

Crisp-DM Retail Analysis



Walmart Store sales

1	1	05-02-2010	1643690.9	0
2	1	12-02-2010	1641957.44	1
3	1	19-02-2010	1611968.17	0
4	1	26-02-2010	1409727.59	0
5	1	05-03-2010	1554806.68	0
6	1	12-03-2010	1439541.59	0
	Store	Date	Weekly_Sales	Holiday_Flag

You are world renowned speaker and authority in Crisp-DM and wrote many award winning books in this topic.

Could you help me do the retail analysis of walmart data available on kaggle using Crisp-DM methodology? I want it to include all the 6 phases: business understanding, data understanding, data preparation, modeling, evaluation, and deployment.



Let's walk through the retail analysis of Walmart sales data using the CRISP-DM methodology, including the six key phases. I'll guide you through each phase, and we can work on the specific analysis together.

Phase 1: Business Understanding

In this phase, the goal is to understand the business context, objectives, and requirements. For a retail analysis of Walmart, typical business objectives might include:

- Identifying trends in sales performance across different stores.
- Understanding the impact of promotions, holidays, or other factors on sales.
- Developing strategies for inventory management, staffing, or marketing.
- Forecasting future sales based on historical data.

Key Questions:

- What are the key sales metrics for Walmart? (e.g., total sales, average sales per store, etc.)
- Are there any seasonal patterns or promotions that affect sales?
- How can we use this analysis to support decision-making in store management?

Phase 2: Data Understanding

We will explore the dataset to understand its structure, variables, and potential quality issues. I'll load the dataset you provided and give you an overview of the data.

Let me start by loading and inspecting the data.

python

```
import pandas as pd

# Load the Walmart Store Sales dataset
file_path = '/mnt/data/Walmart_Store_sales.csv'
walmart_data = pd.read_csv(file_path)

# Display the first few rows of the dataset and basic information
walmart_data_info = walmart_data.info()
walmart_data_head = walmart_data.head()

walmart_data_info, walmart_data_head
```

STDOUT/STDERR

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
#   Column          Non-Null Count  Dtype  
---  -
0   Store            6435 non-null  int64  
1   Date             6435 non-null  object  
2   Weekly_Sales     6435 non-null  float64 
3   Holiday_Flag     6435 non-null  int64  
4   Temperature      6435 non-null  float64 
5   Fuel_Price       6435 non-null  float64 
6   CPI              6435 non-null  float64 
7   Unemployment     6435 non-null  float64 
dtypes: float64(5), int64(2), object(1)
memory usage: 402.3+ KB
```

Result

```
(None,
Store      Date  Weekly_Sales  Holiday_Flag  Temperature  Fuel_Price  \
0      1  05-02-2010    1643690.90           0        42.31      2.572
1      1  12-02-2010    1641957.44           1        38.51      2.548
2      1  19-02-2010    1611968.17           0        39.93      2.514
3      1  26-02-2010    1409727.59           0        46.63      2.561
4      1  05-03-2010    1554806.68           0        46.50      2.625

      CPI  Unemployment
0  211.096358      8.106
1  211.242170      8.106
2  211.289143      8.106
3  211.319643      8.106
4  211.350143      8.106 )
```

Data Understanding

The dataset has 6,435 entries with the following columns:

- **Store:** Identifier for each store.
- **Date:** Date of the sales record.
- **Weekly_Sales:** Sales for the store during that week.
- **Holiday_Flag:** Whether the week includes a holiday (1 for holiday week, 0 otherwise).
- **Temperature:** Temperature in the region for that week.
- **Fuel_Price:** Fuel price during that week in the region.
- **CPI:** Consumer Price Index.
- **Unemployment:** Unemployment rate in the region.

At this stage, we aim to:

- Understand relationships and potential trends (e.g., impact of holidays or temperature on sales).
- Identify data quality issues (e.g., missing values, outliers).

There are no null values, and the data types appear to be correct, though we may need to convert the `Date` column to a proper date format.

Phase 3: Data Preparation

This phase involves cleaning, transforming, and selecting the data for modeling. We'll:

1. Convert the `Date` column to datetime format.
2. Check for any outliers or inconsistencies in sales, temperature, or other columns.
3. Create any new features or aggregate data if necessary (e.g., sales by month).

Let me handle the data preparation.

python

