Professional Data & Business Analytics

Batch - 05

Instructor: Joy Dhon Chakma

Python Portfolio Project: A Case Study

Product Item Analysis on: I00181 and I00177.

Lutfor Rahman Sohan

Table of Contents

Data Loading and Preparation

The analysis begins by loading the necessary Python libraries including pandas, numpy, matplotlib, and seaborn, for data manipulation, numerical operations, and visualizations, as well as scikit-learn for modeling.

My dataset is in Excel format and it is organized in the **Star Schema format**. It represents a dimension or a fact table in the data model: **fact table, transaction dimension, item dimension, customer dimension, time dimension, and store dimension.**

The datasets are then merged to form a comprehensive dataframe, fact_trans_item_cust_time_store. This comprehensive dataset contains 39 columns including information on payment keys, customer keys, time keys, item keys, store keys, quantity sold, unit prices, total price, transaction types, item names, customer information, date, time, and location.

The analysis is then narrowed to filter the dataset focusing on two specific item keys, **'I00181' and 'I00177'** creating filtered_dataset which is then used for the rest of the analysis. Initial data cleaning steps involve checking for missing values using the .isna().sum() method and filling missing values in the 'street', 'name', and 'bank_name' columns with their respective modes.

Code:

```
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!!")
```

Codeshot-1

```
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')
```

Codeshot-2

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

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

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

Codeshot-3

Descriptive Analysis

Descriptive analytics provides crucial information about a company's performance by tracking progress and comparing it with other businesses. It offers insights into **current business performance**, **historical trends**, and **strengths and weaknesses**. This involves identifying key metrics, gathering necessary data, preparing it for analysis, and using tools like spreadsheets to analyze the data. The results are often presented visually using charts or graphs.

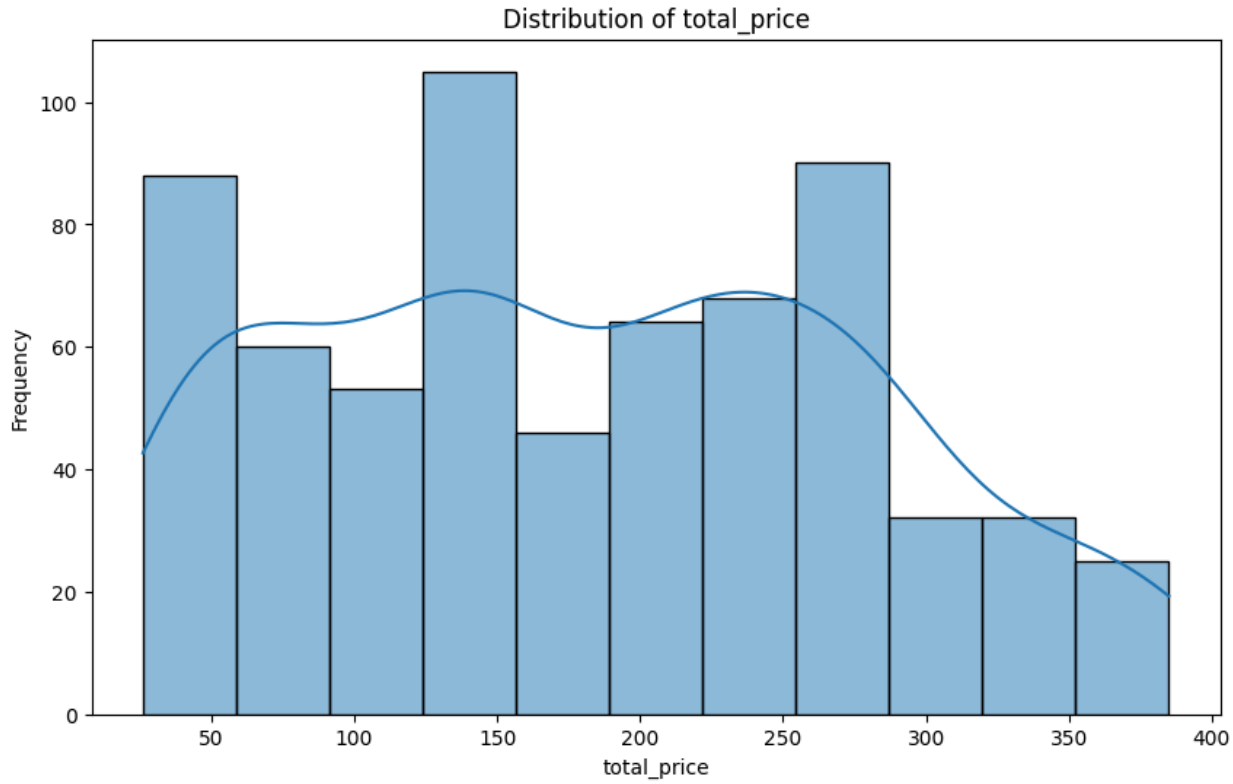
Distribution of Total Price

A histogram is used to visualize the distribution of the ‘total_price’ variable, providing insights into its frequency across different price ranges. This helps understand the typical transaction amounts and identify any potential outliers.

Code:

```
plt.figure(figsize=(10, 6))
sns.histplot(filtered_dataset['total_price'], kde=True)
plt.title('Distribution of total_price')
plt.xlabel('total_price')
plt.ylabel('Frequency')
plt.show()
```
0.1s

Visualization:



Overall Sales Performance

The total sales are calculated by summing up the 'total_price' column which is 121764.0. The average sales, calculated by finding the mean of the 'total_price' column is approximately 183.66. The total quantity of items sold was 3993.

Code:

```python
total_sales = filtered_dataset['total_price'].sum()
average_sales = filtered_dataset['total_price'].mean()
total_quantity_sold = filtered_dataset['quantity_sold'].sum()

print(f"Total Sales: {total_sales}")
print(f"Average Sales: {average_sales}")
print(f"Total Quantity Sold: {total_quantity_sold}")
```

✓ 0.0s

```
Total Sales: 121764.0
Average Sales: 183.65610859728505
Total Quantity Sold: 3993
```

Total Revenue by Product

The total revenue generated by each of the two items is calculated. **Item I00177 generated a total revenue of 69790.0 while Item I00181 generated a total revenue of 51974.0**. A bar plot visualizes the revenue for each item, confirming that **item I00177 is the top-performing item**.

Code:

