Task

Describe the columns available in your sales data (e.g., 'Date', 'Product', 'Region', 'Sales Amount', 'Quantity', 'Customer Segment', 'Service Type').

In [6]:
```
import pandas as pd

# NOTE: The previous attempt failed because 'sales_data.csv' was not found.
# Please upload your 'sales_data.csv' file to the environment or update the path be
# For demonstration purposes and to allow the notebook to run, a sample DataFrame i
# Replace this sample DataFrame creation with your actual data loading code once th

try:
    # Attempt to load the actual sales data if available
    df_sales = pd.read_csv('sales_data.csv')
    print("Successfully loaded 'sales_data.csv'.")
except FileNotFoundError:
    print("Warning: 'sales_data.csv' not found. Creating a sample DataFrame for dem
    # Create a sample DataFrame if 'sales_data.csv' is not found
    data = {
        'Date': pd.to_datetime(['2023-01-01', '2023-01-01', '2023-01-02', '2023-01-
        'Product': ['Product A', 'Product B', 'Product A', 'Product C', 'Product B'
        'Region': ['North', 'South', 'North', 'East', 'West'],
        'Sales Amount': [150.75, 200.50, 160.00, 300.25, 220.00],
        'Quantity': [2, 1, 3, 1, 2],
        'Customer Segment': ['SMB', 'Enterprise', 'SMB', 'Enterprise', 'SMB'],
        'Service Type': ['Voice', 'Data', 'SMS', 'Data', 'Voice']
    }
    df_sales = pd.DataFrame(data)

print('DataFrame Info:')
df_sales.info()

print('First 5 rows of the DataFrame:')
print(df_sales.head())
```

```
Warning: 'sales_data.csv' not found. Creating a sample DataFrame for demonstration.
DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Date              5 non-null      datetime64[ns]
 1   Product           5 non-null      object
 2   Region            5 non-null      object
 3   Sales Amount      5 non-null      float64
 4   Quantity          5 non-null      int64
 5   Customer Segment  5 non-null      object
 6   Service Type      5 non-null      object
dtypes: datetime64[ns](1), float64(1), int64(1), object(4)
memory usage: 412.0+ bytes
First 5 rows of the DataFrame:
        Date    Product Region  Sales Amount  Quantity Customer Segment  \
0 2023-01-01  Product A  North        150.75         2              SMB
1 2023-01-01  Product B  South        200.50         1       Enterprise
2 2023-01-02  Product A  North        160.00         3              SMB
3 2023-01-02  Product C   East        300.25         1       Enterprise
4 2023-01-03  Product B   West        220.00         2              SMB

  Service Type
0        Voice
1         Data
2          SMS
3         Data
4        Voice
```

In [3]:

```python
import pandas as pd

# NOTE: The previous attempt failed because 'sales_data.csv' was not found.
# Please upload your 'sales_data.csv' file to the environment or update the path be
# For demonstration purposes and to allow the notebook to run, a sample DataFrame i
# Replace this sample DataFrame creation with your actual data loading code once th

try:
    # Attempt to load the actual sales data if available
    df_sales = pd.read_csv('sales_data.csv')
    print("Successfully loaded 'sales_data.csv'.")
except FileNotFoundError:
    print("Warning: 'sales_data.csv' not found. Creating a sample DataFrame for dem
    # Create a sample DataFrame if 'sales_data.csv' is not found
    data = {
        'Date': pd.to_datetime(['2023-01-01', '2023-01-01', '2023-01-02', '2023-01-
        'Product': ['Product A', 'Product B', 'Product A', 'Product C', 'Product B'
        'Region': ['North', 'South', 'North', 'East', 'West'],
        'Sales Amount': [150.75, 200.50, 160.00, 300.25, 220.00],
        'Quantity': [2, 1, 3, 1, 2],
        'Customer Segment': ['SMB', 'Enterprise', 'SMB', 'Enterprise', 'SMB'],
        'Service Type': ['Voice', 'Data', 'SMS', 'Data', 'Voice']
    }
    df_sales = pd.DataFrame(data)
```

```
print('DataFrame Info:')
df_sales.info()

print('\nFirst 5 rows of the DataFrame:')
print(df_sales.head())
```

```
Warning: 'sales_data.csv' not found. Creating a sample DataFrame for demonstration.
DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Date              5 non-null      datetime64[ns]
 1   Product           5 non-null      object
 2   Region            5 non-null      object
 3   Sales Amount      5 non-null      float64
 4   Quantity          5 non-null      int64
 5   Customer Segment  5 non-null      object
 6   Service Type      5 non-null      object
dtypes: datetime64[ns](1), float64(1), int64(1), object(4)
memory usage: 412.0+ bytes

First 5 rows of the DataFrame:
        Date    Product Region  Sales Amount  Quantity Customer Segment  \
0 2023-01-01  Product A  North        150.75         2              SMB
1 2023-01-01  Product B  South        200.50         1       Enterprise
2 2023-01-02  Product A  North        160.00         3              SMB
3 2023-01-02  Product C   East        300.25         1       Enterprise
4 2023-01-03  Product B   West        220.00         2              SMB

  Service Type
0        Voice
1         Data
2          SMS
3         Data
4        Voice
```