```
# Convert the 'Date' column to datetime format
walmart_data['Date'] = pd.to_datetime(walmart_data['Date'], format='%d-%m-%Y')

# Check for outliers in 'Weekly_Sales' using summary statistics
walmart_data_description = walmart_data.describe()

walmart_data_description
```

Result

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	\
count	6435.000000	6.435000e+03	6435.000000	6435.000000	6435.000000	
mean	23.000000	1.046965e+06	0.069930	60.663782	3.358607	
std	12.988182	5.643666e+05	0.255049	18.444933	0.459020	
min	1.000000	2.099862e+05	0.000000	-2.060000	2.472000	
25%	12.000000	5.533501e+05	0.000000	47.460000	2.933000	
50%	23.000000	9.607460e+05	0.000000	62.670000	3.445000	
75%	34.000000	1.420159e+06	0.000000	74.940000	3.735000	
max	45.000000	3.818686e+06	1.000000	100.140000	4.468000	

	CPI	Unemployment
count	6435.000000	6435.000000
mean	171.578394	7.999151
std	39.356712	1.875885
min	126.064000	3.879000
25%	131.735000	6.891000
50%	182.616521	7.874000
75%	212.743293	8.622000
max	227.232807	14.313000

Data Preparation Insights:

1. **Weekly Sales:** The sales range from about 209,986 to 3.8 million dollars, which suggests significant variability across stores and weeks.
2. **Temperature:** The temperature ranges from -2.06°C to 100.14°C, indicating possible weather-related factors in different regions.
3. **Fuel Price:** The fuel price ranges from \$2.47 to \$4.47.
4. **CPI and Unemployment:** These have reasonable ranges for economic indicators.

Next, we should:

- Normalize the data for modeling (optional).
- Handle any potential outliers (high sales values might represent promotional weeks).

Phase 4: Modeling

In this phase, we will create a model to predict sales or understand relationships in the data. A common modeling approach for retail analysis includes:

- Time series forecasting for weekly sales trends.
- Regression models to assess the impact of external factors (holiday, temperature, etc.).

Let's start by creating a simple regression model to predict **Weekly Sales** based on available features (Holiday_Flag, Temperature, Fuel_Price, CPI, and Unemployment).

I'll prepare the data and run a regression model to evaluate the predictive power of these variables.

python

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Prepare the features (X) and target (y)
X = walmart_data[['Holiday_Flag', 'Temperature', 'Fuel_Price', 'CPI', 'Unemployment']]
y = walmart_data['Weekly_Sales']

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

# Train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on the test data
y_pred = model.predict(X_test)

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

mse, r2
```

Result

(316554511405.115, 0.01738312550370069)

Modeling Evaluation

The linear regression model produced the following results:

- **Mean Squared Error (MSE):** 316,554,511,405, which indicates a high error in prediction.
- **R-squared (R^2):** 0.017, showing that the model explains only 1.7% of the variance in weekly sales based on the selected features.

This suggests that the features used (Holiday_Flag, Temperature, Fuel_Price, CPI, Unemployment) have limited predictive power for weekly sales. We may need additional features (e.g., time-related factors or store-specific attributes) to improve the model.

Phase 5: Evaluation

From the modeling results, it is clear that our current feature set doesn't strongly predict sales. To improve the model:

- We could try using time series forecasting techniques since sales typically follow a seasonal pattern.
- Incorporating store-level information (store size, location) or external factors (e.g., promotions) could also improve the model.

Phase 6: Deployment

In a business context, deployment would involve integrating the model into a system to predict future sales or provide insights for store managers. For our case, this could mean:

- Automating weekly sales predictions for each store.
- Building a dashboard that visualizes sales trends and forecasts.

