-Commerce Analysis and Prediction

---

1. Problem Definition / Business Understanding

- **Objective**: This project focuses on analyzing Walmart's e-commerce data to derive actionable insights and build predictive models for sales optimization. The workflow comprises data collection, preprocessing, exploratory analysis, feature engineering, model building, deployment, and integration with visualization tools like Power BI. Each technology and technique used is detailed with its rationale and advantages.
- **Goals**:
  - Develop models to predict future sales (forecasted demand).
  - Build an interactive dashboard to visualize key insights (using Power BI).
  - Optimize the sales prediction by incorporating external features such as weather, promotions, and customer demographics.

Technologies and Tools Used

Python Libraries

1. **Pandas**: For data manipulation, handling missing values, and reshaping datasets.
2. **NumPy**: For efficient numerical computations and handling arrays.
3. **Matplotlib**: For creating static visualizations such as line charts and scatter plots.
4. **Seaborn**: For aesthetically pleasing statistical visualizations like heatmaps and boxplots.
5. **Scikit-learn**: For implementing machine learning models, data preprocessing, and model evaluation.
6. **XGBoost**: For building efficient and scalable gradient boosting regression models.
7. **SHAP**: For interpreting model predictions and understanding feature importance.
8. **Joblib**: For saving and loading machine learning models efficiently.
9. **Flask**: For creating REST APIs to deploy machine learning models for real-time predictions.

Visualization Tools

1. **Power BI**: For building interactive dashboards and integrating model predictions with business intelligence.

Development Environments

1. **Jupyter Notebook**: For developing, testing, and documenting Python code in an interactive environment.
2. **Command Prompt**: For installing Python libraries and running scripts locally.

Additional Tools

1. **Postman**: For testing APIs created with Flask by sending and verifying HTTP requests.

2. **Joblib**: For serializing and deserializing machine learning models.

Software and Platforms

1. **Python 3.11**: Programming language used for data analysis, modeling, and API development.
2. **Microsoft Windows**: Operating system for running Jupyter Notebook, Power BI, and other tools.
3. **Kaggle**: For sourcing the dataset and exploring the initial data environment.

---

2. Project Phases

Phase 1: Researching and Preparing the Dataset

Steps Taken:

1. **Problem Definition:**
    - Defined the objective to predict and analyze Walmart's sales to optimize operational strategies and customer satisfaction.
    - Focused on identifying the key factors influencing sales.
2. **Dataset Acquisition:**
    - Acquired the dataset from Kaggle (https://www.kaggle.com/datasets/ankitrajmishra/walmart).
    - Transitioned from Kaggle's coding environment to a local setup using Jupyter Notebook.
3. **Technologies Used:**
    - **Pandas:** Used for data manipulation and handling missing/null values.
        - **Advantages:** Efficient for handling large datasets, versatile for operations like filtering, grouping, and reshaping data.
    - **NumPy:** Utilized for numerical computations.
        - **Advantages:** Fast and memory-efficient for handling arrays and numerical operations.
4. **Data Cleaning:**
    - Checked for missing values and handled them through imputation strategies (e.g., mean, median for numerical fields).
    - Identified and fixed outliers using statistical methods (e.g., IQR-based filtering).
5. **Outcome:**
    - A clean and structured dataset ready for further analysis and model-building processes.

jupyter Untitled Last Checkpoint: 4 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import torch
import cv2

print("All libraries are installed and working correctly!")
```

```
All libraries are installed and working correctly!
```

In [2]:
```python
import pandas as pd

# Load the CSV file
data = pd.read_csv(r"D:\downloads\data science project\Walmart.csv")

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

# Display information about the dataset
print(data.info())
```

```
   transaction_id  customer_id  product_id product_name     category  \
0               1         2824         843       Fridge  Electronics
1               2         1409         135           TV  Electronics
2               3         5506         391       Fridge  Electronics
3               4         5012         710   Smartphone  Electronics
4               5         4657         116       Laptop  Electronics

   quantity_sold  unit_price transaction_date  store_id   store_location  ...  \
0              3      188.46  3/31/2024 21:46         3        Miami, FL  ...
1              4     1912.04  7/28/2024 12:45         5       Dallas, TX  ...
2              4     1377.75   6/10/2024 4:55         1  Los Angeles, CA  ...
3              5      182.31   8/15/2024 1:03         5        Miami, FL  ...
4              3      499.28   9/13/2024 0:45         6      Chicago, IL  ...

   customer_loyalty_level  payment_method  promotion_applied  \
0                  Silver     Credit Card               True
1                    Gold            Cash               True
2                Platinum            Cash              False
3                  Silver            Cash               True
4                  Bronze  Digital Wallet              False

       promotion_type  weather_conditions  holiday_indicator   weekday  \
0                NaN              Stormy              False    Friday
```

walmart

23 New Notebook Download

Data Card Code (0) Discussion (0) Suggestions (0)

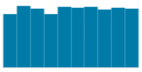
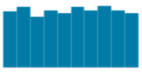
Walmart.csv (881.72 kB)

Detail Compact Column 10 of 28 columns

About this file

Add Suggestion

The dataset provides a comprehensive view of Walmart's operational and sales data, capturing details of transactions, products, customers, stores, and external factors like promotions and weather. Key attributes include **transaction details** (e.g., ID, date, quantity sold, unit price), **customer information** (e.g., loyalty level, payment method), **product details** (e.g., ID, name, category), **store specifics** (e.g., ID, location), and **external factors** (e.g., weather, holidays, promotions). The dataset supports analyses like demand forecasting, inventory management, sales performance, customer behavior, and promotion effectiveness, enabling Walmart to optimize operations and enhance decision-making.

transaction_id	customer_id	product_id	product_name	category	quantit
Unique identifier for each transaction.	Unique identifier for each customer.	Unique identifier for each product.	Name of the product.	The product category (e.g., Electronics, Furniture)	Number o each tran:
 1 5000	 1001 9998	 100 999	Fridge 13% Tablet 13% Other (3698) 74%	Electronics 52% Appliances 48%	 1
1	2824	843	Fridge	Electronics	3
2	1409	135	TV	Electronics	4
3	5506	391	Fridge	Electronics	4

Data Explorer

Version 1 (881.72 kB)

Walmart.csv

Summary

1 file

28 columns

```
   transaction_id  customer_id  product_id product_name     category  \
0               1         2824         843       Fridge  Electronics
1               2         1409         135           TV  Electronics
2               3         5506         391       Fridge  Electronics
3               4         5012         710   Smartphone  Electronics
4               5         4657         116       Laptop  Electronics

   quantity_sold  unit_price transaction_date  store_id   store_location  ...  \
0              3      188.46  3/31/2024 21:46         3        Miami, FL  ...
1              4     1912.04  7/28/2024 12:45         5       Dallas, TX  ...
2              4     1377.75   6/10/2024 4:55         1  Los Angeles, CA  ...
3              5      182.31   8/15/2024 1:03         5        Miami, FL  ...
4              3      499.28   9/13/2024 0:45         6      Chicago, IL  ...

   customer_loyalty_level  payment_method  promotion_applied  \
0                  Silver     Credit Card               True
1                    Gold            Cash               True
2                Platinum            Cash              False
3                  Silver            Cash               True
4                  Bronze  Digital Wallet              False

        promotion_type  weather_conditions  holiday_indicator   weekday  \
0                  NaN              Stormy              False    Friday
1  Percentage Discount               Rainy              False    Monday
2                  NaN               Sunny              False   Tuesday
3  Percentage Discount               Sunny               True    Sunday
4                  NaN               Sunny              False  Thursday

   stockout_indicator forecasted_demand actual_demand
0                True               172           179
1                True               109           484
2                True               289           416
3               False               174           446
4                True               287           469


