B. K. BIRLA COLLEGE (AUTONOMOUS), KALYAN

(DEPARTMENT OF COMPUTER SCIENCE)

CERTIFICATE

*This is to certify that Mr./Ms.*_____
*Roll No.*_____*Exam Seat No.*_____*has satisfactorily completed the Practical in* **Data Science (Data Visualization)** *as laid down in the regulation of University of Mumbai for the purpose of M.Sc. Computer Science* **Semester-III(Practical)** *Examination* **2023-2024**.

Date: 27/01/2024

Place: Kalyan

_____
Signature of Examiners

_____
Professor In-charge

_____
Head
Dept. Of Computer Science

Index

Practical No. 1

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

Background Information:

Series:

The Pandas Series can be defined as a one-dimensional array that can store various data types. We can easily convert the list, tuple, and dictionary into series using the "series' method. The row labels of series are called the index. A Series cannot contain multiple columns. It has the following parameter:

- **data:** It can be any list, dictionary, or scalar value.
- **index:** The value of the index should be unique and hashable. It must be of the same length as data. If we do not pass any index, default **np.arrange(n)** will be used.
- **dtype:** It refers to the data type of series.
- **copy:** It is used for copying the data.

Different Operations in Series:
Mathematical operations

- Sum
- Product
- Square Root
- Exponential
- Logarithm

Statistical operations

- Minimum
- Maximum
- Mean
- Median
- Standard Deviation

Array manipulation

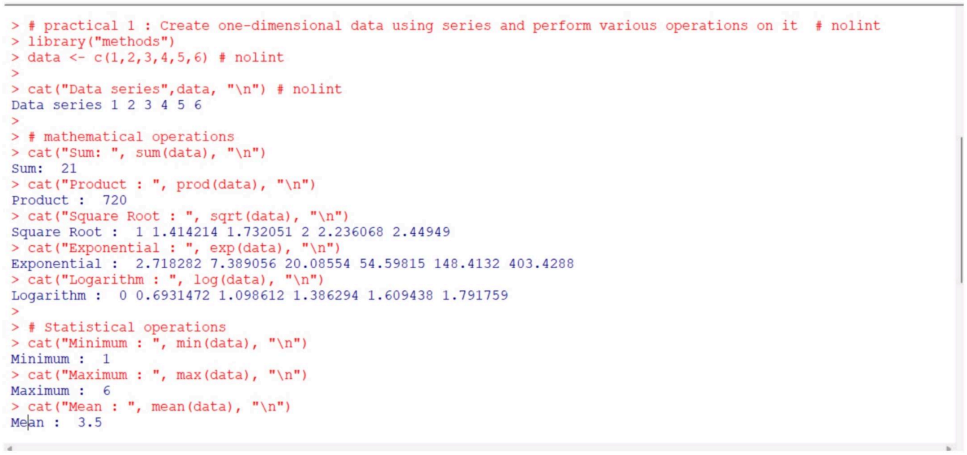
- Sorted array
- Reversed array
- Unique value
- Reshaped data

Code:

```
# practical 1 : Create one-dimensional data using series and perform various operations on it

library("methods")

data <- c(1,2,3,4,5,6)


cat("Data series",data, "\n")
```

```
# mathematical operations
cat("Sum: ", sum(data), "\n")
cat("Product : ", prod(data), "\n")
cat("Square Root : ", sqrt(data), "\n")
cat("Exponential : ", exp(data), "\n")
cat("Logarithm : ", log(data), "\n")

# Statistical operations
cat("Minimum : ", min(data), "\n")
cat("Maximum : ", max(data), "\n")
cat("Mean : ", mean(data), "\n")
cat("Median : ", median(data), "\n")
cat("Standard Deviation : ", sd(data), "\n")

# Array manipulation
cat("Sorted array : ", sort(data), "\n")
cat("Reversed array : ", rev(data), "\n")
cat("Unique value : ", unique(data), "\n")
cat("Reshaped data 2X3" , "\n")
matrix(data,nrow=2,ncol=3)
cat("series to matrix conversion : ",matrix(data,nrow=2,ncol=3) ,"\n")

cat("data series with element wise Addition : ", data + 2, "\n")
cat("data series with element wise multiplication : ", data * 2, "\n")
```

Output:



```
> cat("Median : ", median(data), "\n")
Median :  3.5
> cat("Standard Deviation : ", sd(data), "\n")
Standard Deviation :  1.870829
>
> # Array manipulation
> cat("Sorted array : ", sort(data), "\n")
Sorted array :  1 2 3 4 5 6
> cat("Reversed array : ", rev(data), "\n")
Reversed array :  6 5 4 3 2 1
> cat("Unique value : ", unique(data), "\n")
Unique value :  1 2 3 4 5 6
> cat("Reshaped data 2X3" , "\n")
Reshaped data 2X3
> matrix(data,nrow=2,ncol=3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> cat("series to matrix conversion : ",matrix(data,nrow=2,ncol=3) ,"\n")
series to matrix conversion :  1 2 3 4 5 6
>
> cat("data series with element wise Addition : ", data + 2, "\n")
data series with element wise Addition :  3 4 5 6 7 8
> cat("data series with element wise multiplication : ", data * 2, "\n")
data series with element wise multiplication :  2 4 6 8 10 12
>
```

Practical No. 2

Aim: Create Two-dimensional data with the help of data frames and perform different operations on it.