We can explore creating visualizations or dashboards next if needed.

Let me know if you'd like to focus on any specific areas or improve the model further!



Why did you use linear regression? can we get the accuracy and try improving it?



I initially used linear regression because it's a simple and interpretable model that works well for understanding relationships between variables, especially when dealing with continuous data like sales. However, its performance in this case (low R^2 and high MSE) suggests it may not be the best choice given the complexity of the Walmart sales data.

Since you'd like to improve the model and measure **accuracy**, let's consider a few things:

1. **Linear regression** measures model fit with metrics like R^2 and MSE, but it doesn't provide an accuracy score directly because it isn't a classification model. For regression tasks, improving the

model's prediction power can involve:

- Trying more complex models like **Decision Trees**, **Random Forests**, or **Gradient Boosting**.
- Using **time-series models** such as ARIMA, if we treat the data as time-dependent.

2. To **improve accuracy**, we could:

- Engineer better features (e.g., time-based features like month, year, or sales lag).
- Add more data, such as store-specific attributes, or account for promotions or special events that may drive sales.

Let's try a **Random Forest Regressor**, which is more robust to non-linear relationships, and I'll include new time-based features like **month** and **year** extracted from the Date column to see if it helps.

I'll build and evaluate this model next.

python

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Feature engineering: extract 'month' and 'year' from 'Date'
walmart_data['Month'] = walmart_data['Date'].dt.month
walmart_data['Year'] = walmart_data['Date'].dt.year

# Updated feature set (including 'Month' and 'Year')
X = walmart_data[['Holiday_Flag', 'Temperature', 'Fuel_Price', 'CPI', 'Unemployment',
'Month', 'Year']]

# Split the data into training and testing sets again
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on the test data
y_rf_pred = rf_model.predict(X_test)

# Evaluate the model: MSE, MAE, and R2 Score
rf_mse = mean_squared_error(y_test, y_rf_pred)
rf_mae = mean_absolute_error(y_test, y_rf_pred)
rf_r2 = r2_score(y_test, y_rf_pred)

rf_mse, rf_mae, rf_r2
```

It seems that there was a connection issue while running the model, but I can describe the next steps for improving the model based on what we were working on:

1. **Feature Engineering:** By including new features such as ``Month`` and ``Year`` from the ``Date`` column, we introduce time-based elements that can capture seasonal sales trends.
2. **Random Forest Regressor:** This model is typically more accurate than linear regression because it captures complex relationships in the data through multiple decision trees.

To proceed:

- You could rerun the Random Forest model on your local environment or I can attempt it again.

- After training the model, evaluate its performance using metrics like **Mean Squared Error (MSE)**, **Mean Absolute Error (MAE)**, and **R-squared (R^2)** to assess how well it predicts sales.

Would you like me to retry running the model or would you prefer I share the code to try on your own?



continue



python

```
# Retry the previous code for Random Forest Regressor with time-based features

# Train a Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on the test data
y_rf_pred = rf_model.predict(X_test)

# Evaluate the model: MSE, MAE, and R2 Score
rf_mse = mean_squared_error(y_test, y_rf_pred)
rf_mae = mean_absolute_error(y_test, y_rf_pred)
rf_r2 = r2_score(y_test, y_rf_pred)

rf_mse, rf_mae, rf_r2
```

```
-----
NameError                                Traceback (most recent call last)
Cell In[1], line 4
      1 # Retry the previous code for Random Forest Regressor with time-based features
      2
      3 # Train a Random Forest Regressor
----> 4 rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
      5 rf_model.fit(X_train, y_train)
      7 # Predict on the test data

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

It seems the environment reset and some variables were lost. Let me restore the necessary components and rerun the Random Forest model.