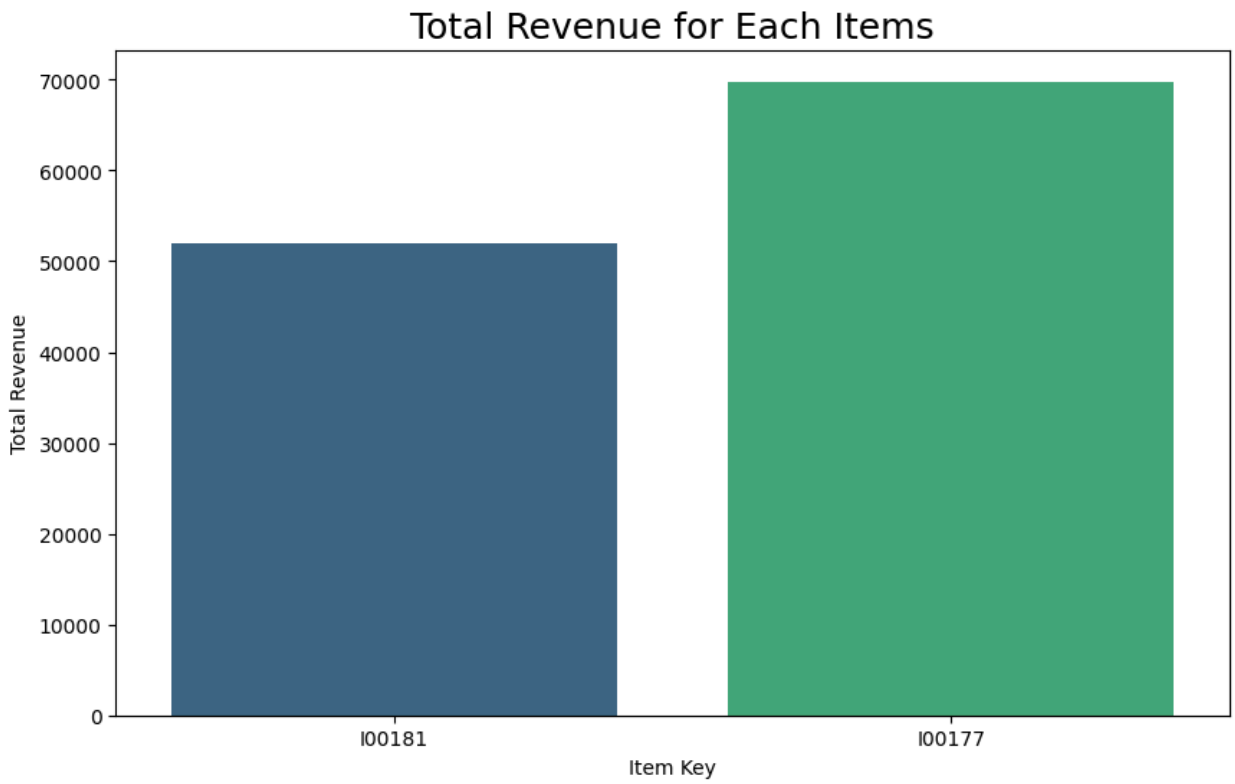
```python
I00181_total_rev = filtered_dataset[filtered_dataset['item_key'] == 'I00181']['total_price'].sum()
I00177_total_rev = filtered_dataset[filtered_dataset['item_key'] == 'I00177']['total_price'].sum()
data = {
    'Item Key': ['I00181', 'I00177'],
    'Total Revenue': [I00181_total_rev, I00177_total_rev]
}
df = pd.DataFrame(data)
print(df)

plt.figure(figsize=(10, 6))
sns.barplot(x='Item Key',
            y='Total Revenue',
            data=df,
            palette='viridis')
plt.title('Total Revenue for Each Items', fontsize = 18)
plt.xlabel('Item Key')
plt.ylabel('Total Revenue')
plt.show()
```

✓ 0.0s

```
  Item Key  Total Revenue
0   I00181        51974.0
1   I00177        69790.0
```

Visualization:



Monthly Revenue for a Specific Year

The code includes a function to calculate and visualize monthly revenue for a given year. When applied to the year 2020, the visualization reveals the months with the highest and lowest sales within that year. **The best revenue in 2020 is seen in the months of November and December**.

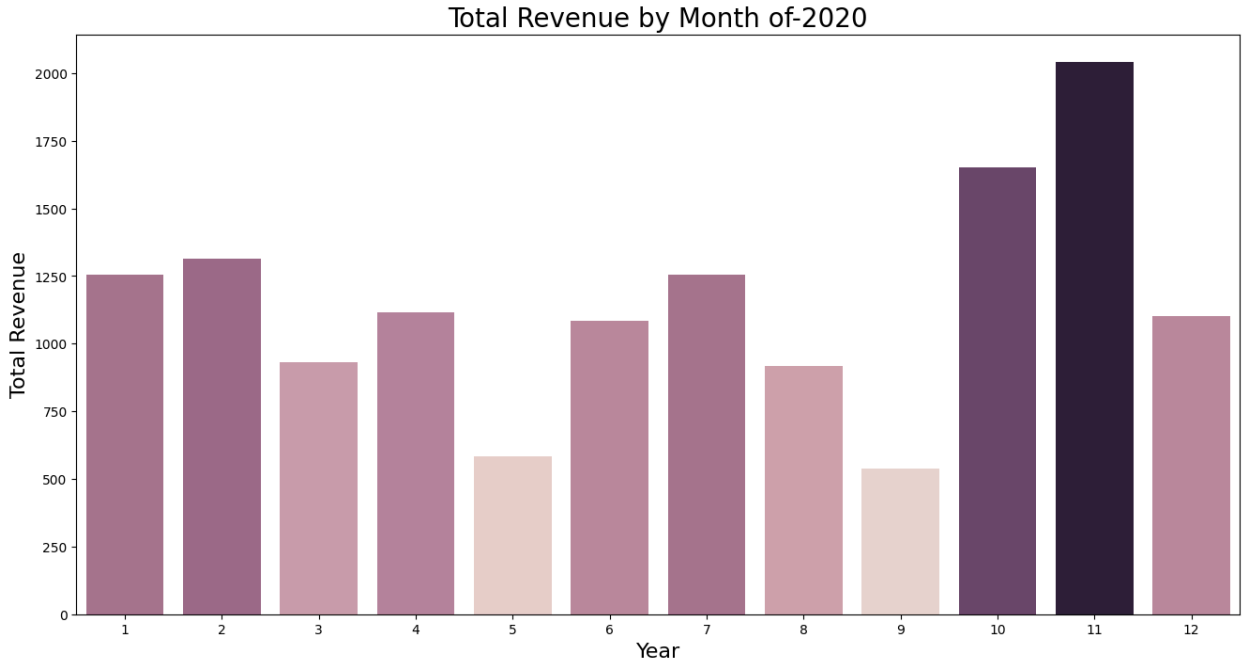
Code:

```python
def rev_by_year(year):
    rev_data = filtered_dataset[['year', 'month','total_price']]
    rev_data1 = rev_data.groupby(['year', 'month'], as_index= False)['total_price'].sum()
    x = rev_data1[rev_data1['year'] == year]

    plt.figure(figsize=(16, 8))
    sns.barplot(x='month',
            y='total_price',
            data=x,
            hue= 'total_price',
            legend=False,
            ) # palette='viridis'
    plt.title(f'Total Revenue by Month of-{year}', fontsize = 20)
    plt.xlabel('Year', fontsize = 16)
    plt.ylabel('Total Revenue', fontsize = 16)
    # plt.xticks(rotation=45)
    return plt.show()

rev_by_year(2020)
```

Visualization:



Quarterly Sales Trends

A line plot visualizes the quarterly sales trends. **Q1 is the best quarter for sales, while Q3 is the worst**.