[5 rows x 28 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 28 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   transaction_id          5000 non-null   int64
 1   customer_id             5000 non-null   int64
 2   product_id              5000 non-null   int64
 3   product_name            5000 non-null   object
 4   category                5000 non-null   object
 5   quantity_sold           5000 non-null   int64
 6   unit_price              5000 non-null   float64
 7   transaction_date        5000 non-null   object
 8   store_id                5000 non-null   int64
 9   store_location          5000 non-null   object
 10  inventory_level         5000 non-null   int64
 11  reorder_point           5000 non-null   int64
 12  reorder_quantity        5000 non-null   int64
 13  supplier_id             5000 non-null   int64
 14  supplier_lead_time      5000 non-null   int64
 15  customer_age            5000 non-null   int64
 16  customer_gender         5000 non-null   object
 17  customer_income         5000 non-null   float64
 18  customer_loyalty_level  5000 non-null   object
 19  payment_method          5000 non-null   object
 20  promotion_applied       5000 non-null   bool
 21  promotion_type          1593 non-null   object
 22  weather_conditions      5000 non-null   object
 23  holiday_indicator       5000 non-null   bool
 24  weekday                 5000 non-null   object
 25  stockout_indicator      5000 non-null   bool
 26  forecasted_demand       5000 non-null   int64
 27  actual_demand           5000 non-null   int64
dtypes: bool(3), float64(2), int64(13), object(10)
memory usage: 991.3+ KB
None
```

```
In [4]: print(data.describe())
```

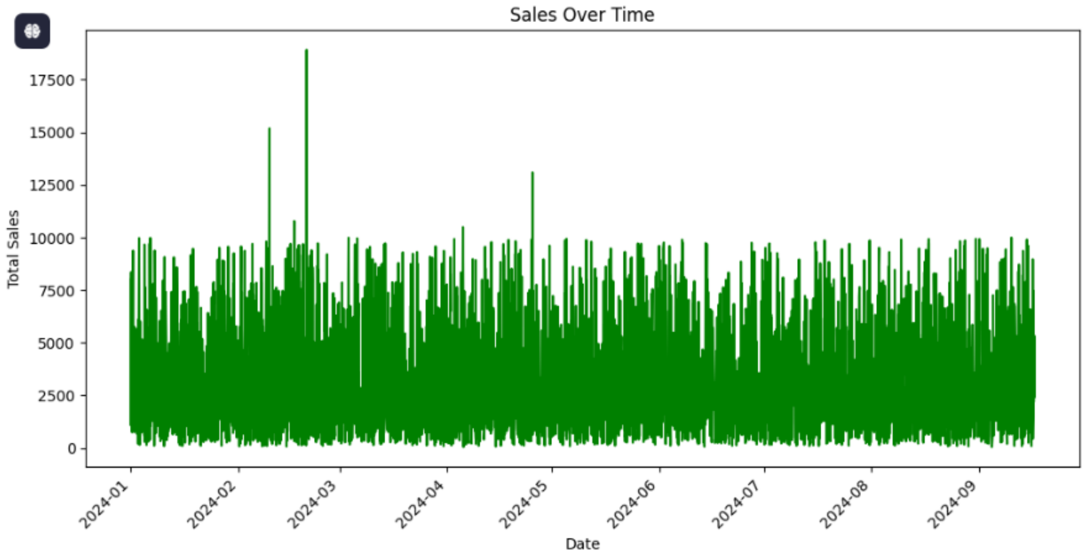
```
       transaction_id  customer_id   product_id  quantity_sold   unit_price  \
count     5000.000000  5000.000000  5000.000000    5000.000000  5000.000000   
mean      2500.500000  5542.497200   551.233400       2.982800  1023.467294   
std       1443.520003  2582.126997   258.826606       1.419474   559.614242   
min          1.000000  1001.000000   100.000000       1.000000    50.100000   
25%       1250.750000  3279.000000   322.000000       2.000000   537.775000   
50%       2500.500000  5558.000000   559.000000       3.000000  1029.175000   
75%       3750.250000  7767.250000   776.000000       4.000000  1506.307500   
max       5000.000000  9998.000000   999.000000       5.000000  1999.850000   

          store_id  inventory_level  reorder_point  reorder_quantity  \
count  5000.000000      5000.000000    5000.000000       5000.000000   
mean     10.525000       253.121800      99.788000        200.517000   
std       5.786888       142.885456      29.132387         58.257381   
min       1.000000         0.000000      50.000000        100.000000   
25%       5.000000       130.000000      75.000000        150.750000   
50%      11.000000       253.000000     100.000000        200.500000   
75%      16.000000       377.250000     125.000000        251.000000   
max      20.000000       500.000000     150.000000        300.000000   

       supplier_id  supplier_lead_time  customer_age  customer_income  \
count   5000.00000         5000.000000   5000.000000      5000.000000   
mean     300.12560            5.523000     44.124000     70041.627846   
std      116.39486            2.863549     15.329358     29053.371736   
min      100.00000            1.000000     18.000000     20005.340000   
25%      199.00000            3.000000     31.000000     44865.417500   
50%      299.00000            6.000000     44.000000     70188.290000   
75%      405.00000            8.000000     58.000000     95395.872500   
max      500.00000           10.000000     70.000000    119999.780000   

