Practical 1

Aim: Create Charts and Reports in Power BI.

Explanation:

Create Charts and Reports in Power BI

1. What is a Chart or Report

In Power BI, a **Chart** is a visual representation of your data (for example, bar charts, line charts, or pie charts).
A **Report** is a collection of visuals and data summaries that help you understand and analyze information in one place.

You can use Power BI to make interactive and attractive reports using simple steps.

---

2. Sample Data

Let's take a small sample dataset.

Product	Category	Month	Sales	Profit
A	Electronics	Jan	5000	1200
B	Electronics	Jan	4000	1000
C	Furniture	Feb	3000	800
D	Furniture	Feb	3500	900
E	Clothing	Mar	4500	1100
F	Clothing	Mar	5000	1300

Create this in **Excel** and save as **SalesReport.xlsx**, then import it into Power BI.

Steps:

1. Open Power BI Desktop
2. Go to **Home → Get Data → Excel**
3. Select the file and click **Load**

Now you will see the data in the **Fields** pane on the right.

---

3. Create Visuals (Charts)

a. Bar Chart – Sales by Category

1. Go to the **Report View** (middle icon on the left side).
2. In **Visualizations**, select **Clustered Column Chart**.
3. Drag **Category** to the Axis field.
4. Drag **Sales** to the Values field.

You will now see a bar chart showing total sales for each category.

---

b. Pie Chart – Profit by Product

1. Select **Pie Chart** from the Visualizations pane.
2. Drag **Product** to the Legend field.
3. Drag **Profit** to the Values field.

You will now see how profit is distributed among products.

---

c. Line Chart – Sales over Months

1. Select **Line Chart**.
2. Drag **Month** to the X-axis field.
3. Drag **Sales** to the Y-axis (Values) field.

This shows how sales change over months.

---

4. Add Slicers (Filters)

Slicers make your report interactive.

1. Select **Slicer** from the Visualizations pane.
2. Drag **Category** into the Field area of the slicer.

Now you can filter all charts by selecting a category.

---

5. Create a Report Page

Arrange your visuals nicely on the report page:

- Place the bar chart on the left

- The pie chart on the right
- The line chart at the bottom
- Add slicers on the top

You can resize visuals by dragging their edges.

---

6. Add Titles and Formatting

To make your report clear and professional:

1. Select each visual and go to the **Format Visual** panel.
2. Turn on **Title** and type a suitable name (for example, “Sales by Category”).
3. Change colors and fonts as needed.

You can also add a **Text box** from the Home ribbon for report titles like “Monthly Sales Report”.

---

7. Publish the Report

When the report is ready:

1. Click **Home → Publish**
2. Sign in to your Power BI account
3. Choose a workspace to upload the report

Once published, you can open and share it online.

---

8. Summary

Task	Description
Get Data	Import data from Excel, CSV, or other sources
Build Visuals	Create bar, line, and pie charts
Add Filters	Use slicers to make reports interactive
Format	Add titles, labels, and colors
Publish	Share reports with others through Power BI Service

Output:

Practical 2

Aim: Time Intelligence and data analysis Functions with DAX

Explanation:

Time Intelligence and Data Analysis Functions with DAX in Power BI

1. What is Time Intelligence

Time Intelligence means analyzing data over time such as days, months, quarters, or years. In Power BI, DAX provides special functions to calculate things like:

- Year to Date (YTD) Sales
- Month to Date (MTD) Sales
- Previous Month or Previous Year Sales
- Growth or change compared to last period

These calculations help understand business trends over time.

---

2. Sample Data

Create a simple dataset in Excel with the following data and save it as **SalesData.xlsx**.

Date	Product	Sales
01-Jan-2023	A	1000
15-Jan-2023	B	1500
01-Feb-2023	A	2000
15-Feb-2023	B	2500
01-Mar-2023	A	3000
15-Mar-2023	B	3500
01-Apr-2023	A	4000
15-Apr-2023	B	4500

Now open Power BI Desktop:

1. Go to **Home → Get Data → Excel**
2. Select the file and click **Load**

---

3. Create a Date Table

Time Intelligence functions work best with a proper date table.

