Pipeline-based Prediction of Gross Income from Supermarket Sales Data

1. Introduction

The retail industry generates large volumes of transactional data, which can be analyzed to predict future business outcomes. In this project, we focus on predicting the gross income of a supermarket based on various sales-related features such as product details, customer demographics, and transaction specifics. Gross income is a key financial metric, representing the total revenue from sales before any deductions, and predicting it accurately can help businesses optimize inventory, pricing, and marketing strategies.Machine learning techniques offer an efficient way to analyze and model this data, providing businesses with actionable insights. In this study, we utilize a pipeline-based approach, which combines data preprocessing and model training in a seamless workflow. Specifically, we use two powerful algorithms: Gradient Boosting and XGBoost, both of which are known for their performance in regression tasks.

2. Objectives

- Develop a predictive model to estimate the gross income from supermarket sales data using machine learning techniques.
- Evaluate the performance of different models, including Gradient Boosting and XGBoost, to identify the most accurate approach for income prediction.
- Implement a streamlined pipeline that integrates data preprocessing, model training, and prediction to facilitate efficient and automated predictions for real-world use.

3. Dataset Description

The dataset used for this project contains transactional data from a supermarket, including details such as product lines, pricing, payment types, customer demographics, and sales metrics. The target variable is gross income, which is predicted using features like product price, quantity sold, and customer details.

<u>Key Features:</u>

Branch, City, Customer Type, Gender, Product Line, Payment Method

Unit Price, Quantity, Tax (5%), Cost of Goods Sold (COGS), Rating

Target: Gross Income

Dataset Shape: (1000, 17) retrieved from the notebook.

```
[303] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Invoice ID               1000 non-null   object
 1   Branch                   1000 non-null   object
 2   City                     1000 non-null   object
 3   Customer type            1000 non-null   object
 4   Gender                   1000 non-null   object
 5   Product line             1000 non-null   object
 6   Unit price               994 non-null    float64
 7   Quantity                 997 non-null    float64
 8   Tax 5%                   998 non-null    float64
 9   Total                    1000 non-null   float64
 10  Date                     1000 non-null   object
 11  Time                     1000 non-null   object
 12  Payment                  1000 non-null   object
 13  cogs                     986 non-null    float64
 14  gross margin percentage  1000 non-null   float64
 15  gross income             998 non-null    float64
 16  Rating                   992 non-null    float64
dtypes: float64(8), object(9)
memory usage: 132.9+ KB
```

4. Methodology

1. Data Exploration and Cleaning

The dataset underwent an initial exploration to ensure data quality:

Missing Values: Handled appropriately using imputers.

Duplicates: Removed if found.

Feature Engineering:

Irrelevant columns (Date, Time, and Invoice ID) were dropped.

Categorical variables were encoded using one-hot and ordinal encoding.

```
[ ]  df = df.drop(columns=['Date', 'Time','Invoice ID'])
     df.info()
```

2.Feature and Target Definition

Features (X): All columns except gross income.

Target (y): Gross income.

```
[ ]  X = df.drop(columns=['gross income'])
     y = df['gross income']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3. Pipeline Design

A modular pipeline was constructed for data preprocessing and model training:

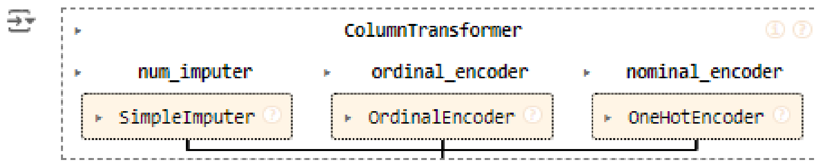
Transformations:

1. Handling missing values.
2. Encoding categorical variables.
3. Scaling numerical features.

trf1 for missing value handle and encodiing categorical variable

```
[ ] ## Perform Imputer for missing value.Also Ordinal encoder and one-hot encoding for handle categorical varibale
    trf1 = ColumnTransformer([
        ('num_imputer', SimpleImputer(strategy='mean'), numeric_columns),
        ('ordinal_encoder', OrdinalEncoder(), ordinal_columns),
        ('nominal_encoder', OneHotEncoder(handle_unknown='ignore', sparse_output=False), nominal_columns)
    ])
```

```
[ ] trf1
```




```
] ## Scaling
  trf2 = ColumnTransformer([
      ('Scale', MinMaxScaler(), slice(0, len(df.columns)))
  ])
```

Feature Selection:

1. Post-encoding feature reduction based on importance.