       forecasted_demand  actual_demand  
count        5000.000000     5000.00000  
mean          297.134000      299.08840  
std           115.568806      121.68078  
min           100.000000       90.00000  
25%           195.000000      194.00000  
50%           297.500000      299.00000  
75%           395.000000      404.00000  
max           500.000000      510.00000  
```

Phase 2: Exploratory Data Analysis (EDA) and Feature Engineering

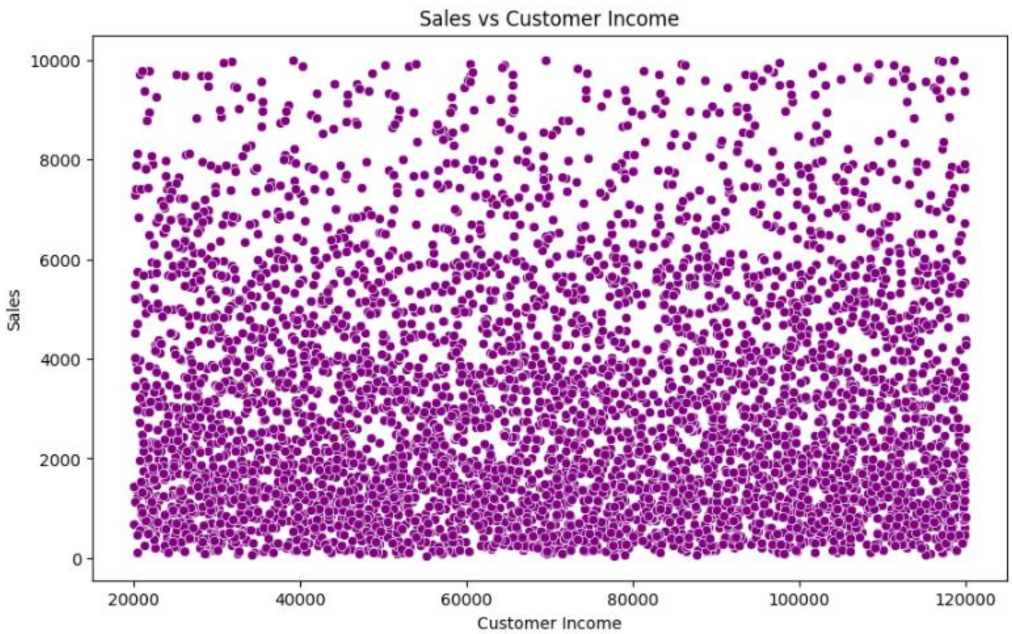
Steps Taken:

1. **Exploratory Data Analysis:**
    - Conducted visualizations using:
        - **Matplotlib and Seaborn:**
            - **Advantages:**
                - Matplotlib: Robust for creating static, interactive, and animated visualizations.
                - Seaborn: Simplifies the creation of aesthetically pleasing statistical plots.
        - Visualizations included:
            - Correlation heatmaps to identify relationships between features.
            - Boxplots and line charts for distribution and trend analysis.
            - Bar and pie charts to visualize sales distributions across various categories (e.g., product, payment methods).
2. **Feature Engineering:**
    - Created new features such as revenue per product and sales per unit.
    - Applied One-Hot Encoding to convert categorical variables into numerical representations.
3. **Data Scaling:**
    - Used Min-Max Scaling and Standardization to normalize features and improve model performance.
4. **Statistical Tests:**
    - Conducted tests such as ANOVA and Chi-Square to evaluate feature significance.
5. **Outcome:**
    - Identified critical features (‘unit_price’ and ‘quantity_sold’) as strong predictors of sales.
    - Reduced dimensionality by excluding non-informative features.

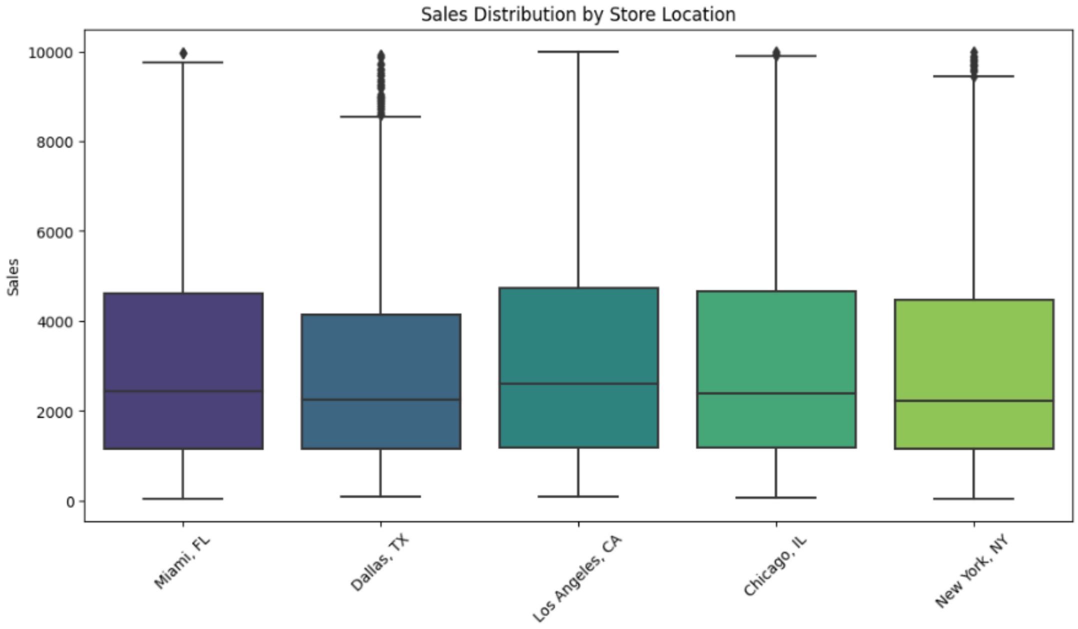
In [15]:
```
# Convert transaction_date to datetime if not already
data['transaction_date'] = pd.to_datetime(data['transaction_date'])

# Group by transaction date and calculate total sales
sales_over_time = data.groupby('transaction_date')['sales'].sum()

# Plot Sales Over Time
plt.figure(figsize=(12, 6))
sales_over_time.plot(kind='line', color='green')
plt.title('Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.xticks(rotation=45)
plt.show()
```



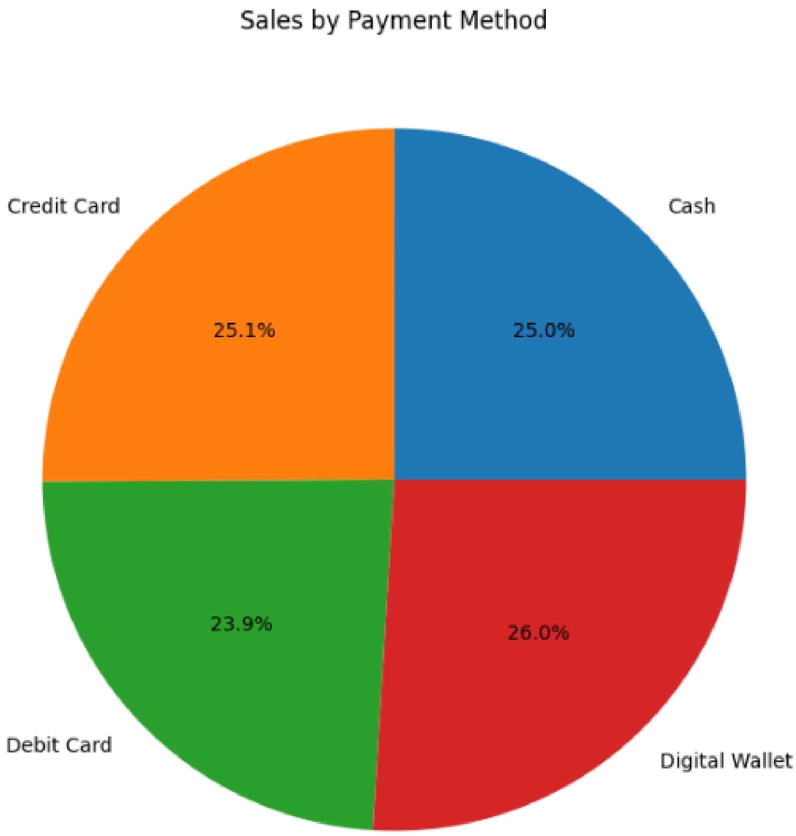
In [16]:
```
# Scatterplot for Sales vs Customer Income
plt.figure(figsize=(10, 6))
sns.scatterplot(x='customer_income', y='sales', data=data, color='purple')
plt.title('Sales vs Customer Income')
plt.xlabel('Customer Income')
plt.ylabel('Sales')
plt.show()
```



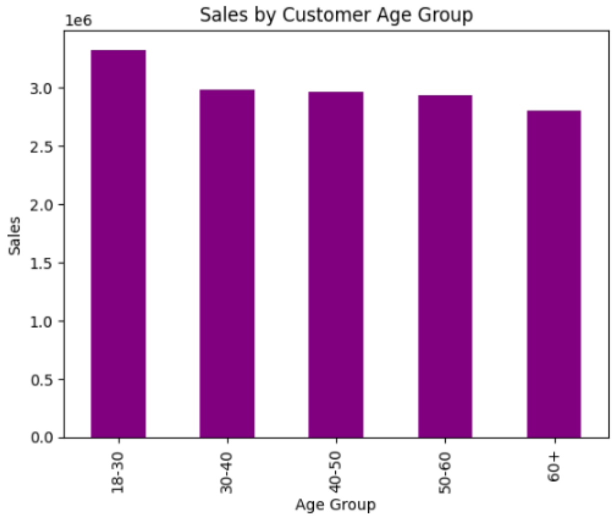
In [17]:
```
# Boxplot for Sales by Store Location
plt.figure(figsize=(12, 6))
sns.boxplot(x='store_location', y='sales', data=data, palette='viridis')
plt.title('Sales Distribution by Store Location')
plt.xlabel('Store Location')
plt.ylabel('Sales')
plt.xticks(rotation=45)
plt.show()
```



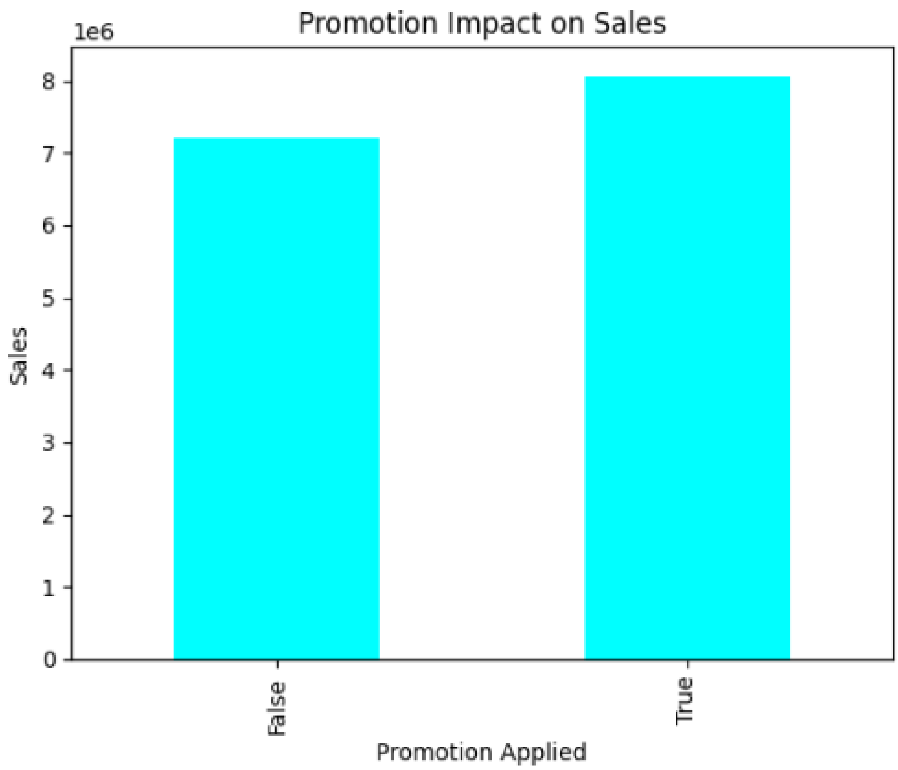
In [20]:
```
payment_sales = data.groupby('payment_method')['sales'].sum()
payment_sales.plot(kind='pie', autopct='%1.1f%%', figsize=(8, 8))
plt.title("Sales by Payment Method")
plt.ylabel("")
plt.show()
```