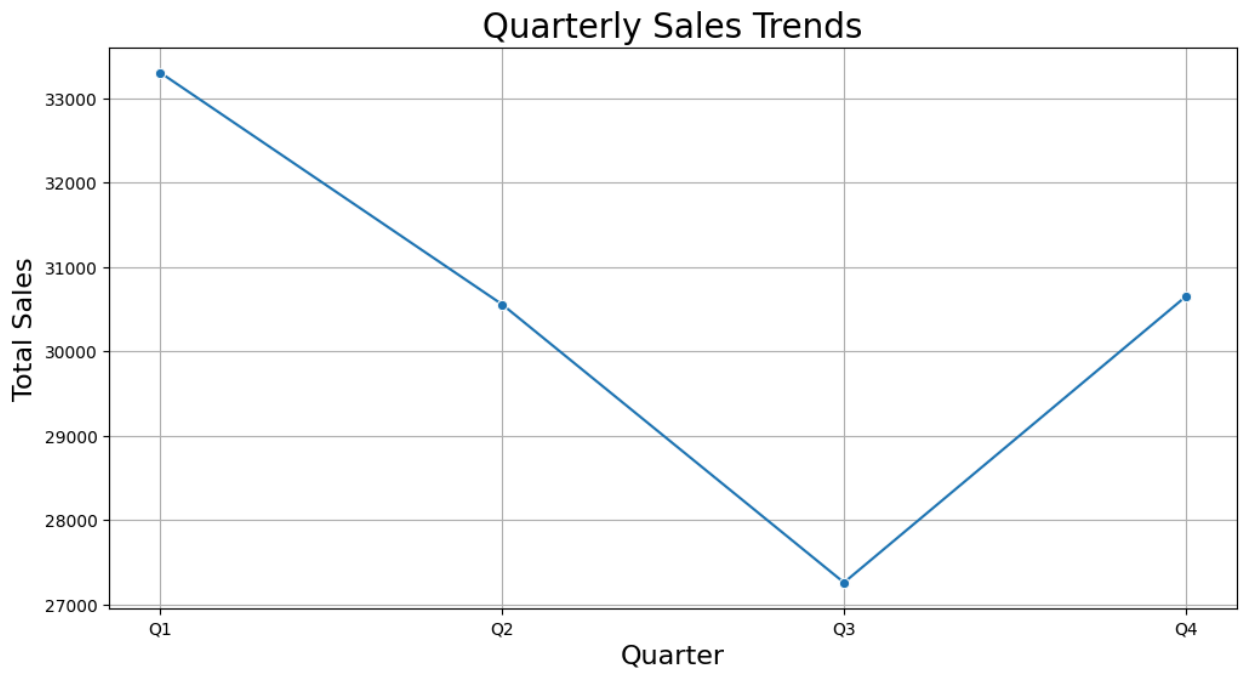
Code:

```
quarterly_sales = filtered_dataset.groupby('quarter')['total_price'].sum().reset_index()
print(quarterly_sales)
plt.figure(figsize=(12, 6))
sns.lineplot(x='quarter', y='total_price', data= quarterly_sales, marker='o')
plt.title('Quarterly Sales Trends', fontsize=20)
plt.xlabel('Quarter', fontsize=16)
plt.ylabel('Total Sales', fontsize=16)
plt.grid(True)
plt.show()
```

✓ 0.0s

```
  quarter  total_price
0      Q1      33303.0
1      Q2      30555.0
2      Q3      27257.0
3      Q4      30649.0
```

Visualization:



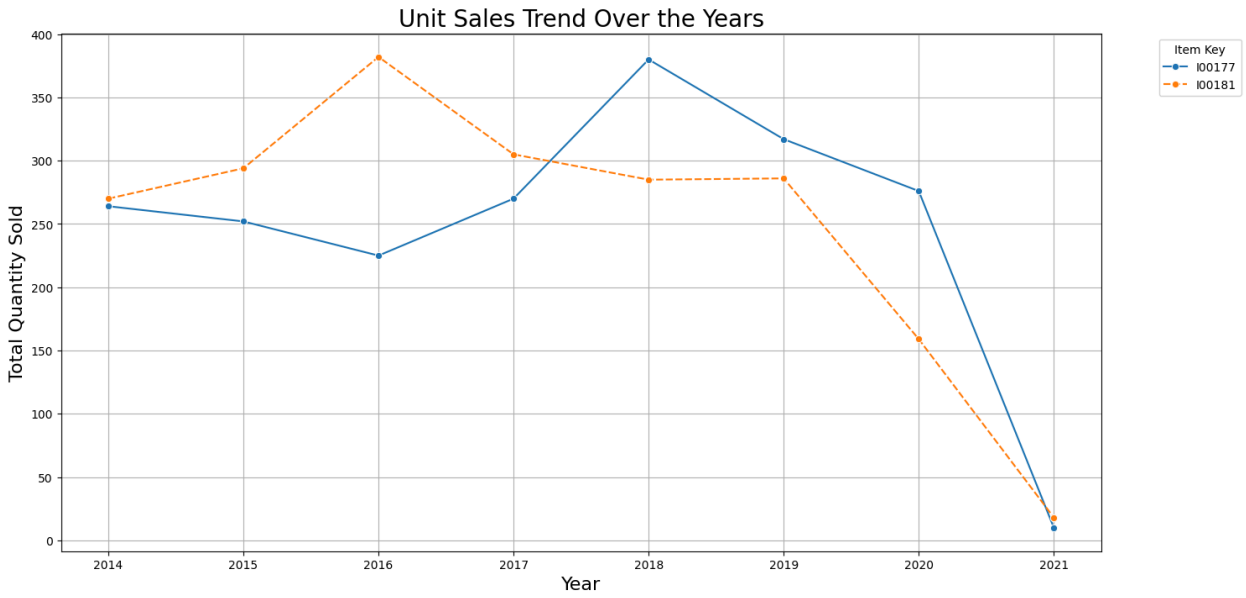
Yearly Sales Trend

Yearly sales are visualized with a line plot. **The best year in terms of sales is 2018, while the worst year is 2020**.

Code:

```
unit_sales_trend = filtered_dataset.groupby(['year', 'item_key'])['quantity_sold'].sum().reset_index()
unit_sales_trend_years = unit_sales_trend.pivot(index='item_key', columns='year', values='quantity_sold').fillna(0)
print(unit_sales_trend_years.T)
plt.figure(figsize=(16, 8))
sns.lineplot(data=unit_sales_trend_years.T, marker='o')
plt.title('Unit Sales Trend Over the Years', fontsize=20)
plt.xlabel('Year', fontsize=16)
plt.ylabel('Total Quantity Sold', fontsize=16)
plt.legend(title='Item Key', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
```

Visualization:



Store Size-wise Total Sales Price

Here, I conduct an analysis based on the store size. The results show that the large stores are performing promptly. But small stores are reportedly performing well compared to the medium size stores.

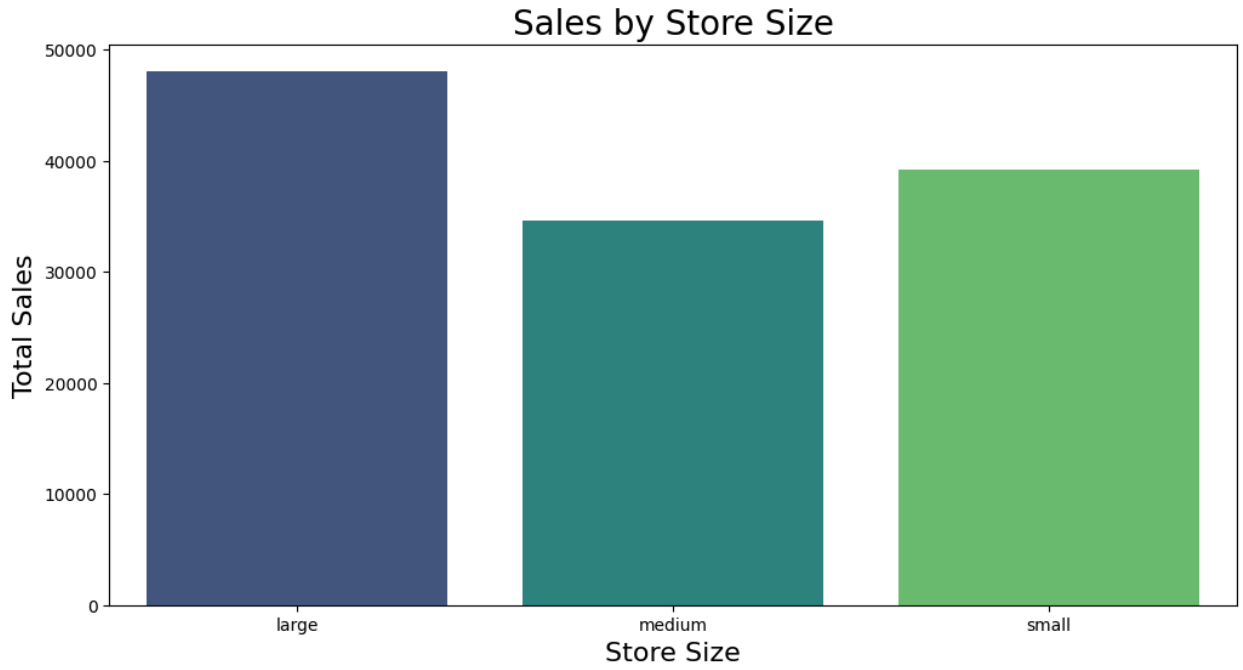
Code:

```
sales_by_store_size = filtered_dataset.groupby('store_size')['total_price'].sum().reset_index()
print(sales_by_store_size)
plt.figure(figsize=(12, 6))
sns.barplot(x='store_size', y='total_price', data=sales_by_store_size, palette='viridis')
plt.title('Sales by Store Size', fontsize=20)
plt.xlabel('Store Size', fontsize=16)
plt.ylabel('Total Sales', fontsize=16)
plt.show()
```

✓ 0.0s

	store_size	total_price
0	large	48065.0
1	medium	34565.0
2	small	39134.0

Visualization:



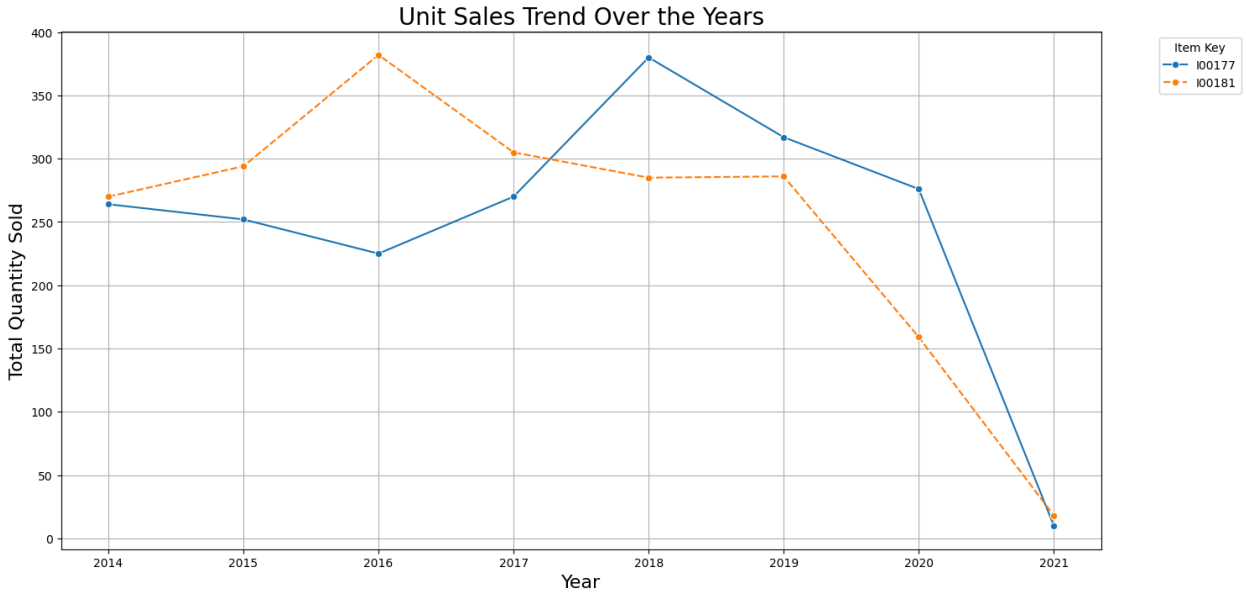
Unit Sales Trend Over the Years

A line plot displays the trend of unit sales for each item over the years. The best and worst years for each item are identified. **Item I00177 performed best in 2016 and had a bad year in 2020, while I00181 performed best in 2018 and had a bad year in 2016**.

Code:

```
unit_sales_trend = filtered_dataset.groupby(['year', 'item_key'])['quantity_sold'].sum().reset_index()
unit_sales_trend_years = unit_sales_trend.pivot(index='item_key', columns='year', values='quantity_sold').fillna(0)
print(unit_sales_trend_years.T)
plt.figure(figsize=(16, 8))
sns.lineplot(data=unit_sales_trend_years.T, marker='o')          Add to chat (Ctrl+L) | Edit highlighted code (Ctrl+I).
plt.title('Unit Sales Trend Over the Years', fontsize=20)
plt.xlabel('Year', fontsize=16)
plt.ylabel('Total Quantity Sold', fontsize=16)
plt.legend(title='Item Key', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
```

Visualization:



Sales Distribution by Division

Sales are analyzed by division, with Dhaka generating the most sales and Barishal generating the least. **The total sales in Dhaka are 93554, while the total sales in Barishal are only 4611**.

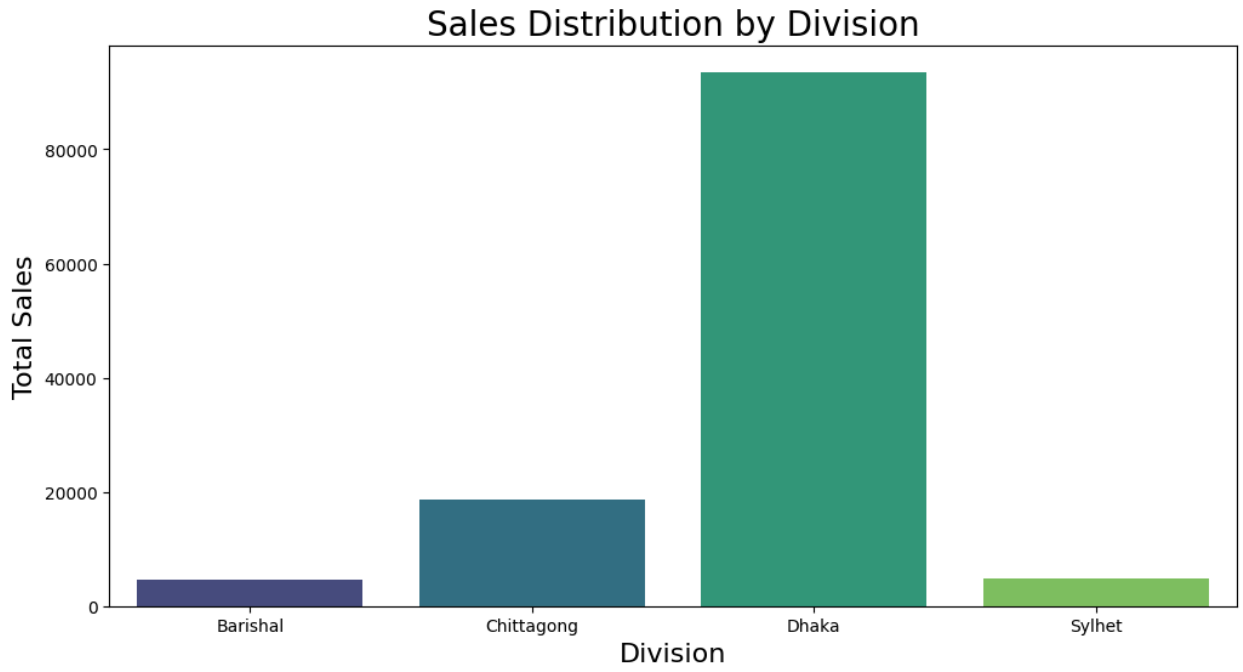
Code:

```python
sales_by_division = filtered_dataset.groupby('division_x')['total_price'].sum().reset_index()
print(sales_by_division)
plt.figure(figsize=(12, 6))
sns.barplot(x='division_x', y='total_price', data=sales_by_division, palette='viridis')
plt.title('Sales Distribution by Division', fontsize=20)
plt.xlabel('Division', fontsize=16)
plt.ylabel('Total Sales', fontsize=16)
plt.show()
```

✓ 0.0s

```
   division_x  total_price
0    Barishal       4611.0
1  Chittagong      18739.0
2       Dhaka      93554.0
3      Sylhet       4860.0
```

Visualization:



Customer Distribution by District

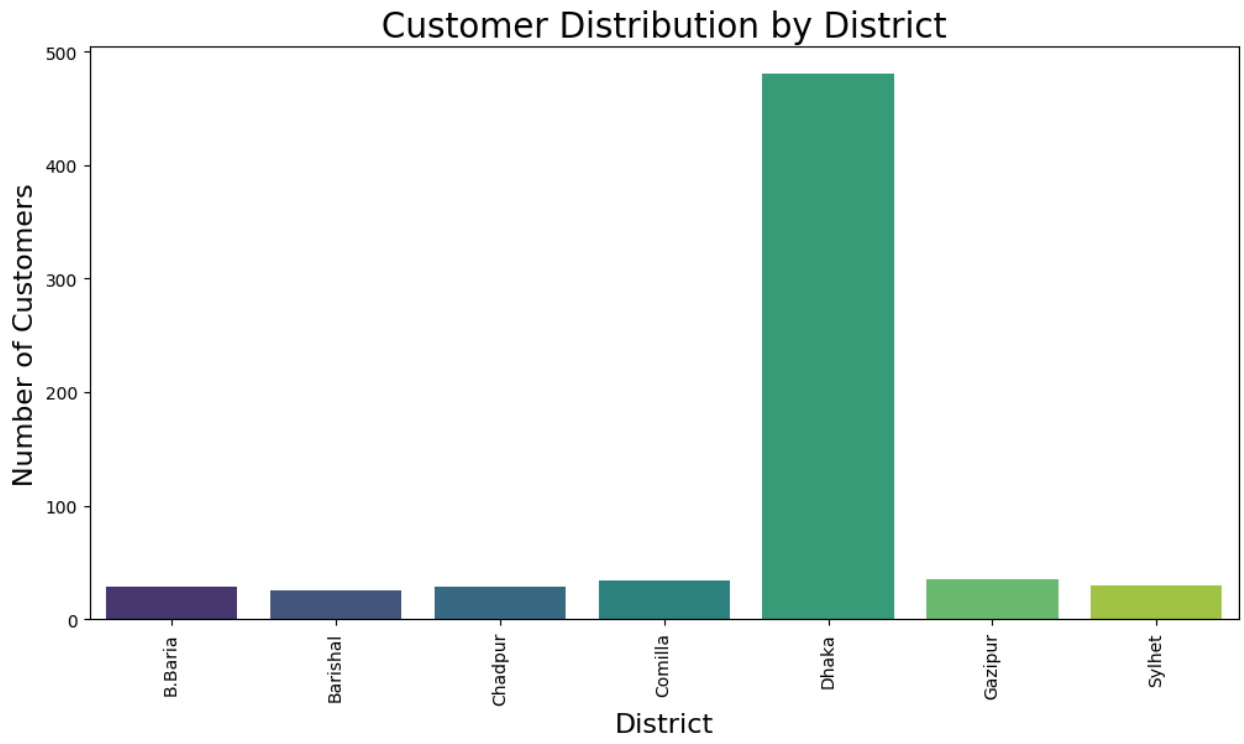
Here, I tried to find out the insights of total customer number based on their district. From that analysis, Dhaka has the hightest numbers of customers while Barishal has the lowest.

Code:

```python
customer_distribution = filtered_dataset.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', fontsize=20)
plt.xlabel('District', fontsize=16)
plt.ylabel('Number of Customers', fontsize=16)
plt.xticks(rotation=90)
plt.show()
```

✓ 0.1s

Visualization:



Total Revenue by Division

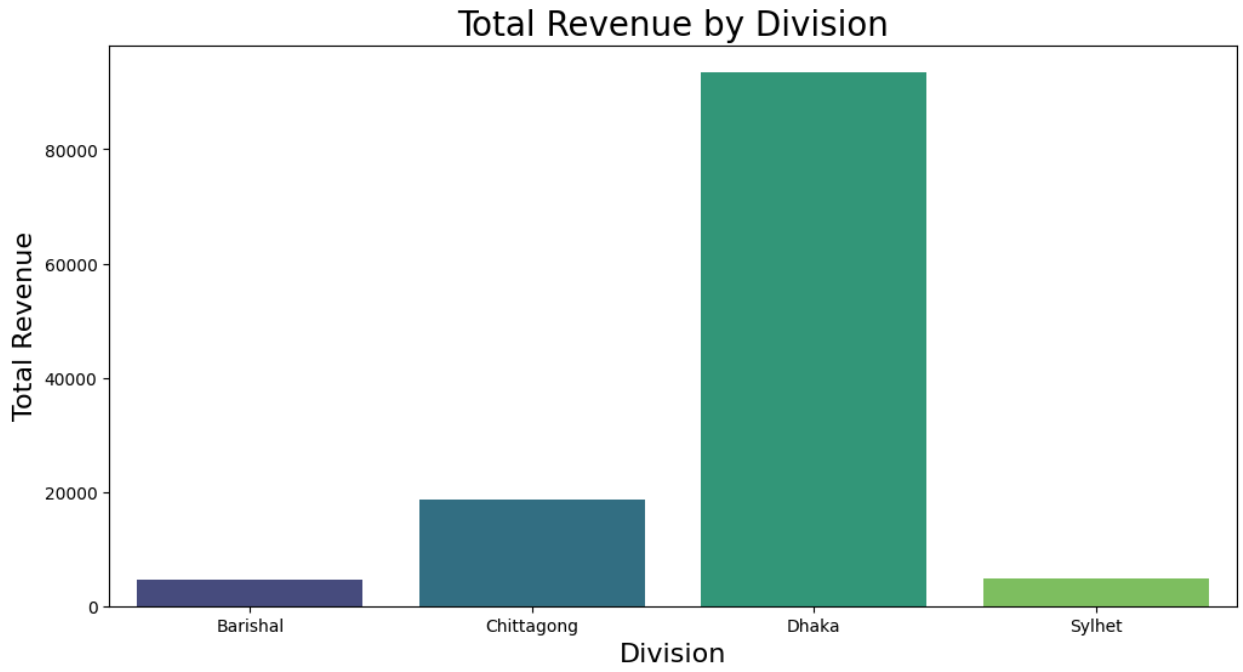
Total revenue is aggregated by division. **Dhaka has the highest revenue, while Barishal has the lowest**, this is also visualized through a bar chart.

