In [1]:
```
import pandas as pd
```

In [64]:
```
# Load the dataset
df = pd.read_csv('AusApparalSales4thQrt2020.csv')
```

In [65]:
```
# Display the few rows
print(df.head())
```

```
         Date        Time State     Group  Unit  Sales
0  1-Oct-2020     Morning    WA      Kids     8  20000
1  1-Oct-2020     Morning    WA       Men     8  20000
2  1-Oct-2020     Morning    WA     Women     4  10000
3  1-Oct-2020     Morning    WA   Seniors    15  37500
4  1-Oct-2020   Afternoon    WA      Kids     3   7500
```

In [66]:
```
# Display the dataframe
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7560 entries, 0 to 7559
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Date    7560 non-null   object
 1   Time    7560 non-null   object
 2   State   7560 non-null   object
 3   Group   7560 non-null   object
 4   Unit    7560 non-null   int64 
 5   Sales   7560 non-null   int64 
dtypes: int64(2), object(4)
memory usage: 354.5+ KB
None
```

Data Cleaning

In [17]:
```
# Check for missing values
missing_values = df.isna().sum()
print(missing_values)
```

```
Date     0
Time     0
State    0
Group    0
Unit     0
Sales    0
dtype: int64
```

In [18]:
```
# Check for non-missing values
non_missing_values = df.notna().sum()
print(non_missing_values)
```

```
Date     7560
Time     7560
State    7560
Group    7560
Unit     7560
Sales    7560
dtype: int64
```

In [70]:
```
# if any value is null
df = df.dropna()
```

In [71]:
```
# no value is null but to replace if we found any null value
df['Sales'].fillna(df['Sales'].mean(), inplace=True)
```

In [69]:
```
print(df.head())
```

```
         Date        Time State     Group  Unit  Sales
0  1-Oct-2020     Morning    WA      Kids     8  20000
1  1-Oct-2020     Morning    WA       Men     8  20000
2  1-Oct-2020     Morning    WA     Women     4  10000
3  1-Oct-2020     Morning    WA   Seniors    15  37500
4  1-Oct-2020   Afternoon    WA      Kids     3   7500
```

Normalization of the data

In [27]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

In [76]:
```
# Create a StandardScaler object and transform the 'Sales' column
scaler = StandardScaler()
df['Sales_standardized'] = scaler.fit_transform(df[['Sales']])
```

In [77]:
```
# Normalize the 'Unit' column
min_max_scaler = MinMaxScaler()
df['Unit_normalized'] = min_max_scaler.fit_transform(df[['Unit']])
```

In [31]:
```
# Display normalized columns
print(df[['Sales_standardized', 'Unit_normalized']].head())
```

```
   Sales_standardized  Unit_normalized
0           -0.775581         0.095238
1           -0.775581         0.095238
2           -1.085645         0.031746
3           -0.232969         0.206349
4           -1.163162         0.015873
```

In [78]:
```
# Group the data by the 'Group' column:
grouped_data = df.groupby('Group')
```

In [79]:
```
# Calculate mean and sum of the sales for each group
mean_sales_by_group = grouped_data['Sales'].mean()
sum_units_by_group = grouped_data['Unit'].sum()
```

In [80]:
```
# Merge and show the results
df = df.merge(mean_sales_by_group, how='left', on='Group', suffixes=('', '_mean'))
df = df.merge(sum_units_by_group, how='left', on='Group', suffixes=('', '_sum'))
```

In [37]:
```
print(df.head())
```

```
         Date        Time State     Group  Unit  Sales  Sales_standardized  \
0  1-Oct-2020     Morning    WA      Kids     8  20000           -0.775581
1  1-Oct-2020     Morning    WA       Men     8  20000           -0.775581
2  1-Oct-2020     Morning    WA     Women     4  10000           -1.085645
3  1-Oct-2020     Morning    WA   Seniors    15  37500           -0.232969
4  1-Oct-2020   Afternoon    WA      Kids     3   7500           -1.163162

   Unit_normalized    Sales_mean  Unit_sum
0         0.095238  45011.904762     34029
1         0.095238  45370.370370     34300
2         0.031746  45207.671958     34177
3         0.206349  44464.285714     33615
4         0.015873  45011.904762     34029
```

Descriptive Statistical Analysis

In [81]:
```
statistics_sales = df['Sales'].describe()
statistics_unit = df['Unit'].describe()

print("Descriptive Statistics for 'Sales' column:")
print(statistics_sales)

print("\nDescriptive Statistics for 'Unit' column:")
print(statistics_unit)
```

```
Descriptive Statistics for 'Sales' column:
count      7560.000000
mean      45013.558201
std       32253.506944
min        5000.000000
25%       20000.000000
50%       35000.000000
75%       65000.000000
max      162500.000000
Name: Sales, dtype: float64

Descriptive Statistics for 'Unit' column:
count    7560.000000
mean       18.005423
std        12.901403
min         2.000000
25%         8.000000
50%        14.000000
75%        26.000000
max        65.000000
Name: Unit, dtype: float64
```