```
In [22]: data['age_group'] = pd.cut(data['customer_age'], bins=[18, 30, 40, 50, 60, 100], labels=['18-30', '30-40', '40-50', '50-60', '60-
         age_sales = data.groupby('age_group')['sales'].sum().sort_values(ascending=False)
         age_sales.plot(kind='bar', color='purple')
         plt.title("Sales by Customer Age Group")
         plt.xlabel("Age Group")
         plt.ylabel("Sales")
         plt.show()
```

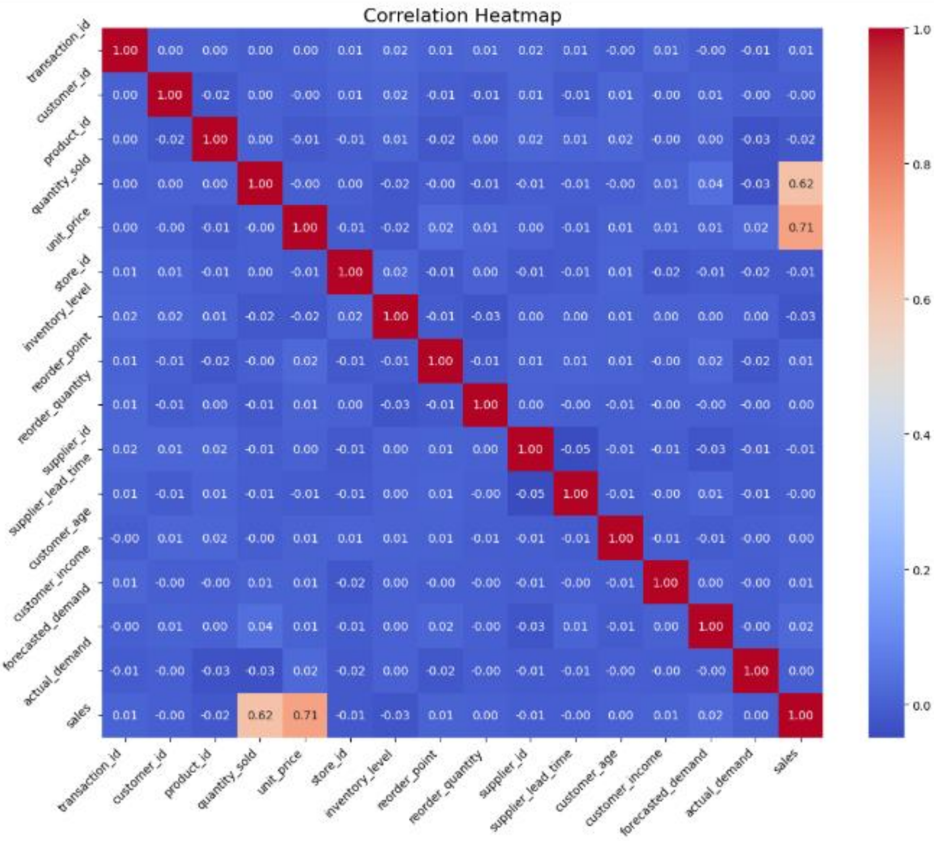



```
In [23]: promotion_sales = data.groupby('promotion_applied')['sales'].sum()
         promotion_sales.plot(kind='bar', color='cyan')
         plt.title("Promotion Impact on Sales")
         plt.xlabel("Promotion Applied")
         plt.ylabel("Sales")
         plt.show()
```



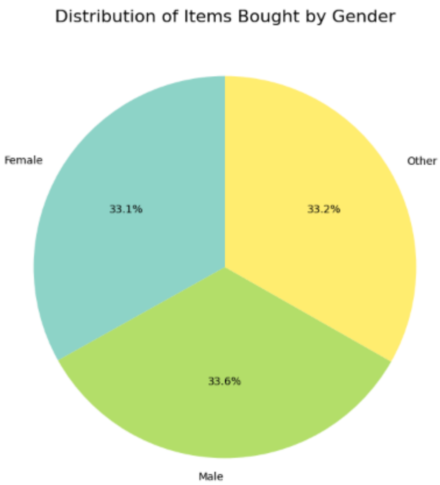
In [26]:

```
# Select only numeric columns
numeric_data = data.select_dtypes(include=[np.number])

# Create the correlation heatmap
plt.figure(figsize=(12, 10))  # Increase figure size
sns.heatmap(numeric_data.corr(), annot=True, cmap='coolwarm', fmt='.2f', annot_kws={"size": 10})  # Adjust annotation font size

# Rotate the axis labels for better readability
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=45, ha='right')

# Add title
plt.title("Correlation Heatmap", fontsize=16)

# Display the plot
plt.tight_layout()
plt.show()
```



In [27]:

```
# Group by gender and product name, and count the number of items bought
gender_sales = data.groupby('customer_gender')['product_name'].count()

# Plot the pie chart
plt.figure(figsize=(8, 8))
gender_sales.plot(kind='pie', autopct='%1.1f%%', startangle=90, cmap='Set3')