Code:

```python
total_revenue_by_division = filtered_dataset.groupby('division_x')['total_price'].sum().reset_index()
print(total_revenue_by_division)

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', fontsize=20)
plt.xlabel('Division', fontsize=16)
plt.ylabel('Total Revenue', fontsize=16)
# plt.xticks(rotation=45)
plt.show()
```

Visualization:



Year-wise Customer Engagement for Each Division

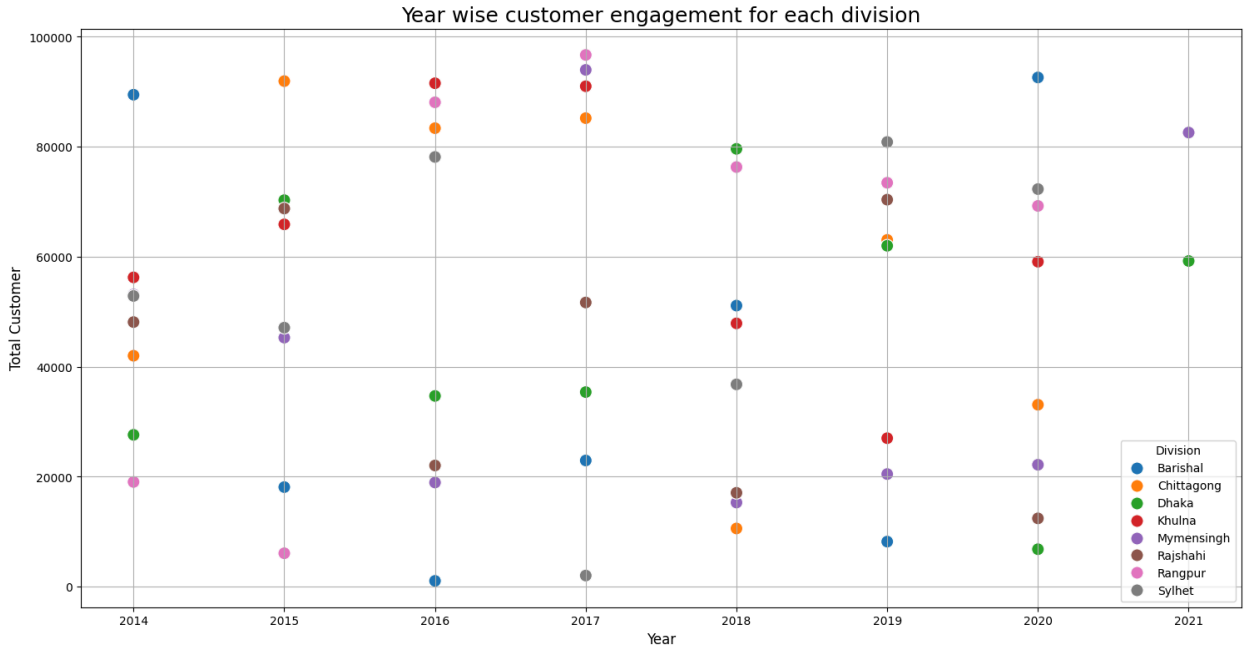
A scatter plot visualizes the customer engagement for each division over the years. The analysis reveals the best and worst-performing divisions for each year. For example, **in 2014 the best division was Barishal, while Rangpur was the underperforming division, and in 2020, Barishal was the best-performing division while Dhaka was the underperforming division**.

Code:

```
yearly_custo_num = filtered_dataset.groupby(['year', 'division_y'], as_index=False)['customer_key'].idxmax()
print(yearly_custo_num)
plt.figure(figsize=(18, 9))
sns.scatterplot(
    data=yearly_custo_num,
    x='year',
    y='customer_key',
    hue='division_y',
    palette='tab10',
    s=120
)

plt.title('Year wise customer engagement for each division', fontsize=18)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Total Customer', fontsize=12)
plt.legend(title='Division')
plt.grid()
plt.show()
```

Visualization:



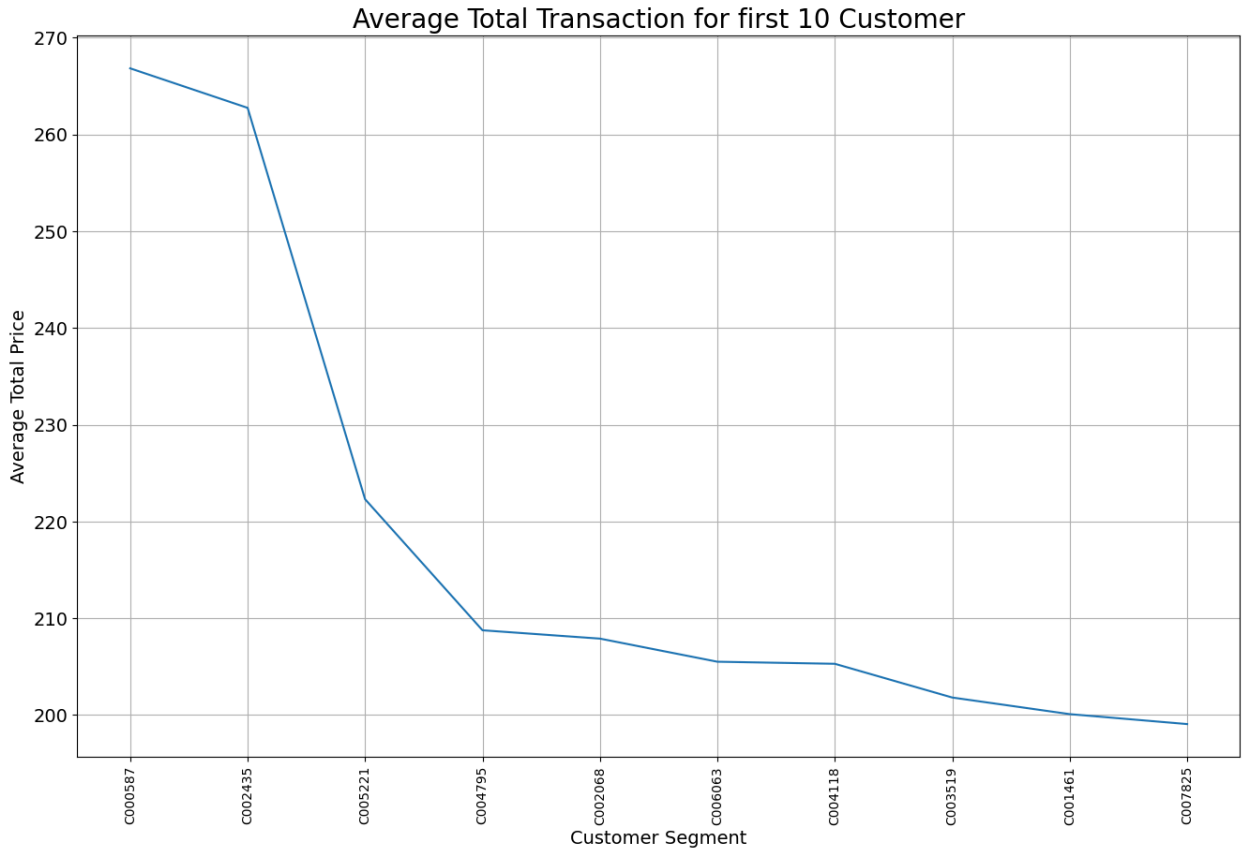
Top 10 Customers by Total Transaction

The top 10 customers with the highest average total transaction amounts are identified. For example, **customer C000587 had an average transaction of approximately 266.83 while customer C007825 had an average transaction of approximately 199.06**.

Code:

```
x = fact_trans_item_cust_time_store[['customer_key', 'total_price']]
avg = x.groupby('customer_key')['total_price'].mean().reset_index()
avg.columns = ['Customer Segment', 'Average Total Price']
y = avg.sort_values(by='Average Total Price', ascending= False).iloc[:10]
print(y)
plt.figure(figsize=(16,10))
sns.lineplot(x='Customer Segment',
             y='Average Total Price',
             data=y,
             )
plt.title('Average Total Transaction for first 10 Customer', fontsize=20)
plt.xlabel('Customer Segment', fontsize=14)
plt.xticks(rotation=90, fontsize=10)
plt.ylabel('Average Total Price', fontsize=14)
plt.yticks(fontsize=14)
plt.grid(True)
plt.show()
```

Visualization:



Total Revenue Generated by Each Store

The total revenue generated by each store is calculated, and the top 5 stores are identified. The top 5 stores are: **S0039 with a total revenue of 6409.0, S0024 with a total revenue of 5065.0, S0012 with a total revenue of 4551.0, S0035 with a total revenue of 4327.0 and S0017 with a total revenue of 4279.0**.