Generating the Highest and Lowest sales Group wise

In [82]:
```
total_sales_by_group = df.groupby('Group')['Sales'].sum()

# Group with the highest sales
highest_sales_group = total_sales_by_group.idxmax()
highest_sales_amount = total_sales_by_group.max()

# Group with the lowest sales
lowest_sales_group = total_sales_by_group.idxmin()
lowest_sales_amount = total_sales_by_group.min()
```

```
print(f"The group generating the highest sales is '{highest_sales_group}' with total sales of {highest_sales_amount:.2f}.")
print(f"The group generating the lowest sales is '{lowest_sales_group}' with total sales of {lowest_sales_amount:.2f}.")
```

```
The group generating the highest sales is ' Men' with total sales of 85750000.00.
The group generating the lowest sales is ' Seniors' with total sales of 84037500.00.
```

Generating the Highest and Lowest sales State wise

In [83]:
```
total_sales_by_state = df.groupby('State')['Sales'].sum()

# State with the highest sales
highest_sales_state = total_sales_by_state.idxmax()
highest_sales_amount_state = total_sales_by_state.max()

# State with the lowest sales
lowest_sales_state = total_sales_by_state.idxmin()
lowest_sales_amount_state = total_sales_by_state.min()

print(f"The state generating the highest sales is '{highest_sales_state}' with total sales of {highest_sales_amount_state:.2f}.")
print(f"The state generating the lowest sales is '{lowest_sales_state}' with total sales of {lowest_sales_amount_state:.2f}.")
```

```
The state generating the highest sales is ' VIC' with total sales of 105565000.00.
The state generating the lowest sales is ' WA' with total sales of 22152500.00.
```

Generate weekly, monthly and quarterly reports

In [41]:
```
import pandas as pd
```

In [85]:
```
# Convert the date function into Datetime format
df['Date'] = pd.to_datetime(df['Date'])
```

In [45]:
```
df.set_index('Date', inplace=True)
```

In [46]:
```
#weekly report
weekly_report = df.resample('W').agg({'Sales': 'sum', 'Unit': 'sum'})
weekly_report['avg_sales'] = df.resample('W').agg({'Sales': 'mean'})
print("Weekly Report:")
print(weekly_report)
```

```
Weekly Report:
              Sales   Unit     avg_sales
Date
2020-10-04  15045000   6018  44776.785714
2020-10-11  27002500  10801  45922.619048
2020-10-18  26640000  10656  45306.122449
2020-10-25  26815000  10726  45603.741497
2020-11-01  21807500   8723  43268.849206
2020-11-08  20865000   8346  35484.693878
2020-11-15  21172500   8469  36007.653061
2020-11-22  21112500   8445  35905.612245
2020-11-29  21477500   8591  36526.360544
2020-12-06  29622500  11849  50378.401361
2020-12-13  31525000  12610  53613.945578
2020-12-20  31655000  12662  53835.034014
2020-12-27  31770000  12708  54030.612245
2021-01-03  13792500   5517  54732.142857
```

In [48]:
```
# monthly report
monthly_report = df.resample('M').agg({'Sales': 'sum', 'Unit': 'sum'})
monthly_report['avg_sales'] = df.resample('M').agg({'Sales': 'mean'})
print("\nMonthly Report:")
print(monthly_report)
```

```
Monthly Report:
               Sales   Unit     avg_sales
Date
2020-10-31  114290000  45716  45353.174603
2020-11-30   90682500  36273  35985.119048
2020-12-31  135330000  54132  53702.380952
```

In [49]:
```
# quarterly report
quarterly_report = df.resample('Q').agg({'Sales': 'sum', 'Unit': 'sum'})
quarterly_report['avg_sales'] = df.resample('Q').agg({'Sales': 'mean'})
print("\nQuarterly Report:")
print(quarterly_report)
```

```
Quarterly Report:
               Sales    Unit     avg_sales
Date
2020-12-31  340302500  136121  45013.558201
```