```
## Feature Selection
## After performing one hot encodding column will be total 24. So here k = 23
trf3 = SelectKBest(score_func=f_regression, k=23)
```

Regression Models:

1. Gradient Boosting Regressor (pipe1).
2. XGBoost Regressor (pipe2).

```
[ ] trf4 = GradientBoostingRegressor()
```

```
[ ] trf5 = XGBRegressor()
```

Create Pipeline

```
[ ] pipe1 = Pipeline([
        ('trf1', trf1),
        ('trf2', trf2),
        ('trf3', trf3),
        ('trf4', trf4)
    ])
    pipe2 = Pipeline([
        ('trf1', trf1),
        ('trf2', trf2),
        ('trf3', trf3),
        ('trf5', trf5)
    ])
```

Fit the Pipeline

```
[ ] pipe1.fit(X_train, y_train)
```

```
pipe2.fit(X_train, y_train)
```

4. Model Training

Train-test split was performed for supervised learning.
Both pipelines (pipe1 and pipe2) were trained on the dataset.

5. Model Evaluation

Models were evaluated using:

1. Mean Absolute Error (MAE).
2. Mean Squared Error (MSE).
3. R^2 Score.
4. Cross-validation was performed to ensure robustness.

```
[ ] from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
    mae = mean_absolute_error(y_test, y_pred1)
    mse = mean_squared_error(y_test,y_pred1)
    r2 = r2_score(y_test, y_pred1)

    print(f'Mean Absolute Error: {mae}')
    print(f'Mean Squared Error: {mse}')
    print("R^2 Score:", r2)
```

```
Mean Absolute Error: 0.07046721017424538
Mean Squared Error: 0.01012512516618506 4
R^2 Score: 0.9999206009302002
```

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
mae = mean_absolute_error(y_test, y_pred2)
mse = mean_squared_error(y_test,y_pred2)
r2 = r2_score(y_test, y_pred2)

print(f'Mean Absolute Error: {mae}')
print(f'Mean Squared Error: {mse}')
print("R^2 Score:", r2)
```

```
Mean Absolute Error: 0.11372242305278782
Mean Squared Error: 0.03597456636560478
R^2 Score: 0.99971789513127
```

Cross-Validation:

Gradient Boosting showed (summary of performance).

```
GradientBoostingRegressor MAE scores:
 [0.06969951 0.13276825 0.26744283 0.0856367  0.07121853]
GradientBoostingRegressor Average MAE score: 0.125353166929588
GradientBoostingRegressor MSE scores:
 [0.0088423  0.41685142 3.46176932 0.0142582  0.00951372]
GradientBoostingRegressor Average MSE score: 0.7822469923549723
```

XGBoost outperformed/underperformed with (summary).

```
XGBRegressor MAE scores:
 [0.11168343 0.15855787 0.13399188 0.12766621 0.11642066]
XGBRegressor Average MAE score: 0.12966400885085727
XGBRegressor MSE scores:
 [0.03328893 0.43455184 0.09544452 0.04888297 0.04036376]
XGBRegressor Average MSE score: 0.13050640243093026
```

Output:

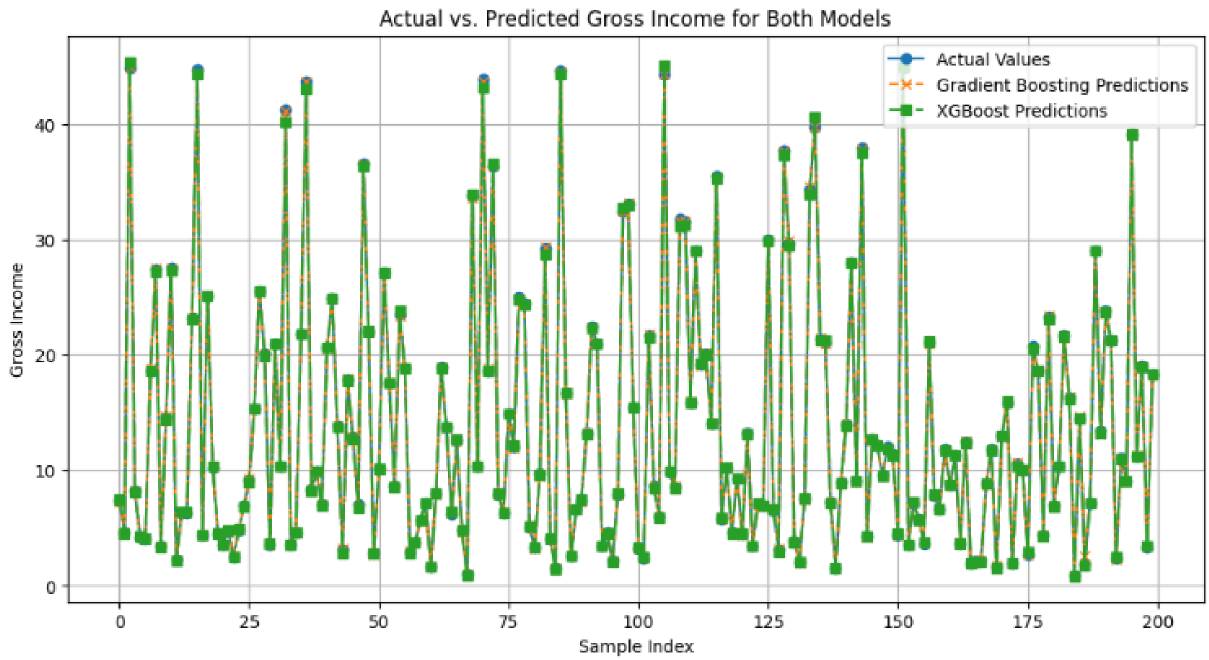
```python
test_data = pd.DataFrame({
    'Branch': ['A'],
    'City': ['Yangon'],
    'Customer type': ['Member'],
    'Gender': ['Female'],
    'Product line': ['Health and beauty'],
    'Unit price': [74.69],
    'Quantity': [7.0],
    'Tax 5%': [26.1415],
    'Total': [548.9715],
    'Payment': ['Ewallet'],
    'cogs': [522.83],
    'Rating': [9.1]
})


# Make the prediction
predicted_gross_income1 = pipe1.predict(test_data)
predicted_gross_income2 = pipe2.predict(test_data)
print(f"Actual Gross Income in dataset:26.1415")
print(f"Predicted Gross Income1: {predicted_gross_income1[0]}")
print(f"Predicted Gross Income2: {predicted_gross_income2[0]}")
```

```
Actual Gross Income in dataset:26.1415
Predicted Gross Income1: 26.118068348506366
Predicted Gross Income2: 26.143829345703125
```

7. Visualization

Actual vs. predicted values for gross income were plotted to visualize the performance.



Conclusion

The pipeline-based approach efficiently predicted gross income with high accuracy. Among the two models, (best-performing model) demonstrated superior performance based on evaluation metrics and cross-validation. This framework is scalable and can be adapted for similar regression tasks in the retail domain.

Future Work

- Test on larger datasets for better generalization.
- Explore additional machine learning models like Random Forest or Neural Networks.
- Integrate external factors like seasonal trends or promotions to enhance predictions.

Artifacts

The trained pipelines (pipe1.pkl and pipe2.pkl) are available for deployment.