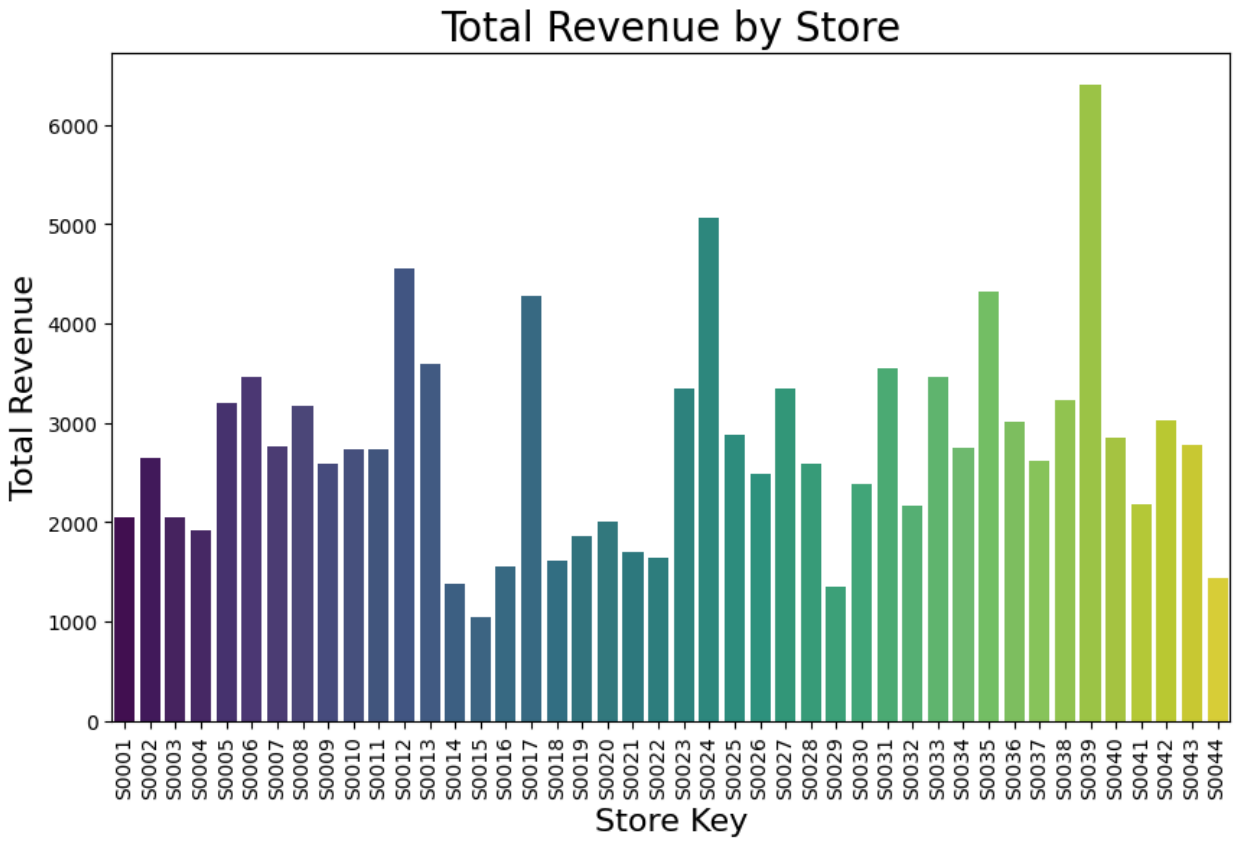
Code:

```
store_rev = filtered_dataset[['store_key', 'total_price']]
store_rev_data = store_rev.groupby('store_key')['total_price'].sum().reset_index()
print(store_rev_data.sort_values(by= 'total_price', ascending= False).head(5))

plt.figure(figsize=(10, 6))
sns.barplot(x='store_key',
            y='total_price',
            data=store_rev_data,
            hue='store_key',
            palette='viridis',
            )

plt.title('Total Revenue by Store', fontsize = 20)
plt.xlabel('Store Key', fontsize = 16)
plt.ylabel('Total Revenue', fontsize = 16)
plt.xticks(rotation=90)
plt.show()
```

Visualization:



Best and Worst Performing Stores

The top 5 best-performing stores and the 5 worst-performing stores are identified based on total revenue and quantity sold. **Store S0039 has the highest revenue and quantity sold (6409.0 and 206 respectively) while store S0015 has the lowest revenue and quantity sold (1050.0 and 30 respectively).**

Code:

```
best_performing_store = filtered_dataset.groupby('store_key').agg({'total_price': 'sum', 'quantity_sold': 'sum'}).sort_values(by='total_price',
ascending=False).head(5)
worst_performing_store = filtered_dataset.groupby('store_key').agg({'total_price': 'sum', 'quantity_sold': 'sum'}).sort_values
(by='total_price', ascending=True).head(5)

print("Best Performing Store:")
print(best_performing_store)
print("**" * 20)
print("Worst Performing Store:")
print(worst_performing_store)
```

Visualization:

```
Best Performing Store:
           total_price  quantity_sold
store_key
S0039           6409.0            206
S0024           5065.0            164
S0012           4551.0            147
S0035           4327.0            137
S0017           4279.0            140
**************************************
Worst Performing Store:
           total_price  quantity_sold
store_key
S0015           1050.0             30
S0029           1348.0             47
S0014           1376.0             46
S0044           1431.0             45
S0016           1560.0             51
```

Prescriptive Analysis

Prescriptive analytics is a form of data analysis that attempts to answer the question, "**What do we need to do to achieve this?**". It uses technology to help businesses make better decisions by analyzing raw data and factoring in information about possible scenarios, available resources, past performance, and current performance.

Prescriptive analytics suggests a course of action or strategy. It relies on **artificial intelligence (AI)** techniques, such as machine learning, to understand and advance from the data, adapting as it acquires new data. It also works with predictive analytics to determine future performance and recommends what course to take.

Machine Learning Model

A linear regression model is trained to predict the total price based on store size, division, item type, and quantity sold. Label encoding is used to transform categorical variables into numerical values.

The model's performance is evaluated using Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared values which indicate the model performs relatively well with an R^2 score of 90.44%.

Here, I used the model to predict the targeted revenue based on the store size, division, and item type. For that, I picked the **target revenue = 7000**, and now I have to find out the **predicted store size, division, and item type** that will help to get this revenue.

Code:

```
# Step-1: Prepare the data
X = filtered_dataset[['store_size', 'division_x', 'item_type', 'quantity_sold']]
y = filtered_dataset['total_price']


lb_encoders = {}
label_cols = ['store_size', 'division_x', 'item_type']

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

# 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)

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

Codeshot-1

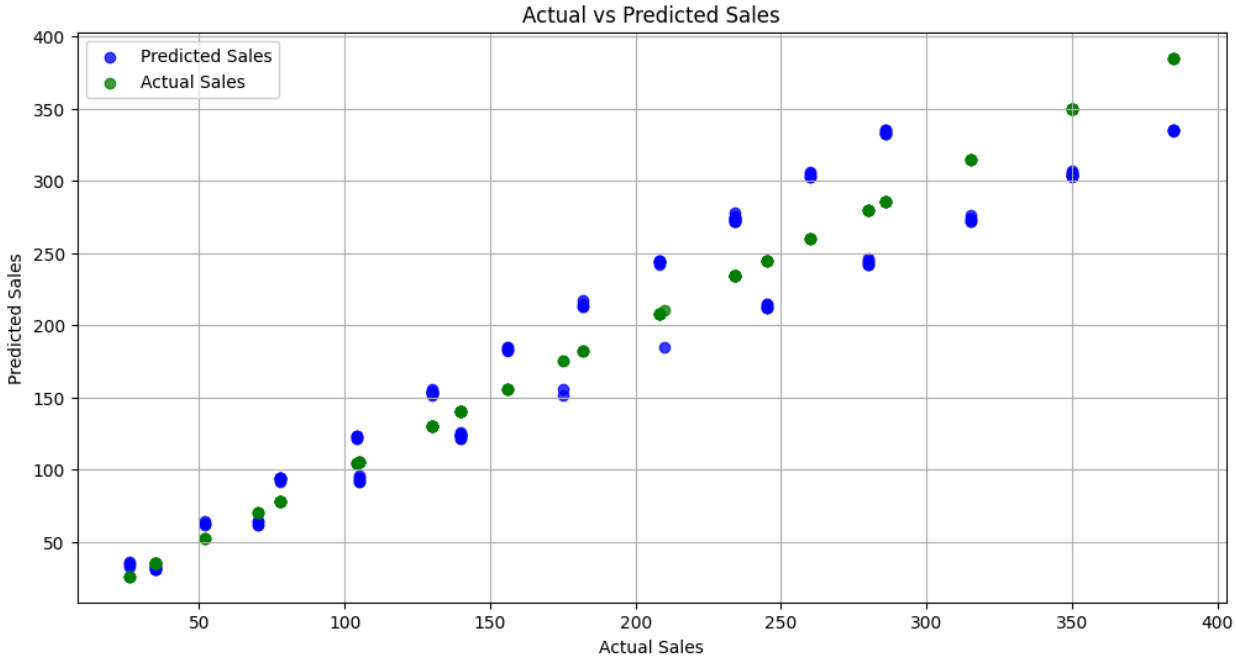
```python
# Step-5: Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

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()
```

Codeshot-2

Visualization:



Prediction Model:

```
target_revenue = 70000

store_sizes = lb_encoders['store_size'].classes_
divisions = lb_encoders['division_x'].classes_
item_types = lb_encoders['item_type'].classes_

best_combination = None
best_revenue_diff = float('inf')

for store_size in store_sizes:
    for division in divisions:
        for item_type in item_types:
            desired_features = pd.DataFrame({
                'store_size': [lb_encoders['store_size'].transform([store_size])[0]],
                'division_x': [lb_encoders['division_x'].transform([division])[0]],
                'item_type': [lb_encoders['item_type'].transform([item_type])[0]],
                'quantity_sold': [1]
            })
```

Codeshot-2

```
            predicted_revenue = model.predict(desired_features)
            desired_features['quantity_sold'] = target_revenue / predicted_revenue
            final_predicted_revenue = model.predict(desired_features)
            revenue_diff = abs(target_revenue - final_predicted_revenue[0])
            if revenue_diff < best_revenue_diff:
                best_revenue_diff = revenue_diff
                best_combination = {
                    'store_size': store_size,
                    'division': division,
                    'item_type': item_type,
                    'quantity_sold': desired_features['quantity_sold'].values[0],
                    'predicted_revenue': final_predicted_revenue[0]
                }

print(f'Best Combination to Achieve Target Revenue of {target_revenue}:')
print(f'Store Size: {best_combination["store_size"]}')
print(f'Division: {best_combination["division"]}')
print(f'Item Type: {best_combination["item_type"]}')
print(f'Quantity Sold: {best_combination["quantity_sold"]}')
print(f'Predicted Revenue: {best_combination["predicted_revenue"]}')
```

Codeshot-3

```
Best Combination to Achieve Target Revenue of 70000:
Store Size: small
Division: Sylhet
Item Type: Food - Chocolate
Quantity Sold: 2355.814878453405
Predicted Revenue: 70949.87596469214
```

Codeshot-4

Demand Forecasting for Inventory Management

Time series analysis is used to forecast demand for inventory management. Exponential Smoothing is applied to forecast both quantity sold and total revenue. The 'date' column is converted to a datetime format.

The demand forecast for the next 12 periods shows the expected quantity to be sold. A similar approach is used to generate a revenue forecast for the next 12 months.

Code:

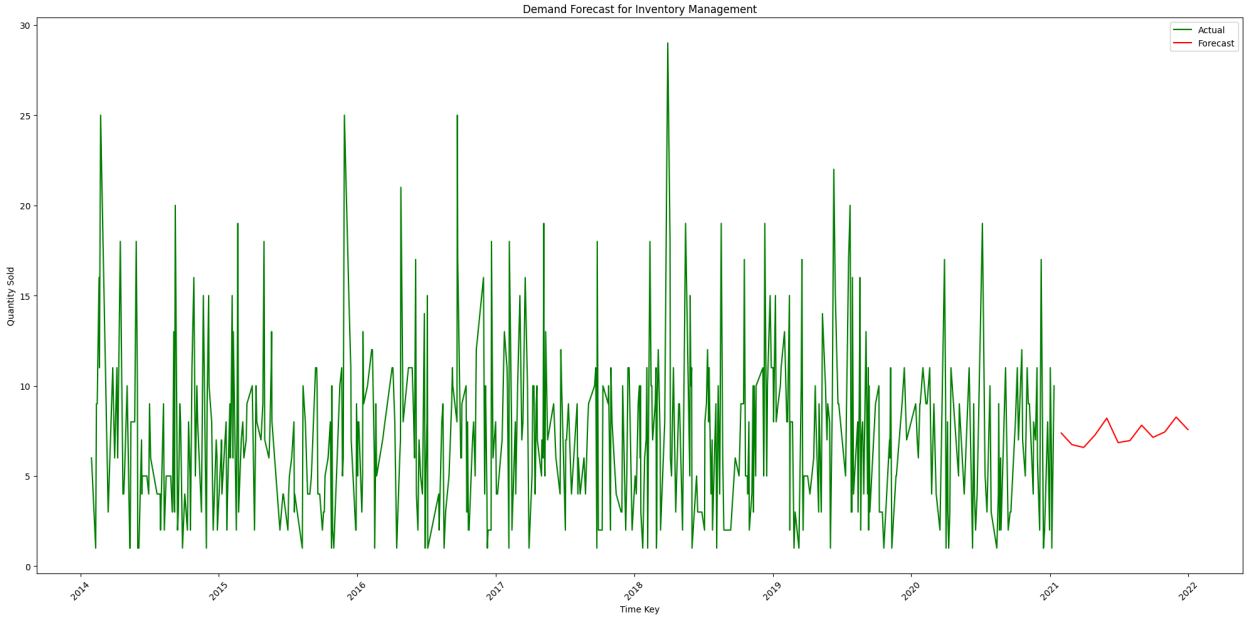
```
time_series_data = filtered_dataset.groupby('date')['quantity_sold'].sum().reset_index()
time_series_data['date'] = pd.to_datetime(time_series_data['date'])

model = ExponentialSmoothing(time_series_data['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_data['date'].iloc[-1], periods=12, freq='M')

plt.figure(figsize=(20, 10))
plt.plot(time_series_data['date'], time_series_data['quantity_sold'], label='Actual', color='green')
plt.plot(forecast_index, forecast, label='Forecast', color='red')
plt.title('Demand Forecast for Inventory Management')
plt.xlabel('Time Key')
plt.ylabel('Quantity Sold')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Visualization:


Demand Forecast for Inventory Management
Actual
Forecast
Quantity Sold
Time Key
0
5
10
15
20
25
30
2014
2015
2016
2017
2018
2019
2020
2021
2022