# Title and adjustments
plt.title("Distribution of Items Bought by Gender", fontsize=16)
plt.ylabel('')  # Hide the y-axis label for clarity
plt.show()
```



Phase 3: Model Building and Validation

Steps Taken:

1. **Model Selection and Planning:**
   - Selected models:
     - Linear Regression: For baseline predictions.
     - Random Forest Regressor: For handling non-linear relationships.
     - XGBoost Regressor: For efficient boosting and accuracy.
   - **Advantages:**
     - Random Forest: Handles missing data and reduces overfitting through ensemble learning.
     - XGBoost: Optimized for speed and performance in large datasets.
2. **Data Splitting:**
   - Split dataset into training (80%) and testing (20%) subsets using train_test_split from Scikit-learn.
3. **Correlation Analysis:**
   - Analyzed relationships between predictors and the target variable ('sales').
   - Eliminated weak predictors to avoid multicollinearity.
4. **Model Training and Evaluation:**
   - Trained models using Scikit-learn and XGBoost libraries.
   - Evaluated performance metrics:
     - Mean Squared Error (MSE)
     - Root Mean Squared Error (RMSE)
     - R-Squared (R^2)
   - Identified overfitting risks in Linear Regression through cross-validation.
5. **Outcome:**
   - Random Forest achieved the best performance with a test MSE of 3.67e-05.
   - XGBoost provided faster computation with slightly higher error.

In [45]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Sample dataset (replace with your actual dataset)
# Assuming 'data' is your DataFrame and 'sales' is the target variable
features_to_scale = ['unit_price', 'quantity_sold', 'revenue_per_product', 'sales_per_unit', 'forecasted_demand']

# Split data into training and testing sets
X = data[features_to_scale]  # Independent variables
y = data['sales']  # Dependent variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize StandardScaler
scaler = StandardScaler()

# Fit and transform the scaler on the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the test data
X_test_scaled = scaler.transform(X_test)

# Update the training and test sets with scaled features
X_train[features_to_scale] = X_train_scaled
X_test[features_to_scale] = X_test_scaled

# Verify the scaling (mean should be 0, std should be 1)
print(X_train[features_to_scale].mean())  # Should be close to 0
print(X_train[features_to_scale].std())   # Should be close to 1
```

```
unit_price            -1.776357e-17
quantity_sold          1.776357e-18
revenue_per_product   -3.019807e-17
sales_per_unit         1.243450e-17
forecasted_demand      2.122746e-16
dtype: float64
unit_price             1.000125
quantity_sold          1.000125
revenue_per_product    1.000125
sales_per_unit         1.000125
forecasted_demand      1.000125
dtype: float64
```

In [43]:
```
import scipy.stats as stats

# Example: ANOVA test for store_location vs sales
anova_result = stats.f_oneway(
    data[data['store_location_Dallas, TX']]['sales'],
    data[data['store_location_Los Angeles, CA']]['sales'],
    data[data['store_location_Miami, FL']]['sales'],
    data[data['store_location_New York, NY']]['sales']
)

print("ANOVA result for store_location vs sales:", anova_result)
```

```
ANOVA result for store_location vs sales: F_onewayResult(statistic=2.0090214915218088, pvalue=0.11048380921077508)
```

In [44]:
```
# Example: Chi-Square test for promotion_type vs sales
# First, we will need to convert 'sales' into categorical data (e.g., high/low sales)

# Create a new column for sales categories
data['sales_category'] = pd.cut(data['sales'], bins=[0, 500, 1000, 1500, 2000, float('inf')], labels=['Low', 'Medium', 'High', '\

# Create contingency table for promotion_type and sales_category
contingency_table = pd.crosstab(data['promotion_type'], data['sales_category'])

# Perform Chi-Square test
chi2_result = stats.chi2_contingency(contingency_table)
print("Chi-Square result for promotion_type vs sales:", chi2_result)
```

```
Chi-Square result for promotion_type vs sales: Chi2ContingencyResult(statistic=0.0, pvalue=1.0, dof=0, expected_freq=array([[17
42.],
       [ 323.]]))
```

In [46]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Initialize the Linear Regression model
model = LinearRegression()

# Train the model on the scaled training data
model.fit(X_train[features_to_scale], y_train)

# Make predictions on the test data
y_pred = model.predict(X_test[features_to_scale])

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = mse**0.5
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R²): {r2}")
```

```
Mean Squared Error (MSE): 9.48419156811565e-31
Root Mean Squared Error (RMSE): 9.738681413885377e-16
R-squared (R²): 1.0
```

In [47]:
```python
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(model, X[features_to_scale], y, cv=5)
print(f"Cross-validation scores: {cv_scores}")
print(f"Mean cross-validation score: {cv_scores.mean()}")
```

```
Cross-validation scores: [1. 1. 1. 1. 1.]
Mean cross-validation score: 1.0
```

In [48]:
```python
coefficients = model.coef_
feature_importance = pd.DataFrame(list(zip(features_to_scale, coefficients)), columns=['Feature', 'Coefficient'])
print(feature_importance)
```

```
              Feature   Coefficient
0          unit_price  7.063264e-01
1       quantity_sold  6.122268e-01
2  revenue_per_product  3.367031e-01
3      sales_per_unit  0.000000e+00
4   forecasted_demand  2.675811e-16
```

In [49]:
```python
important_features = ['unit_price', 'quantity_sold', 'revenue_per_product']
X_refined = X[important_features]
model.fit(X_refined, y)
```

Out[49]:
```
LinearRegression
LinearRegression()
```

In [50]:
```python
test_predictions = model.predict(X_test[important_features])
test_mse = mean_squared_error(y_test, test_predictions)
test_rmse = np.sqrt(test_mse)
test_r2 = r2_score(y_test, test_predictions)
print(f"Test MSE: {test_mse}, Test RMSE: {test_rmse}, Test R²: {test_r2}")
```

```
Test MSE: 0.000275011681047537, Test RMSE: 0.01658347614487195, Test R²: 0.9997335580014167
```

In [51]:
```python
from sklearn.linear_model import Ridge
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_refined, y)
```

Out[51]:
```
Ridge
Ridge()
```

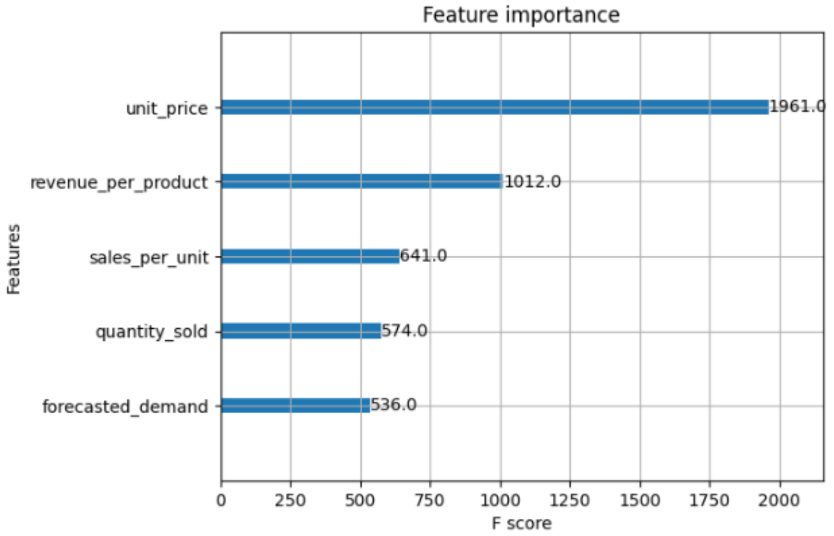
In [11]:
```python
# Define features (adjusted based on available columns) and target variable
important_features = ['unit_price', 'quantity_sold', 'revenue_per_product', 'promotion_applied_True', 'sales_per_unit']
X = dataset[important_features]
y = dataset['sales']  # Target variable

# Split the dataset into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Print shapes to confirm split
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)
```