Sales Data Columns Description

Based on the loaded sales data, the following columns are available:

- **Date** (Data Type: `datetime64[ns]`): This column represents the specific date when a sales transaction occurred. It is crucial for time-series analysis and tracking sales trends over time.
- **Product** (Data Type: `object`): This column indicates the name or identifier of the product that was sold. Examples include 'Product A', 'Product B', 'Product C', which would likely correspond to different telco products or services.
- **Region** (Data Type: `object`): This column specifies the geographical region where the sales transaction took place (e.g., 'North', 'South', 'East', 'West'). This helps in understanding regional sales performance.

- **Sales Amount** (Data Type: `float64`): This column records the total monetary value of each sales transaction. It's a key metric for revenue analysis.
- **Quantity** (Data Type: `int64`): This column denotes the number of units of a specific product sold in a single transaction. It's useful for understanding product popularity and inventory management.
- **Customer Segment** (Data Type: `object`): This column categorizes the customer into different segments (e.g., 'SMB' for Small and Medium Businesses, 'Enterprise'). This helps in tailoring marketing strategies and understanding customer behavior across different groups.
- **Service Type** (Data Type: `object`): This column specifies the type of service associated with the sale (e.g., 'Voice', 'Data', 'SMS'). This is particularly relevant for a telco dataset, indicating the nature of the service provided.

In [7]:
```
print('--- Column Descriptions ---')
for col in df_sales.columns:
    col_type = df_sales[col].dtype
    print(f"\nColumn Name: '{col}'")
    print(f"Data Type: {col_type}")

    # For object type columns (often categorical), list unique values for better de
    if col_type == 'object':
        unique_values = df_sales[col].unique()
        print(f"Unique Values (first 5): {unique_values[:5]}")
        if len(unique_values) > 5:
            print(f"Total Unique Values: {len(unique_values)}")
    elif pd.api.types.is_datetime64_any_dtype(df_sales[col]):
        print(f"Date Range: {df_sales[col].min()} to {df_sales[col].max()}")
    elif pd.api.types.is_numeric_dtype(df_sales[col]):
        print(f"Min Value: {df_sales[col].min()}, Max Value: {df_sales[col].max()}"
```

```
--- Column Descriptions ---

Column Name: 'Date'
Data Type: datetime64[ns]
Date Range: 2023-01-01 00:00:00 to 2023-01-03 00:00:00

Column Name: 'Product'
Data Type: object
Unique Values (first 5): ['Product A' 'Product B' 'Product C']

Column Name: 'Region'
Data Type: object
Unique Values (first 5): ['North' 'South' 'East' 'West']

Column Name: 'Sales Amount'
Data Type: float64
Min Value: 150.75, Max Value: 300.25

Column Name: 'Quantity'
Data Type: int64
Min Value: 1, Max Value: 3

Column Name: 'Customer Segment'
Data Type: object
Unique Values (first 5): ['SMB' 'Enterprise']

Column Name: 'Service Type'
Data Type: object
Unique Values (first 5): ['Voice' 'Data' 'SMS']
```

Load the Actual Sales Data

To ensure the analysis proceeds with your actual sales data, please make sure your `sales_data.csv` file is uploaded to the Colab environment. If the file is not yet uploaded or if it has a different name/path, please upload it now or modify the file path in the code below accordingly.

The following code block will attempt to load `sales_data.csv`. If successful, you will see 'Successfully loaded 'sales_data.csv'.'; otherwise, it will continue to use the sample DataFrame, indicating that the file was still not found.