Background Information:

Data Frame:

- Data Frames are data displayed in a format as a table.
- Data Frames can have different types of data inside it. While the first column can be character, the second and third can be numeric or logical. However, each column should have the same type of data.
- Use the data.frame() function to create a data frame:

Different Operations in Data Frame:

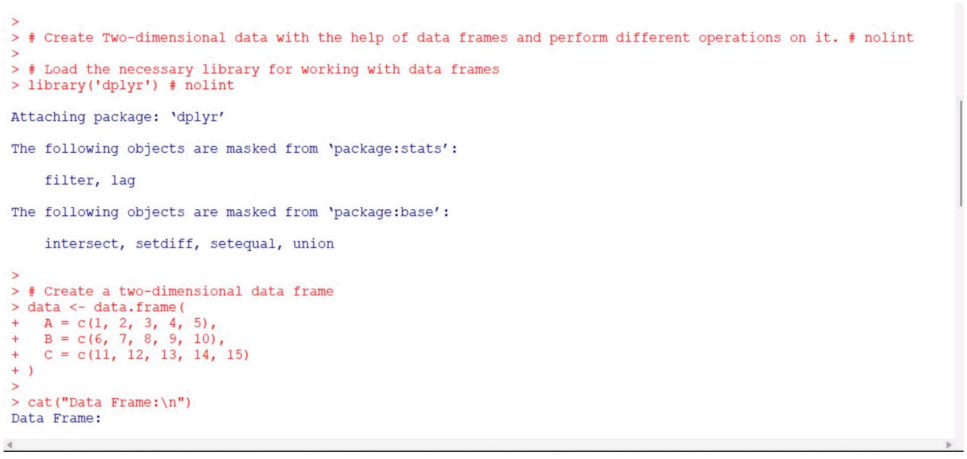
- Accessing columns
- Accessing rows
- Sum of columns
- Mean of rows
- Transpose

Code:

```
# Create Two-dimensional data with the help of data frames and perform different operations on it.


# Load the necessary library for working with data frames

library('dplyr')


# Create a two-dimensional data frame

data <- data.frame(

  A = c(1, 2, 3, 4, 5),

  B = c(6, 7, 8, 9, 10),

  C = c(11, 12, 13, 14, 15)

)


cat("Data Frame:\n")

print(data)


# Accessing columns

cat("Column A:\n")

print(data$A)
```

```
cat("Column B:\n")
print(data$B)
cat("Column C:\n")
print(data$C)

# Accessing rows
cat("Row 1:\n")
print(data[1,])

# Sum of columns
cat("Sum of Column A:", sum(data$A), "\n")
cat("Sum of Column B:", sum(data$B), "\n")
cat("Sum of Column C:", sum(data$C), "\n")

# Mean of rows
mean(as.numeric(data[1, ]))
mean(as.numeric(data[2, ]))

# Transpose the data frame
transposed_data <- t(data)
cat("Transposed Data Frame:\n")
print(transposed_data)
```

Output:



```
> cat("Data Frame:\n")
Data Frame:
> print(data)
  A  B  C
1 1  6 11
2 2  7 12
3 3  8 13
4 4  9 14
5 5 10 15
>
> # Accessing columns
> cat("Column A:\n")
Column A:
> print(data$A)
[1] 1 2 3 4 5
> cat("Column B:\n")
Column B:
> print(data$B)
[1]  6  7  8  9 10
> cat("Column C:\n")
Column C:
> print(data$C)
[1] 11 12 13 14 15
>
> # Accessing rows
> cat("Row 1:\n")
```

```
> # Accessing rows
> cat("Row 1:\n")
Row 1:
> print(data[1,])
  A B  C
1 1 6 11
>
>
> # Sum of columns
> cat("Sum of Column A:", sum(data$A), "\n")
Sum of Column A: 15
> cat("Sum of Column B:", sum(data$B), "\n")
Sum of Column B: 40
> cat("Sum of Column C:", sum(data$C), "\n")
Sum of Column C: 65
>
> # Mean of rows
> mean(as.numeric(data[1, ]))
[1] 6
> mean(as.numeric(data[2, ]))
[1] 7
>
> # Transpose the data frame
> transposed_data <- t(data)
> cat("Transposed Data Frame:\n")
Transposed Data Frame:
```

```
>
> # Transpose the data frame
> transposed_data <- t(data)
> cat("Transposed Data Frame:\n")
Transposed Data Frame:
> print(transposed_data)
  [,1] [,2] [,3] [,4] [,5]
A    1    2    3    4    5
B    6    7    8    9   10
C   11   12   13   14   15
>
```

Practical No. 3

Aim: Write a code to read data from the different file formats like JSON, HTML, XML, and CSV files and check for missing data and outlier values and handle them.