```
Training data shape: (4000, 5)
Testing data shape: (1000, 5)
```

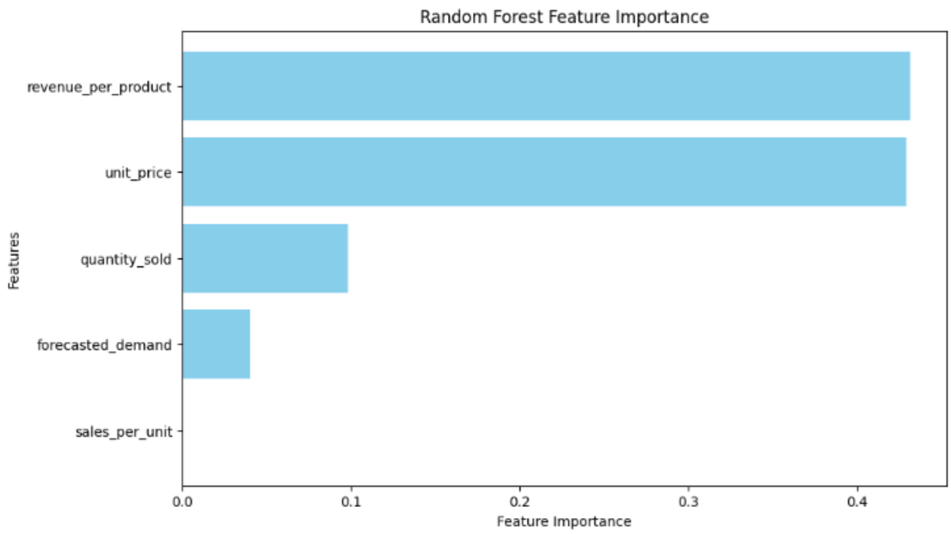
In [12]:
```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import joblib

# Initialize and train the Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Evaluate the model
rf_predictions = rf_model.predict(X_test)
rf_mse = mean_squared_error(y_test, rf_predictions)
print(f"Random Forest MSE (Important Features): {rf_mse}")

# Save the model
joblib.dump(rf_model, r'D:\downloads\data science project\random_forest_model_important.pkl')
print("Random Forest model with important features saved successfully.")
```

```
Random Forest MSE (Important Features): 3.6706571062076665e-05
Random Forest model with important features saved successfully.
```

In [20]:
```python
import matplotlib.pyplot as plt  # Import the matplotlib.pyplot module
from xgboost import plot_importance

# Plot feature importance for the XGBoost model
plot_importance(xgb_model)
plt.show()  # Display the plot
```

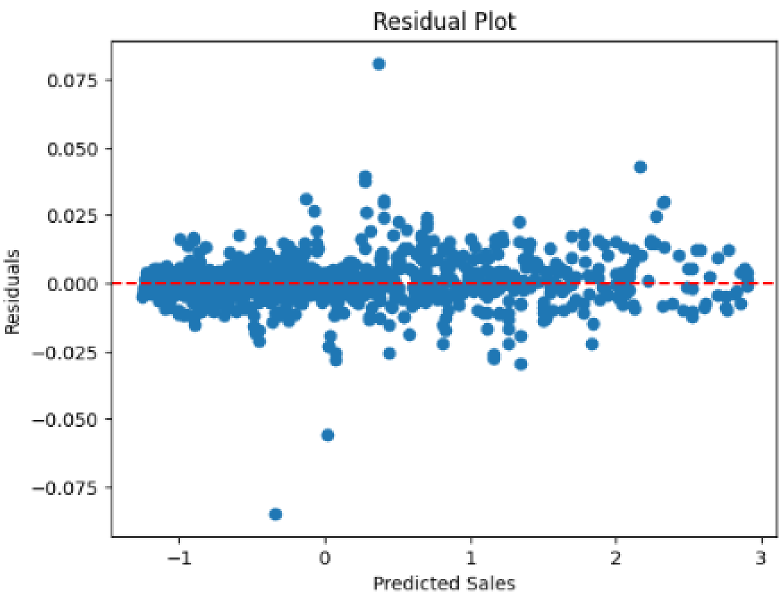


In [21]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Get feature importance from the Random Forest model
rf_feature_importances = rf_model.feature_importances_

# Create a DataFrame for better visualization
features = X_train.columns  # Assuming X_train contains feature names
importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': rf_feature_importances
}).sort_values(by='Importance', ascending=False)

# Plot the feature importance
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.title("Random Forest Feature Importance")
plt.gca().invert_yaxis()  # Invert y-axis to display most important features at the top
plt.show()
```



- **Residual Plots**: Visualize residuals to check for patterns (indicating model inadequacy) or randomness (indicating a good fit).

In [32]:
```
import matplotlib.pyplot as plt
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(0, color='r', linestyle='--')
plt.xlabel("Predicted Sales")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```



Hyperparameter Tuning

- You can use **GridSearchCV** or **RandomizedSearchCV** to find the best hyperparameters for your models.

In [33]:
```
from sklearn.model_selection import GridSearchCV

# Random Forest Hyperparameters
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(estimator=RandomForestRegressor(), param_grid=param_grid, cv=3)
grid_search.fit(X_train, y_train)
print("Best parameters:", grid_search.best_params_)
```

```
Best parameters: {'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 100}
```

Phase 4: Model Deployment

Steps Taken:

1. **Saving the Model:**
   - Stored the trained Random Forest model as a .pkl file using Joblib.
2. **Flask API Development:**
   - Created a REST API for real-time predictions.
     - **Why Flask? Why Are We Doing This?**

1. **Making the Model Accessible:** By deploying the model as a web service, we make it accessible to other applications, websites, or users. Instead of running the model on a local machine or directly in a script, we allow remote systems to send input data and get predictions through an HTTP API.
2. **Real-Time Predictions:** With the model deployed as an API, it can receive real-time data (such as product features or sales data) and immediately return predictions. This is particularly useful in applications like e-commerce, where users or systems can submit new data and get predictions on-demand.
3. **Separation of Model and Application Logic:** Deploying the model as an API separates the model's logic from the application logic. This allows for better scalability, flexibility, and easier integration with different systems. The model can be used in various applications without having to share or expose the model code itself.
4. **Scalability:** By deploying the model as an API, it becomes easier to scale the system. Multiple clients can make requests to the API concurrently, and the system can be optimized for high availability and performance.

   - Defined endpoints to receive input data and return predictions in JSON format.

```
D:\downloads\data science project>python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 665-538-137
C:\Users\USER\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validation.py:2739: UserWarning: X
 does not have valid feature names, but RandomForestRegressor was fitted with feature names
  warnings.warn(
127.0.0.1 - - [09/Jan/2025 14:32:44] "POST /predict HTTP/1.1" 200 -
```

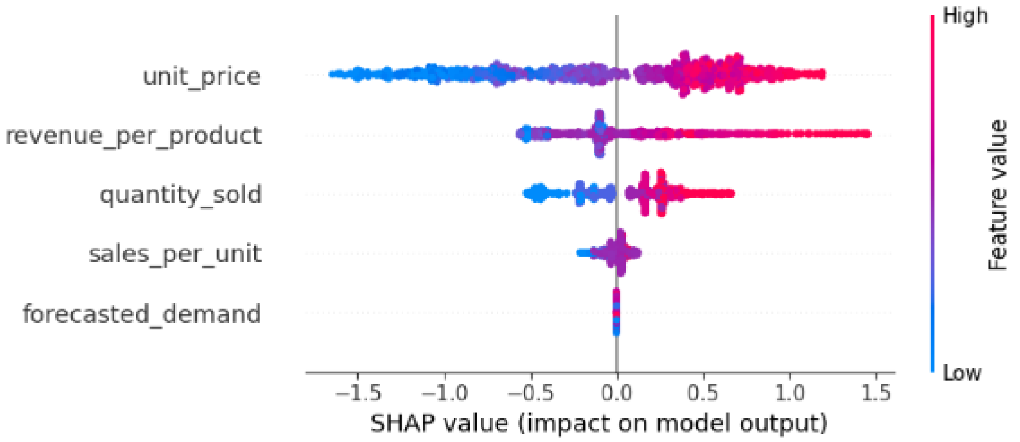
```
Microsoft Windows [Version 10.0.22631.4602]
(c) Microsoft Corporation. All rights reserved.