python

```
# Reimport necessary libraries and reload the dataset
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Reload the dataset
walmart_data = pd.read_csv(file_path)
walmart_data['Date'] = pd.to_datetime(walmart_data['Date'], format='%d-%m-%Y')

# Feature engineering: extract 'month' and 'year' from 'Date'
walmart_data['Month'] = walmart_data['Date'].dt.month
walmart_data['Year'] = walmart_data['Date'].dt.year

# Updated feature set (including 'Month' and 'Year')
```

```
X = walmart_data[['Holiday_Flag', 'Temperature', 'Fuel_Price', 'CPI', 'Unemployment',
'Month', 'Year']]
y = walmart_data['Weekly_Sales']

# Split the data into training and testing sets again
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on the test data
y_rf_pred = rf_model.predict(X_test)

# Evaluate the model: MSE, MAE, and R2 Score
rf_mse = mean_squared_error(y_test, y_rf_pred)
rf_mae = mean_absolute_error(y_test, y_rf_pred)
rf_r2 = r2_score(y_test, y_rf_pred)

rf_mse, rf_mae, rf_r2
```

```
-----
NameError                                Traceback (most recent call last)
Cell In[2], line 6
      3 from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
      5 # Reload the dataset
----> 6 walmart_data = pd.read_csv(file_path)
      7 walmart_data['Date'] = pd.to_datetime(walmart_data['Date'], format='%d-%m-%Y')
      9 # Feature engineering: extract 'month' and 'year' from 'Date'

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

python

```
# Reimport pandas and reload the dataset from the previous steps
import pandas as pd

# Reload the dataset
walmart_data = pd.read_csv(file_path)
walmart_data['Date'] = pd.to_datetime(walmart_data['Date'], format='%d-%m-%Y')

# Feature engineering: extract 'month' and 'year' from 'Date'
walmart_data['Month'] = walmart_data['Date'].dt.month
walmart_data['Year'] = walmart_data['Date'].dt.year

# Updated feature set (including 'Month' and 'Year')
X = walmart_data[['Holiday_Flag', 'Temperature', 'Fuel_Price', 'CPI', 'Unemployment',
'Month', 'Year']]
y = walmart_data['Weekly_Sales']

# Split the data into training and testing sets again
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on the test data
y_rf_pred = rf_model.predict(X_test)

# Evaluate the model: MSE, MAE, and R2 Score
rf_mse = mean_squared_error(y_test, y_rf_pred)
rf_mae = mean_absolute_error(y_test, y_rf_pred)
rf_r2 = r2_score(y_test, y_rf_pred)

rf_mse, rf_mae, rf_r2
```

```
-----
NameError                                Traceback (most recent call last)
```



```
Cell In[3], line 5
      2 import pandas as pd
      4 # Reload the dataset
----> 5 walmart_data = pd.read_csv(file_path)
      6 walmart_data['Date'] = pd.to_datetime(walmart_data['Date'], format='%d-%m-%Y')
      8 # Feature engineering: extract 'month' and 'year' from 'Date'
```

NameError: name 'file_path' is not defined

It seems that the file path has been lost as well. Let me reload the file you uploaded earlier and continue with the model training.

python

```
# Use the provided file path to reload the Walmart dataset and proceed with analysis
file_path = '/mnt/data/Walmart_Store_sales.csv'

# Reload the dataset
walmart_data = pd.read_csv(file_path)
walmart_data['Date'] = pd.to_datetime(walmart_data['Date'], format='%d-%m-%Y')

# Feature engineering: extract 'month' and 'year' from 'Date'
walmart_data['Month'] = walmart_data['Date'].dt.month
walmart_data['Year'] = walmart_data['Date'].dt.year

# Updated feature set (including 'Month' and 'Year')
X = walmart_data[['Holiday_Flag', 'Temperature', 'Fuel_Price', 'CPI', 'Unemployment',
                  'Month', 'Year']]
y = walmart_data['Weekly_Sales']

# Split the data into training and testing sets again
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on the test data
y_rf_pred = rf_model.predict(X_test)