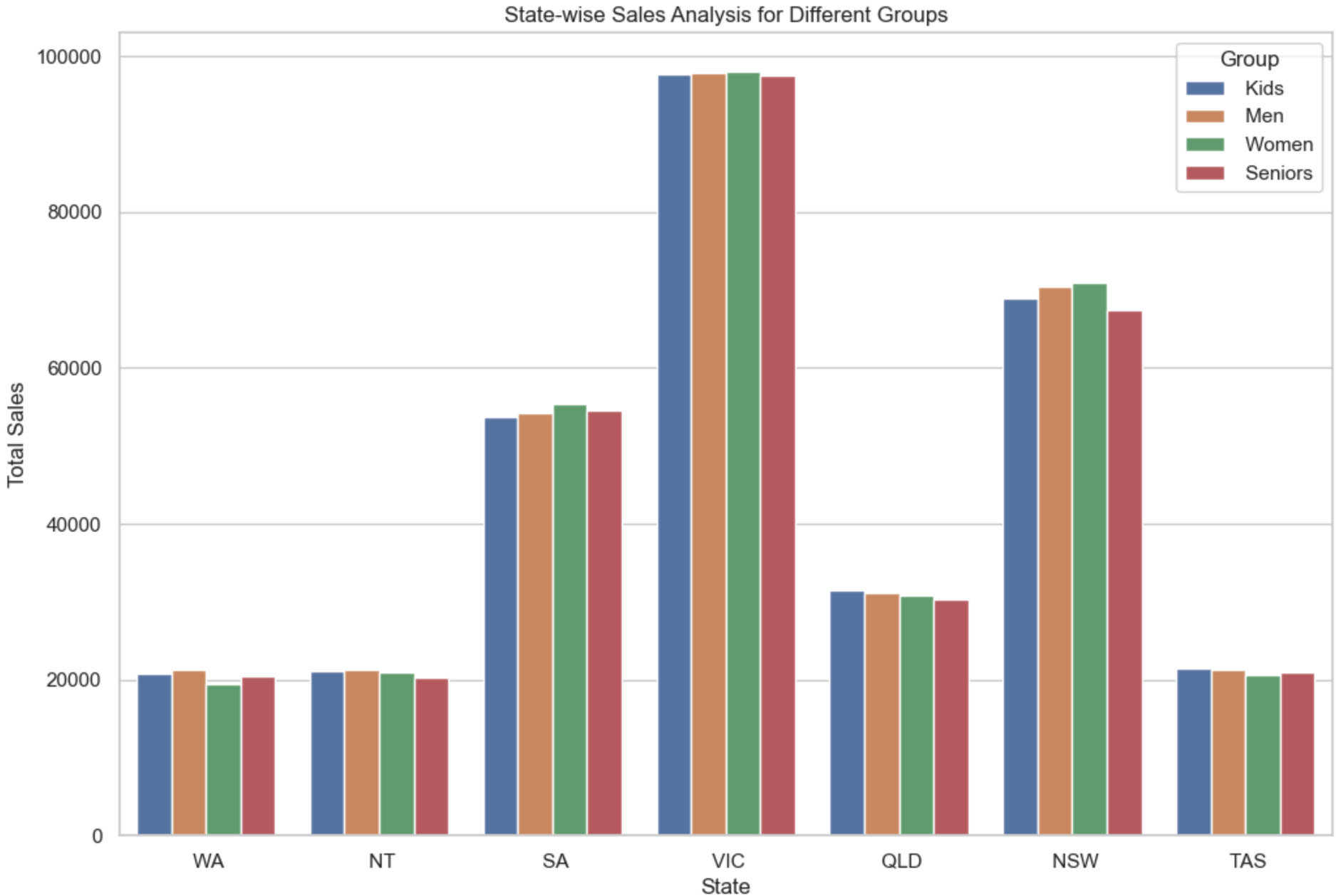
Data Visualization

In [50]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [51]:
```
sns.set(style="whitegrid")
```