D:\downloads\data science project>curl -X POST -H "Content-Type: application/json" -d "{\"features\": [10.5, 200, 20.0,
150, 1800]}" http://127.0.0.1:5000/predict
{
  "prediction": [
    2.9282564520737058
  ]
}
```

3. **SHAP for Model Interpretation:**
   - Used SHAP (SHapley Additive exPlanations) to understand feature contributions to predictions.
     - **Advantages:**
       - Provides transparency in model predictions.
       - Highlights key predictors for each decision.

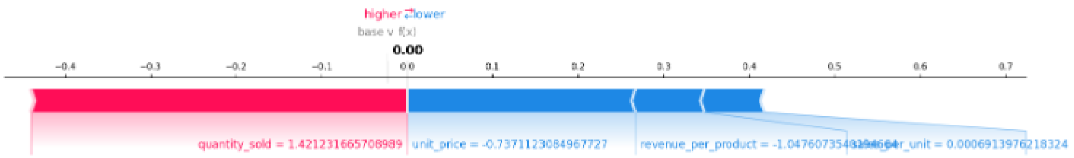
```
In [22]: import shap
         import matplotlib.pyplot as plt
```

```
In [23]: # Create a SHAP explainer for the XGBoost model
         explainer = shap.Explainer(xgb_model, X_test)

         # Calculate SHAP values for the test set
         shap_values = explainer(X_test)
```

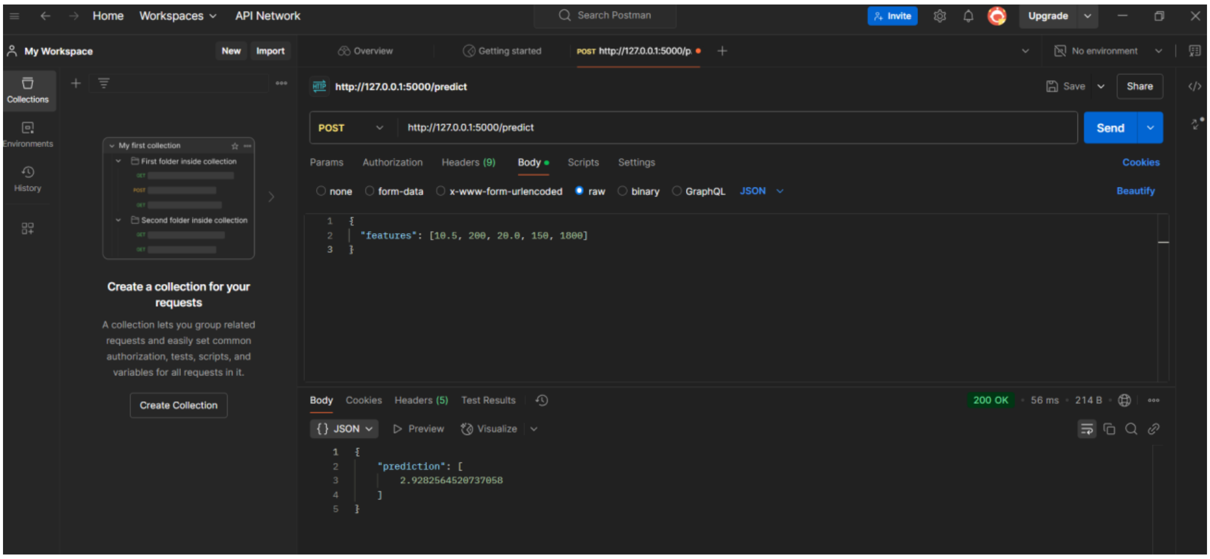
```
In [24]: # Generate a summary plot
         shap.summary_plot(shap_values, X_test)
```