Background Information:

JSON:

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

HTML:

HTML stands for HyperText Markup Language. It creates a complete website structure of web pages. HTML is a combination of Hypertext and Markup language. Hypertext defines the link between the web pages and markup language defines the text document within the tag.

XML:

Extensible Markup Language (XML) is a markup language that provides rules to define any data. Unlike other programming languages, XML cannot perform computing operations by itself.

Code:

```
# Load the required libraries

library(rjson)

library(XML)


# Read data from JSON file

json_data <- fromJSON(file = " File Path")


# Read data from HTML file

html_data <- readHTMLTable(" File Path")


# Read data from XML filed

xml_data <- xmlParse(" File Path")

xml_data <- xmlToList(xml_data)

# Read data from CSV file

csv_data <- read.csv("File Path")
```

```
# Check for missing data
missing_data <- sapply(json_data, function(x) sum(is.na(x)))
missing_data <- missing_data[missing_data > 0]

# Check for outlier values
outlier_data <- sapply(csv_data, function(x) sum(x < 0))
outlier_data <- outlier_data[outlier_data > 0]

# Handle missing data and outlier values
json_data[is.na(json_data)] <- 0
csv_data[csv_data < 0] <- 0

# Print the data
print(json_data)
print(html_data)
print(xml_data)
print(csv_data)
```

Output:

```
> # Load the required libraries
> library(rjson)
> library(XML)
>
> # Read data from JSON file
> json_data <- fromJSON(file = "D:\\Downloads\\aa\\DV\\Practical3\\data.json")
>
> # Read data from HTML file
> html_data <- readHTMLTable("D:\\Downloads\\aa\\DV\\Practical3\\data.html")
>
> # Read data from XML filed
> xml_data <- xmlParse("D:\\Downloads\\aa\\DV\\Practical3\\data.xml")
> xml_data <- xmlToList(xml_data)
>
> # Read data from CSV file
> csv_data <- read.csv("D:\\Downloads\\aa\\DV\\Practical3\\data.csv")
>
> # Check for missing data
> missing_data <- sapply(json_data, function(x) sum(is.na(x)))
> missing_data <- missing_data[missing_data > 0]
>
> # Check for outlier values
> outlier_data <- sapply(csv_data, function(x) sum(x < 0))
> outlier_data <- outlier_data[outlier_data > 0]
>
> # Handle missing data and outlier values
```

```
>
> # Handle missing data and outlier values
> json_data[is.na(json_data)] <- 0
> csv_data[csv_data < 0] <- 0
>
> # Print the data
> print(json_data)
$fruit
[1] "Apple"

$size
[1] "Large"

$color
[1] "Red"

> print(html_data)
$`NULL`
                        Company          Contact Country
1          Alfreds Futterkiste     Maria Anders Germany
2   Centro comercial Moctezuma  Francisco Chang  Mexico
3                 Ernst Handel    Roland Mendel Austria
4               Island Trading    Helen Bennett      UK
5 Laughing Bacchus Winecellars  Yoshi Tannamuri  Canada
6 Magazzini Alimentari Riuniti Giovanni Rovelli   Italy
```

```
> print(xml_data)
$employees
$employees$employee
$employees$employee$id
[1] "1"

$employees$employee$name
[1] "John Doe"

$employees$employee$position
[1] "Software Engineer"

$employees$employee$department
[1] "Engineering"


$employees$employee
$employees$employee$id
[1] "2"

$employees$employee$name
[1] "Steven Broad"

$employees$employee$position
[1] "IT Engineer"
```

```
[1] "3"

$employees$employee$name
[1] "Jane Warner"

$employees$employee$position
[1] "Software Engineer"

$employees$employee$department
[1] "Engineering"


> print(csv_data)
   Numeric Numeric.2 Numeric.Suffix
1        1         1            1st
2        2         2            2nd
3        3         3            3rd
4        4         4            4th
5        5         5            5th
6        6         6            6th
7        7         7            7th
8        8         8            8th
9        9         9            9th
10      10        10           10th
>
```

Practical No. 4

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

Background Information:

Pivoting data refers to the process of reorganizing and reshaping data to analyze it from a different perspective. This transformation is often necessary to make the data more suitable for analysis, reporting, or visualization. The specific steps involved in pivoting data may vary depending on the tools or programming languages used, but the general idea is to convert data from a long format to a wide format or vice versa. Here are two common types of data pivoting:

1. Wide to Long (Unpivoting):

This type of pivoting involves transforming wide-format data into a long-format.