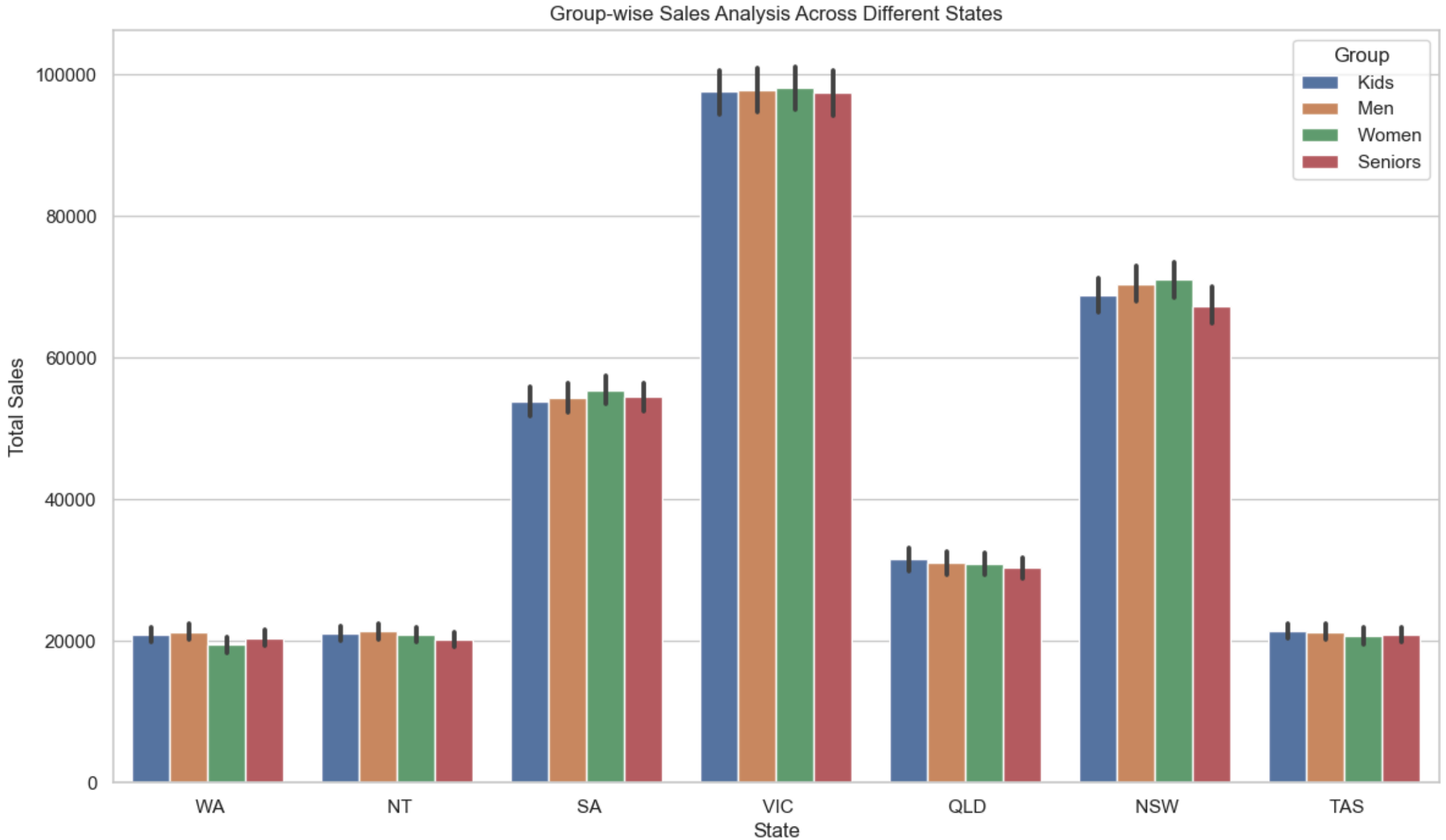
A bar plot for state-wise sales analysis for different groups

In [56]:
```
plt.figure(figsize=(12, 8))
sns.barplot(x='State', y='Sales', hue='Group', data=df,  errorbar=None)
plt.title('State-wise Sales Analysis for Different Groups')
plt.xlabel('State')
plt.ylabel('Total Sales')
plt.legend(title='Group')
plt.show()
```