```
In [28]: # Use shap.force_plot with scalar expected_value
         shap.force_plot(
             explainer.expected_value,      # Scalar base value
             shap_values.values[0, :],      # SHAP values for the first instance
             X_test.iloc[0, :],             # Feature values for the first instance
             matplotlib=True                # Enable matplotlib for rendering
         )
```



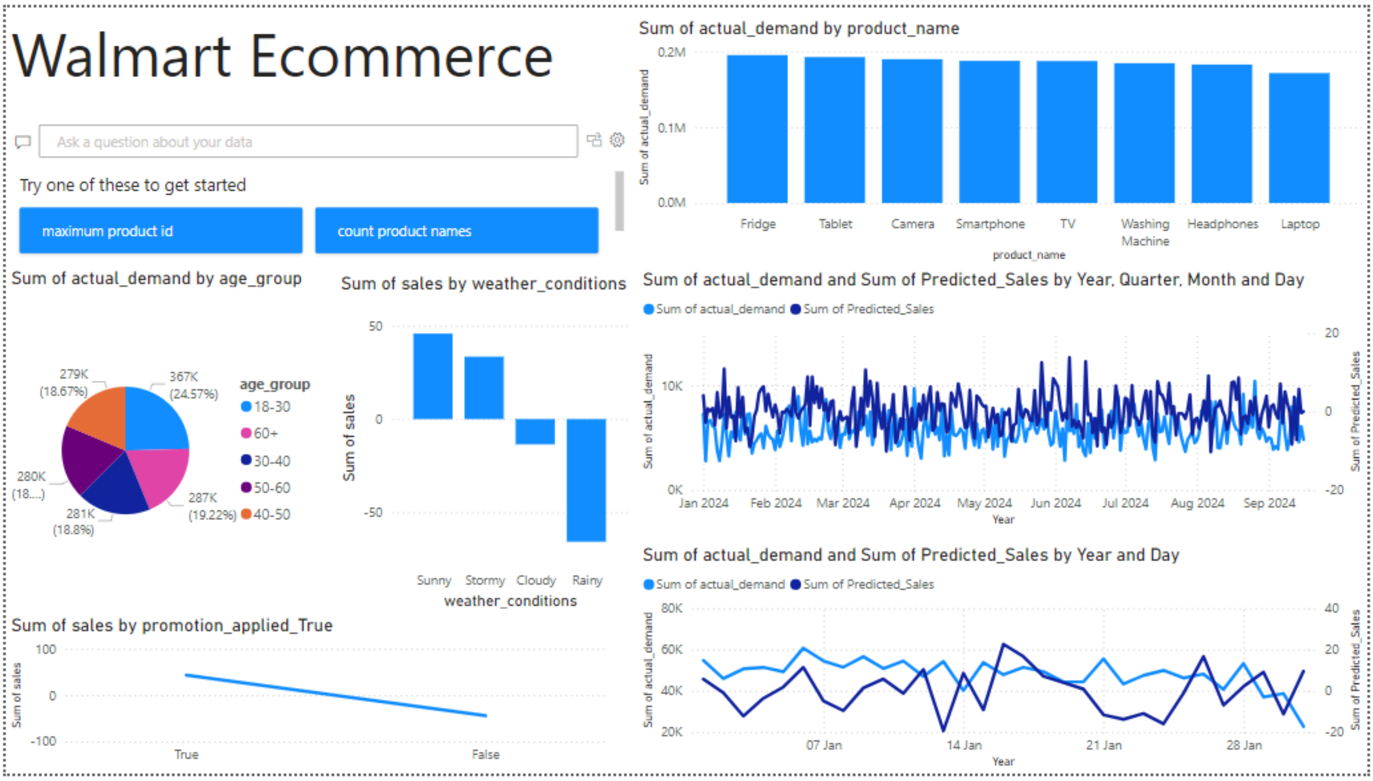
4. **Postman Testing:**
    - Validated API functionality by sending test POST requests to the Flask server.
        - **Why Postman?**
            - Simplifies API testing with an intuitive interface.
            - Ensures the API is functioning correctly before deployment.
5. **Outcome:**
    - Successfully deployed a scalable model for real-time sales predictions.

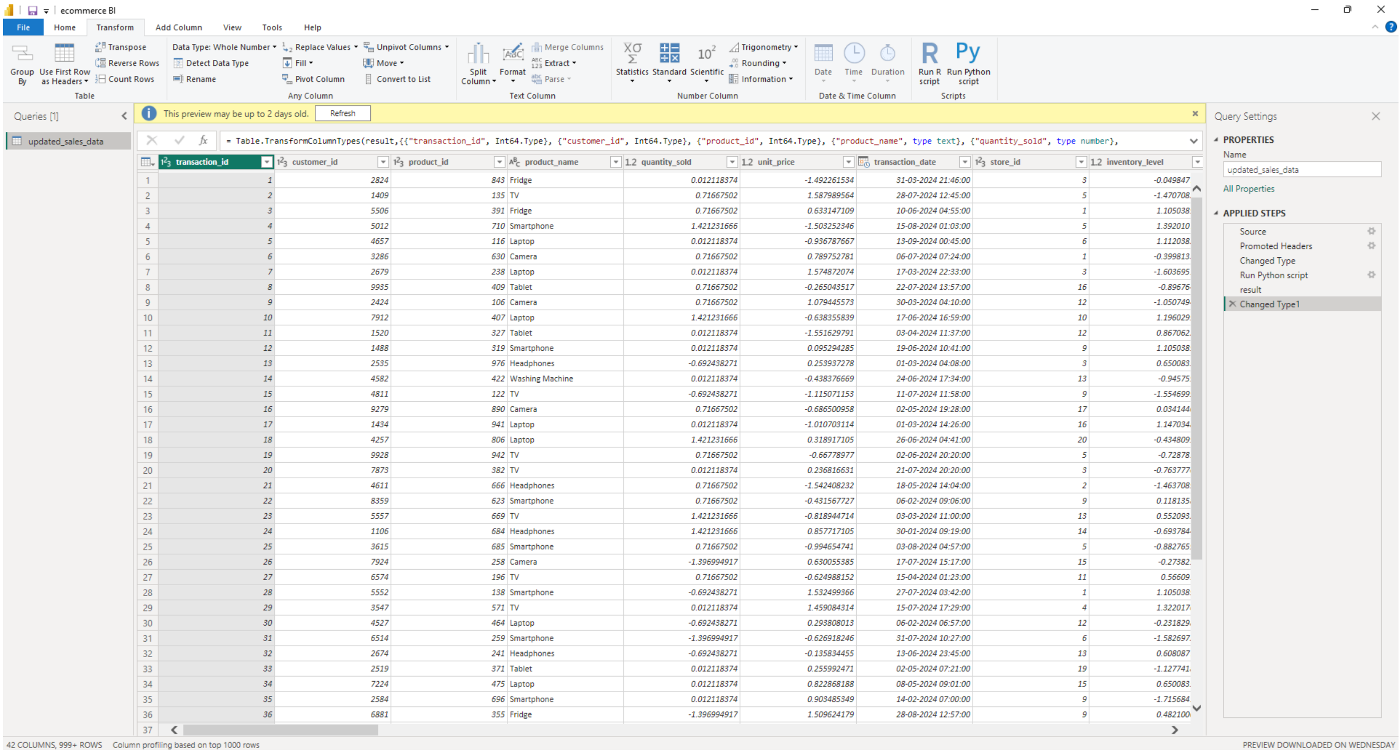

Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Overview
Getting started
POST http://127.0.0.1:5000/p
No environment
Collections
Environments
History
http://127.0.0.1:5000/predict
Save
Share
POST
http://127.0.0.1:5000/predict
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"features": [10.5, 200, 20.0, 150, 1800]
}
My first collection
First folder inside collection
Second folder inside collection
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
Body
Cookies
Headers (5)
Test Results
200 OK
56 ms
214 B
{ } JSON
Preview
Visualize
{
"prediction": [
2.9282564520737058
]
}

Phase 5: Integration with Power BI

Steps Taken:

1. **Loading Data:**
   - Imported the cleaned dataset and model predictions into Power BI using Python scripting.
2. **Creating Interactive Dashboards:**
   - Built visual dashboards to display:
     - Predicted vs. actual sales trends.
     - Feature contributions to sales.
   - Enhanced user experience with slicers and filters for real-time interaction.
3. **Technologies Used:**
   - **Power BI:**
     - **Advantages:**
       - Facilitates interactive and shareable dashboards.
       - Supports integration with Python for advanced analytics.
   - **Python Scripting in Power BI:**
     - Imported pre-trained models for seamless integration.
4. **Outcome:**
   - Enabled dynamic visualization of sales data and predictions for strategic decision-making.





---

3. Key Learnings and Outcomes

1. **Predictive Insights:**
   - unit_price and quantity_sold emerged as the strongest predictors of sales.
   - Features like store_location and promotion_type had minimal impact.
2. **Model Selection:**
   - Random Forest demonstrated robust performance and interpretability.
   - XGBoost offered speed and scalability for larger datasets.
3. **Deployment and Usability:**
   - Flask API enabled real-time predictions accessible via REST endpoints.
   - Power BI dashboards provided actionable insights for business stakeholders.

---

4. Other steps

- **Model Refinement:**
  - Used Ridge or Lasso regression to reduce overfitting risks.
  - Optimized hyperparameters for better performance.
- **Scalability and Security:**
  - Deploy Flask API on a cloud platform (e.g., AWS, Azure) for wider access.
  - Secure API endpoints with authentication mechanisms.
- **Business Applications:**
  - Leverage insights for inventory optimization and pricing strategies.

- Use Power BI dashboards to track performance and refine business strategies.

---

5. Pointwise Summary of the Entire Project

1. Problem Definition: Defined the objective to predict Walmart e-commerce sales and derive actionable insights.
2. Data Collection: Acquired the dataset from Kaggle.
3. Transition to Local Setup: Moved from Kaggle to Jupyter Notebook for better control and flexibility.
4. Data Loading: Loaded the dataset into Python using Pandas.
5. Data Cleaning: Handled missing values and outliers to prepare a clean dataset.
6. Exploratory Data Analysis (EDA): Used Matplotlib and Seaborn to analyze trends, correlations, and distributions.
7. Feature Engineering: Created new features like revenue per product and applied one-hot encoding for categorical variables.
8. Feature Scaling: Applied Min-Max Scaling and Standardization to normalize data for modeling.
9. Statistical Tests: Conducted ANOVA and Chi-Square tests to identify significant features.
10. Feature Selection: Selected key predictors like unit_price, quantity_sold, and revenue_per_product.
11. Model Planning: Chose Linear Regression, Random Forest, and XGBoost for modeling based on data characteristics.
12. Data Splitting: Split the dataset into training (80%) and testing (20%) sets using Scikit-learn.
13. Linear Regression: Built, trained, and evaluated a baseline regression model.
14. Random Forest: Developed and saved a robust Random Forest regression model.
15. XGBoost Regression: Built and evaluated a high-performance gradient boosting model.
16. Model Evaluation: Assessed models using metrics like MSE, RMSE, and R^2.
17. Model Saving: Saved the trained models as .pkl files using Joblib for deployment.
18. SHAP Analysis: Interpreted feature importance and model predictions using SHAP visualizations.
19. Power BI Integration: Loaded the cleaned dataset and model predictions into Power BI for visualization.
20. Flask API Development: Created a REST API with Flask to enable real-time predictions.
21. API Testing: Used Postman to test the Flask API endpoints for functionality.
22. Power BI Dashboards: Built interactive dashboards to visualize sales trends, feature importance, and predictions.
23. Documentation: Maintained detailed notes and reports for each step of the project.

Conclusion

This project effectively utilized data science methodologies and tools to analyze Walmart's e-commerce sales. The integration of predictive modeling, interpretability (SHAP), and interactive dashboards (Power BI) provided actionable insights to drive business decisions. The workflow is scalable and adaptable for real-world applications, ensuring its relevance in dynamic retail environments.

Vinit Satardekar 09 Jan 2024