2. Long to Wide (Pivoting):

This type of pivoting involves transforming long-format data into a wide-format.

Code:

```
# install.packages("tidyverse", type="source")
library(tidyverse)


head(billboard)


#pivot_longer(): Pivot data from wide to long
billboard_long <- billboard |>
  pivot_longer(
    cols = starts_with('wk'),
    names_to = 'week',
    values_to = 'rank'
  )


billboard_long


#pivot_wider(): Pivot data from long to wide
billboard_long |>
  pivot_wider(
    names_from = week,
```

values_from = rank

)

Output:

```
> library(tidyverse)
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.4
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.4.4     ✔ tibble    3.2.1
✔ lubridate 1.9.3     ✔ tidyr     1.3.0
✔ purrr     1.0.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
>
> head(billboard)
# A tibble: 6 × 79
  artist      track date.entered   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8
  <chr>       <chr> <date>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 2 Pac       Baby… 2000-02-26      87    82    72    77    87    94    99    NA
2 2Ge+her     The … 2000-09-02      91    87    92    NA    NA    NA    NA    NA
3 3 Doors Do… Kryp… 2000-04-08      81    70    68    67    66    57    54    53
4 3 Doors Do… Loser 2000-10-21      76    76    72    69    67    65    55    59
5 504 Boyz    Wobb… 2000-04-15      57    34    25    17    17    31    36    49
6 98^0        Give… 2000-08-19      51    39    34    26    26    19     2     2
# ℹ 68 more variables: wk9 <dbl>, wk10 <dbl>, wk11 <dbl>, wk12 <dbl>,
#   wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>, wk18 <dbl>,
#   wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>, wk24 <dbl>,
#   wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>, wk30 <dbl>,
#   wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>, wk35 <dbl>, wk36 <dbl>,
#   wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>, wk41 <dbl>, wk42 <dbl>,
#   wk43 <dbl>, wk44 <dbl>, wk45 <dbl>, wk46 <dbl>, wk47 <dbl>, wk48 <dbl>, …
>
> #pivot_longer(): Pivot data from wide to long
> billboard_long <- billboard |>
+   pivot_longer(
+     cols = starts_with('wk'),
+     names_to = 'week',
+     values_to = 'rank'
+   )
>
> billboard_long
# A tibble: 24,092 × 5
   artist track                   date.entered week   rank
   <chr>  <chr>                   <date>       <chr> <dbl>
 1 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk1      87
 2 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk2      82
 3 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk3      72
 4 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk4      77
 5 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk5      87
 6 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk6      94
 7 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk7      99
 8 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk8      NA
 9 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk9      NA
> #pivot_wider(): Pivot data from long to wide
> billboard_long |>
+   pivot_wider(
+     names_from = week,
+     values_from = rank
+   )
# A tibble: 317 × 79
   artist      track date.entered   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8
   <chr>       <chr> <date>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 2 Pac       Baby… 2000-02-26      87    82    72    77    87    94    99    NA
 2 2Ge+her     The … 2000-09-02      91    87    92    NA    NA    NA    NA    NA
 3 3 Doors D…  Kryp… 2000-04-08      81    70    68    67    66    57    54    53
 4 3 Doors D…  Loser 2000-10-21      76    76    72    69    67    65    55    59
 5 504 Boyz    Wobb… 2000-04-15      57    34    25    17    17    31    36    49
 6 98^0        Give… 2000-08-19      51    39    34    26    26    19     2     2
 7 A*Teens     Danc… 2000-07-08      97    97    96    95   100    NA    NA    NA
 8 Aaliyah     I Do… 2000-01-29      84    62    51    41    38    35    35    38
 9 Aaliyah     Try … 2000-03-18      59    53    38    28    21    18    16    14
10 Adams, Yo…  Open… 2000-08-26      76    76    74    69    68    67    61    58
# ℹ 307 more rows
# ℹ 68 more variables: wk9 <dbl>, wk10 <dbl>, wk11 <dbl>, wk12 <dbl>,
#   wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>, wk18 <dbl>,
#   wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>, wk24 <dbl>,
#   wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>, wk30 <dbl>,
#   wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>, wk35 <dbl>, wk36 <dbl>,
#   wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>, wk41 <dbl>, wk42 <dbl>, …
```

Practical No. 5

Aim: Connecting and extracting with various data resources in tableau.