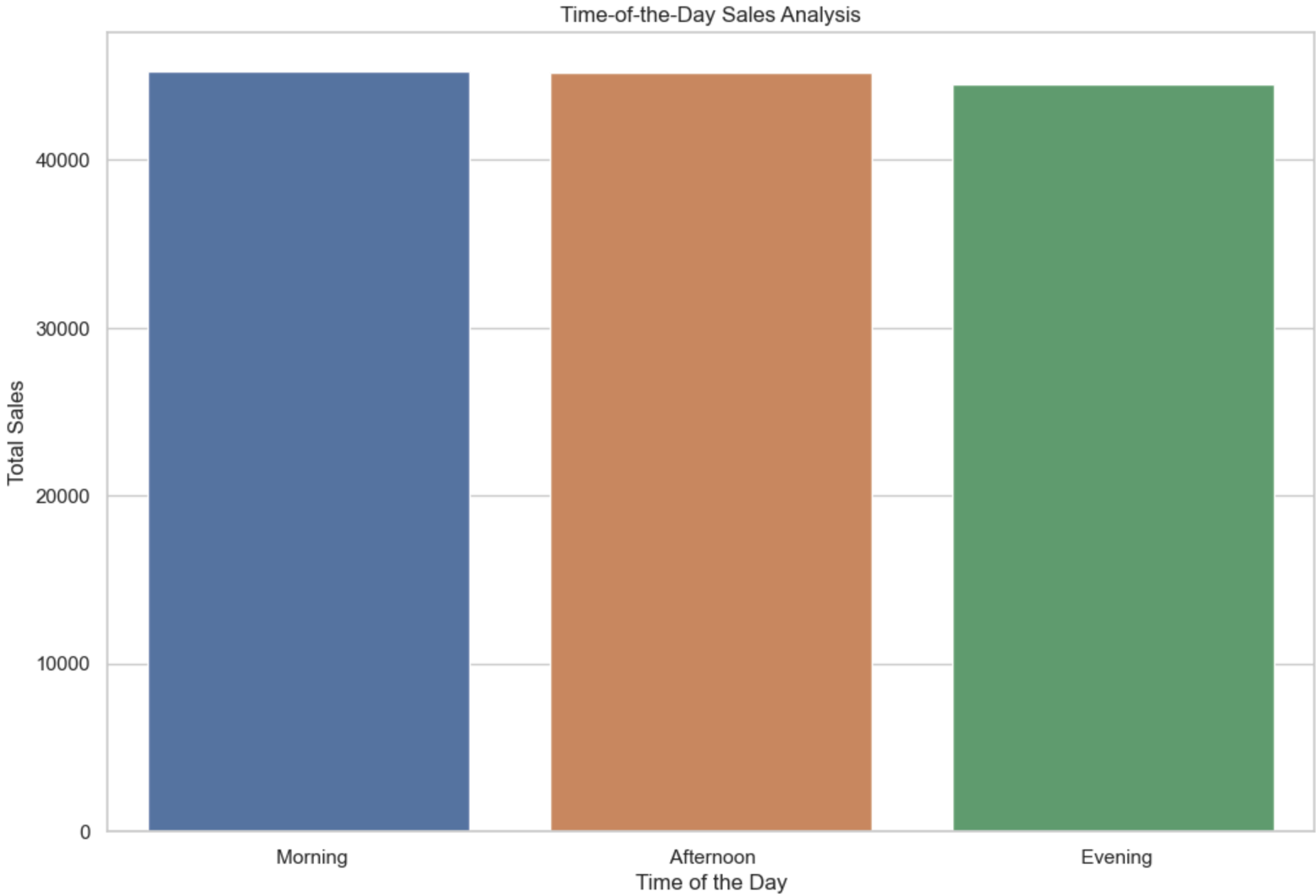
A bar plot for Group-wise sales analysis (kids, women, men, and seniors) across different states.

In [57]:
```
plt.figure(figsize=(14, 8))
sns.barplot(x='State', y='Sales', hue='Group', data=df)
plt.title('Group-wise Sales Analysis Across Different States')
plt.xlabel('State')
plt.ylabel('Total Sales')
plt.legend(title='Group')
plt.show()
```



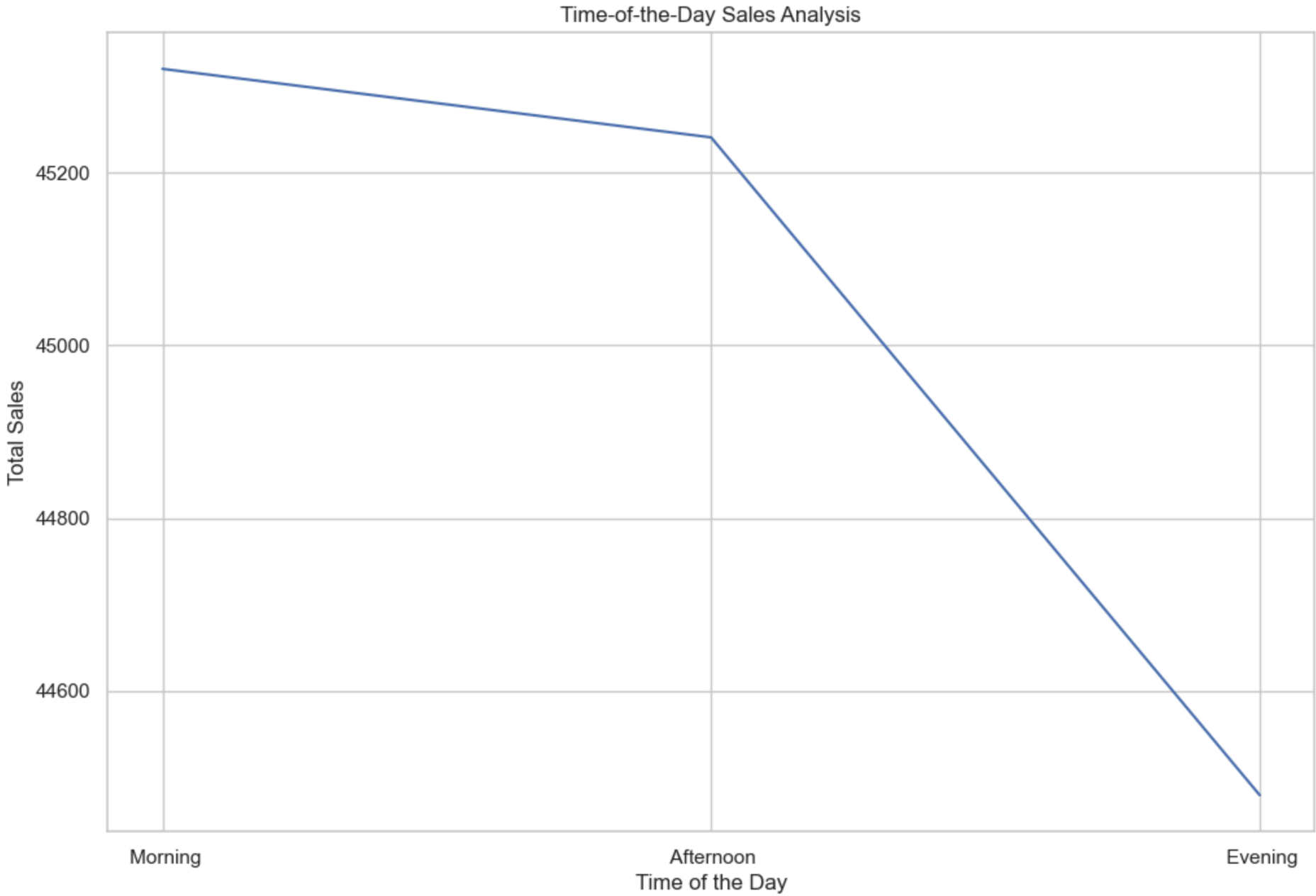
A bar plot for time-of-the-day analysis

In [59]:
```
plt.figure(figsize=(12, 8))
sns.barplot(x='Time', y='Sales', data=df, errorbar=None)
plt.title('Time-of-the-Day Sales Analysis')
plt.xlabel('Time of the Day')
plt.ylabel('Total Sales')
plt.show()
```



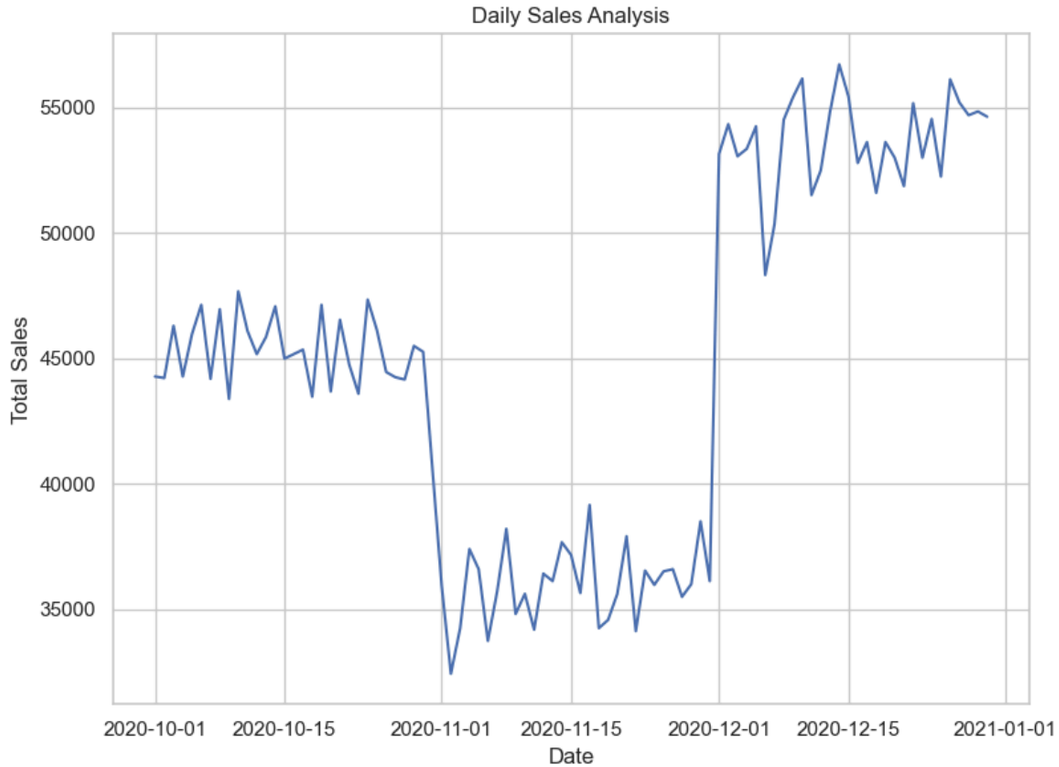
A line plot for time-of-the-day analysis

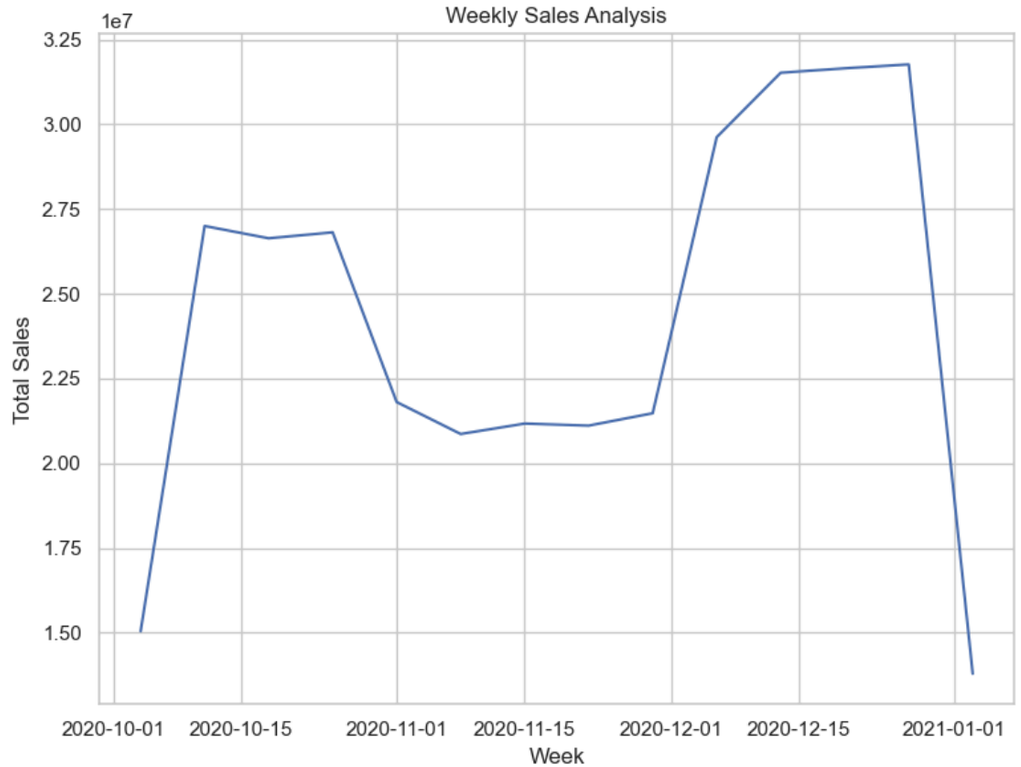
In [61]:
```
plt.figure(figsize=(12, 8))
sns.lineplot(x='Time', y='Sales', data=df, errorbar=None)
plt.title('Time-of-the-Day Sales Analysis')
plt.xlabel('Time of the Day')
plt.ylabel('Total Sales')
plt.show()
```



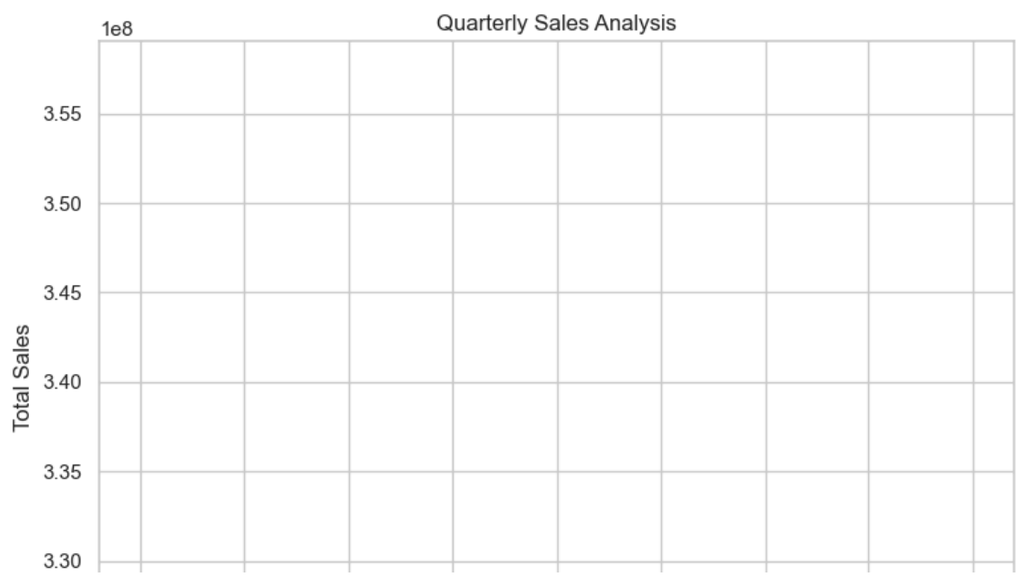
Creating subplots for daily, weekly, monthly, and quarterly charts

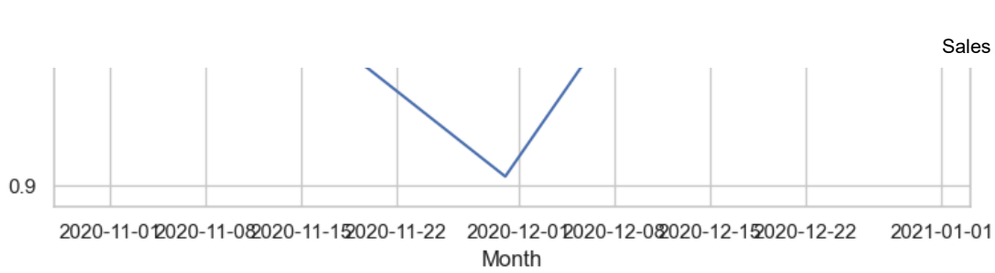
In [63]:
```
# Creating subplots
fig, axes = plt.subplots(2, 2, figsize=(16, 12))

# Daily chart
sns.lineplot(x='Date', y='Sales', data=df, errorbar=None, ax=axes[0, 0])
axes[0, 0].set_title('Daily Sales Analysis')
axes[0, 0].set_xlabel('Date')
axes[0, 0].set_ylabel('Total Sales')

# Weekly chart
weekly_data = df.resample('W').agg({'Sales': 'sum'})
sns.lineplot(x=weekly_data.index, y='Sales', data=weekly_data, errorbar=None, ax=axes[0, 1])
axes[0, 1].set_title('Weekly Sales Analysis')
axes[0, 1].set_xlabel('Week')
axes[0, 1].set_ylabel('Total Sales')

# Monthly chart
monthly_data = df.resample('M').agg({'Sales': 'sum'})
sns.lineplot(x=monthly_data.index, y='Sales', data=monthly_data, errorbar=None, ax=axes[1, 0])
axes[1, 0].set_title('Monthly Sales Analysis')
axes[1, 0].set_xlabel('Month')
axes[1, 0].set_ylabel('Total Sales')

# Quarterly chart
quarterly_data = df.resample('Q').agg({'Sales': 'sum'})
sns.lineplot(x=quarterly_data.index, y='Sales', data=quarterly_data, errorbar=None, ax=axes[1, 1])
axes[1, 1].set_title('Quarterly Sales Analysis')
axes[1, 1].set_xlabel('Quarter')
axes[1, 1].set_ylabel('Total Sales')

plt.tight_layout()
plt.show()
```

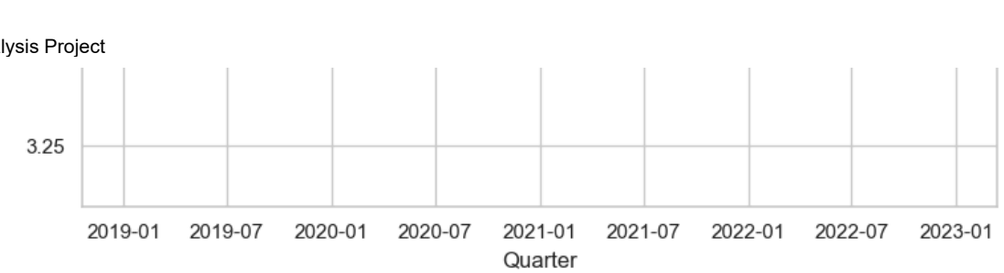

Daily Sales Analysis
Total Sales
55000
50000
45000
40000
35000
2020-10-01
2020-10-15
2020-11-01
2020-11-15
2020-12-01
2020-12-15
2021-01-01
Date


Weekly Sales Analysis
1e7
Total Sales
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
2020-10-01
2020-10-15
2020-11-01
2020-11-15
2020-12-01
2020-12-15
2021-01-01
Week


Monthly Sales Analysis
1e8
Total Sales
1.3
1.2
1.1
1.0


Quarterly Sales Analysis
1e8
Total Sales
3.55
3.50
3.45
3.40
3.35
3.30





I recommend using Seaborn and Matplotlib for statistical data visualization in your dashboard. Seaborn, built on Matplotlib, offers a high-level interface with default styles, making it ideal for creating attractive statistical graphics. It simplifies complex visualizations, handles categorical data, and integrates seamlessly with Pandas DataFrames. The library's simplicity and ease of use make it efficient for creating insightful visualizations with minimal code. Additionally, Matplotlib provides fine-grained customization when needed. While Seaborn is excellent for general statistical visualizations, for interactive dashboards or specialized charts, Plotly or Bokeh may be considered. The choice ultimately depends on your specific analysis requirements, with Seaborn and Matplotlib providing a strong foundation for statistical data exploration and visualization.