Professional Data & Business Analytics

Batch - 05

Instructor: Joy Dhon Chakma

Python Portfolio Project: A Case Study

Store Analysis: S0020

Lutfor Rahman Sohan

Table of Contents

Data Loading and Preparation

The analysis begins by importing necessary libraries such as pandas, numpy, matplotlib, and seaborn for data manipulation and visualization, as well as scikit-learn for machine learning tasks. Several data files are loaded from an Excel file ("case-study-data.xlsx") into pandas DataFrames: fact_table, trans_dim, item_dim, customer_dim, time_dim, and store_dim.

These DataFrames are then merged to create a comprehensive dataset named fact_trans_item_cust_time_store. A subset of this data is selected, focusing on the store with the key 'S0020', and stored in a DataFrame called store_data. This selection ensures that all subsequent analysis is specific to this particular store.

The head() function displays the first few rows of the store_data DataFrame to verify that the data has been loaded correctly.

Codeshot-1:

```
fact_table = pd.read_excel("case-study-data.xlsx", sheet_name = "Fact_table", engine='openpyxl')
trans_dim = pd.read_excel("case-study-data.xlsx", sheet_name = "Trans_dim", engine='openpyxl')
item_dim = pd.read_excel("case-study-data.xlsx", sheet_name = "Item_dim", engine='openpyxl')
customer_dim = pd.read_excel("case-study-data.xlsx", sheet_name = "Customer_dim", engine='openpyxl')
time_dim = pd.read_excel("case-study-data.xlsx", sheet_name = "Time_dim", engine='openpyxl')
store_dim =  pd.read_excel("case-study-data.xlsx", sheet_name = "Store_dim", engine='openpyxl')

print("data has been loaded successfully!!")
```

✓ 8.7s

```
data has been loaded successfully!!
```

Codeshot-2:

```
fact_trans = pd.merge(fact_table, trans_dim,on= 'payment_key')
fact_trans_item = pd.merge(fact_trans, item_dim, on='item_key')
fact_trans_item_cust = pd.merge(fact_trans_item, customer_dim, on= 'customer_key')
fact_trans_item_cust_time = pd.merge(fact_trans_item_cust, time_dim, on= 'time_key')
fact_trans_item_cust_time_store = pd.merge(fact_trans_item_cust_time, store_dim, on= 'store_key')
```

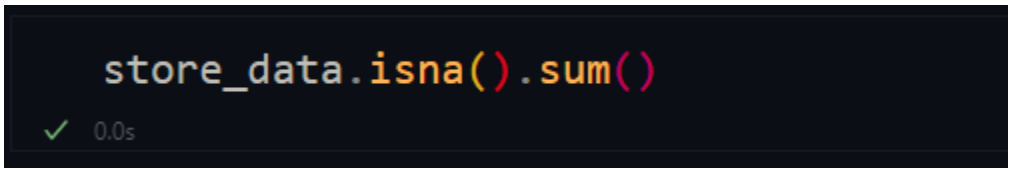
✓ 0.1s

Data Exploration and Cleaning

Initial descriptive statistics (describe().T) is generated for store_data to understand the distribution of numerical features like 'quantity_sold', 'unit_price_x', and 'total_price', among others. Information about the data types and non-null counts is obtained using info(), which shows that there are 1801 entries with 39 columns, some of which are numerical (int64, float64) and others are object types.

The number of missing values in each column is then assessed using isna().sum(), revealing missing values in columns such as 'unit_x', 'bank_name', 'name', and 'street'. To handle these missing values, the mode of each column is calculated and used to fill the missing entries. The 'unit_y' column is dropped. After cleaning the data, the first 5 rows of the cleaned store_data are displayed.

Codeshot-3:



Codeshot-4:

```
mode_street= store_data['street'].mode()[0]
store_data['street'].fillna(mode_street, inplace=True)
```
✓ 0.0s

```
mode_name= store_data['name'].mode()[0]
store_data['name'].fillna(mode_name, inplace=True)
```
✓ 0.0s

```
mode_bank= store_data['bank_name'].mode()[0]
store_data['bank_name'].fillna(mode_bank, inplace=True)
```
✓ 0.0s

```
mode_unit= store_data['unit_x'].mode()[0]
store_data['unit_x'].fillna(mode_bank, inplace=True)
```
✓ 0.0s

Descriptive Analysis

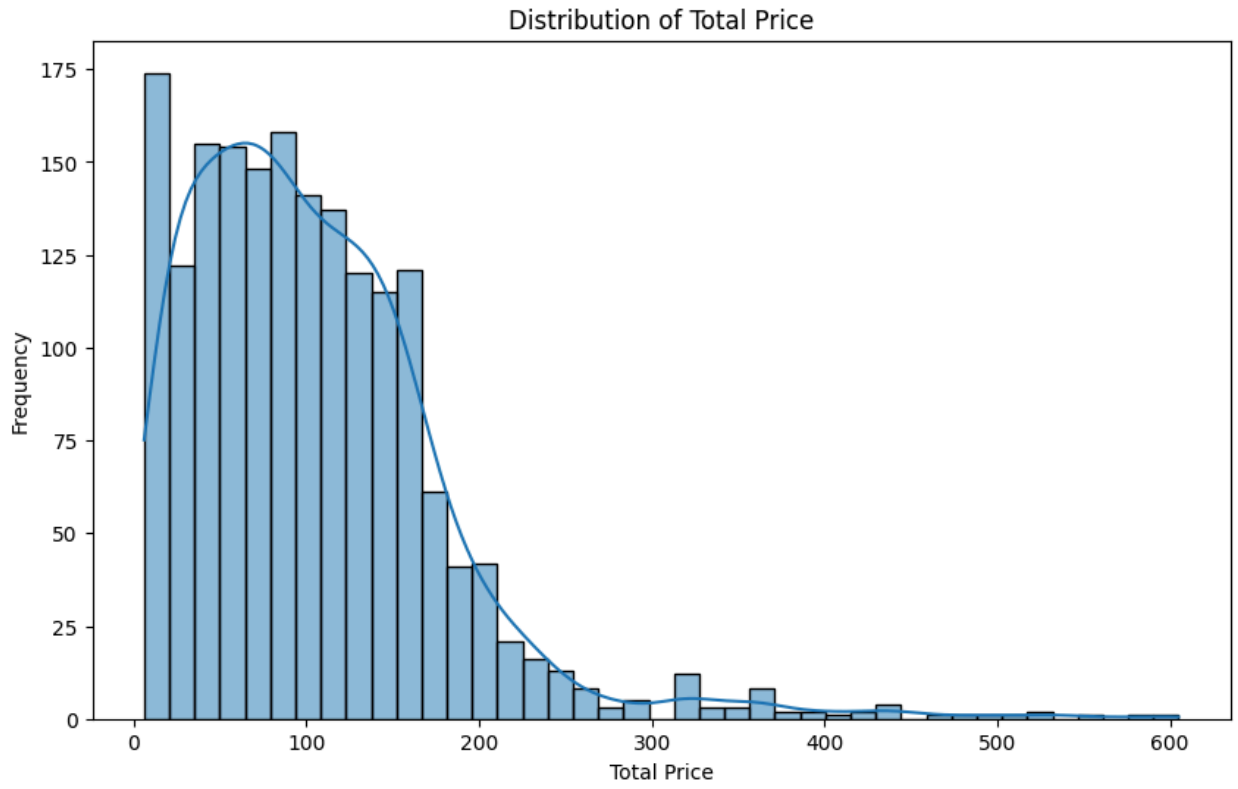
Distribution of Total Price

The distribution of 'total_price' is visualized using a histogram with a Kernel Density Estimate (KDE) to understand its shape.