In [8]:
```
import pandas as pd

try:
    # Attempt to load the actual sales data
    df_sales = pd.read_csv('sales_data.csv')
    print("Successfully loaded 'sales_data.csv'.")
except FileNotFoundError:
    print("Warning: 'sales_data.csv' not found. Continuing with the sample DataFram
    # Re-create the sample DataFrame if 'sales_data.csv' is not found
    data = {
        'Date': pd.to_datetime(['2023-01-01', '2023-01-01', '2023-01-02', '2023-01-
        'Product': ['Product A', 'Product B', 'Product A', 'Product C', 'Product B'
```

```
        'Region': ['North', 'South', 'North', 'East', 'West'],
        'Sales Amount': [150.75, 200.50, 160.00, 300.25, 220.00],
        'Quantity': [2, 1, 3, 1, 2],
        'Customer Segment': ['SMB', 'Enterprise', 'SMB', 'Enterprise', 'SMB'],
        'Service Type': ['Voice', 'Data', 'SMS', 'Data', 'Voice']
    }
    df_sales = pd.DataFrame(data)

print('\nDataFrame Info:')
df_sales.info()

print('\nFirst 5 rows of the DataFrame:')
print(df_sales.head())
```

```
Warning: 'sales_data.csv' not found. Continuing with the sample DataFrame for demons
tration.

DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Date              5 non-null      datetime64[ns]
 1   Product           5 non-null      object
 2   Region            5 non-null      object
 3   Sales Amount      5 non-null      float64
 4   Quantity          5 non-null      int64
 5   Customer Segment  5 non-null      object
 6   Service Type      5 non-null      object
dtypes: datetime64[ns](1), float64(1), int64(1), object(4)
memory usage: 412.0+ bytes

First 5 rows of the DataFrame:
        Date    Product Region  Sales Amount  Quantity Customer Segment  \
0 2023-01-01  Product A  North        150.75         2              SMB
1 2023-01-01  Product B  South        200.50         1       Enterprise
2 2023-01-02  Product A  North        160.00         3              SMB
3 2023-01-02  Product C   East        300.25         1       Enterprise
4 2023-01-03  Product B   West        220.00         2              SMB

  Service Type
0        Voice
1         Data
2          SMS
3         Data
4        Voice
```

Data Cleaning and Preprocessing

Subtask:

Perform necessary data cleaning and preprocessing steps, such as handling missing values, converting data types, and extracting relevant features like 'Month' or 'Year' from date

columns, specific to telco product sales analysis in Nigeria.

Handling Missing Values

First, I'll check for any missing values across all columns in the `df_sales` DataFrame. If missing values are present, I will drop the corresponding rows to ensure data integrity for subsequent analysis, as specified in the instructions. This is a common strategy when missing data is not extensive and dropping rows does not significantly reduce the dataset size.

In [9]:
```
print('Missing values before dropping:')
print(df_sales.isnull().sum())

initial_rows = df_sales.shape[0]
df_sales.dropna(inplace=True)

print(f"\nNumber of rows before dropping NaNs: {initial_rows}")
print(f"Number of rows after dropping NaNs: {df_sales.shape[0]}")

print('\nMissing values after dropping:')
print(df_sales.isnull().sum())
```

```
Missing values before dropping:
Date                0
Product             0
Region              0
Sales Amount        0
Quantity            0
Customer Segment    0
Service Type        0
dtype: int64

Number of rows before dropping NaNs: 5
Number of rows after dropping NaNs: 5

Missing values after dropping:
Date                0
Product             0
Region              0
Sales Amount        0
Quantity            0
Customer Segment    0
Service Type        0
dtype: int64
```

Converting Data Types and Feature Engineering

Next, I'll ensure the 'Date' column is correctly formatted as a datetime object. This is crucial for time-series analysis. After confirming the correct data type, I will extract 'Month' and 'Year' into new, separate columns. These new features will be valuable for analyzing seasonal trends and annual performance in sales data.

In [10]:
```
print('DataFrame info before type conversion and feature extraction:')
df_sales.info()

# Ensure 'Date' column is datetime type
if not pd.api.types.is_datetime64_any_dtype(df_sales['Date']):
    df_sales['Date'] = pd.to_datetime(df_sales['Date'])
    print("\n'Date' column converted to datetime type.")
else:
    print("\n'Date' column is already in datetime type.")

# Extract 'Month' and 'Year' from 'Date'
df_sales['Month'] = df_sales['Date'].dt.month
df_sales['Year'] = df_sales['Date'].dt.year

print('\nDataFrame Info after feature extraction:')
df_sales.info()

print('\nFirst 5 rows of the DataFrame with new features:')
print(df_sales.head())
```

```
DataFrame info before type conversion and feature extraction:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Date              5 non-null      datetime64[ns]
 1   Product           5 non-null      object
 2   Region            5 non-null      object
 3   Sales Amount      5 non-null      float64
 4   Quantity          5 non-null      int64
 5   Customer Segment  5 non-null      object
 6   Service Type      5 non-null      object
dtypes: datetime64[ns](1), float64(1), int64(1), object(4)
memory usage: 412.0+ bytes

'Date' column is already in datetime type.

DataFrame Info after feature extraction:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Date              5 non-null      datetime64[ns]
 1   Product           5 non-null      object
 2   Region            5 non-null      object
 3   Sales Amount      5 non-null      float64
 4   Quantity          5 non-null      int64
 5   Customer Segment  5 non-null      object
 6   Service Type      5 non-null      object
 7   Month             5 non-null      int32
 8   Year              5 non-null      int32
dtypes: datetime64[ns](1), float64(1), int32(2), int64(1), object(4)
memory usage: 452.0+ bytes

First 5 rows of the DataFrame with new features:
        Date    Product Region  Sales Amount  Quantity Customer Segment  \
0 2023-01-01  Product A  North        150.75         2              SMB
1 2023-01-01  Product B  South        200.50         1       Enterprise
2 2023-01-02  Product A  North        160.00         3              SMB
3 2023-01-02  Product C   East        300.25         1       Enterprise
4 2023-01-03  Product B   West        220.00         2              SMB

  Service Type  Month  Year
0        Voice      1  2023
1         Data      1  2023
2          SMS      1  2023
3         Data      1  2023
4        Voice      1  2023
```

Calculating Key Sales Metrics

To understand the sales performance, I will calculate several key performance indicators (KPIs) as instructed:

1. **Total Sales**: This provides an overall measure of revenue generated.
2. **Sales by Product**: This helps identify which products are performing well and which might need attention.
3. **Sales by Region**: This reveals geographical sales strengths and weaknesses.
4. **Sales by Service Type**: Relevant for telco, this shows which types of services (e.g., Voice, Data, SMS) contribute most to sales.
5. **Sales Over Time (Monthly)**: This allows for the identification of trends, seasonality, and overall sales growth or decline over periods.

In [12]:
```
print('--- Calculating Key Sales Metrics ---')

# 1. Calculate Total Sales
total_sales = df_sales['Sales Amount'].sum()
print(f"\nTotal Sales: {total_sales:.2f}")

# 2. Calculate Sales by Product
sales_by_product = df_sales.groupby('Product')['Sales Amount'].sum().reset_index()
print("\nSales by Product:")
print(sales_by_product)

# 3. Calculate Sales by Region
sales_by_region = df_sales.groupby('Region')['Sales Amount'].sum().reset_index()
print("\nSales by Region:")
print(sales_by_region)

# 4. Calculate Sales by Service Type
sales_by_service_type = df_sales.groupby('Service Type')['Sales Amount'].sum().rese
print("\nSales by Service Type:")
print(sales_by_service_type)

# 5. Calculate Sales Over Time (Monthly)
sales_over_time_monthly = df_sales.groupby(['Year', 'Month'])['Sales Amount'].sum()
print("\nSales Over Time (Monthly):")
print(sales_over_time_monthly)
```

```
--- Calculating Key Sales Metrics ---

Total Sales: 1031.50

Sales by Product:
     Product  Sales Amount
0  Product A        310.75
1  Product B        420.50
2  Product C        300.25

Sales by Region:
  Region  Sales Amount
0   East        300.25
1  North        310.75
2  South        200.50
3   West        220.00

Sales by Service Type:
  Service Type  Sales Amount
0         Data        500.75
1          SMS        160.00
2        Voice        370.75

Sales Over Time (Monthly):
   Year  Month  Sales Amount
0  2023      1        1031.5
```

Create Dashboard Visualizations

Subtask:

Generate interactive visualizations using a Python dashboarding library (e.g., Plotly Dash) to display the key sales metrics.

In [13]:
```
import plotly.express as px

# 1. Sales by Product Bar Chart
fig_product = px.bar(sales_by_product, x='Product', y='Sales Amount', title='Sales
fig_product.show()

# 2. Sales by Region Bar Chart
fig_region = px.bar(sales_by_region, x='Region', y='Sales Amount', title='Sales by
fig_region.show()

# 3. Sales by Service Type Bar Chart
fig_service_type = px.bar(sales_by_service_type, x='Service Type', y='Sales Amount'
fig_service_type.show()

# 4. Sales Over Time (Monthly) Line Chart
# Create a combined date column for plotting
sales_over_time_monthly['Date_YM'] = pd.to_datetime(sales_over_time_monthly['Year']

fig_time = px.line(sales_over_time_monthly, x='Date_YM', y='Sales Amount', title='M
fig_time.show()
```

Build Dashboard Layout and Functionality

Subtask:

Construct the overall layout of the dashboard, incorporating the generated visualizations and adding interactive components like filters (e.g., by date range, product, region, or service type) to allow for dynamic analysis of the Nigerian telco sales data.

In [14]:
```
import dash
from dash import dcc
from dash import html
from dash.dependencies import Input, Output
import plotly.express as px

# Initialize the Dash app
app = dash.Dash(__name__)

# Get unique values for dropdowns from the existing df_sales DataFrame
product_options = [{'label': i, 'value': i} for i in df_sales['Product'].unique()]
region_options = [{'label': i, 'value': i} for i in df_sales['Region'].unique()]
service_type_options = [{'label': i, 'value': i} for i in df_sales['Service Type'].
```

```
# Get min and max dates for date picker from the existing df_sales DataFrame
min_date = df_sales['Date'].min()
max_date = df_sales['Date'].max()

# Define the layout of the dashboard
app.layout = html.Div(children=[
    html.H1(children='Nigerian Telco Sales Dashboard', style={'textAlign': 'center'
    html.P('An interactive dashboard to analyze sales performance by product, regio

    html.Div([
        # Filter Controls Column
        html.Div([
            html.H2('Filters', style={'textAlign': 'center'}),
            html.Label('Product:'),
            dcc.Dropdown(
                id='product-dropdown',
                options=product_options,
                multi=True,
                placeholder="Select Products..."
            ),
            html.Br(),
            html.Label('Region:'),
            dcc.Dropdown(
                id='region-dropdown',
                options=region_options,
                multi=True,
                placeholder="Select Regions..."
            ),
            html.Br(),
            html.Label('Service Type:'),
            dcc.Dropdown(
                id='service-type-dropdown',
                options=service_type_options,
                multi=True,
                placeholder="Select Service Types..."
            ),
            html.Br(),
            html.Label('Date Range:'),
            dcc.DatePickerRange(
                id='date-range-picker',
                min_date_allowed=min_date,
                max_date_allowed=max_date,
                start_date=min_date,
                end_date=max_date,
                display_format='YYYY-MM-DD'
            ),
        ], style={'width': '25%', 'display': 'inline-block', 'verticalAlign': 'top'

        # Visualizations Column
        html.Div([
            html.H2('Sales Performance Overview', style={'textAlign': 'center'}),
            dcc.Graph(
                id='sales-by-product-graph',
                figure=fig_product
            ),
```

```
            dcc.Graph(
                id='sales-by-region-graph',
                figure=fig_region
            ),
            dcc.Graph(
                id='sales-by-service-type-graph',
                figure=fig_service_type
            ),
            dcc.Graph(
                id='monthly-sales-graph',
                figure=fig_time
            )
        ], style={'width': '73%', 'display': 'inline-block', 'verticalAlign': 'top'
    ], style={'display': 'flex', 'flexDirection': 'row', 'flexWrap': 'wrap'}), # Us
])

# --- Callback Functions for Interactivity ---
# This callback updates the 'Sales by Product' graph based on all filters.
# You would implement similar callbacks for other graphs to make the dashboard full
@app.callback(
    Output('sales-by-product-graph', 'figure'),
    [Input('product-dropdown', 'value'),
     Input('region-dropdown', 'value'),
     Input('service-type-dropdown', 'value'),
     Input('date-range-picker', 'start_date'),
     Input('date-range-picker', 'end_date')]
)
def update_product_sales_graph(selected_products, selected_regions, selected_servic
    filtered_df = df_sales.copy()

    # Apply product filter
    if selected_products and len(selected_products) > 0:
        filtered_df = filtered_df[filtered_df['Product'].isin(selected_products)]

    # Apply region filter
    if selected_regions and len(selected_regions) > 0:
        filtered_df = filtered_df[filtered_df['Region'].isin(selected_regions)]

    # Apply service type filter
    if selected_service_types and len(selected_service_types) > 0:
        filtered_df = filtered_df[filtered_df['Service Type'].isin(selected_service

    # Apply date range filter
    if start_date and end_date:
        filtered_df = filtered_df[(filtered_df['Date'] >= start_date) & (filtered_d

    sales_by_product_filtered = filtered_df.groupby('Product')['Sales Amount'].sum(
    if sales_by_product_filtered.empty:
        # Return an empty graph or a message if no data matches filters
        return px.bar(title='No Data Available for Selected Filters')
    fig = px.bar(sales_by_product_filtered, x='Product', y='Sales Amount', title='S
    return fig

# NOTE: To run this Dash application, you would typically save this code as a Pytho
# (e.g., app.py) and run it from your terminal using `python app.py`.
# In a Colab environment, running `app.run_server()` directly can block execution
```

```
# and may require additional setup (e.g., ngrok for tunneling) to access the dashbo
# Uncomment the following lines if you have the necessary environment setup to run
# if __name__ == '__main__':
#     app.run_server(debug=True, port=8050) # You can specify a port

print("Dash application code generated successfully. To interact with the dashboard
print("This code provides the layout and one example callback. You would extend the
```

```
Dash application code generated successfully. To interact with the dashboard, uncomm
ent the `app.run_server` block and run this cell. Further setup might be needed for
Colab.
This code provides the layout and one example callback. You would extend the callbac
ks for all visualizations.
```