1. In Power BI, go to **Modeling → New Table**
2. Type this formula in the formula bar:

DateTable =

CALENDAR (DATE(2023,1,1), DATE(2023,12,31))

3. Then, go to **Table Tools → Mark as Date Table** and select the **Date** column.

---

4. Create Relationship

Open **Model View**.
Drag and connect:

- **Sales[Date] → DateTable[Date]**

This tells Power BI how dates in both tables are connected.

---

5. Create Measures Using DAX

Now we will create a few simple DAX measures.

a. Total Sales

Total Sales = SUM(Sales[Sales])

This adds up all the sales amounts.

---

b. Year to Date (YTD) Sales

YTD Sales =

TOTALYTD(

  [Total Sales],

  DateTable[Date]

)

This calculates sales from the start of the year up to the current date.

---

c. Month to Date (MTD) Sales

```
MTD Sales =
TOTALMTD(
    [Total Sales],
    DateTable[Date]
)
```

This shows total sales from the beginning of the month to the current date.

---

d. Previous Month Sales

```
Previous Month Sales =
CALCULATE(
    [Total Sales],
    PREVIOUSMONTH(DateTable[Date])
)
```

This returns sales from the previous month.

---

e. Sales Growth Percentage

```
Sales Growth % =
DIVIDE(
    [Total Sales] - [Previous Month Sales],
    [Previous Month Sales],
    0
)
```

This shows the percentage increase or decrease compared to the previous month.

---

6. Create Visuals

To see the results:

1. Add a **Line chart** or **Table visual**.
2. Put **DateTable[Date]** on the X-axis (or rows).
3. Add the measures like **Total Sales**, **YTD Sales**, **MTD Sales**, and **Sales Growth %** to the values area.

You will now see how sales change over time with all the DAX measures.

---

7. Summary

Function	Purpose
TOTALYTD	Calculates year-to-date values
TOTALMTD	Calculates month-to-date values
PREVIOUSMONTH	Returns previous month's values
DATEADD	Shifts time periods forward or backward
SAMEPERIODLASTYEAR	Compares current data with the same period last year

Output:

Practical 3

Aim: Operations on Pinned Reports and Visuals using Power BI

Explanation:

Operations on Pinned Reports and Visuals using Power BI

1. Understanding Pinning in Power BI

In Power BI, when you **pin** a visual or report, it means you are saving it to a **dashboard** so that it can be viewed quickly without reopening the full report.
You can pin charts, tables, or entire report pages.
Pinned visuals become **tiles** on a Power BI dashboard.

Example: You create a bar chart of monthly sales in a report and pin it to your Sales Dashboard. You can then view that chart anytime on the dashboard without opening the full report.

---

2. Steps to Pin Visuals or Reports

Step 1: Open Your Report

- Open Power BI Desktop and publish your report to **Power BI Service** (online).
- Go to the Power BI Service (https://app.powerbi.com) and open the report you want.

Step 2: Pin a Visual

- Hover over the visual you want to pin (for example, a bar chart or line chart).
- Click the **Pin** icon at the top-right corner of the visual.
- Choose an existing dashboard or create a new one.
- Click **Pin**.

Now that visual appears as a **tile** on your dashboard.

Step 3: Pin an Entire Report Page

- Open the report page in Power BI Service.
- On the top menu, click **Pin Live Page**.
- Choose a dashboard and click **Pin**.

This will add the entire report page as a live, interactive tile.

---

3. View Pinned Visuals on the Dashboard

Once you pin visuals or pages, go to the **Dashboards** section in Power BI Service.
Open your dashboard — you will see all pinned visuals displayed as tiles.
You can click on any tile to go back to the original report.

---

4. Operations You Can Perform on Pinned Visuals

a. Resize Tiles

- Hover over a tile's corner.
- Drag to make it bigger or smaller.

b. Move Tiles

- Click and drag a tile to rearrange it on the dashboard.

c. Edit Tile Details

- Click the **More options (three dots)** on the tile.
- Choose **Edit details**.
- You can rename the tile or add a subtitle.

d. Add a Custom Link

- In the **Edit details** panel, you can set a custom link so that clicking the tile opens a specific report, page, or external website.

e. Refresh Data

- Dashboards automatically refresh when the dataset refreshes, but you can manually refresh tiles if needed.

f. Delete a Tile

- Click the **More options (three dots)** on a tile.
- Select **Delete tile** to remove it from the dashboard.

---

5. Pinning from Power BI Desktop (Indirect Method)

In Power BI Desktop, you cannot directly pin visuals.
You first **publish** your report to Power BI Service, and then perform all pinning operations online.
This is because dashboards exist only in the Power BI Service, not in Desktop.

---

6. Benefits of Pinning

Benefit	Description
Quick Access	You can view key visuals instantly without opening full reports
Real-Time Updates	Pinned visuals update automatically when data changes
Custom Layout	Dashboards allow you to arrange visuals freely
Sharing	Dashboards can be shared with others for collaboration
Monitoring	Helps track performance indicators at a glance

---

7. Summary of Operations

Operation	Description
Pin Visual	Add individual charts or tables to a dashboard
Pin Live Page	Add an entire report page to a dashboard
Move Tile	Change position of a visual on dashboard
Resize Tile	Adjust size for better layout
Edit Details	Change title or link for the tile
Delete Tile	Remove unwanted visuals from dashboard
Refresh Tile	Update visuals with latest data

---

8. Practical Example

Suppose you have a report showing:

- Total Sales by Month (Line Chart)
- Profit by Product (Bar Chart)
- Top 5 Customers (Table)

You can pin each of these visuals to a dashboard called “Business Overview.”
From the dashboard, you can view sales trends and profits at a glance and rearrange the visuals based on importance

Output:

Practical 4

Aim: Create one-dimensional data using series and perform various operations on it

Explanation:

1. Introduction

In Python, the **Pandas** library provides a powerful data structure called a **Series**.
A **Series** is a **one-dimensional array-like object** that can hold data of any type (numbers, strings, etc.), along with labels called **indexes**.

We can use a Series to store and manipulate one-dimensional data easily.

---

2. Install and Import Pandas

If Pandas is not installed, install it first:

pip install pandas

Now import it:

import pandas as pd

---

3. Create a Series

Let's create a simple Series containing marks of students.

```
import pandas as pd


# Creating a Series
marks = pd.Series([85, 90, 78, 92, 88], index=['Amit', 'Riya', 'John', 'Sara', 'Neha'])


print("Student Marks Series:")
print(marks)
```

Output:

```
Student Marks Series:
Amit    85
Riya    90
John    78
```

Sara 92

Neha 88

dtype: int64

Here:

- The values [85, 90, 78, 92, 88] are the data.
- The names are **indexes** (like labels).

---

4. Access Elements of Series

You can access elements by **index label** or **position**.

```
# Access by label
print("Marks of Sara:", marks['Sara'])


# Access by position
print("Marks of 2nd student:", marks[1])
```

---

5. Perform Basic Operations

a. Arithmetic Operations

```
# Add 5 marks to each student
print("\nAfter adding 5 marks:")
print(marks + 5)


# Subtract 2 marks
print("\nAfter subtracting 2 marks:")
print(marks - 2)
```

---

b. Statistical Operations

```
print("\nMaximum Marks:", marks.max())
print("Minimum Marks:", marks.min())
```

```
print("Average Marks:", marks.mean())
print("Sum of Marks:", marks.sum())
print("Standard Deviation:", marks.std())
```

---

c. Conditional Operations

```
# Students who scored more than 85
print("\nStudents scoring above 85:")
print(marks[marks > 85])
```

---

d. Sorting the Series

```
# Sort by values
print("\nSorted by Marks:")
print(marks.sort_values())

# Sort by index
print("\nSorted by Names:")
print(marks.sort_index())
```

---

e. Update and Delete Elements

```
# Update a value
marks['John'] = 82
print("\nAfter updating John's marks:")
print(marks)

# Delete an element
marks = marks.drop('Neha')
print("\nAfter deleting Neha's record:")
print(marks)
```

---

f. Check for Missing Values

```
# Create a new Series with missing value
new_marks = pd.Series([85, None, 78, 92, None], index=['Amit', 'Riya', 'John', 'Sara', 'Neha'])

print("\nSeries with missing values:")
print(new_marks)

print("\nCheck for missing values:")
print(new_marks.isnull())

# Fill missing values with average
filled = new_marks.fillna(new_marks.mean())
print("\nAfter filling missing values:")
print(filled)
```

---

6. Summary of Operations

Operation	Description
marks[‘Sara’]	Access element by label
marks[1]	Access element by position
marks + 5	Add 5 to all elements
marks.mean()	Calculate average
marks[marks > 85]	Filter based on condition
marks.sort_values()	Sort by values
marks.drop('Neha')	Delete element
fillna()	Fill missing values

---

7. Conclusion

- A **Pandas Series** is a simple one-dimensional data structure.
- You can perform mathematical, statistical, and conditional operations easily.
- Series are useful for handling labeled data and performing analysis quickly.

Code:

```
# Import pandas library
import pandas as pd

# ----------------------------------------
# 1. Create a Series
# ----------------------------------------
marks = pd.Series([85, 90, 78, 92, 88],
                  index=['Amit', 'Riya', 'John', 'Sara', 'Neha'])

print("Student Marks Series:")
print(marks)
print("-" * 40)

# ----------------------------------------
# 2. Access Elements
# ----------------------------------------
print("Marks of Sara:", marks['Sara'])
print("Marks of 2nd student (Riya):", marks[1])
print("-" * 40)

# ----------------------------------------
# 3. Arithmetic Operations
# ----------------------------------------
print("After adding 5 marks to each student:")
print(marks + 5)

print("\nAfter subtracting 2 marks from each student:")
print(marks - 2)
print("-" * 40)

# ----------------------------------------
# 4. Statistical Operations
# ----------------------------------------
print("Maximum Marks:", marks.max())
print("Minimum Marks:", marks.min())
print("Average Marks:", marks.mean())
print("Sum of Marks:", marks.sum())
```

```
print("Standard Deviation:", marks.std())
print("-" * 40)

# ----------------------------------------
# 5. Conditional Operations
# ----------------------------------------
print("Students scoring above 85:")
print(marks[marks > 85])
print("-" * 40)

# ----------------------------------------
# 6. Sorting
# ----------------------------------------
print("Sorted by Marks (Values):")
print(marks.sort_values())

print("\nSorted by Names (Index):")
print(marks.sort_index())
print("-" * 40)

# ----------------------------------------
# 7. Update and Delete Elements
# ----------------------------------------
marks['John'] = 82
print("After updating John's marks:")
print(marks)

marks = marks.drop('Neha')
print("\nAfter deleting Neha's record:")
print(marks)
print("-" * 40)

# ----------------------------------------
# 8. Missing Values
# ----------------------------------------
new_marks = pd.Series([85, None, 78, 92, None],
                      index=['Amit', 'Riya', 'John', 'Sara', 'Neha'])

print("Series with missing values:")
print(new_marks)

print("\nCheck for missing values (True means missing):")
print(new_marks.isnull())

filled = new_marks.fillna(new_marks.mean())
print("\nAfter filling missing values with average:")
print(filled)
```

```
print("-" * 40)

# ----------------------------------------
# 9. Summary
# ----------------------------------------
print("Original Series:")
print(marks)
print("\nFinal Series after operations:")
print(filled)
```

Output:

Practical 5

Aim: Perform Reshaping of the hierarchical data and pivoting data frame data

Explanation:

1. Introduction

In data analysis, **reshaping** means changing the structure or layout of your data.
In **Pandas**, reshaping can be done using methods like:

- stack() and unstack() (for hierarchical or multi-level data)
- pivot() and pivot_table() (for rearranging columns and rows)

These operations help us view and analyze data in different formats easily.

---

2. Import Pandas

```
import pandas as pd
```

---

3. Create a Sample DataFrame

Let's create simple sales data for different cities and years.

```
# Creating sample data
data = {
   'City': ['Delhi', 'Delhi', 'Mumbai', 'Mumbai', 'Chennai', 'Chennai'],
   'Year': [2022, 2023, 2022, 2023, 2022, 2023],
   'Sales': [25000, 30000, 22000, 27000, 20000, 26000],
   'Profit': [4000, 5000, 3500, 4500, 3000, 4200]
}

df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
```

---

4. Create a Hierarchical (MultiIndex) DataFrame

We can create a **MultiIndex** using City and Year.

```
# Setting hierarchical index
df_hier = df.set_index(['City', 'Year'])
print("\nHierarchical (MultiIndex) DataFrame:")
print(df_hier)
```

---

5. Reshaping using Stack and Unstack

a. Using stack()

stack() moves columns into rows, making the data more compact.

```
stacked = df_hier.stack()
print("\nStacked DataFrame (columns to rows):")
print(stacked)
```

b. Using unstack()

unstack() is the reverse of stack(). It moves rows back into columns.

```
unstacked = stacked.unstack()
print("\nUnstacked DataFrame (rows back to columns):")
print(unstacked)
```

---

6. Pivoting Data

a. Using pivot()

The pivot() function reshapes data based on column values.

```
pivot_df = df.pivot(index='City', columns='Year', values='Sales')
print("\nPivoted DataFrame (Sales per City per Year):")
print(pivot_df)
```

---

b. Using pivot_table()

pivot_table() is similar to pivot(), but it can apply aggregation (like sum, mean).

```
pivot_table_df = df.pivot_table(values='Profit', index='City', columns='Year', aggfunc='mean')
```

print("\nPivot Table showing average profit per city per year:")

print(pivot_table_df)

---

7. Resetting Index

If you want to convert a hierarchical index back to normal columns:

reset_df = df_hier.reset_index()

print("\nAfter resetting the index:")

print(reset_df)

---

8. Summary of Functions

Function	Purpose
set_index()	Create a multi-level (hierarchical) index
stack()	Convert columns into rows
unstack()	Convert rows back into columns
pivot()	Reshape data without aggregation
pivot_table()	Reshape data with aggregation (sum, mean, etc.)
reset_index()	Convert index levels back into columns

Code:

```
import pandas as pd

# ------------------------------------------
# Create sample data
# ------------------------------------------
data = {
   'City': ['Delhi', 'Delhi', 'Mumbai', 'Mumbai', 'Chennai', 'Chennai'],
   'Year': [2022, 2023, 2022, 2023, 2022, 2023],
   'Sales': [25000, 30000, 22000, 27000, 20000, 26000],
   'Profit': [4000, 5000, 3500, 4500, 3000, 4200]
}

df = pd.DataFrame(data)
```

```
print("Original DataFrame:")
print(df)
print("-" * 50)

# ------------------------------------------
# Hierarchical (MultiIndex) Data
# ------------------------------------------
df_hier = df.set_index(['City', 'Year'])
print("Hierarchical (MultiIndex) DataFrame:")
print(df_hier)
print("-" * 50)

# ------------------------------------------
# Reshaping using Stack and Unstack
# ------------------------------------------
stacked = df_hier.stack()
print("Stacked DataFrame (columns to rows):")
print(stacked)
print("-" * 50)

unstacked = stacked.unstack()
print("Unstacked DataFrame (rows back to columns):")
print(unstacked)
print("-" * 50)

# ------------------------------------------
# Pivoting Data
# ------------------------------------------
pivot_df = df.pivot(index='City', columns='Year', values='Sales')
print("Pivoted DataFrame (Sales per City per Year):")
print(pivot_df)
print("-" * 50)

pivot_table_df = df.pivot_table(values='Profit', index='City', columns='Year', aggfunc='mean')
print("Pivot Table showing average profit per city per year:")
print(pivot_table_df)
print("-" * 50)

# ------------------------------------------
# Reset Index
# ------------------------------------------
reset_df = df_hier.reset_index()
print("After resetting the index:")
print(reset_df)
print("-" * 50)
```