Background Information:

Data Sources

Tableau can extract data from all the popular data sources. These include:

1) **File System**

The simplest data source you can use with Tableau is a file. These could be files like an Excel spreadsheet, a CSV file or a text file.

2) **Cloud System**

You can also source data from popular cloud sources. Some of the options are:

- Google Analytics
- Google BigQuery
- Windows Azure
- Amazon Redshift

3) **Relational systems**

You can connect to many types of relational databases such as SQL Server, Oracle, and DB2.

4) **Live Data Sources**

Connect live is a feature of Tableau that allows you to connect real-time data. Tableau does this by constantly reading the data, so your visualizations are constantly up to date.

5) **Using In-Memory Data**

The alternative to connecting to a live data source is to load one into memory. This is a better option for static data that won't change anytime soon, as it will only be loaded once. The in-memory database will then be analyzed by Tableau.

6) **Connecting Multiple Data Sources**

One of the great features of Tableau is the ability to combine data sources. You can work with data from a file system and data from a relational database all at the same time. All you need to do is define multiple data connections.

Data Extraction Techniques

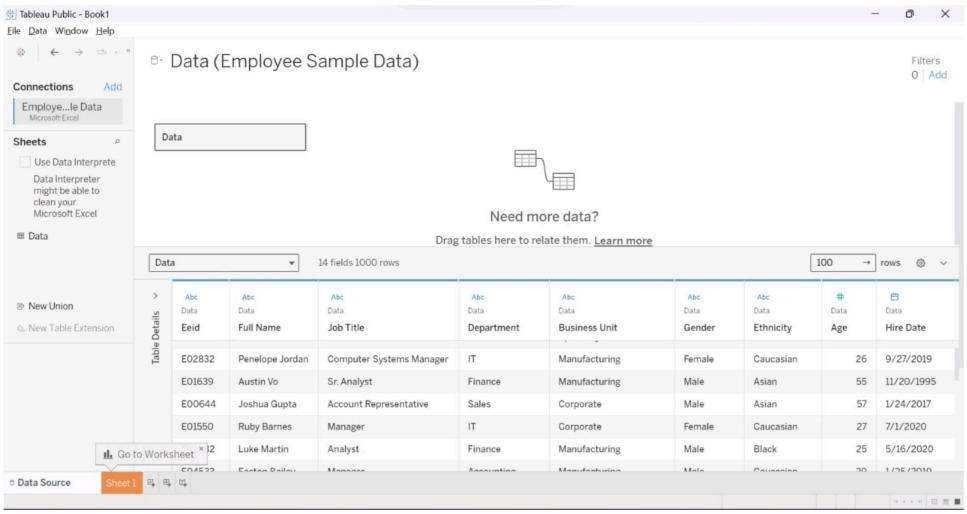
- Once you've decided on your data sources, the next step is to extract the data you need from those sources.
- Whether you are connecting to a live database or storing your data in memory, you may well want to cut it down to only what you need for your application. This will mean you'll have less data to extract from a live source or a smaller amount of data to store in memory.
- It also converts the data to a form that works well with the Tableau engine, meaning things will speed up even more.
- With Tableau, this is done with data extracts.

- A data extract is simply a subset of a total data source. When extracting data, you can choose exactly what you want and how much of underlying data to extract using extract data dialog box.
- To create a new Tableau data extract, go to Data -> Extract Data. You'll be presented with many options to limit the number of rows and aggregate for dimensions. Here is where you can use filters to cut down your data to just the things you need.