Code:

```
plt.figure(figsize=(10, 6))
sns.histplot(store_data['total_price'], kde=True)
plt.title('Distribution of Total Price')
plt.xlabel('Total Price')
plt.ylabel('Frequency')
plt.show()
```

Visualization:



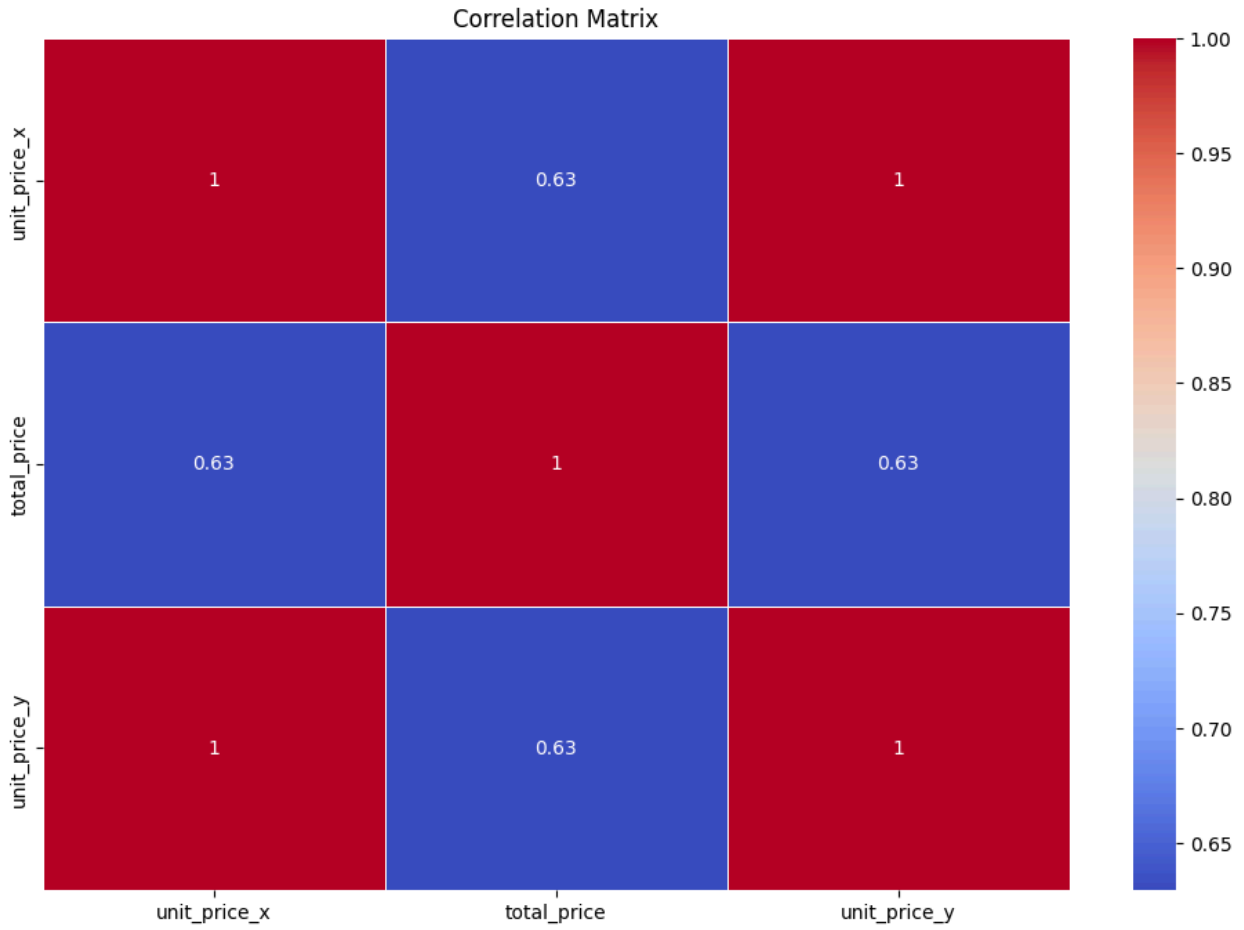
Correlation Matrix

A correlation matrix heatmap is generated using the select_dtypes function on store_data to identify relationships between numerical features, showing how these variables are related to each other.

Code:

```python
# Correlation matrix
correlation_matrix = store_data.select_dtypes('float64', 'int64').corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

Visualization



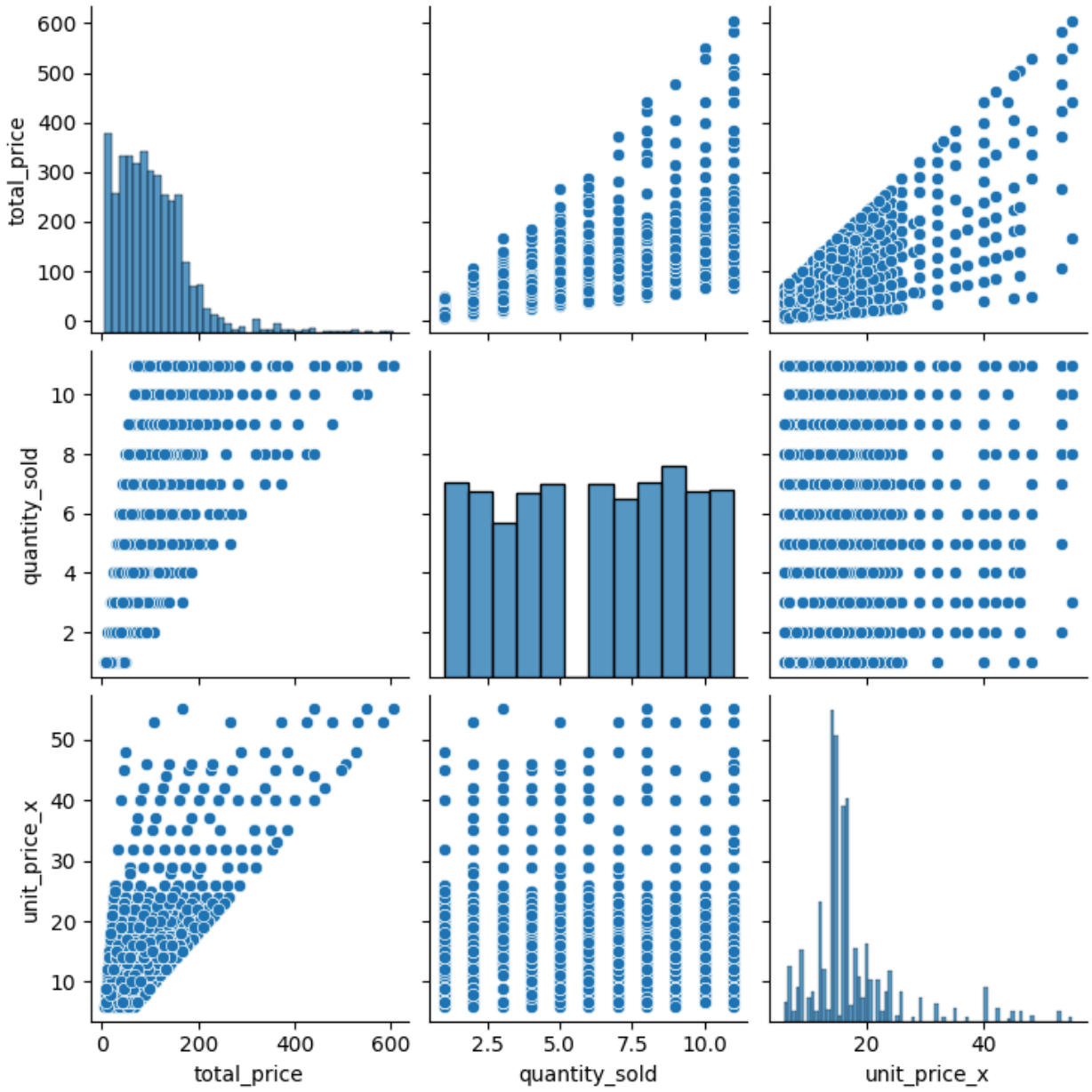
Pairplot to Visualize Relationships

Pair plots are created to visualize relationships between 'total_price', 'quantity_sold', 'unit_price_x', and 'unit_x'.

Code:

```python
# Pairplot to visualize relationships
sns.pairplot(store_data[['total_price', 'quantity_sold', 'unit_price_x', 'unit_x']])
plt.show()
```

Visualization:



Total Revenue by Division

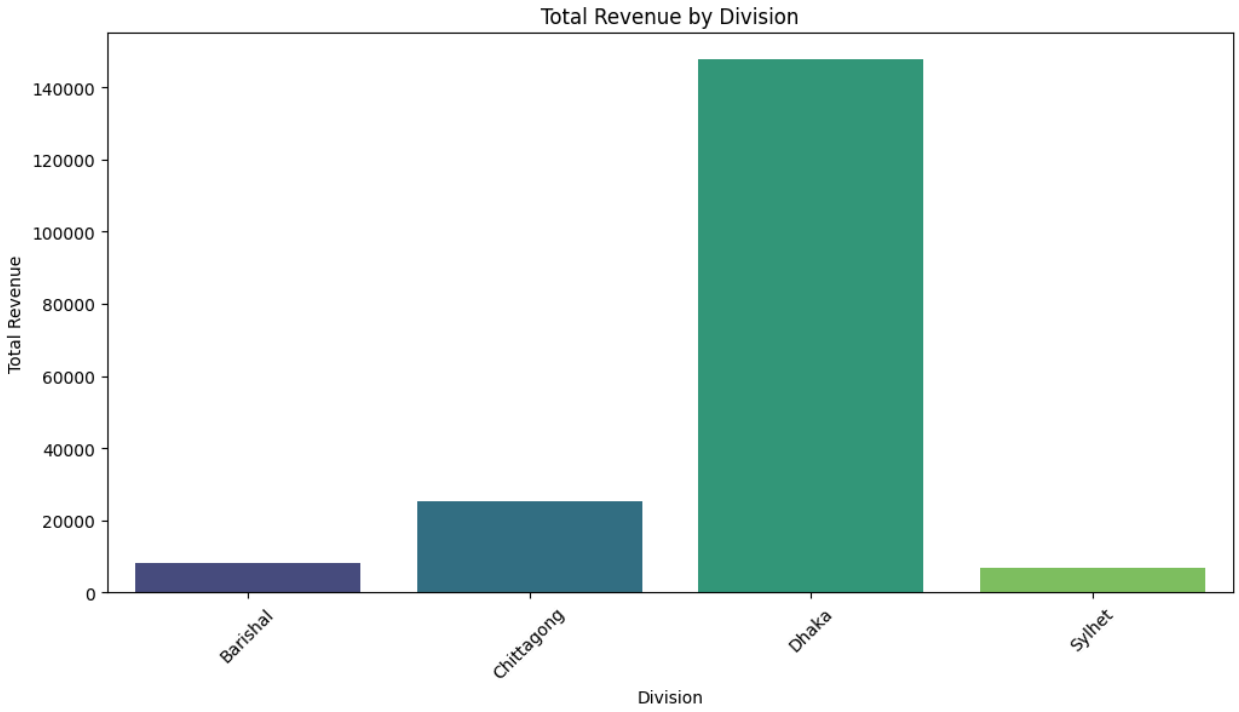
Total revenue is calculated and grouped by division and then displayed in descending order, revealing that **Dhaka generates the highest revenue**.

A bar chart visualizes total revenue by division, offering a visual comparison of revenue generation across different regions.

Code:

```python
total_revenue_by_division = store_data.groupby('division_x')['total_price'].sum().reset_index()
print(total_revenue_by_division.sort_values(by= 'total_price', ascending= False))
plt.figure(figsize=(12, 6))
sns.barplot(x='division_x', y='total_price', data=total_revenue_by_division, palette='viridis')
plt.title('Total Revenue by Division')
plt.xlabel('Division')
plt.ylabel('Total Revenue')
plt.xticks(rotation=45)
plt.show()
```

Visualization:



Monthly sales trends

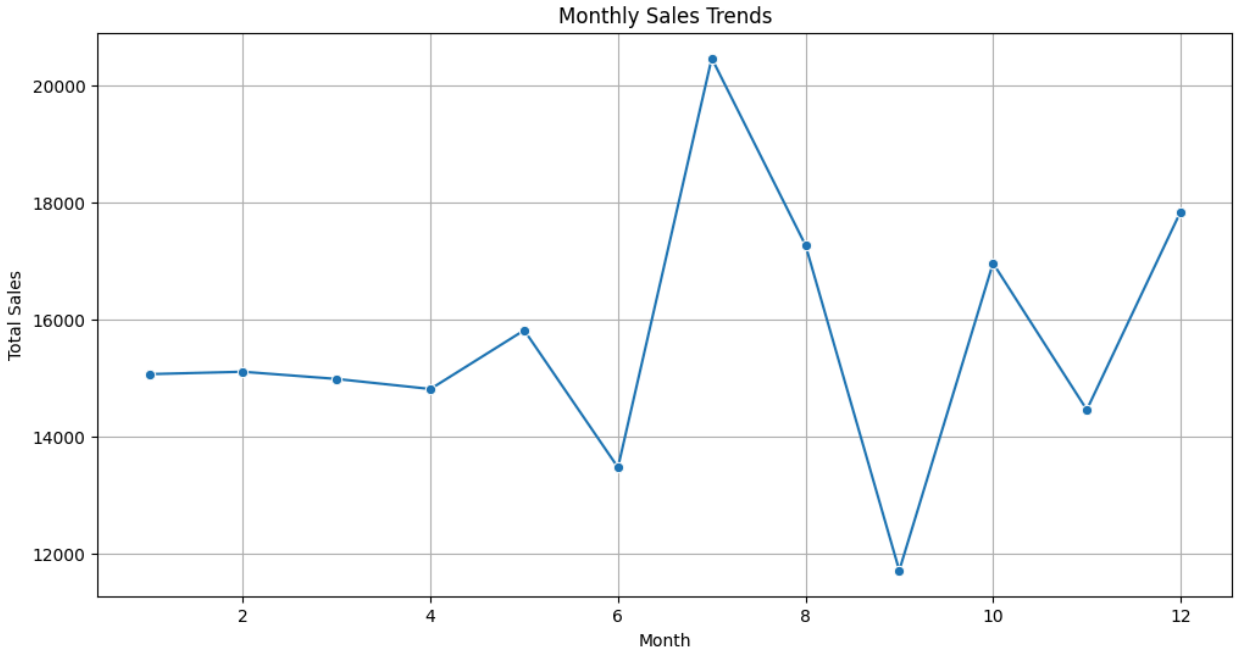
Monthly sales trends are analyzed by summing the total_price for each month and then visualized using a line plot. The plot shows fluctuations in sales across the months, with **July being the highest-grossing month**.

Code:

```
monthly_sales = store_data.groupby('month')['total_price'].sum().reset_index()
print(monthly_sales.sort_values(by= 'month'))
plt.figure(figsize=(12, 6))
sns.lineplot(x='month', y='total_price', data=monthly_sales, marker='o')
plt.title('Monthly Sales Trends')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.grid(True)
plt.show()
```

0.0s

Visualization:



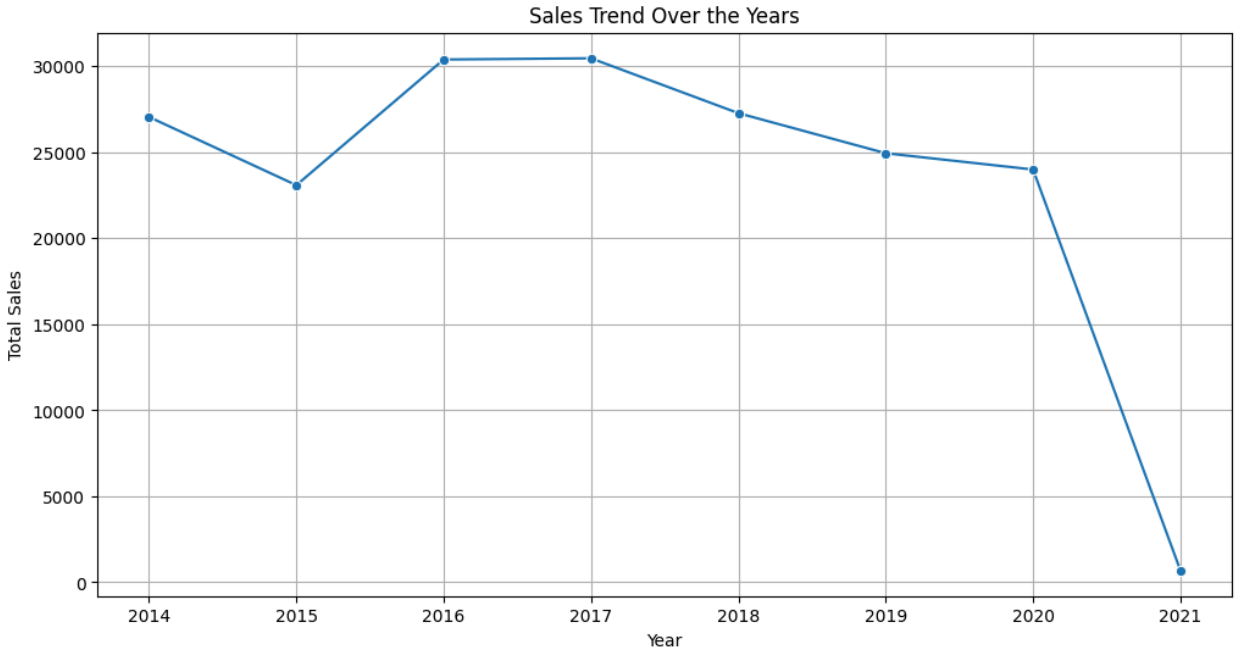
Sales trends over the years

Sales trends over the years are examined using a line plot, and it is revealed that **2017 had the highest total sales**. Whereas the lowest total sales was in 2015. Here, 2021 data is deprecated as the dataset doesn't contain complete year data.

Code:

```
yearly_sales = store_data.groupby('year')['total_price'].sum().reset_index()
print(yearly_sales)
plt.figure(figsize=(12, 6))
sns.lineplot(x='year', y='total_price', data=yearly_sales, marker='o')
plt.title('Sales Trend Over the Years')
plt.xlabel('Year')
plt.ylabel('Total Sales')
plt.grid(True)
plt.show()
```

Visualization:



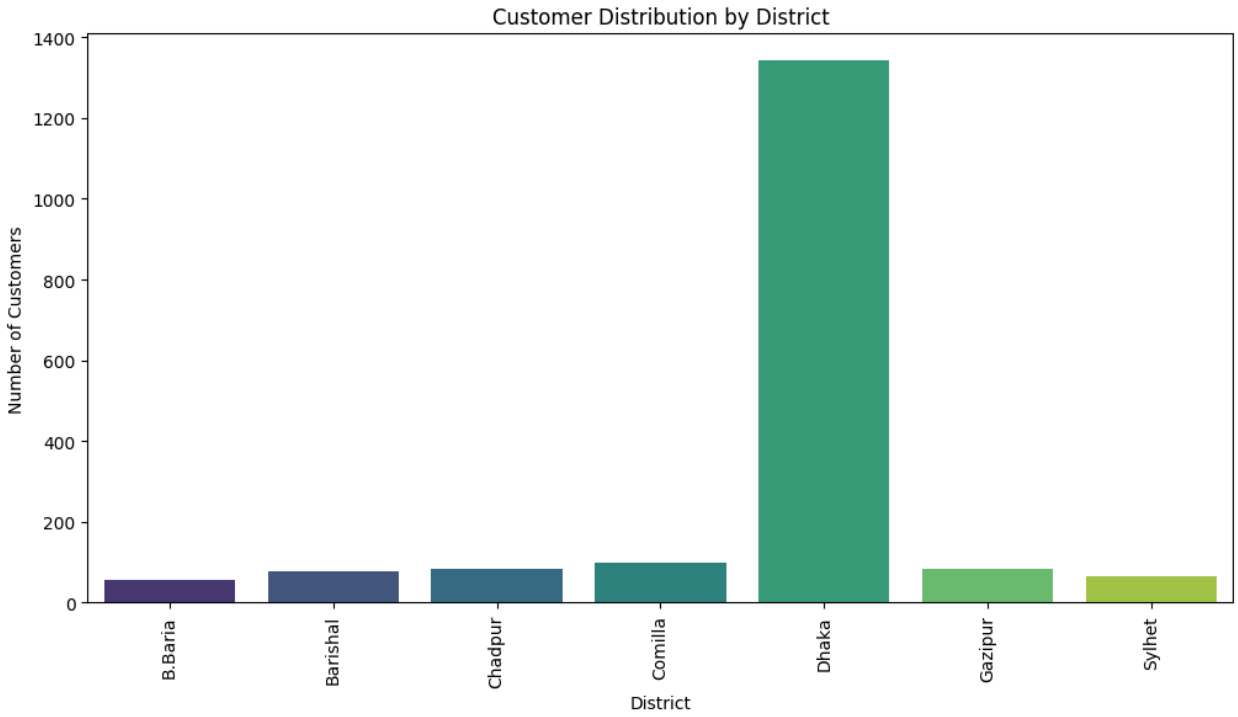
Customer distribution by district

Customer distribution by district is examined by counting the number of customers in each district, which is then displayed in a bar chart. The visualization shows that **Dhaka has the highest number of customers**.

Code:

```
customer_distribution = store_data.groupby('district_x')['customer_key'].count().reset_index()
print(customer_distribution)
plt.figure(figsize=(12, 6))
sns.barplot(x='district_x', y='customer_key', data=customer_distribution, palette='viridis')
plt.title('Customer Distribution by District')
plt.xlabel('District')
plt.ylabel('Number of Customers')
plt.xticks(rotation=90)
plt.show()
```

Visualization:



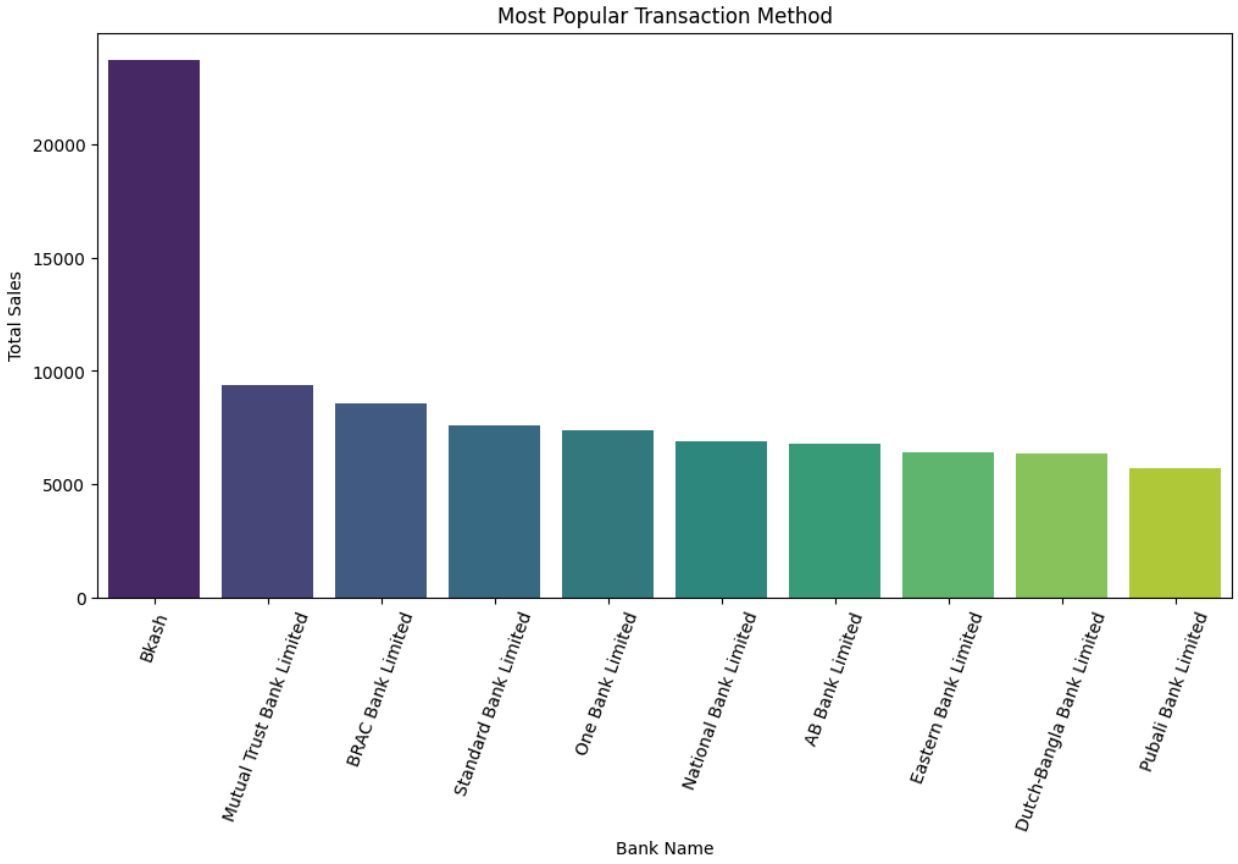
Transaction Method Analysis

Here, I identify the major transaction methods by our valuable customers and I displayed that with a bar chart for the top 10. **Bkash is the highest revenue-generating transaction method**.

Code:

```
sales_by_bank = store_data.groupby('bank_name')['total_price'].sum().reset_index().sort_values(by='total_price', ascending=False).head(10)
print(sales_by_bank)
plt.figure(figsize=(12, 6))
sns.barplot(x='bank_name', y='total_price', data=sales_by_bank, palette='viridis')
plt.title('Sales by Bank Name')
plt.xlabel('Bank Name')
plt.ylabel('Total Sales')
plt.xticks(rotation=70)
plt.show()
```

Visualization:



Profit analysis based on the item type

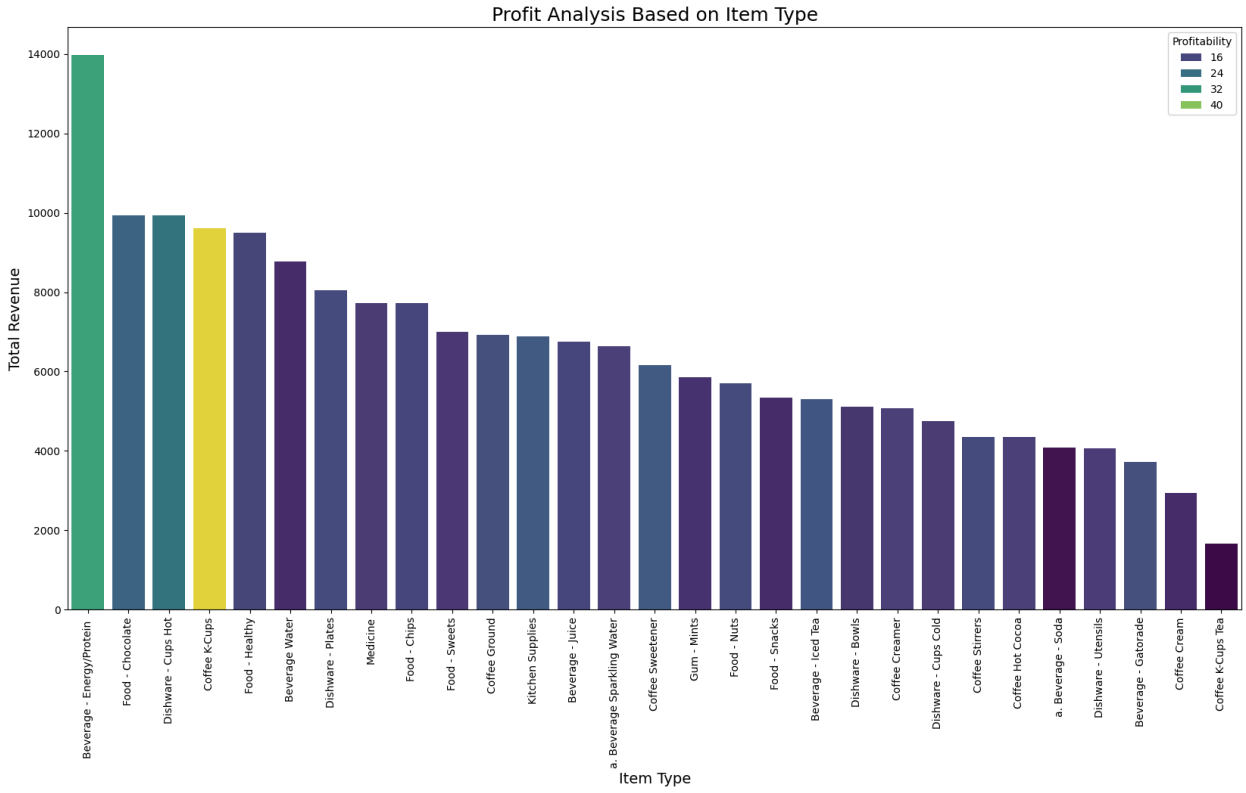
Profitability analysis is conducted based on the item type. The analysis calculates the total quantity sold, total revenue, and the number of unique transactions for each item type.

It then calculates profitability by dividing total revenue by total quantity sold and then the product performance is displayed in descending order based on total revenue with a bar chart.

Code:

```python
product_performance = store_data.groupby('item_type').agg(
    total_quantity_sold=('quantity_sold', 'sum'),
    total_revenue=('total_price', 'sum'),
    num_transactions=('payment_key', 'nunique')
).reset_index()

product_performance['profitability'] = (
    product_performance['total_revenue'] / product_performance['total_quantity_sold']
)

product_performance = product_performance.sort_values(by='total_revenue', ascending=False)
print(product_performance.head(10))

plt.figure(figsize=(20, 10))
sns.barplot(x='item_type',
            y='total_revenue',
            hue='profitability',
            data=product_performance,
            palette='viridis')

plt.title('Profit Analysis Based on Item Type', fontsize=18)
plt.xlabel('Item Type', fontsize=14)
plt.ylabel('Total Revenue', fontsize=14)
plt.legend(title='Profitability')
plt.xticks(rotation=90)
plt.show()
```

Visualization:



Customer Segment Analysis

Customer segmentation is done by calculating total spend, purchase frequency, average basket size, and the last purchase date for each customer. Customers are then segmented into categories: 'Low-Value', 'Medium-Value', 'High-Value', and 'VIP' based on their total spending.

- Low-Value = Customers who have spent equal or less than 100
- Medium-value = Customers who have spent equal or less than 500
- Hight-value = Customers who have spent more than 500.

Code:

```
customer_segmentation = store_data.groupby('customer_key').agg(
    total_spend=('total_price', 'sum'),
    purchase_frequency=('payment_key', 'nunique'),
    avg_basket_size=('quantity_sold', 'mean'),
    last_purchase_date=('date', 'max')
).reset_index()

customer_segmentation['segment'] = pd.cut(
    customer_segmentation['total_spend'],
    bins=[0, 100, 500, float('inf')],
    labels=['Low-Value', 'Medium-Value', 'High-Value']
)

print(customer_segmentation.sort_values(by= 'total_spend', ascending= False))
```

Visualization:

	customer_key	total_spend	purchase_frequency	avg_basket_size	\
1377	C007738	640.00	2	9.5	
1139	C006492	605.00	1	11.0	
159	C000959	583.00	1	11.0	
1192	C006779	550.00	1	10.0	
902	C005224	540.00	2	9.5	
...	...	...	...	...	
826	C004725	8.00	1	1.0	
1414	C007977	8.00	1	1.0	
217	C001339	6.75	1	1.0	
1529	C008681	6.00	1	1.0	
150	C000900	6.00	1	1.0	

	last_purchase_date	segment
1377	2017-02-09	High-Value
1139	2020-06-11	High-Value
159	2014-04-27	High-Value
1192	2020-07-15	High-Value
902	2019-03-24	High-Value
...	...	...
826	2016-07-25	Low-Value
1414	2020-05-06	Low-Value
217	2019-01-15	Low-Value
1529	2015-03-11	Low-Value
150	2018-11-25	Low-Value

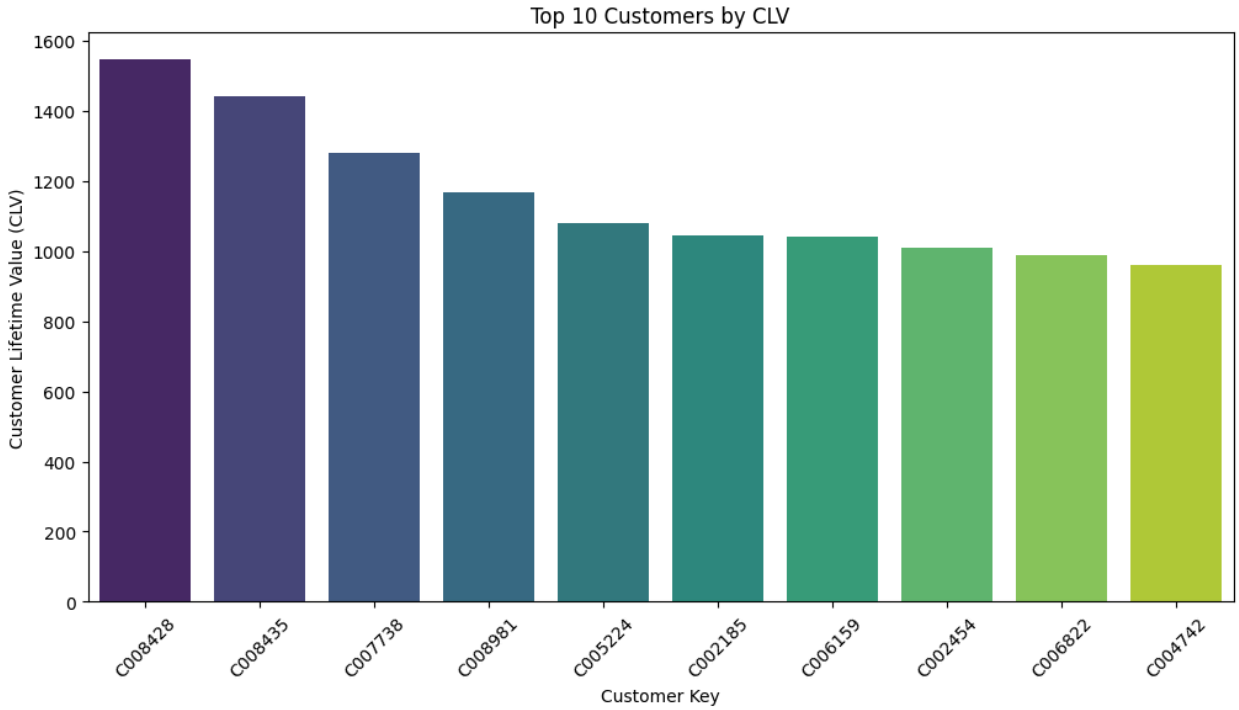
Customer lifetime value prediction

Customer Lifetime Value (CLV) is computed for each customer by multiplying their total spend by their purchase frequency. A bar chart shows the top 10 customers with the highest CLV.

Code:

```python
clv_data = store_data.groupby('customer_key').agg(
    total_spend=('total_price', 'sum'),
    purchase_frequency=('payment_key', 'nunique'),
    avg_order_value=('total_price', 'mean'),
).reset_index()

clv_data['clv'] = clv_data['total_spend'] * clv_data['purchase_frequency']
clv_data = clv_data.sort_values(by='clv', ascending=False)
print(clv_data.head(10))

top_10_clv = clv_data.head(10)
plt.figure(figsize=(12, 6))
sns.barplot(x='customer_key', y='clv', data=top_10_clv, palette='viridis')
plt.title('Top 10 Customers by CLV')
plt.xlabel('Customer Key')
plt.ylabel('Customer Lifetime Value (CLV)')
plt.xticks(rotation=45)
plt.show()
```

Visualization:



Predictive Analysis

Linear Regression Model

In this section, I trained the dataset using the Linear Regression model to predict the total price value for the respective store. I used 'item_type', 'quantity_sold', 'unit_price_x', and 'quarter' as features to predict the total price.

From the dataset, I had two categorical variables ('item_type' and 'quarter') and encoded them by using **Label Encoding.** Here, I trained the 80% dataset, where 20% of the data was for testing.

Therefore, I evaluated the model's performance using Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared score. Below, I listed the respective scores for these terms:

- **Mean Absolute Error:** 14.354436470650079 - makes an absolute error of **14.35 units** in its predictions.
- **Mean Squared Error:** 808.69751348827
- **R^2 Score:** 88.81946973030972

The MAE and R^2 score is fairly acceptable, but our MSE could be better. However, this can be adjusted by adding more features to the training dataset.

However, I have done real-world problem solutions based on the model! Let's say, the respective **Store wants to know how much they can generate by selling Beverage Water in Quarter 4**. I found out the result and it is very much closer to the other Q4 average earnings. The prediction **result was: 150.914.**

Code:

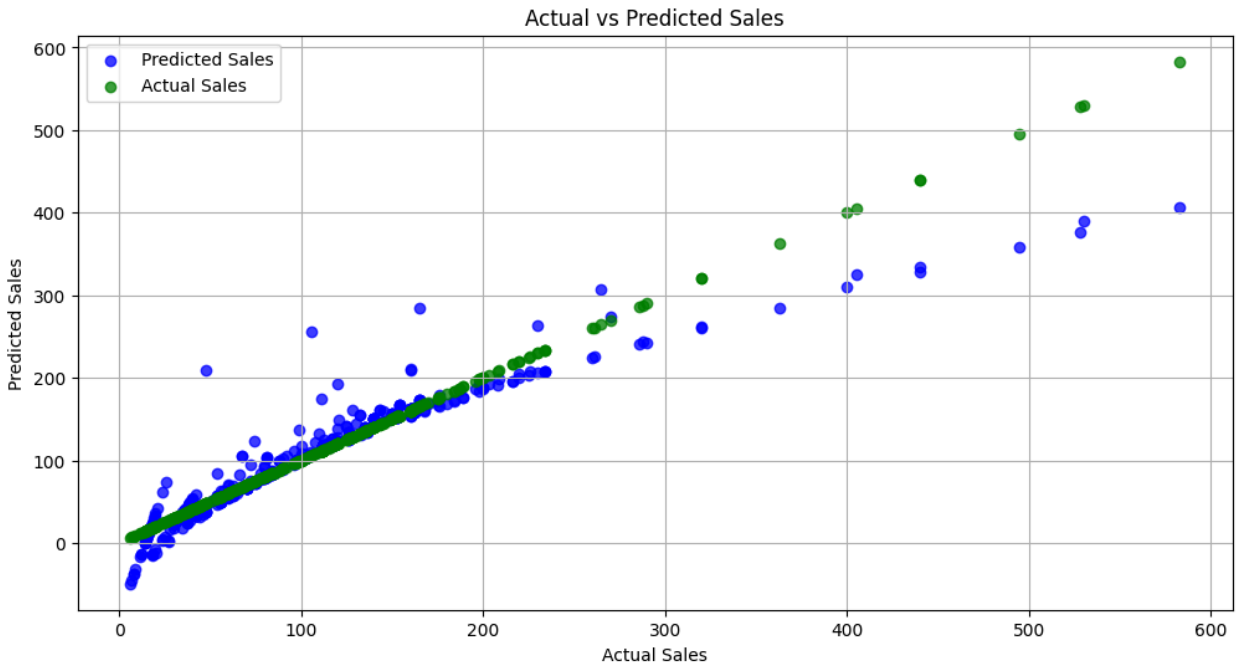
```
pred_data = store_data[[ 'item_type', 'quantity_sold', 'unit_price_x','quarter']]

X = pred_data
y = store_data['total_price']

# Encode categorical variables
lb_encoders = {}
label_cols = pred_data.select_dtypes('object')

for col in label_cols:
    lb_encoders[col] = LabelEncoder()
    X[col] = lb_encoders[col].fit_transform(X[col])

# Step-2: Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step-3: Train the model
model = LinearRegression()
model.fit(X_train, y_train)
```

Code-1

```
# Step-4: Make predictions
y_pred = model.predict(X_test)

# Step-5: Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Absolute Error: {mae}')
print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2 * 100}')

# Step-6: Plot actual vs predicted values
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred, alpha=0.75, color='blue', label='Predicted Sales')
plt.scatter(y_test, y_test, alpha=0.75, color='green', label='Actual Sales')
plt.xlabel('Actual Sales')
plt.ylabel('Predicted Sales')
plt.title('Actual vs Predicted Sales')
plt.legend()
plt.grid()
plt.show()
```

Code-2

Visualization:



Prediction:

```
# Prediction
# Beverage Water - 4
# avg_quantity_sold - 7, target-10
# avg_unit_price - 14
# Q4- 3

task = [4, 10, 14, 3]
task_prediction = model.predict([task])
print(task_prediction[0])
```

✓ 0.0s

```
150.91460749860866
```

Sales Prediction and Revenue Forecasting

The sales forecasting analysis in the provided data uses time series analysis with the Exponential Smoothing model to predict future revenue and sales. This method analyzes historical sales data, specifically the "**total price**" and "**quantity sold**" aggregated by date, to forecast future sales.

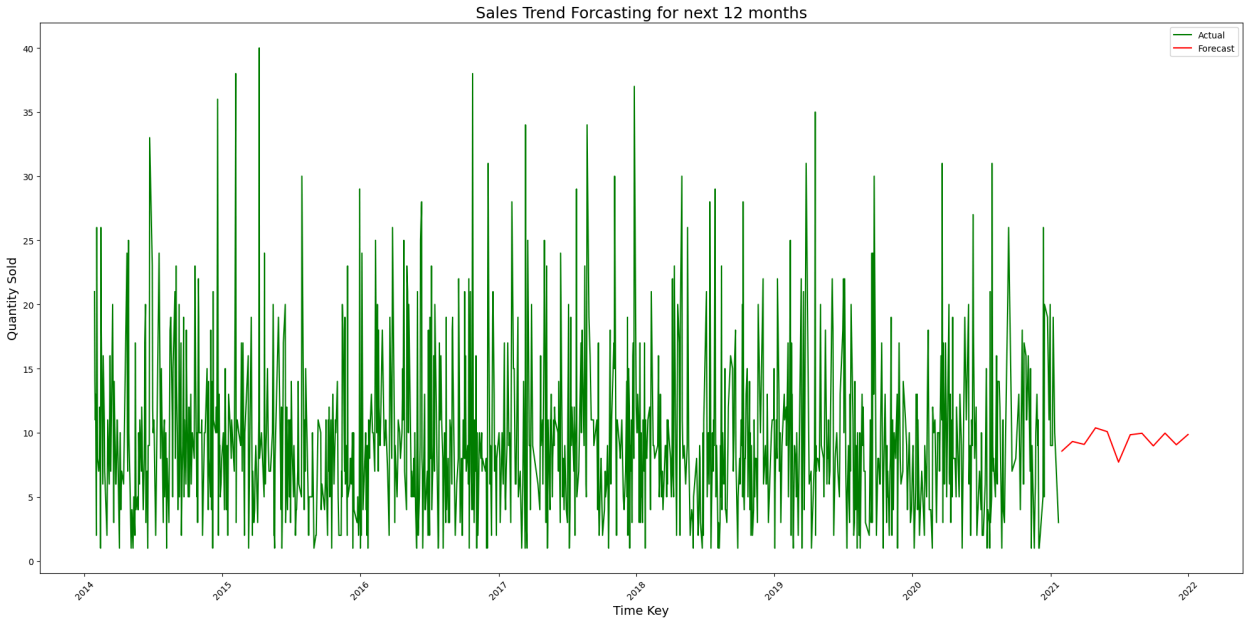
The Exponential Smoothing model is configured with an 'add' trend and an 'add' seasonal component, which is appropriate for data that shows a consistent pattern of change over time along with recurring seasonal fluctuations. A seasonal period of 12 months is applied to capture yearly sales patterns.

The time series analysis with the Exponential Smoothing model provides a comprehensive overview of sales trends and seasonality, which helps in strategic business planning and forecasting future revenue. The approach focuses on broader trends and seasonal patterns rather than specific factors that impact sales.

Sales Trend Forecasting Code:

```
store_data['date'] = pd.to_datetime(store_data['date'])
store_data['date'] = store_data['date'].dt.date
store_data['date']

time_series_df = store_data.groupby('date')['quantity_sold'].sum().reset_index()
time_series_df['date'] = pd.to_datetime(time_series_df['date'])

model = ExponentialSmoothing(time_series_df['quantity_sold'], trend='add', seasonal='add', seasonal_periods=12)
fit = model.fit()

forecast = fit.forecast(steps=12)
forecast_index = pd.date_range(start=time_series_df['date'].iloc[-1], periods=12, freq='M')

plt.figure(figsize=(20, 10))
plt.plot(time_series_df['date'], time_series_df['quantity_sold'], label='Actual', color='green')
plt.plot(forecast_index, forecast, label='Forecast', color='red')
plt.title('Sales Forcasting for next 12 months', fontsize = 18)
plt.xlabel('Time Key', fontsize = 14)
plt.ylabel('Total Price', fontsize = 14)
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

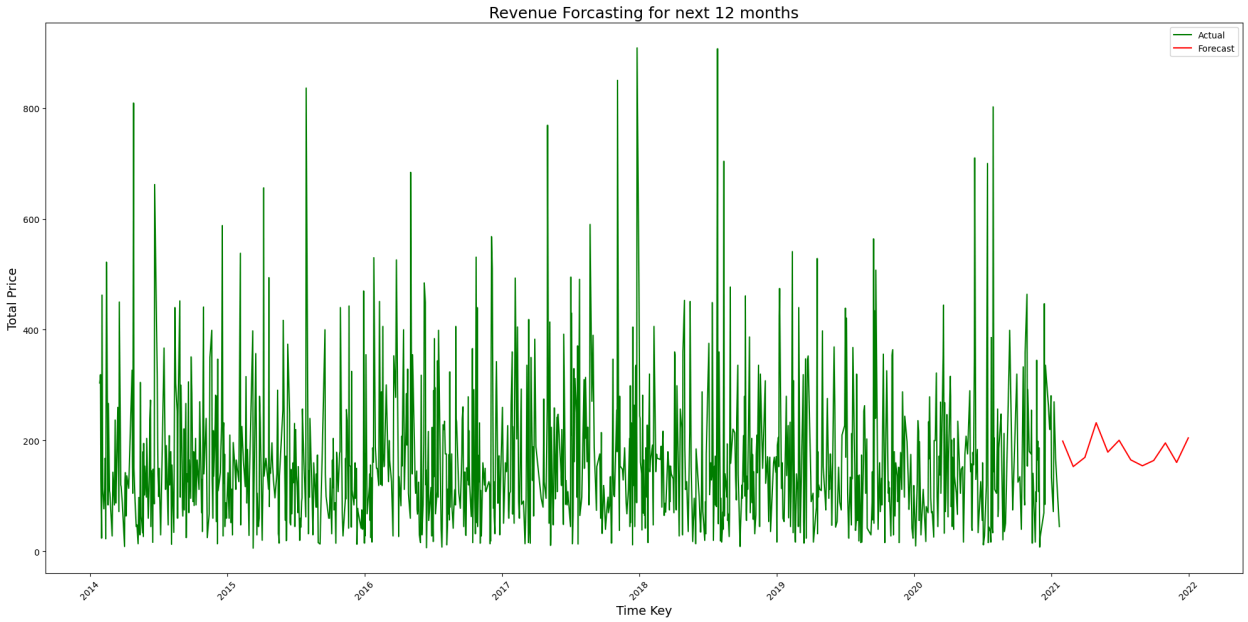
Sales Trend Forecasting Visualization:



Revenue Trend Forecasting Code:

```
time_series_df = store_data.groupby('date')['total_price'].sum().reset_index()
time_series_df['date'] = pd.to_datetime(time_series_df['date'])

model = ExponentialSmoothing(time_series_df['total_price'], trend='add', seasonal='add', seasonal_periods=12)
fit = model.fit()

forecast = fit.forecast(steps=12)
forecast_index = pd.date_range(start=time_series_df['date'].iloc[-1], periods=12, freq='M')

plt.figure(figsize=(20, 10))
plt.plot(time_series_df['date'], time_series_df['total_price'], label='Actual', color='green')
plt.plot(forecast_index, forecast, label='Forecast', color='red')
plt.title('Revenue Forcasting for next 12 months', fontsize = 18)
plt.xlabel('Time Key', fontsize = 14)
plt.ylabel('Total Price', fontsize = 14)
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Revenue Trend Forecasting Visualization:



— The End —

Table of Contents with Numbers