# Evaluate the model: MSE, MAE, and R2 Score
rf_mse = mean_squared_error(y_test, y_rf_pred)
rf_mae = mean_absolute_error(y_test, y_rf_pred)
rf_r2 = r2_score(y_test, y_rf_pred)

rf_mse, rf_mae, rf_r2
```

```
-----
NameError                                Traceback (most recent call last)
Cell In[4], line 17
      14 y = walmart_data['Weekly_Sales']
      16 # Split the data into training and testing sets again
----> 17 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
      19 # Train a Random Forest Regressor
      20 rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

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

python

```
# Reimport necessary functions
from sklearn.model_selection import train_test_split

# Now, split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Random Forest Regressor
```

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

# Predict on the test data
y_rf_pred = rf_model.predict(X_test)

# Evaluate the model: MSE, MAE, and R2 Score
rf_mse = mean_squared_error(y_test, y_rf_pred)
rf_mae = mean_absolute_error(y_test, y_rf_pred)
rf_r2 = r2_score(y_test, y_rf_pred)

rf_mse, rf_mae, rf_r2
```

Result

(289698026296.0537, 385017.4891359771, 0.10074834225794782)

The **Random Forest Regressor** model yielded the following evaluation metrics:

- **Mean Squared Error (MSE):** 289,698,026,296 (slightly improved from the linear regression).
- **Mean Absolute Error (MAE):** 385,017, indicating the average error between the predicted and actual sales values.
- **R-squared (R^2):** 0.101, meaning that 10.1% of the variance in weekly sales is explained by the model. This is an improvement over the linear regression model but still not highly predictive.

Recommendations for Improvement:

- **Feature Engineering:** More features such as store location, promotions, or store-specific attributes could improve the model's performance.
- **Time Series Models:** Since sales data often follows temporal patterns, applying a time series forecasting model like ARIMA might yield better results.

Would you like to proceed with additional improvements, such as trying a time series model, or further refining this model?



Please compare accuracies of previous and new model and also plot some graphs for visualizations



python

```
import matplotlib.pyplot as plt
import numpy as np

# Calculate accuracy for both models
# For regression models,  $R^2$  is often used as a measure of accuracy
linear_regression_r2 = 0.017 # from the linear regression result
random_forest_r2 = rf_r2

# Comparison of  $R^2$  values (accuracy)
accuracy_comparison = {
    'Linear Regression  $R^2$ ': linear_regression_r2,
    'Random Forest  $R^2$ ': random_forest_r2
}
```

```
# Create a bar chart for comparison of accuracies
models = list(accuracy_comparison.keys())
scores = list(accuracy_comparison.values())

plt.figure(figsize=(8, 6))
plt.bar(models, scores, color=['blue', 'green'])
plt.title('Comparison of R2 Values (Model Accuracy)')
plt.ylabel('R2 Score')
plt.ylim(0, 0.2)
plt.show()

# Plot actual vs predicted values for Random Forest Model
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_rf_pred, alpha=0.3)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', lw=2) # Perfect prediction line
plt.title('Actual vs Predicted Weekly Sales (Random Forest)')
plt.xlabel('Actual Weekly Sales')
plt.ylabel('Predicted Weekly Sales')
plt.show()

# Plot feature importances from the Random Forest model
importances = rf_model.feature_importances_
feature_names = X.columns

# Sort the feature importances in descending order
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))
plt.bar(range(len(importances)), importances[indices], align='center')
plt.xticks(range(len(importances)), feature_names[indices], rotation=90)
plt.title('Feature Importance (Random Forest)')
plt.show()
```


Comparison of R² Values (Model Accuracy)

 Output image

Actual vs Predicted Weekly Sales (Random Forest)

 Output image

Feature Importance (Random Forest)

 Output image

Here are the visualizations for comparing the models and understanding the Random Forest's performance:

1. Comparison of R^2 Values (Accuracy):

- **Linear Regression:** $R^2 = 0.017$
- **Random Forest:** $R^2 = 0.101$ The Random Forest model outperforms the linear regression in terms of accuracy, explaining more variance in the sales data.

2. Actual vs Predicted Weekly Sales (Random Forest):

- This scatter plot shows the relationship between actual and predicted sales. The closer the points are to the red line, the better the predictions. The Random Forest model still has a lot of variance but is an improvement over linear regression.

3. Feature Importance (Random Forest):

- The bar chart shows the importance of each feature in predicting weekly sales. This helps identify which factors have the most significant impact, with **Year**, **Temperature**, and **Holiday_Flag** being key features.

Would you like to further refine the analysis or explore any specific parts of the data in more depth?



generate the confusion matrix



Confusion matrices are used for classification models, not regression models. Since we're working with a regression task (predicting continuous sales values), a confusion matrix is not applicable in this context.

For regression models, we typically evaluate performance using metrics like **Mean Squared Error (MSE)**, **Mean Absolute Error (MAE)**, and **R-squared (R^2)**, as we have already done.

If you are interested in classification-based tasks, such as predicting a specific sales range or classifying stores into categories (e.g., high vs. low sales), we can convert the regression problem into a classification problem and then generate a confusion matrix.

Would you like to explore this direction, or should we continue improving the current regression model?



Print the confusion matrix



Since a confusion matrix applies to classification problems, we can create a classification task from the current regression problem by converting sales into categories. For example, we could classify the **Weekly Sales** into categories like "Low", "Medium", and "High."

Let's proceed by creating three categories of sales and then generating the confusion matrix for a classification model (e.g., a decision tree or random forest classifier).

I'll begin by creating categories for the sales data and generating a confusion matrix.

python

```
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier

# Convert weekly sales into categories (Low, Medium, High)
# Define thresholds based on percentiles
sales_labels = pd.cut(y, bins=[-1, 500000, 1500000, np.inf], labels=['Low', 'Medium', 'High'])

# Split the data into training and testing sets for classification
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(X,
sales_labels, test_size=0.2, random_state=42)

# Train a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train_class, y_train_class)

# Predict on the test data
y_pred_class = rf_classifier.predict(X_test_class)
```

```
# Generate the confusion matrix
conf_matrix = confusion_matrix(y_test_class, y_pred_class)

conf_matrix
```

Result

```
array([[123, 16, 152],
       [ 20, 103, 131],
       [ 90, 93, 559]])
```

Here is the confusion matrix for the classification task, where we categorized **Weekly Sales** into "Low", "Medium", and "High" categories:

	Predicted Low	Predicted Medium	Predicted High
Actual Low	123	16	152
Actual Medium	20	103	131
Actual High	90	93	559

Interpretation:

- **123** instances were correctly classified as "Low".
- **103** instances were correctly classified as "Medium".
- **559** instances were correctly classified as "High".
- Misclassifications include, for example, 152 "Low" sales being predicted as "High" sales.

This matrix helps evaluate the classification performance by comparing predicted vs. actual categories. If you'd like, we can also calculate metrics like precision, recall, and F1-score based on this matrix. Let me know if you'd like to proceed!



Please proceed



python

```
from sklearn.metrics import classification_report

# Generate classification report for precision, recall, and F1-score
class_report = classification_report(y_test_class, y_pred_class, target_names=['Low',
'Medium', 'High'])

class_report
```

Result

```
'          precision    recall  f1-score   support\n\n         Low           0.53      0.42      0.47
```

Here is the **classification report** for the Random Forest classifier:

- **Low Sales:**
 - Precision: 0.53
 - Recall: 0.42
 - F1-score: 0.47
- **Medium Sales:**
 - Precision: 0.49
 - Recall: 0.41
 - F1-score: 0.44
- **High Sales:**
 - Precision: 0.66
 - Recall: 0.75
 - F1-score: 0.71
- **Overall Accuracy:** 61%

The classification report shows that the model performs better at predicting "High" sales compared to "Low" or "Medium" sales. Precision, recall, and F1-scores are generally moderate, but we could explore feature engineering or more advanced models to improve this.

Let me know if you'd like to delve deeper into any specific area or further improve the model!

ChatGPT can make mistakes. Check important info.