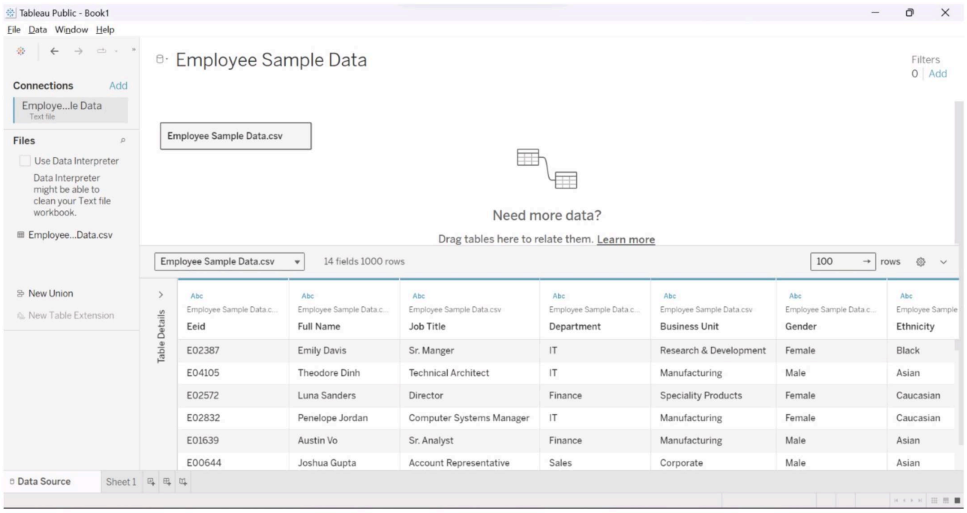
Filtering Extracted Data

- You might not need every single field and row in the data you've extracted. By cutting it down to just the things you need, you can improve performance and make life easier for yourself.
- There are three main types of filters to use in Tableau
  1) Dimension filer
  2) Measure filter
  3) Date filter
- Each works on a different type of data field. To apply a filter, simply drag a field into the filter pane, it looks like this.
- Then you'll be prompted with some options for your filter. Choose the ones you need and click apply.
- Once you've created a data extract, you can add more data to it from the data pane. Do this by going to Data -> Extract -> Append to File. You can do this with new data types, just make sure they are the same type and have the same number of fields as the original data.
- It's possible to work on large data sets using Tableau. Things do, however, get a little more difficult if your dataset doesn't fit in memory. This is where data extracts and filters really come in handy. If your data is still too big to fit in RAM after extracting and filtering it down, it will still work but will run a lot more slowly.

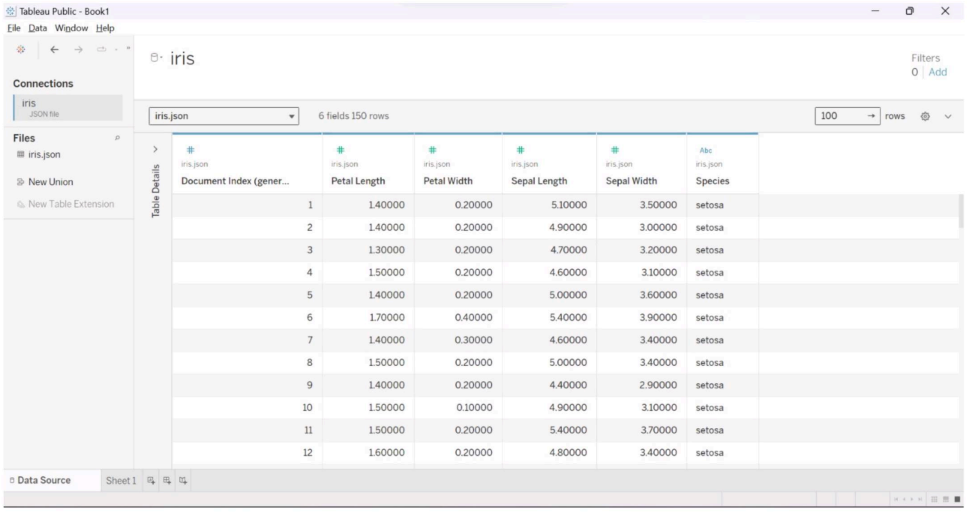
Excel File Data Source



Text File Data Source



JSON File Data Source



Practical No. 6

Aim: Performing calculations and creating parameters in Tableau.

Background Information:

In Tableau, calculated fields are user-defined fields that you can create to perform custom calculations on your data. These fields are created using Tableau's formula language and can involve mathematical operations, aggregations, logical expressions, string manipulations, and more. Calculated fields allow you to derive new information from existing data or perform complex calculations that are not present in the original dataset.

Steps:

1. Open Tableau:

- Launch Tableau Desktop and open the workbook to which you want to add a calculated field.

2. Navigate to a Worksheet:

- Click on the worksheet tab at the bottom of the screen.

3. Open Data Pane:

- On the left side of the screen, locate the "Data" pane.

4. Right-Click and Choose "Create Calculated Field":

- Right-click in the Data pane and choose "Create Calculated Field."

5. Name Your Calculated Field:

- Enter a meaningful name for your calculated field in the "Name" field.

6. Define the Calculation:

- In the calculation editor that appears, define your calculation using Tableau's calculation syntax. For example:

  - Basic arithmetic: `Profit - Cost`

  - Aggregation: `SUM(Sales) / COUNTD(CustomerID)`

  - Logical operations: `IF Category = 'Electronics' THEN Sales * 1.1 ELSE Sales END`

7. Click OK:

- Once you've defined your calculation, click "OK" to create the calculated field.

8. Use the Calculated Field:

- Now, you can use your calculated field in the Rows or Columns shelf, in the Color or Size marks card, or in other parts of the visualization.