In [11]:
```
print('Installing Dash and its dependencies...')
!pip install dash==2.14.2 dash-core-components==2.0.0 dash-html-components==2.0.0
print('Dash and its dependencies installed.')
```

```
Installing Dash and its dependencies...
Collecting dash==2.14.2
  Downloading dash-2.14.2-py3-none-any.whl.metadata (11 kB)
Collecting dash-core-components==2.0.0
  Downloading dash_core_components-2.0.0-py3-none-any.whl.metadata (2.9 kB)
Collecting dash-html-components==2.0.0
  Downloading dash_html_components-2.0.0-py3-none-any.whl.metadata (3.8 kB)
Collecting Flask<3.1,>=1.0.4 (from dash==2.14.2)
  Downloading flask-3.0.3-py3-none-any.whl.metadata (3.2 kB)
Collecting Werkzeug<3.1 (from dash==2.14.2)
  Downloading werkzeug-3.0.6-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.12/dist-packa
ges (from dash==2.14.2) (5.24.1)
Collecting dash-table==5.0.0 (from dash==2.14.2)
  Downloading dash_table-5.0.0-py3-none-any.whl.metadata (2.4 kB)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.1
2/dist-packages (from dash==2.14.2) (4.15.0)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages
(from dash==2.14.2) (2.32.4)
Collecting retrying (from dash==2.14.2)
  Downloading retrying-1.4.2-py3-none-any.whl.metadata (5.5 kB)
Collecting ansi2html (from dash==2.14.2)
  Downloading ansi2html-1.9.2-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.12/dist-packag
es (from dash==2.14.2) (1.6.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages
(from dash==2.14.2) (75.2.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.12/dist-
packages (from dash==2.14.2) (8.7.1)
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.12/dist-packa
ges (from Flask<3.1,>=1.0.4->dash==2.14.2) (3.1.6)
Requirement already satisfied: itsdangerous>=2.1.2 in /usr/local/lib/python3.12/dis
t-packages (from Flask<3.1,>=1.0.4->dash==2.14.2) (2.2.0)
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.12/dist-packag
es (from Flask<3.1,>=1.0.4->dash==2.14.2) (8.3.1)
Requirement already satisfied: blinker>=1.6.2 in /usr/local/lib/python3.12/dist-pack
ages (from Flask<3.1,>=1.0.4->dash==2.14.2) (1.9.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.12/dist-pac
kages (from plotly>=5.0.0->dash==2.14.2) (9.1.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages
(from plotly>=5.0.0->dash==2.14.2) (26.0)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.12/dist-p
ackages (from Werkzeug<3.1->dash==2.14.2) (3.0.3)
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.12/dist-packages
(from importlib-metadata->dash==2.14.2) (3.23.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.1
2/dist-packages (from requests->dash==2.14.2) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packag
es (from requests->dash==2.14.2) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-
packages (from requests->dash==2.14.2) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-
packages (from requests->dash==2.14.2) (2026.1.4)
Downloading dash-2.14.2-py3-none-any.whl (10.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.2/10.2 MB 70.2 MB/s eta 0:00:00
Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
```

```
Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Downloading flask-3.0.3-py3-none-any.whl (101 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 101.7/101.7 kB 8.2 MB/s eta 0:00:00
Downloading werkzeug-3.0.6-py3-none-any.whl (227 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 228.0/228.0 kB 18.2 MB/s eta 0:00:00
Downloading ansi2html-1.9.2-py3-none-any.whl (17 kB)
Downloading retrying-1.4.2-py3-none-any.whl (10 kB)
Installing collected packages: dash-table, dash-html-components, dash-core-component
s, Werkzeug, retrying, ansi2html, Flask, dash
  Attempting uninstall: Werkzeug
    Found existing installation: Werkzeug 3.1.5
    Uninstalling Werkzeug-3.1.5:
      Successfully uninstalled Werkzeug-3.1.5
  Attempting uninstall: Flask
    Found existing installation: Flask 3.1.2
    Uninstalling Flask-3.1.2:
      Successfully uninstalled Flask-3.1.2
Successfully installed Flask-3.0.3 Werkzeug-3.0.6 ansi2html-1.9.2 dash-2.14.2 dash-c
ore-components-2.0.0 dash-html-components-2.0.0 dash-table-5.0.0 retrying-1.4.2
Dash and its dependencies installed.
```

In [16]:

```python
import dash
from dash import dcc
from dash import html
from dash.dependencies import Input, Output
import plotly.express as px

# Initialize the Dash app
app = dash.Dash(__name__)

# Get unique values for dropdowns from the existing df_sales DataFrame
product_options = [{'label': i, 'value': i} for i in df_sales['Product'].unique()]
region_options = [{'label': i, 'value': i} for i in df_sales['Region'].unique()]
service_type_options = [{'label': i, 'value': i} for i in df_sales['Service Type'].

# Get min and max dates for date picker from the existing df_sales DataFrame
min_date = df_sales['Date'].min()
max_date = df_sales['Date'].max()

# Define the layout of the dashboard
app.layout = html.Div(children=[
    html.H1(children='Nigerian Telco Sales Dashboard', style={'textAlign': 'center'
    html.P('An interactive dashboard to analyze sales performance by product, regio

    html.Div([
        # Filter Controls Column
        html.Div([
            html.H2('Filters', style={'textAlign': 'center'}),
            html.Label('Product:'),
            dcc.Dropdown(
                id='product-dropdown',
                options=product_options,
                multi=True,
                placeholder="Select Products..."
            ),
```

```
            html.Br(),
            html.Label('Region:'),
            dcc.Dropdown(
                id='region-dropdown',
                options=region_options,
                multi=True,
                placeholder="Select Regions..."
            ),
            html.Br(),
            html.Label('Service Type:'),
            dcc.Dropdown(
                id='service-type-dropdown',
                options=service_type_options,
                multi=True,
                placeholder="Select Service Types..."
            ),
            html.Br(),
            html.Label('Date Range:'),
            dcc.DatePickerRange(
                id='date-range-picker',
                min_date_allowed=min_date,
                max_date_allowed=max_date,
                start_date=min_date,
                end_date=max_date,
                display_format='YYYY-MM-DD'
            ),
        ], style={'width': '25%', 'display': 'inline-block', 'verticalAlign': 'top'

        # Visualizations Column
        html.Div([
            html.H2('Sales Performance Overview', style={'textAlign': 'center'}),
            dcc.Graph(
                id='sales-by-product-graph',
                figure=fig_product
            ),
            dcc.Graph(
                id='sales-by-region-graph',
                figure=fig_region
            ),
            dcc.Graph(
                id='sales-by-service-type-graph',
                figure=fig_service_type
            ),
            dcc.Graph(
                id='monthly-sales-graph',
                figure=fig_time
            )
        ], style={'width': '73%', 'display': 'inline-block', 'verticalAlign': 'top'
    ], style={'display': 'flex', 'flexDirection': 'row', 'flexWrap': 'wrap'}), # Us
])

# --- Callback Functions for Interactivity ---
# This callback updates the 'Sales by Product' graph based on all filters.
# You would implement similar callbacks for other graphs to make the dashboard full
@app.callback(
    Output('sales-by-product-graph', 'figure'),
```

```
    [Input('product-dropdown', 'value'),
     Input('region-dropdown', 'value'),
     Input('service-type-dropdown', 'value'),
     Input('date-range-picker', 'start_date'),
     Input('date-range-picker', 'end_date')]
)
def update_product_sales_graph(selected_products, selected_regions, selected_servic
    filtered_df = df_sales.copy()

    # Apply product filter
    if selected_products and len(selected_products) > 0:
        filtered_df = filtered_df[filtered_df['Product'].isin(selected_products)]

    # Apply region filter
    if selected_regions and len(selected_regions) > 0:
        filtered_df = filtered_df[filtered_df['Region'].isin(selected_regions)]

    # Apply service type filter
    if selected_service_types and len(selected_service_types) > 0:
        filtered_df = filtered_df[filtered_df['Service Type'].isin(selected_service

    # Apply date range filter
    if start_date and end_date:
        filtered_df = filtered_df[(filtered_df['Date'] >= start_date) & (filtered_d

    sales_by_product_filtered = filtered_df.groupby('Product')['Sales Amount'].sum(
    if sales_by_product_filtered.empty:
        # Return an empty graph or a message if no data matches filters
        return px.bar(title='No Data Available for Selected Filters')
    fig = px.bar(sales_by_product_filtered, x='Product', y='Sales Amount', title='S
    return fig

# NOTE: To run this Dash application, you would typically save this code as a Pytho
# (e.g., app.py) and run it from your terminal using `python app.py`.
# In a Colab environment, running `app.run_server()` directly can block execution
# and may require additional setup (e.g., ngrok for tunneling) to access the dashbo
# Uncomment the following lines if you have the necessary environment setup to run
# if __name__ == '__main__':
 #    app.run_server(debug=True, port=8050) # You can specify a port

print("Dash application code generated successfully. To interact with the dashboard
print("This code provides the layout and one example callback. You would extend the
```

```
Dash application code generated successfully. To interact with the dashboard, uncomm
ent the `app.run_server` block and run this cell. Further setup might be needed for
Colab.
This code provides the layout and one example callback. You would extend the callbac
ks for all visualizations.
```

Final Task

Subtask:

Provide the complete Python code for the interactive sales dashboard and instructions on

how to run it locally, summarizing the insights that can be gained from the dashboard.

Summary:

Data Analysis Key Findings

- **Data Availability**: The `sales_data.csv` file was not found, leading to the use of a sample DataFrame for all analysis steps. This sample data provided a structured representation of typical telco sales information.
- **Data Structure**: The sample DataFrame contained 7 columns: `Date` (datetime), `Product` (object), `Region` (object), `Sales Amount` (float), `Quantity` (int), `Customer Segment` (object), and `Service Type` (object).
- **Data Cleaning**: The sample data was found to have no missing values. The `Date` column was already correctly formatted as a datetime object. New `Month` and `Year` features were successfully extracted from the `Date` column, expanding the DataFrame to 9 columns.
- **Key Performance Indicators (KPIs) Calculated (based on sample data):**
  - Total Sales: $1031.50
  - Sales by Product: Product B led with $420.50, followed by Product A ($310.75) and Product C ($300.25).
  - Sales by Region: North region generated the highest sales at $310.75, with East ($300.25), West ($220.00), and South ($200.50) following.
  - Sales by Service Type: Data services accounted for the largest share with $500.75, followed by Voice ($370.75) and SMS ($160.00).
  - Monthly Sales Over Time: For the single month present (January 2023), total sales were $1031.50.
- **Visualizations**: Interactive bar charts were generated for Sales by Product, Sales by Region, and Sales by Service Type using `plotly.express`. A line chart for Monthly Sales Over Time was also created.
- **Dashboard Construction**: A Dash application layout was successfully built, incorporating the generated visualizations and adding interactive filter components for Product, Region, Service Type, and Date Range. An example callback was implemented to update the 'Sales by Product' graph based on filter selections.

Insights or Next Steps

- **Upload Real Data**: For accurate analysis, the `sales_data.csv` file must be uploaded to the environment. The current findings are based on sample data and may not reflect actual sales performance.
- **Complete Dashboard Interactivity**: Implement additional callback functions within the Dash application to ensure all visualizations dynamically update based on the selected filters, providing a fully interactive dashboard experience.