9. Edit or Delete Calculated Field (if needed):

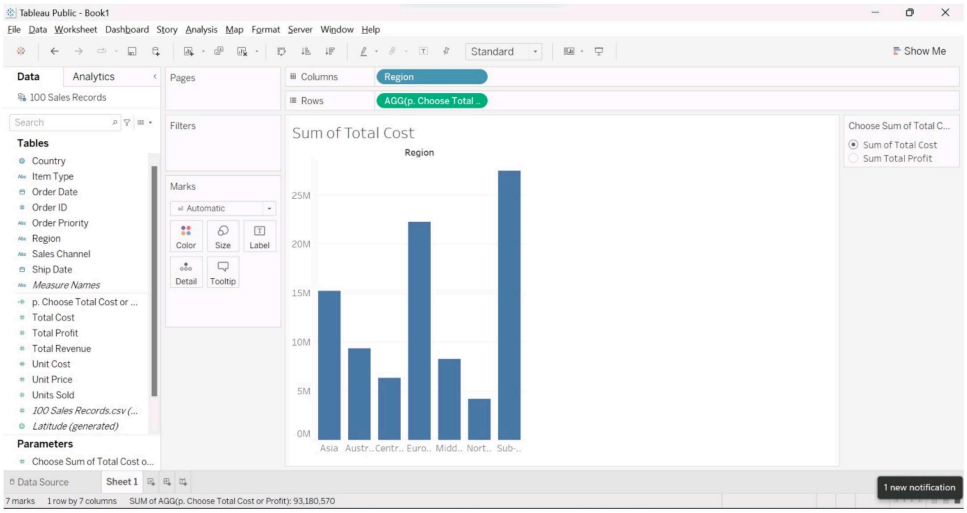
- If you need to make changes to your calculated field, right-click on it in the Data pane and choose "Edit" or "Delete."

10. Save Your Workbook:

- Save your Tableau workbook to preserve the calculated field for future use.

Calculated fields are dynamic and update as your data changes. They provide a powerful way to extend the capabilities of your visualizations by incorporating custom logic and calculations.

Output:



Practical No. 7

Aim: Create a Trend model using data, Analyse-it and use it for forecasting.

Background Information:

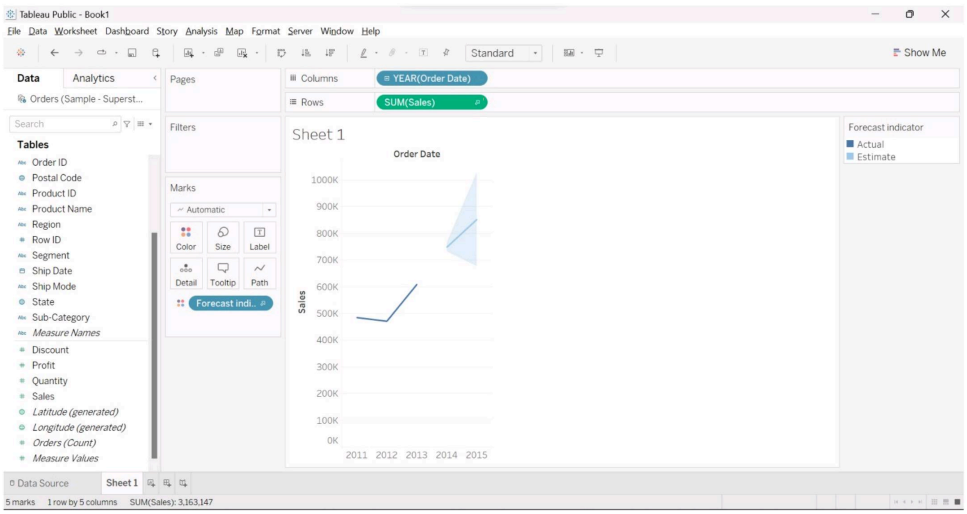
A trend model refers to a statistical or analytical approach used to identify and analyze trends in data. It can be applied in various fields such as finance, economics, and data analysis to recognize patterns or directional movements over time. Trend models often involve mathematical algorithms or statistical methods to help predict future trends based on historical data.

Steps:

- Connect Tableau desktop to the **Sample Superstore** data set.
- Once you have connected to the **Orders** sheet, go to **Sheet 1**.
- Drag **Order Date** to the **Columns** shelf. (Make sure the date is set to continuous in a Month / Year format.)
- Drag **Sales** to the **Rows** shelf.
- Go to the **Analysis** menu and select **Show Forecast** from the **Forecast** menu.

Tableau will automatically add a line to your chart forecasting the sales for the next 12 months. The light color around the line is the estimated range for the sales.

Output:



Practical No. 8

Aim: Create Dashboard and Storytelling using tableau.

Background Information:

Dashboard:

A dashboard is a way of displaying various types of visual data in one place. Usually, a dashboard is intended to convey different, but related information in an easy-to-digest form. And oftentimes, this includes things like key performance indicators (KPI)s or other important business metrics that stakeholders need to see and understand immediately.

Dashboards are useful across different industries and verticals because they're highly customizable. They can include data of all sorts with varying date ranges to help you understand what happened, why it happened, what may happen, and what action you should take. And since dashboards use visualizations like tables, graphs, and charts, others who aren't as close to the data can quickly and easily understand the story it tells or the insights it reveals.

Storytelling:

In Tableau, a story is a sequence of visualizations that work together to convey information. You can create stories to tell a data narrative, provide context, demonstrate how decisions relate to outcomes, or to simply make a compelling case.

A story is a sheet, so the methods you use to create, name, and manage worksheets and dashboards also apply to stories. At the same time, a story is also a collection of sheets, arranged in a sequence. Each individual sheet in a story is called a story point.

When you share a story —for example, by publishing a workbook to Tableau Public, Tableau Server, or Tableau Cloud—users can interact with the story to reveal new findings or ask new questions about the data.

Dashboard Steps:

1. Connect to Data:

- Open Tableau, connect to your data source.

2. Create Visualizations:

- Build the charts and graphs you want.

3. Go to Dashboard Tab:

- Click on the "Dashboard" tab.

4. Drag Visualizations:

- Drag your visualizations onto the dashboard canvas.

5. Arrange and Size:

- Organize using layout containers, adjust size.

6. Add Interactivity:

- Set up actions for interactivity (e.g., filters).

7. Add Titles and Text:

- Include titles, text, and annotations.

8. Format and Style:

- Adjust formatting and styling.

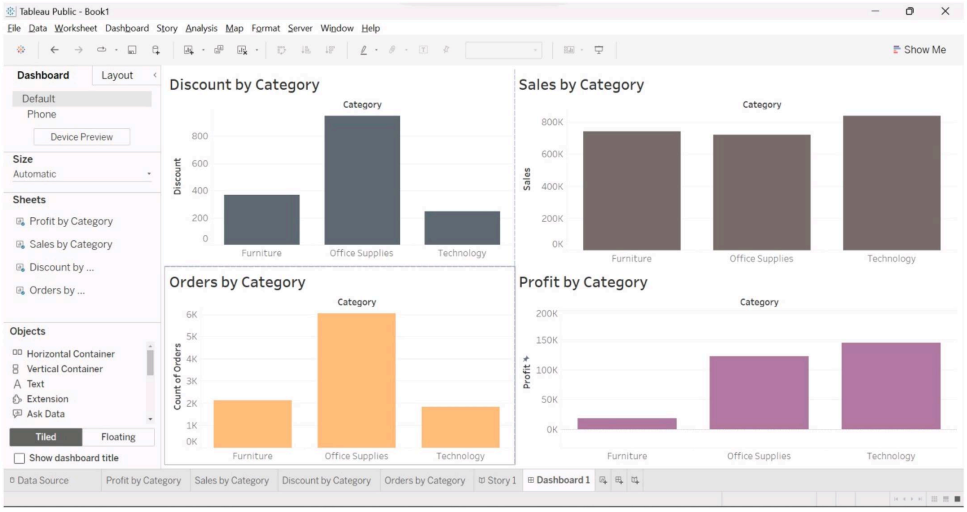
9. Test and Refine:

- Test interactivity, make refinements.

10. Save and Publish:

- Save the workbook and publish it to Tableau Server or Tableau Online if needed.

Output:



Story Steps:

1. Prepare Your Data:

- Connect to your data source and organize your data as needed.

2. Create Worksheets and Dashboards:

- Build the individual visualizations or dashboards that you want to include in your story.

3. Open a New Story:

- Click on the "Story" tab at the bottom of the screen.

4. Add a Blank Story Point:

- Click on the "Blank" option to add a new story point.

5. Add Sheets or Dashboards:

- Drag and drop sheets or dashboards onto the story point canvas. Each story point can include one or more visualizations.

6. Arrange and Annotate:

- Organize the order of your story points by dragging them into the desired sequence.

- Add annotations, titles, or text to provide context for each story point.

7. Add Captions:

- Click on a specific sheet or dashboard in a story point to add a caption. This helps explain the content to your audience.

8. Format and Style:

- Adjust the formatting and style of your story to ensure consistency and clarity.

9. Preview Your Story:

- Click the "Show/Hide Dashboard" button to preview how your story will appear to viewers.

10. Add More Story Points:

- Continue adding additional story points to create a complete narrative.

11. Add Story Transitions (Optional):

- Use the "Animate" option to add transitions between story points for a smoother flow.

12. Test and Refine:

- Test your story to ensure that the sequence and information flow smoothly.

13. Save and Share:

- Save your Tableau workbook containing the story.

- If you have Tableau Server or Tableau Online access, you can publish the story for sharing.

Output: