GR20 Regulations

III B.Tech II Semester

Data Visualization Lab

(GR20B3005)

B.Tech with Minor in Artificial Intelligence and Machine Learning

GOKARAJU RANGARAJU
INSTITUTE OF ENGINEERING AND TECHNOLOGY
(Autonomous)

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

DATA VISUALIZATION LAB

Course Code: GR20B3005 **L/T/P/C: 3/0/0/3**

Course Objectives:

- Make more effective visualizations for data.
- Understand how fundamental principles of design and human cognition inform effective visualizations.
- Utilize popular visualization applications such as Tableau and Excel.
- Develop web pages that allow others to interact with data.
- Create visualizations using interactive web graphics programming in SVG format, java script, and D3.js.

Course Outcomes: On completion of the course the student should be able to

- Demonstrate knowledge of technical advances through active participation in life-long
- Discuss concepts and principles of data visualization particularly related to decisionmaking.
- Investigate technologies and practices for visualizing data as part of a datamanagement and analytics system
- Conduct research relevant data visualization topics
- Use existing visualization tools and techniques to analyze basic datasets.

Task-1:

Defining data visualization; Visualization workflow: describing data visualization workflow, process in practice.

Task-2:

Data Representation: chart types: categorical, hierarchical, relational, temporal &spatial

Task-3:
2D: Bar charts, Clustered bar charts, dot plots, connected dot plots

Task-4:
2D: pictograms, proportional shape charts, bubble charts, radar charts, polar charts

Task-5:
2 D: Range chart, Box-and- whisker plots, univariate scatter plots, histograms word cloud

Task-6:
2 D: Pie chart, waffle chart, stacked bar chart, back-to-back bar chart, tree map

Task-7:
3-D: Surfaces, contours, hidden surface

Task-8:
3-D: pm3d coloring, 3Dmapping

Task-9:
Programs on multi-dimensional data visualization

Task-10:
Programs on manifold visualization

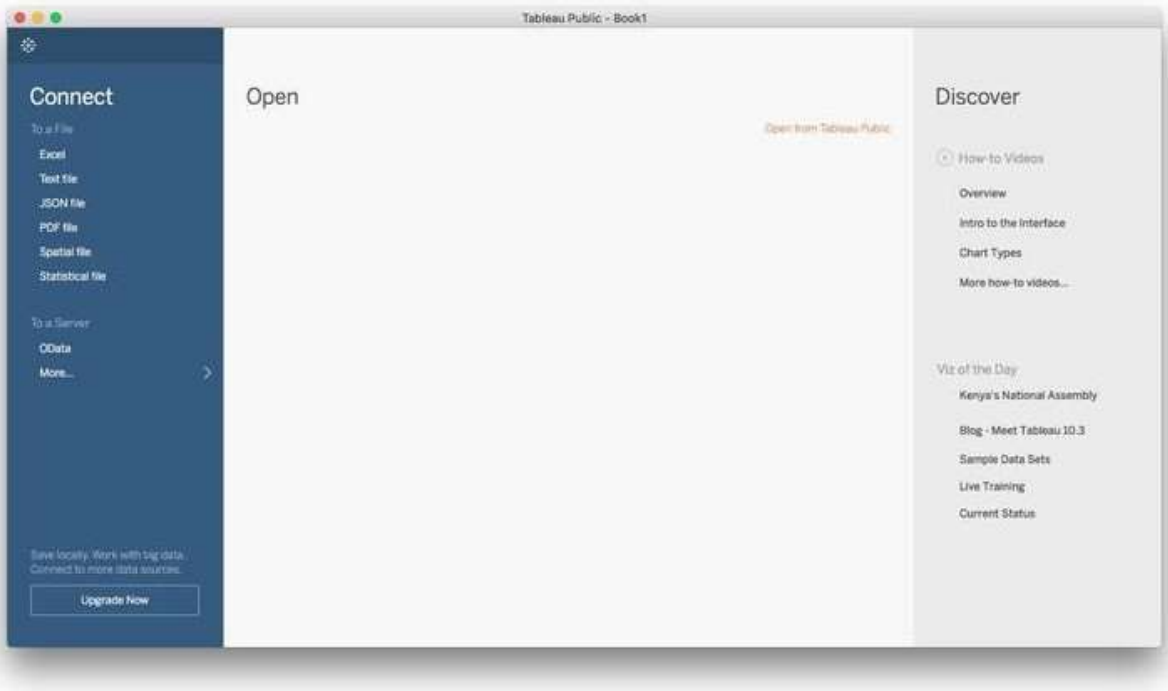
Task-11:
Programs on graph data visualization

Task-12:
Programs on Annotation

INDEX

Task1: Defining data visualization; Visualization workflow:
describing data visualization workflow, process in practice.

Aim: Understanding of Data Visualization process
Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data.
In the world of Big Data, data visualization tools and technologies are essential to analyze massive amounts of information and make data-driven decisions.
Tableau Interface:



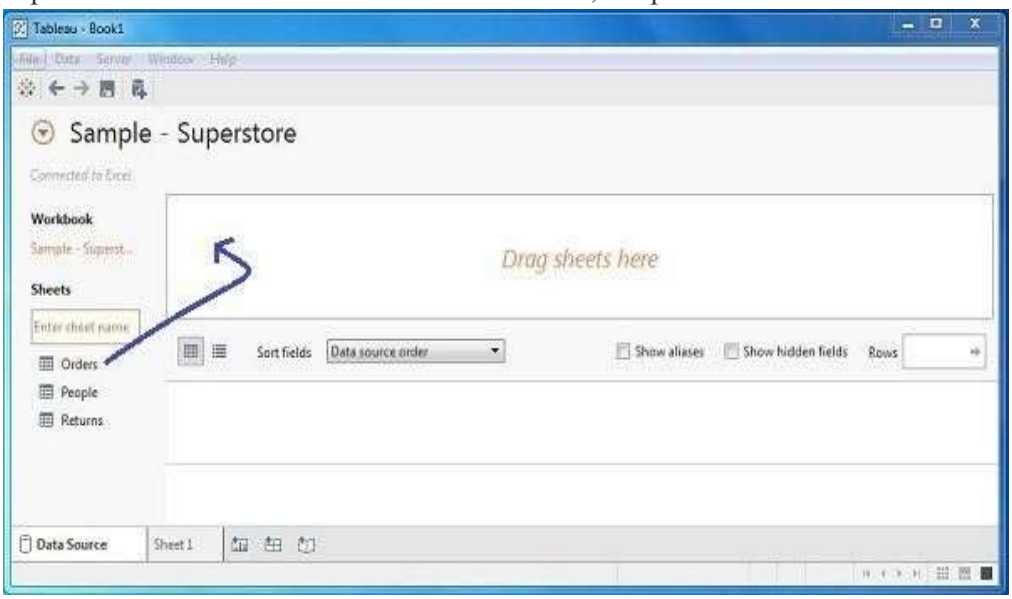
There are three basic steps involved in creating any Tableau data analysis report.
These three steps are −
Connect to a data source − It involves locating the data and using an appropriate type of connection to read the data.

Choose dimensions and measures − This involves selecting the required columns from the source data for analysis.

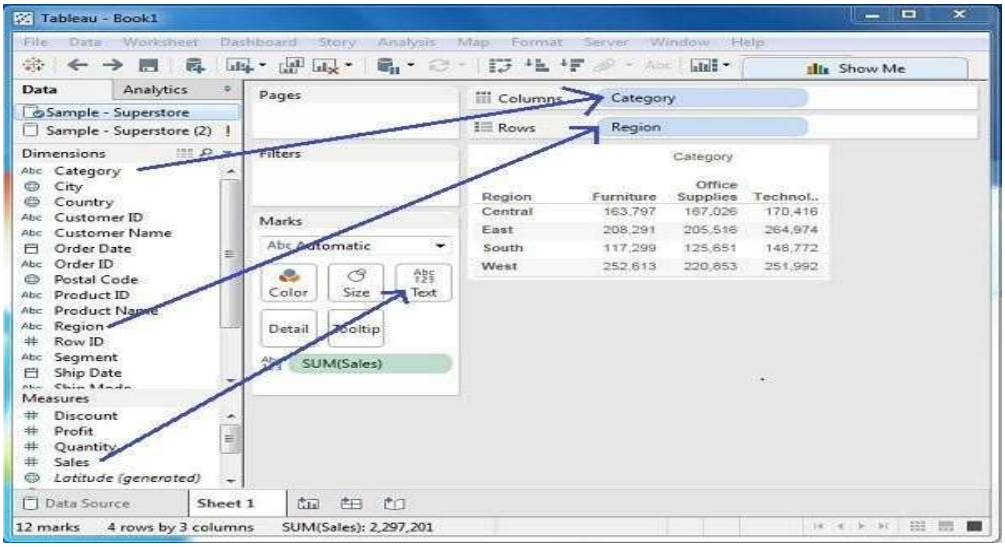
Apply visualization technique − This involves applying required visualization methods, such as a specific chart or graph type to the data being analyzed.

Connect to a data source

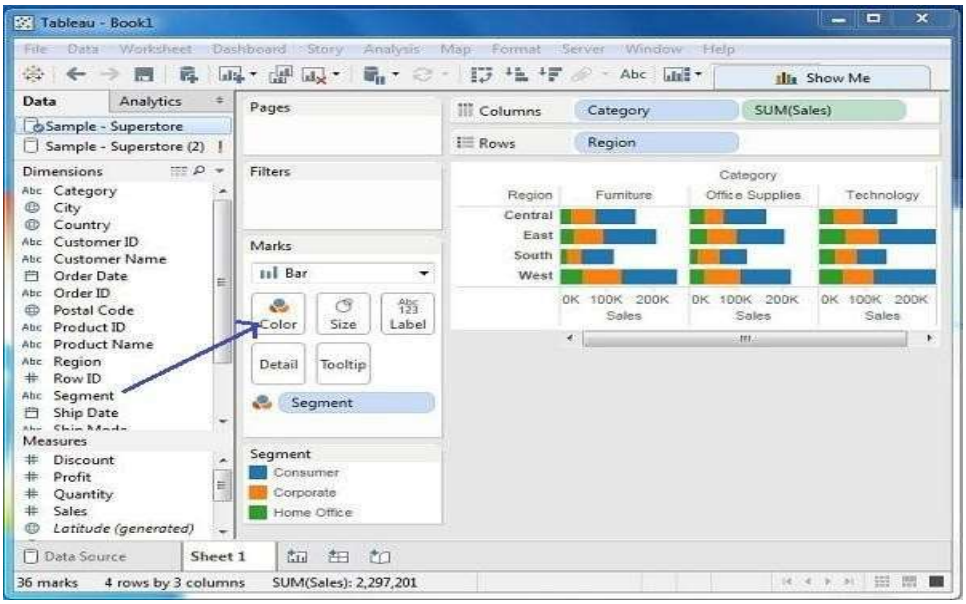
Superstore.xls file has three sheets named Orders, People and Returns. Choose Orders.



Choose dimensions and measures



Apply visualization technique



Common general types of data visualization:

- Charts
- Tables
- Graphs
- Maps
- Infographics
- Dashboards

More specific examples of methods to visualize data:

- Area Chart
- Bar Chart
- Box-and-whisker Plots
- Bubble Cloud
- Bullet Graph
- Cartogram
- Circle View
- Dot Distribution Map
- Gantt Chart
- Heat Map
- Highlight Table
- Histogram
- Matrix
- Network
- Polar Area
- Radial Tree
- Scatter Plot (2D or 3D)
- Streamgraph

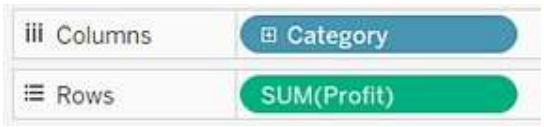
- Text Tables
- Timeline
- Treemap
- Wedge Stack Graph
- Word Cloud
- And any mix-and-match combination in a dashboard!

Task 2: Experiment Data Representation: chart types: categorical, hierarchical, relational, temporal & spatial.

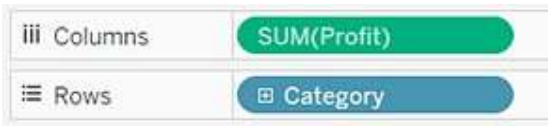
Aim : To visualize categorical, hierarchical, relational, temporal& spatial data

Categorical Data Representation:
Use bar charts to compare data across categories. You create a bar chart by placing a dimension on the Rows shelf and a measure on the Columns shelf, or vice versa.
A bar chart uses the Bar mark type. Tableau selects this mark type when the data view matches one of the two field arrangements shown below. You can add additional fields to these shelves.

Creates Vertical Bars



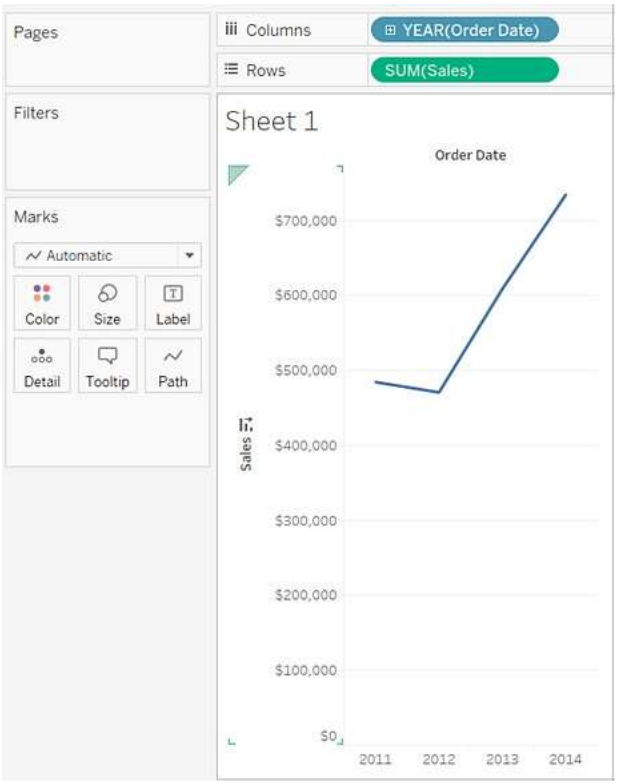
Creates Horizontal Bars



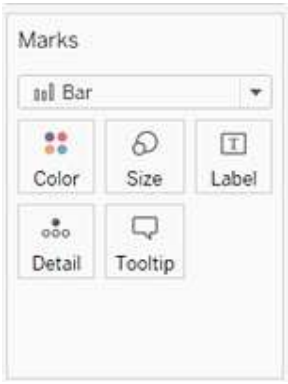
To create a bar chart that displays total sales over a four-year period, follow these steps:

Connect to the Sample - Superstore data source.

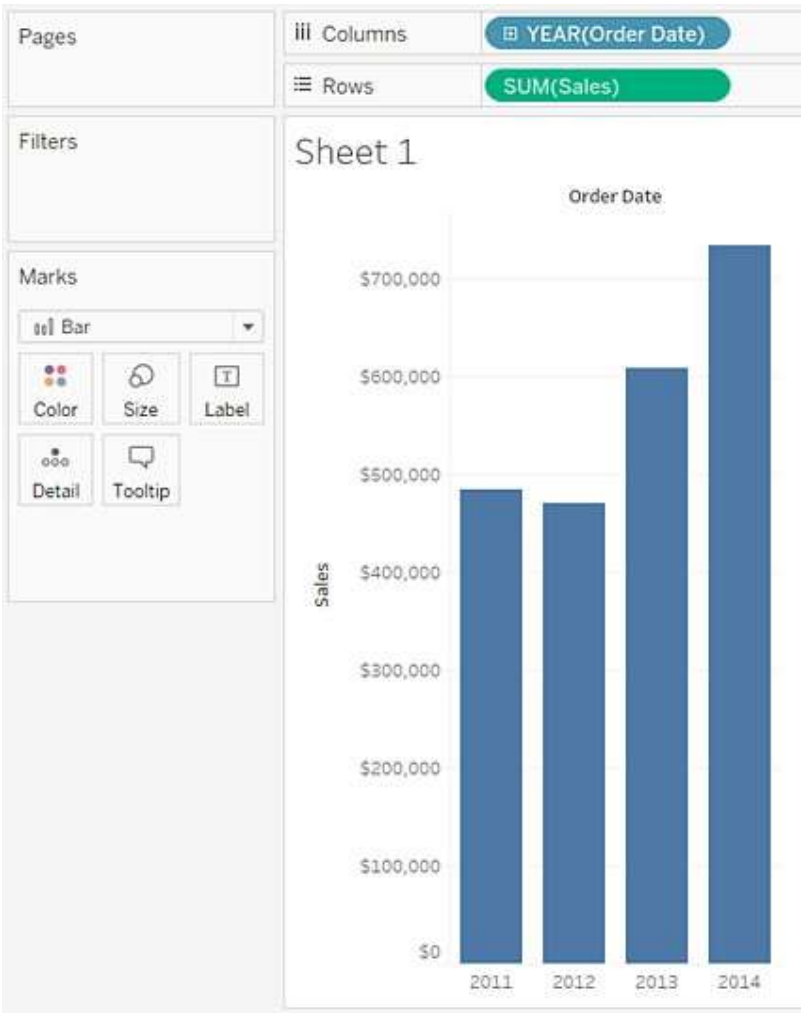
Drag the Order Date dimension to Columns and drag the Sales measure to Rows.



On the Marks card, select Bar from the drop-down list.

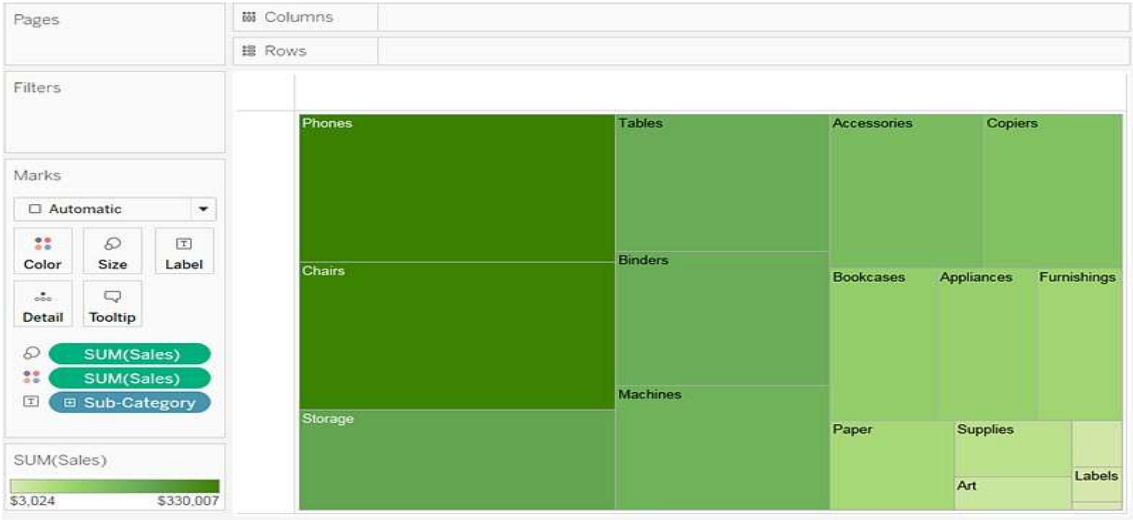


The view changes to a bar chart.

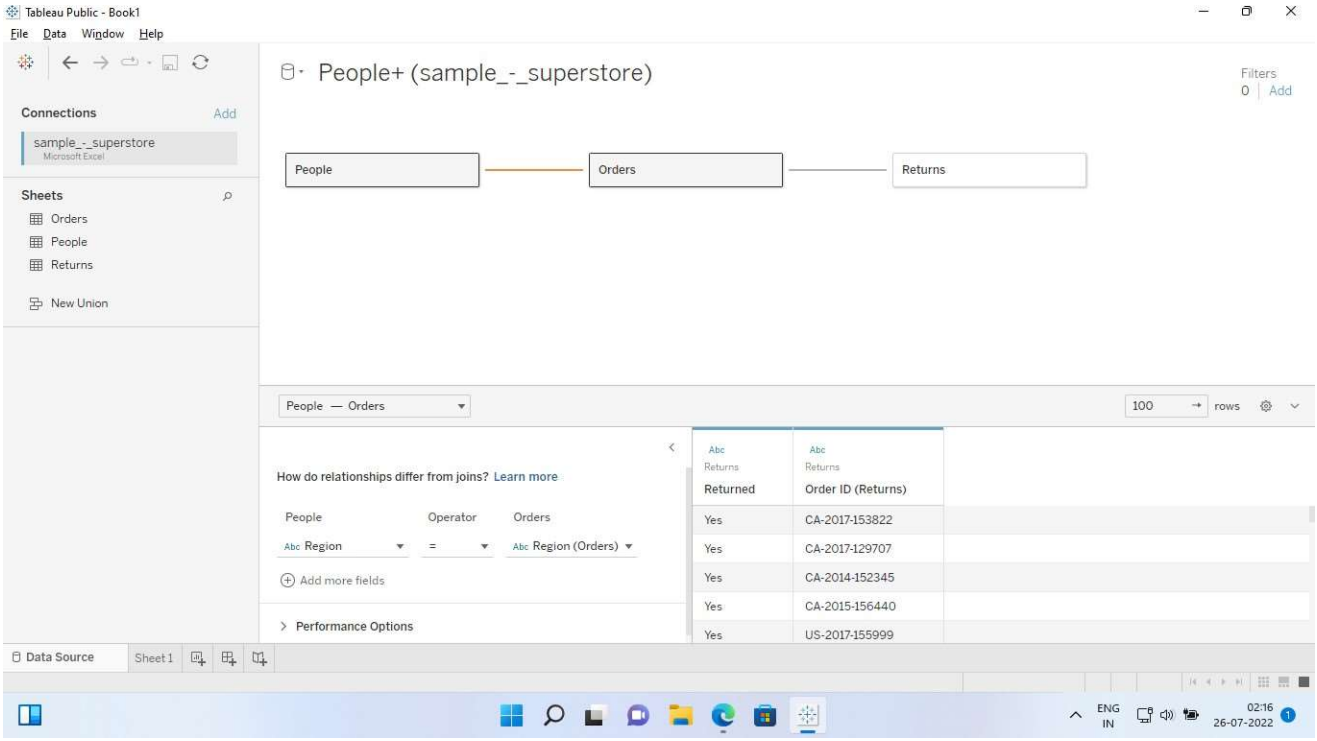


Hierarchical Data Representation:

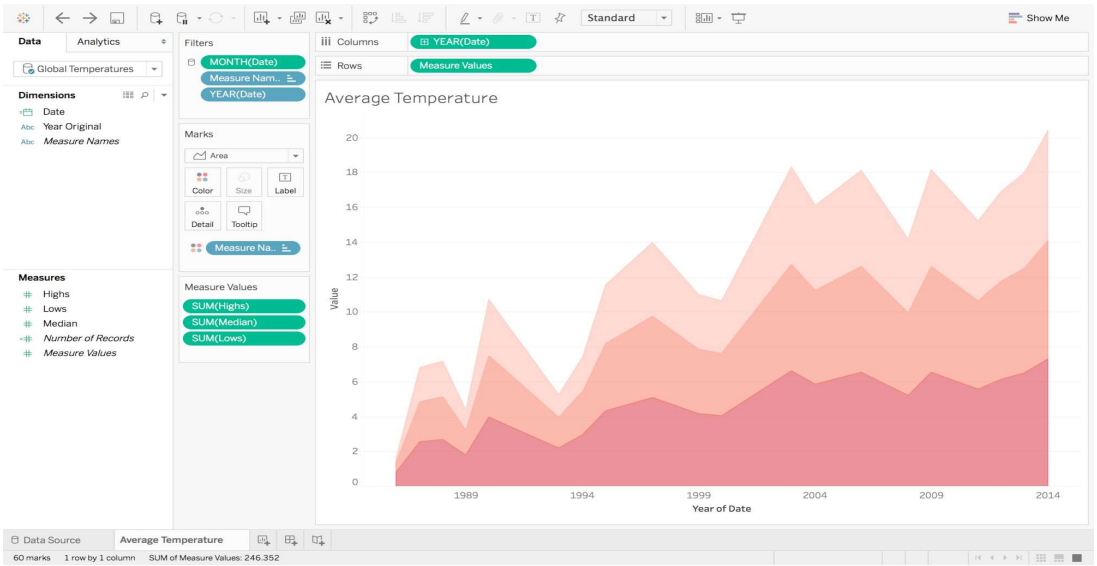
Use treemaps to display data in nested rectangles. You use dimensions to define the structure of the treemap, and measures to define the size or color of the individual rectangles. Treemaps are a relatively simple data visualization that can provide insight in a visually attractive format.



Relational Data Representation:



Temporal Data Representation: Time series analysis is crucial to understanding your data. The ability to look forward and backward, to drill down from years to days and see data trends over different periods of time is essential for the most comprehensive analysis. Tableau's built-in date and time functions let you drag and drop to analyze time trends, drill down with a click, analyze times by day of the week, and easily perform time comparisons like year-over-year growth and moving averages.



Task 3: 2Dexperiments:Bar charts, Clustered bar charts, dot plots, connected dot plots

Aim: To visualize Bar charts, Clustered bar charts, dot plots, connected dot plots

2D- Bar Charts:

Clustered bar charts:

Tableau Clustered Bar Chart Instructions

- First, drag a measure to the Rows shelf
- Second, drag any dimension to the Columns shelf
- Lastly, drag another dimension to the Columns shelf at the top of the screen

Viewing your measure by two dimensions creates the grouping or clustered bar chart view.

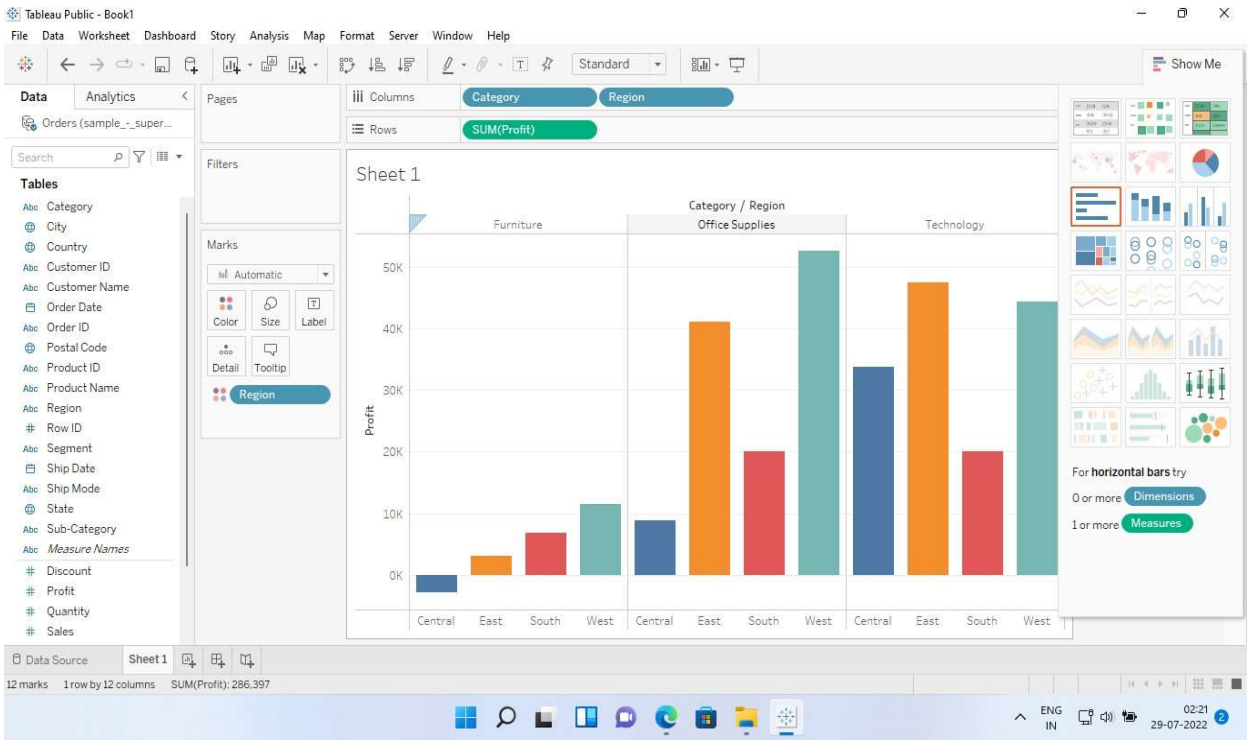
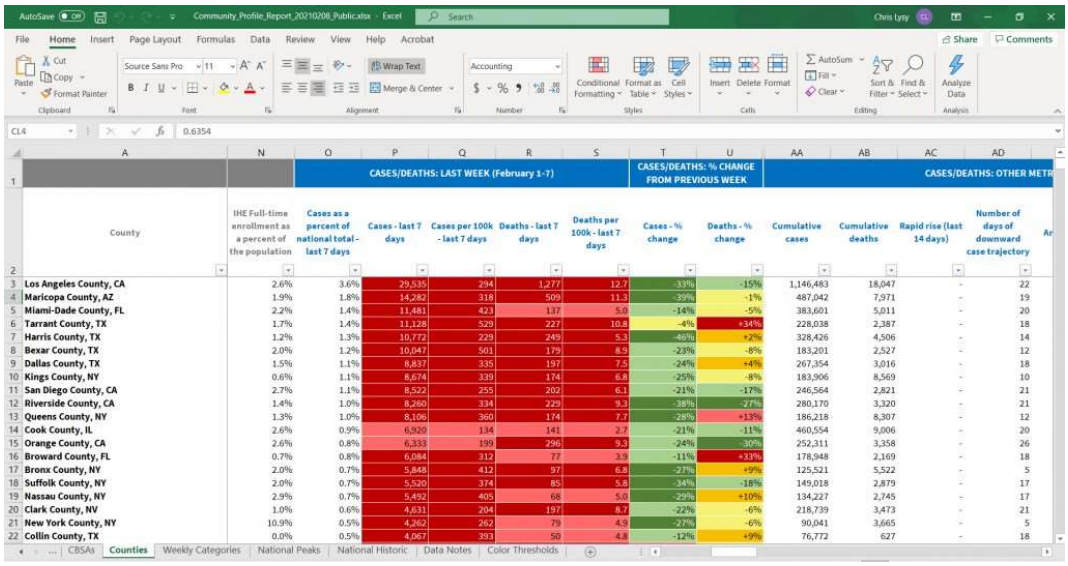
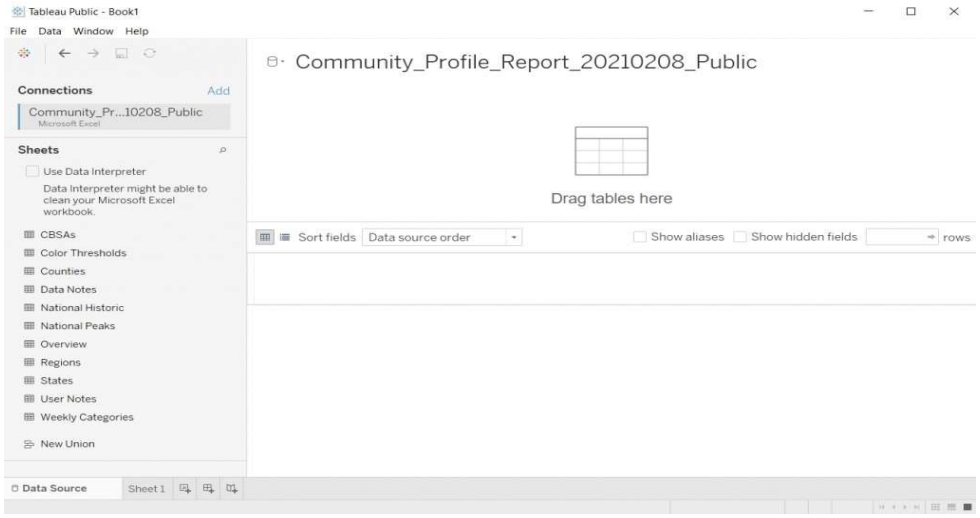


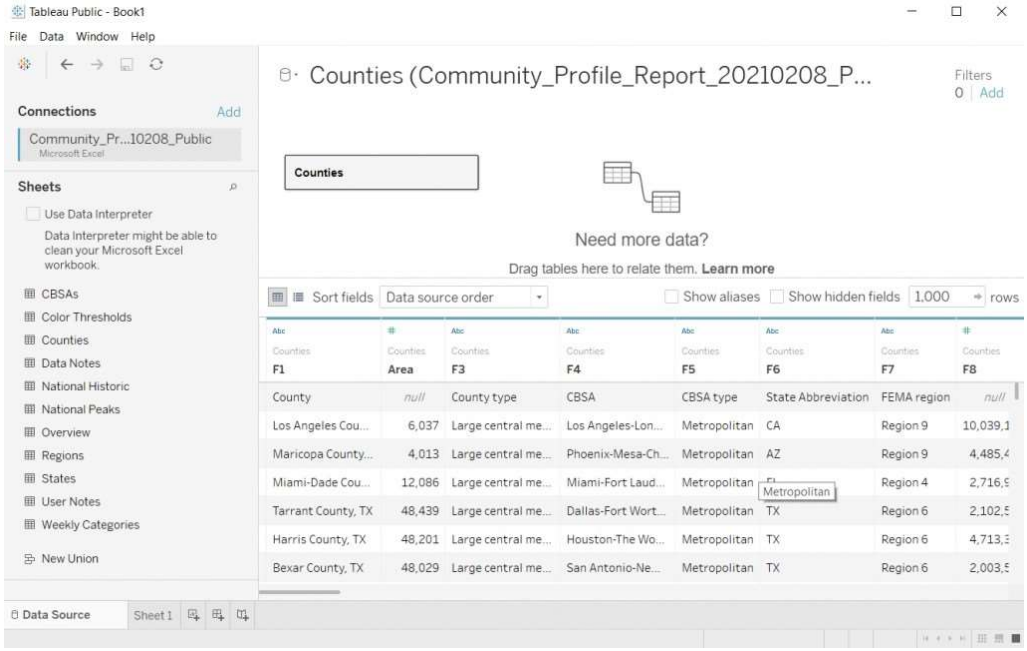
Tableau Public - Book1
File Data Worksheet Dashboard Story Analysis Map Format Server Window Help
Standard
Show Me
Data
Analytics
Pages
Filters
Marks
Automatic
Color
Size
Label
Detail
Tooltip
Region
Columns
Category
Region
Rows
SUM(Profit)
Sheet 1
Category / Region
Furniture
Office Supplies
Technology
Profit
50K
40K
30K
20K
10K
0K
Central
East
South
West
For horizontal bars try
0 or more Dimensions
1 or more Measures
Tables
Orders (sample_-_super...
Abc Category
City
Country
Abc Customer ID
Abc Customer Name
Order Date
Abc Order ID
Postal Code
Abc Product ID
Abc Product Name
Abc Region
Row ID
Abc Segment
Ship Date
Abc Ship Mode
State
Abc Sub-Category
Abc Measure Names
Discount
Profit
Quantity
Sales
Data Source
Sheet 1
12 marks
1 row by 12 columns
SUM(Profit): 286,397
ENG
IN
02:21
29-07-2022

Dot plots:

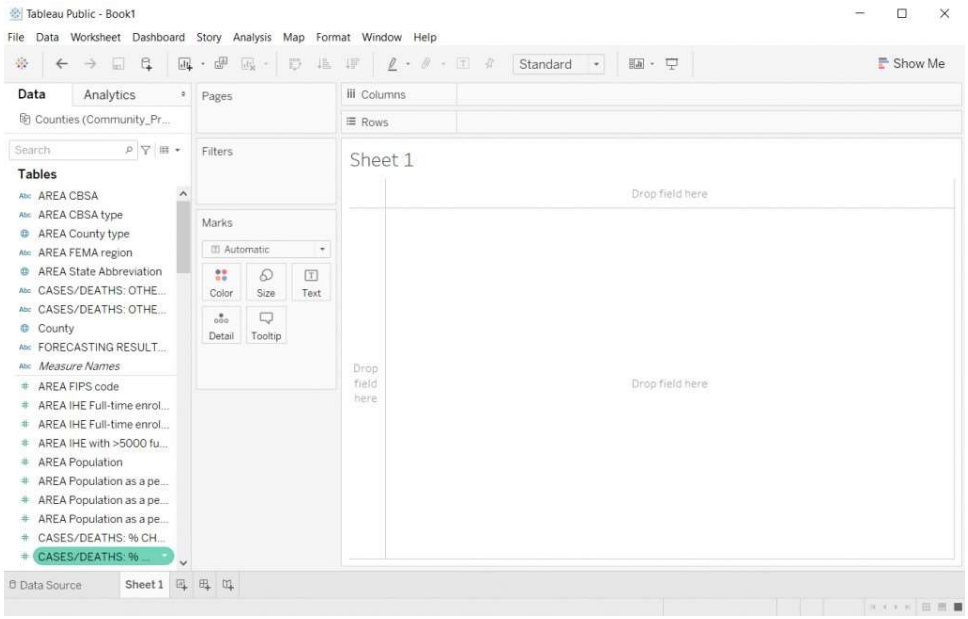

CASES/DEATHS: LAST WEEK (February 1-7)
CASES/DEATHS: % CHANGE FROM PREVIOUS WEEK
CASES/DEATHS: OTHER METR
County
IHE Full-time enrollment as a percent of the population
Cases as a percent of national total - last 7 days
Cases - last 7 days
Cases per 100k - last 7 days
Deaths - last 7 days
Deaths per 100k - last 7 days
Cases - % change
Deaths - % change
Cumulative cases
Cumulative deaths
Rapid rise (last 14 days)
Number of days of downward case trajectory
Los Angeles County, CA 2.6% 3.6% 29,535 294 1,277 12.7 -33% -15% 1,146,483 18,047 22
Maricopa County, AZ 1.9% 1.8% 14,282 318 509 11.3 -39% -1% 487,042 7,971 19
Miami-Dade County, FL 2.2% 1.4% 11,481 423 137 5.0 -14% -5% 383,601 5,011 20
Tarrant County, TX 1.7% 1.4% 11,128 529 227 10.8 -4% +34% 228,038 2,387 18
Harris County, TX 1.2% 1.3% 10,772 229 249 5.3 -46% +2% 328,426 4,506 14
Bexar County, TX 2.0% 1.2% 10,047 501 179 8.9 -23% -8% 183,201 2,527 12
Dallas County, TX 1.5% 1.1% 8,837 335 197 7.5 -24% +4% 267,354 3,016 18
Kings County, NY 0.6% 1.1% 8,674 339 174 6.8 -25% -8% 183,906 8,569 10
San Diego County, CA 2.7% 1.1% 8,522 255 202 6.1 -21% -17% 246,564 2,821 21
Riverside County, CA 1.4% 1.0% 8,260 334 229 9.3 -38% -27% 280,170 3,320 21
Queens County, NY 1.3% 1.0% 8,106 360 174 7.7 -28% +13% 186,218 8,307 12
Cook County, IL 2.6% 0.9% 6,920 134 141 2.7 -21% -11% 460,554 9,006 20
Orange County, CA 2.6% 0.8% 6,333 199 296 9.3 -24% -30% 252,311 3,358 26
Broward County, FL 0.7% 0.8% 6,084 312 77 3.9 -11% +33% 178,948 2,169 18
Bronx County, NY 2.0% 0.7% 5,848 412 97 6.8 -27% +9% 125,521 5,522 5
Suffolk County, NY 2.0% 0.7% 5,520 374 85 5.8 -34% -18% 149,018 2,879 17
Nassau County, NY 2.9% 0.7% 5,492 405 68 5.0 -29% +10% 134,227 2,745 17
Clark County, NV 1.0% 0.6% 4,631 204 197 8.7 -22% -6% 218,739 3,473 21
New York County, NY 10.9% 0.5% 4,262 262 79 4.9 -27% -6% 90,041 3,665 5
Collin County, TX 0.0% 0.5% 4,067 393 50 4.8 -12% +9% 76,772 627 18
CBSAs
Counties
Weekly Categories
National Peaks
National Historic
Data Notes
Color Thresholds

Drag and Drop your Spreadsheet

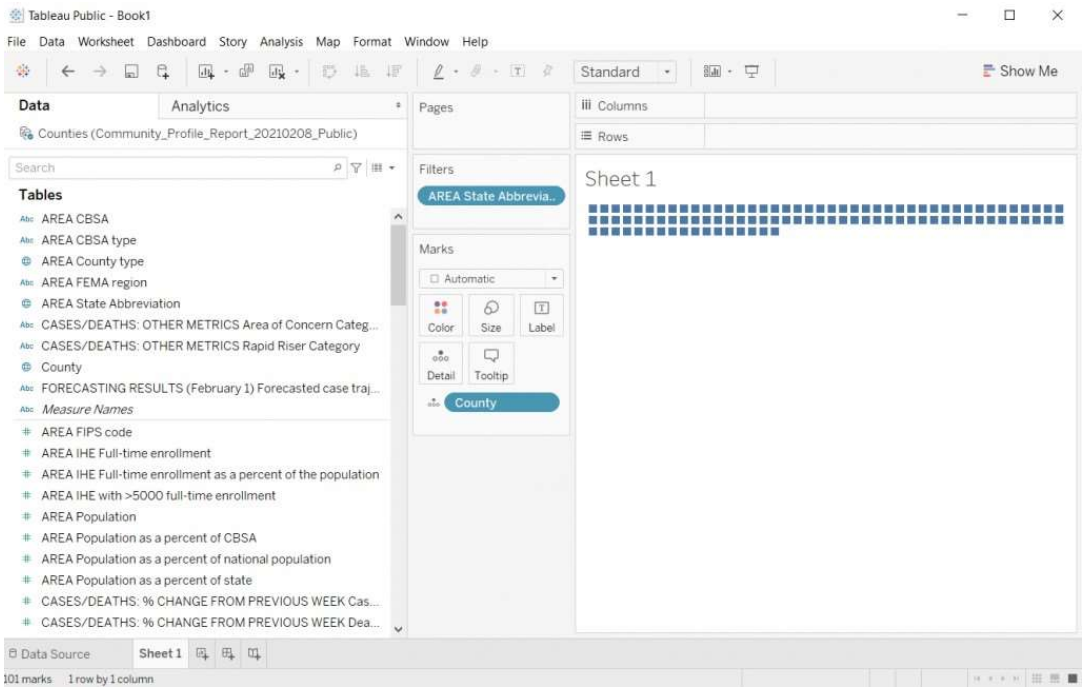




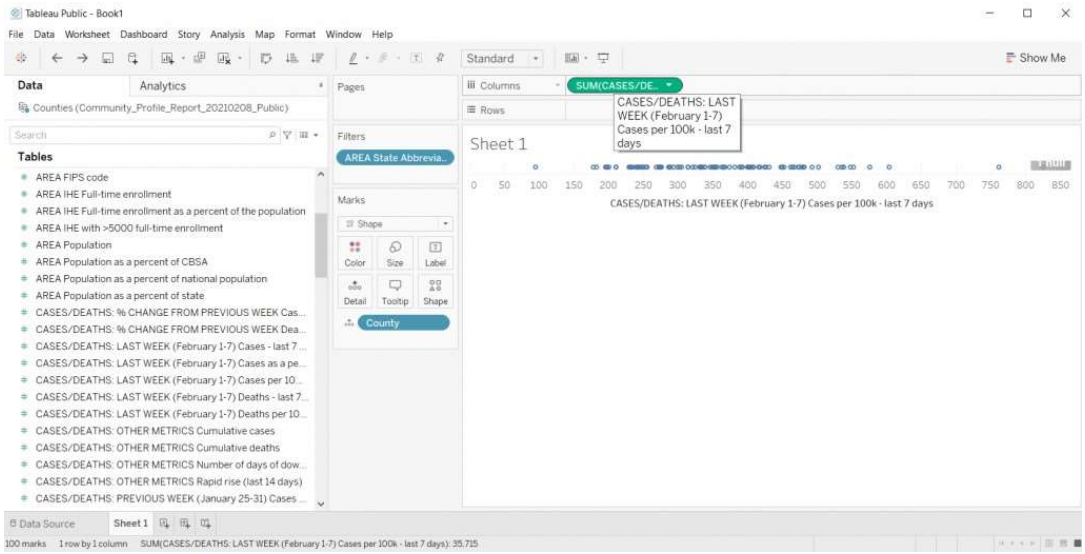
Clicking on Sheet 1



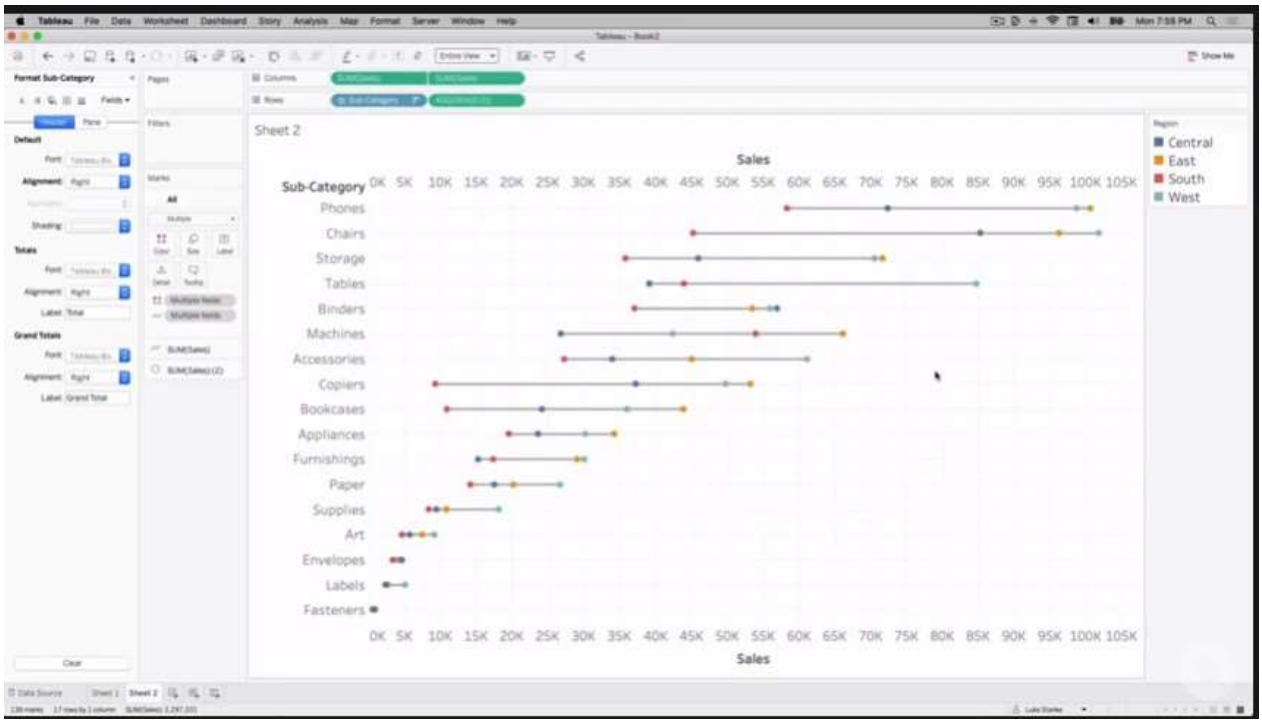
Showing the data



Creating the Dot Plot



Connected dot plots

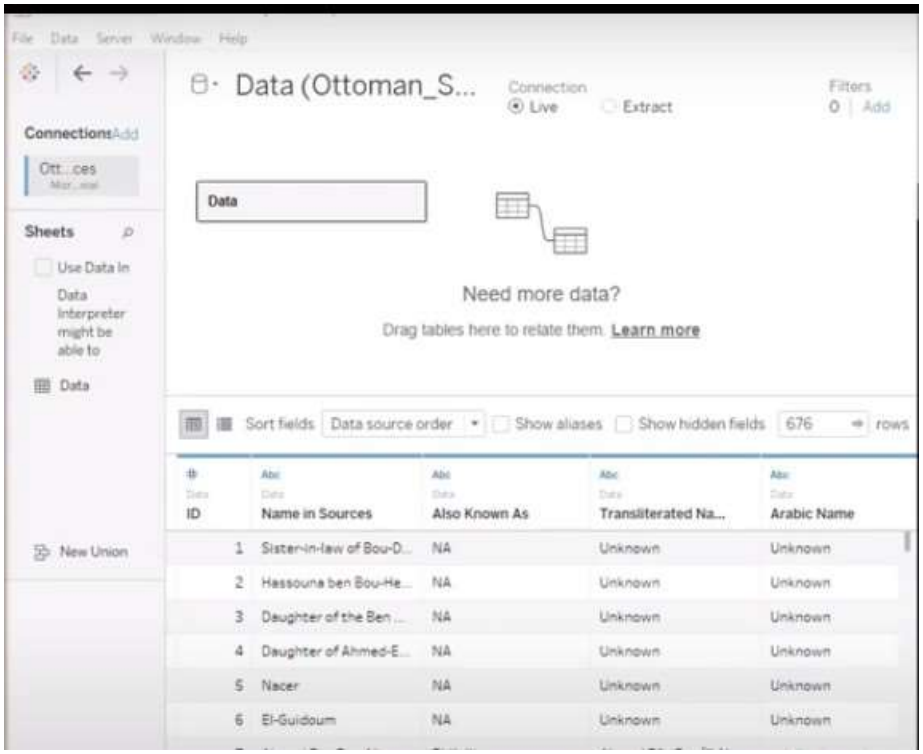


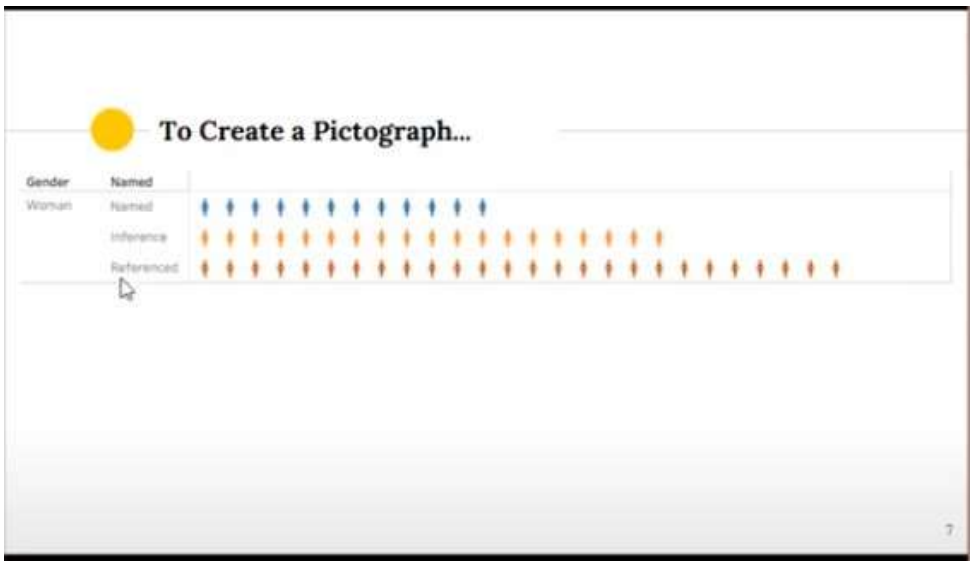
Task -4: pictograms, proportional shape charts, bubble charts, radar charts, polar charts.

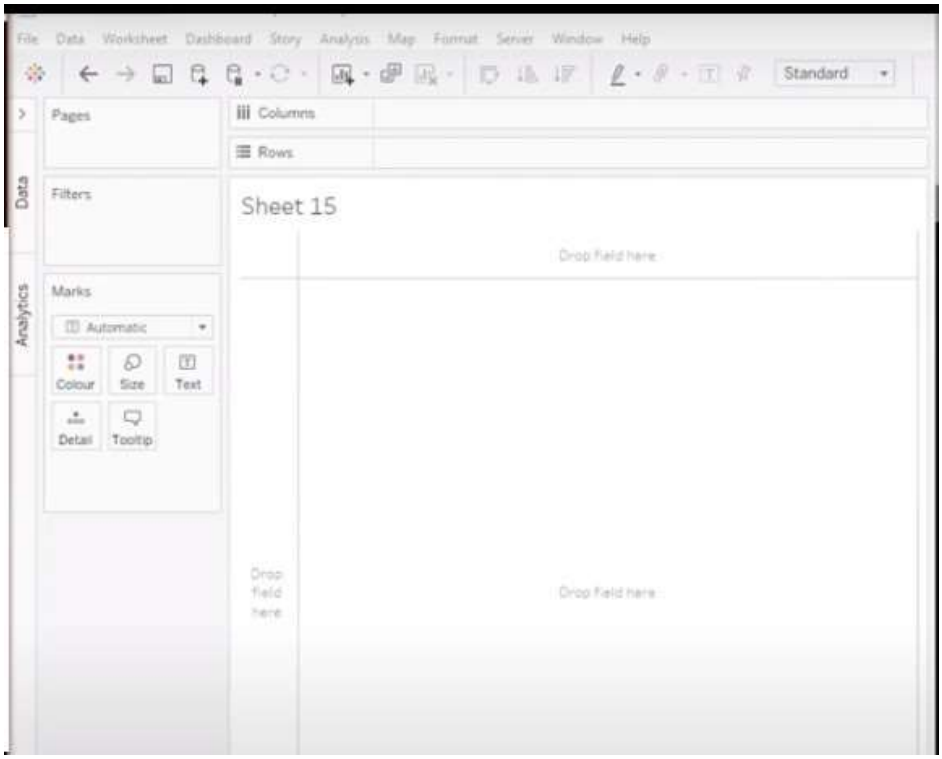
Aim: To visualize pictograms, proportional shape charts, bubble charts, radar charts, polar charts.

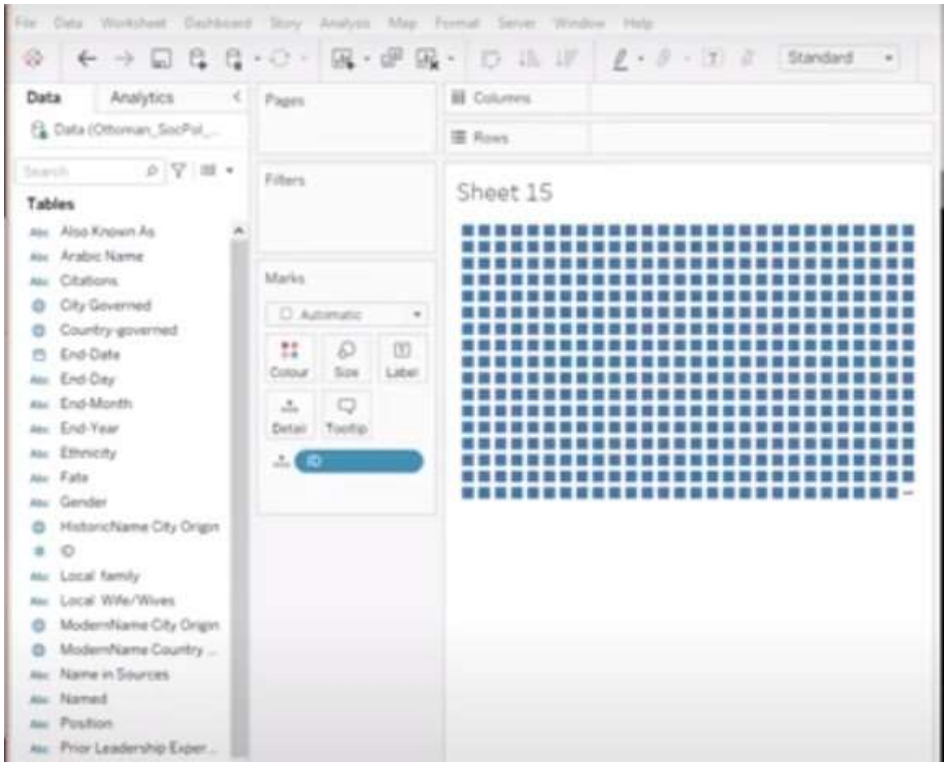
Pictograms

Load Data set





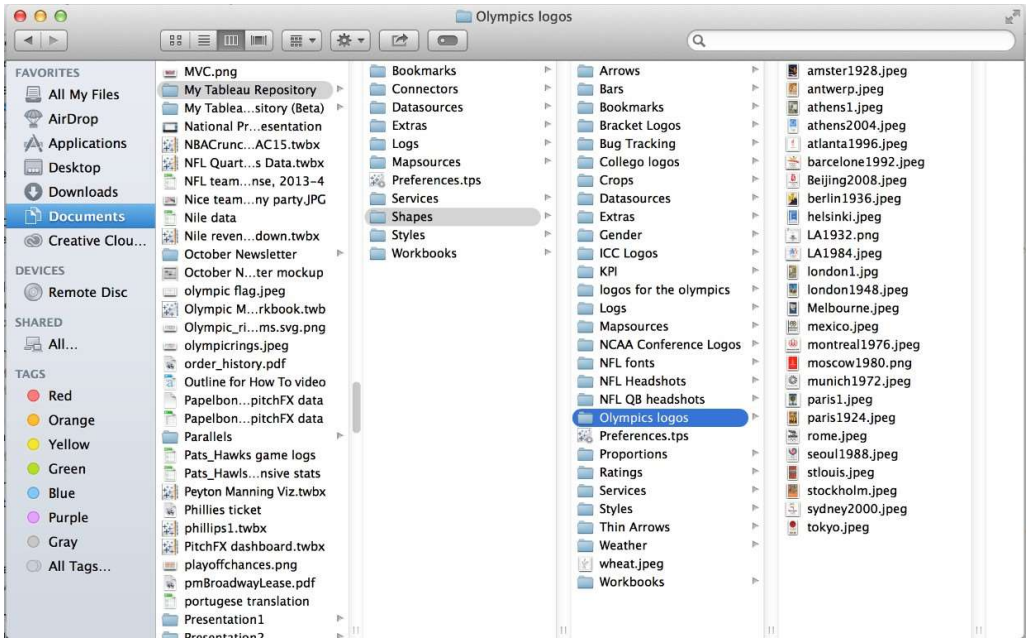

File Data Worksheet Dashboard Story Analysis Map Format Server Window Help
Standard
Pages
Columns
Rows
Data
Analytics
Filters
Sheet 15
Drop field here
Marks
Automatic
Colour
Size
Text
Detail
Tooltip
Drop field here
Drop field here


File Data Worksheet Dashboard Story Analysis Map Format Server Window Help
Standard
Data
Analytics
Pages
Columns
Rows
Filters
Sheet 15
Marks
Automatic
Colour
Size
Label
Detail
Tooltip
Tables
Also Known As
Arabic Name
Citations
City Governed
Country-governed
End-Date
End-Day
End-Month
End-Year
Ethnicity
Fate
Gender
HistoricName City Origin
ID
Local family
Local Wife/Wives
ModernName City Origin
ModernName Country
Name in Sources
Named
Position
Prior Leadership Exper

Proportional shape charts

- Find image file on the internet. For example - use a set of shapes such as logos or flags.
- Download the image to your computer.
- Drag images into your "my Tableau repository" -> "shapes" folder.
- Open Tableau and your new shapes will automatically be included in your "edit shapes" menu.

"my Tableau repository" -> "shapes" -> "Olympics logos"



- Created new worksheet and put the year of the Games field ("edition") on columns. Then change the marks card type from "automatic" -> "shapes" and dropped another copy of "edition" onto "shape" in the marks card.

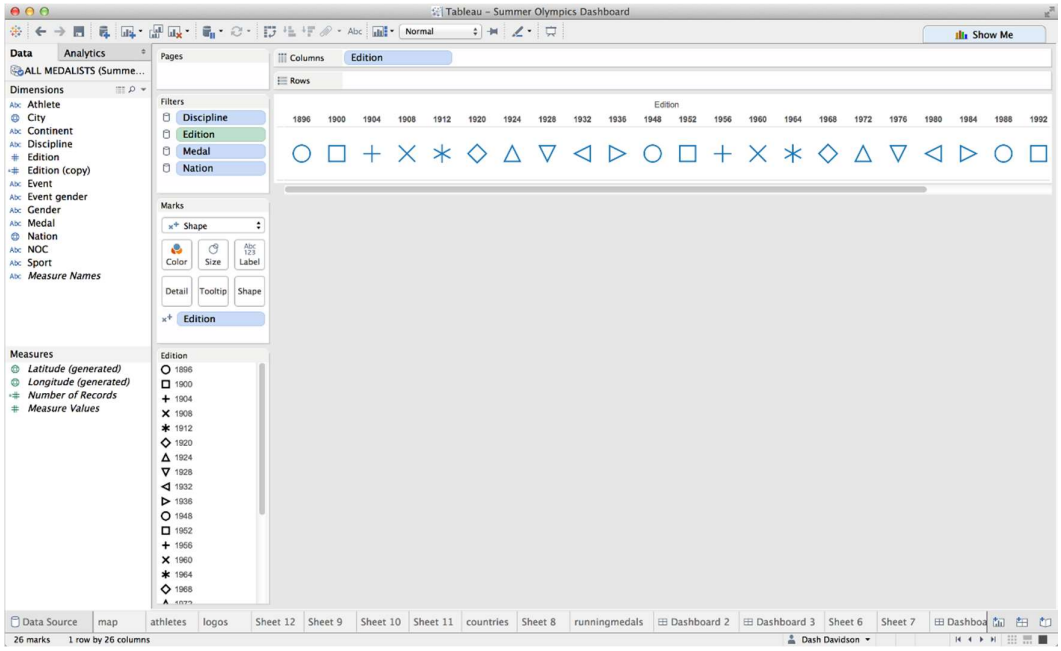
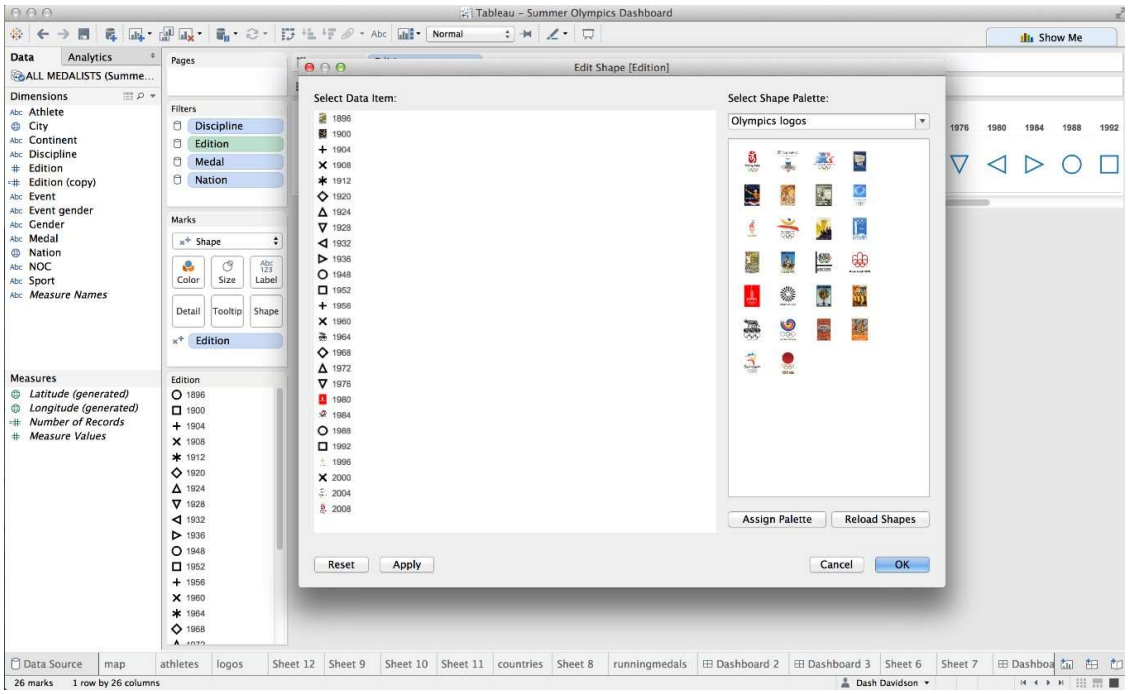
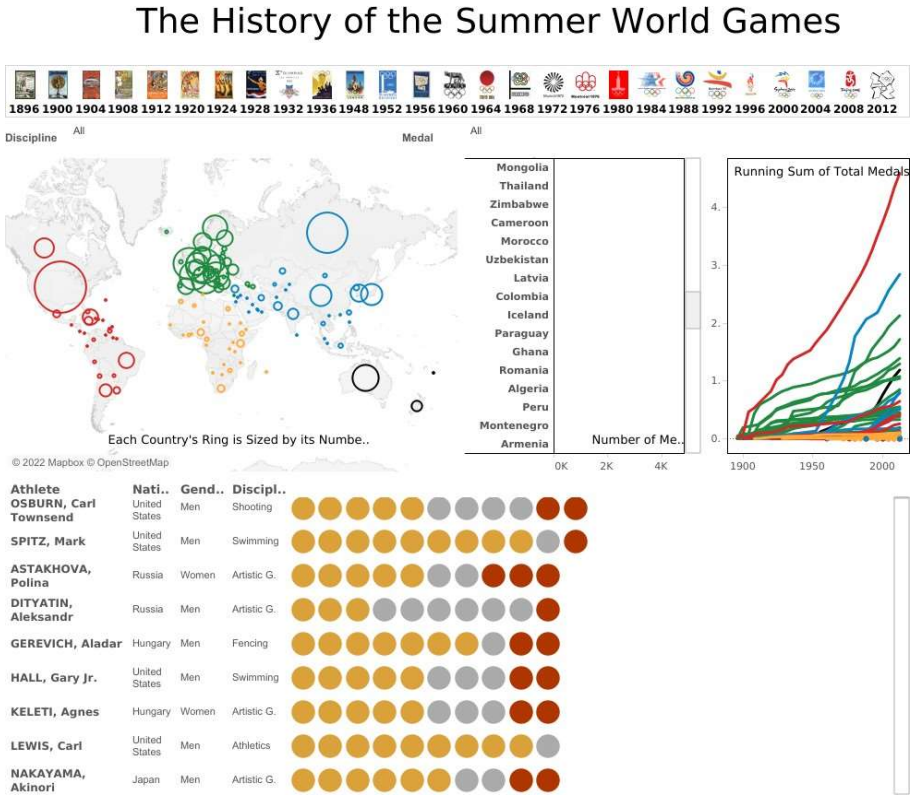


Tableau – Summer Olympics Dashboard
Data
Analytics
ALL MEDALISTS (Summe...
Dimensions
Athlete
City
Continent
Discipline
Edition
Edition (copy)
Event
Event gender
Gender
Medal
Nation
NOC
Sport
Measure Names
Measures
Latitude (generated)
Longitude (generated)
Number of Records
Measure Values
Pages
Filters
Discipline
Edition
Medal
Nation
Marks
Shape
Color
Size
Label
Detail
Tooltip
Shape
Edition
Edition
1896
1900
1904
1908
1912
1920
1924
1928
1932
1936
1948
1952
1956
1960
1964
1968
Columns
Edition
Rows
Edition
1896 1900 1904 1908 1912 1920 1924 1928 1932 1936 1948 1952 1956 1960 1964 1968 1972 1976 1980 1984 1988 1992
Show Me
Data Source
map
athletes
logos
Sheet 12
Sheet 9
Sheet 10
Sheet 11
countries
Sheet 8
runningmedals
Dashboard 2
Dashboard 3
Sheet 6
Sheet 7
26 marks
1 row by 26 columns
Dash Davidson

- Select my new custom shapes folder and match up the editions of the Games with their respective posters. Click on "shape" on the marks card, hit the drop-down "shape palette" menu and switched from "default" to "Olympics logos" and matched up each Games with its poster.





Mark type:	Circle
Detail:	Dimension
Size:	Measure
Color:	Dimension or Measure
Label (optional):	Dimension or Measure

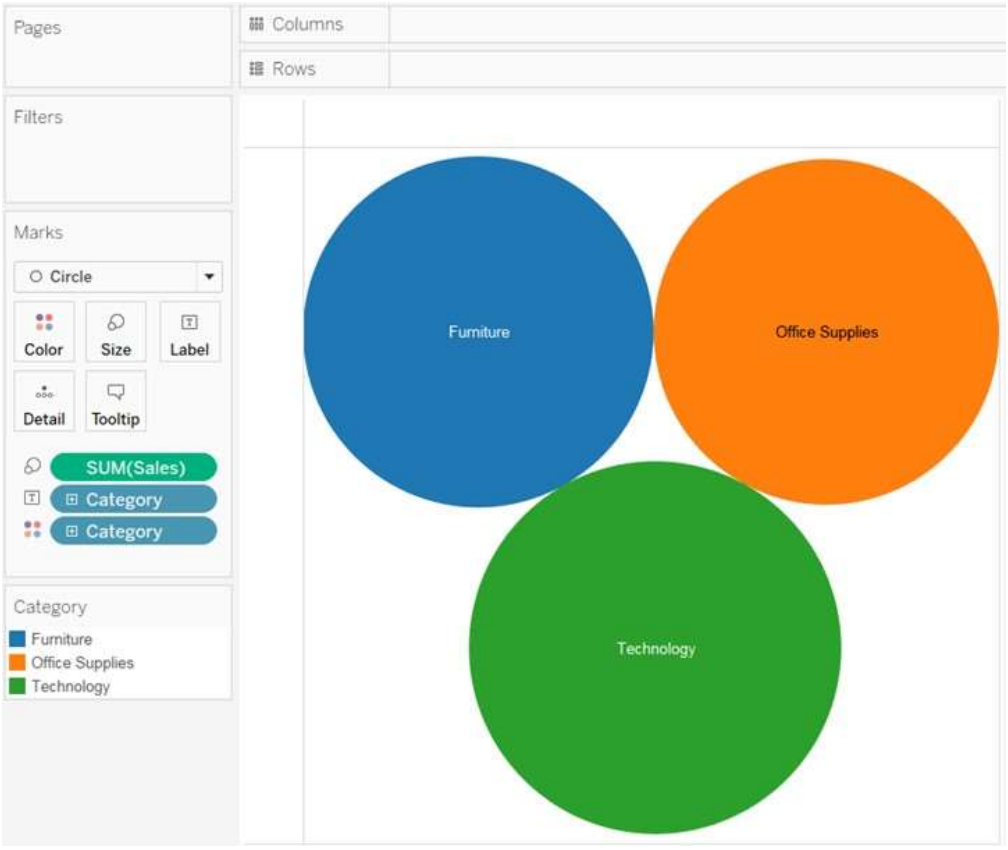
Bubble charts:

Use packed bubble charts to display data in a cluster of circles. Dimensions define the individual bubbles, and measures define the size and color of the individual circles.

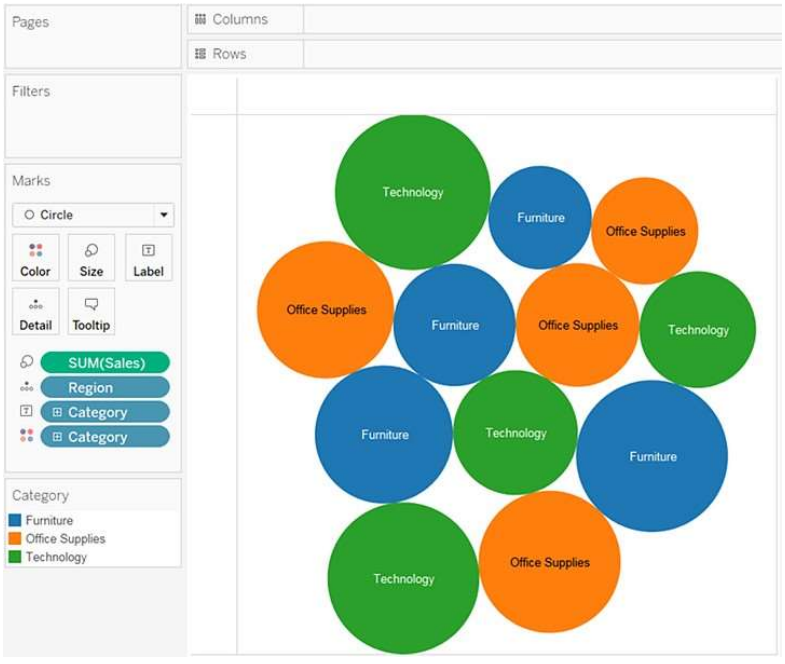
To create a basic packed bubble chart that shows sales and profit information for different product categories, follow these steps:

- Connect to the Sample - Superstore data source.
- Drag the Category dimension to Columns.
- A horizontal axis displays product categories.
- Drag the Sales measure to Rows.
- The measure is aggregated as a sum and a vertical axis appears.
- Tableau displays a bar chart—the default chart type when there is a dimension on the Columns shelf and a measure on the Rows shelf.
- Click Show Me on the toolbar, then select the packed bubbles chart type.

Tableau displays the following packed bubble chart:



Drag Region to Detail on the Marks card to include more bubbles in the view.



Radar Charts:

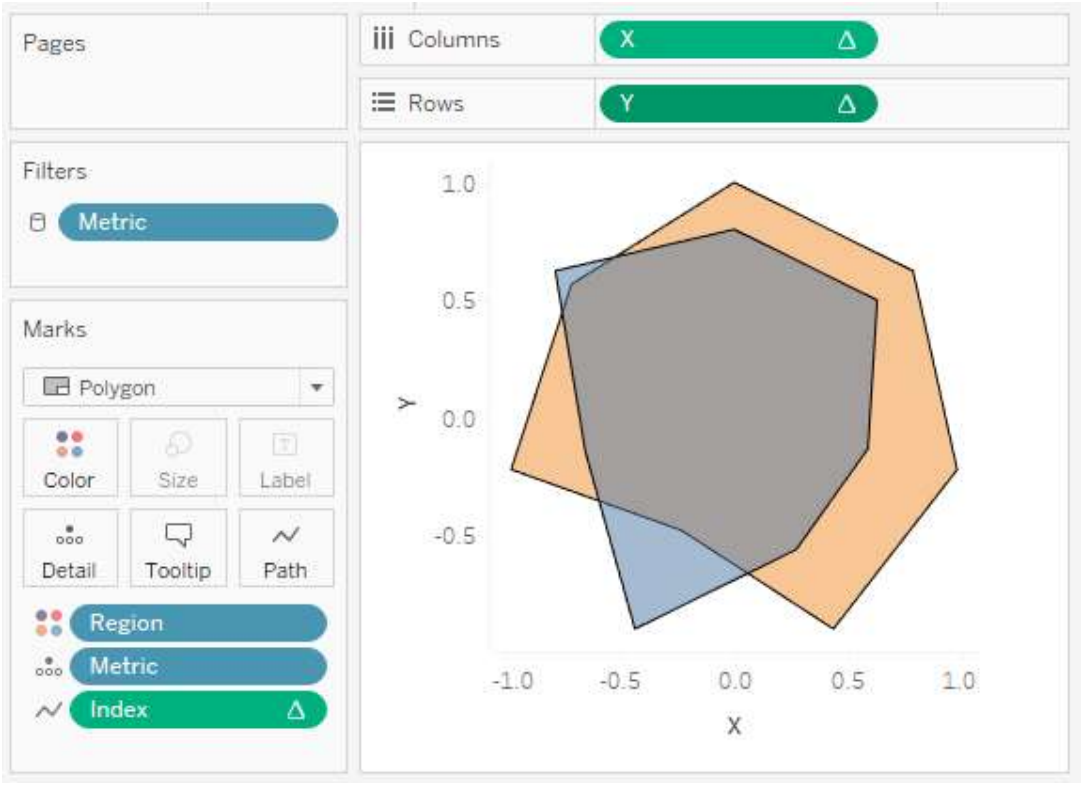
Load the following data into Tableau Public.

Region	Metric	Value
Europe	Sales Revenue	1,500,000
Europe	Operating Cost	500,000
Europe	Profit	1,000,000
Europe	Quantity Sold	80,000
Europe	Average Discount	10
Europe	Total Headcount	200
Europe	Sales Headcount	80
APAC	Sales Revenue	1,000,000
APAC	Operating Cost	400,000
APAC	Profit	600,000
APAC	Quantity Sold	50,000
APAC	Average Discount	8
APAC	Total Headcount	220
APAC	Sales Headcount	150

So to build our first worksheet:

- Drag **Metric** onto the **Filters Card**.
    - Select all Metrics.
    - Right-click on the object, go to **Apply to Worksheets** and select **All Using This Data Source**.
- Change the **Marks Type** to **Polygon**.
- Drag **Region** onto **Color**.

- Drag **Metric** onto **Detail**.
- Drag **Index** onto **Path**.
    - Right-click on the object, go to **Compute Using** and select **Metric**.
- Drag **X** onto **Columns**.
    - Right-click on the object and go to **Edit Table Calculations**.
    - In Nested Calculations choose **Index**.
        - In **Compute Using** select **Specific Dimensions**.
        - Ensure that only **Metric** is checked.
    - In Nested Calculations choose **TC_Metric Count**.
        - In **Compute Using** select **Specific Dimensions**.
        - Ensure that only **Metric** is checked.
    - In Nested Calculations choose **TC_Max Value**.
        - In **Compute Using** select **Specific Dimensions**.
        - Ensure that **Metric** and **Region** are both checked and ensure that **Metric is on top**.
        - In **Restarting Every** select **Metric**.
- Drag **Y** onto **Rows**.
    - Right-click on the object and go to **Edit Table Calculations**.
    - In Nested Calculations choose **Index**.
        - In **Compute Using** select **Specific Dimensions**.
        - Ensure that only **Metric** is checked.
    - In Nested Calculations choose **TC_Metric Count**.
        - In **Compute Using** select **Specific Dimensions**.
        - Ensure that only **Metric** is checked.
    - In Nested Calculations choose **TC_Max Value**.
        - In **Compute Using** select **Specific Dimensions**.
        - Ensure that **Metric** and **Region** are both checked and ensure that **Metric is on top**.
        - In **Restarting Every** select **Metric**.
- Click on **Color** and change the **Opacity** to **50%**.



Polar Charts:

Load the data-For example - Order data

Calculate the fields

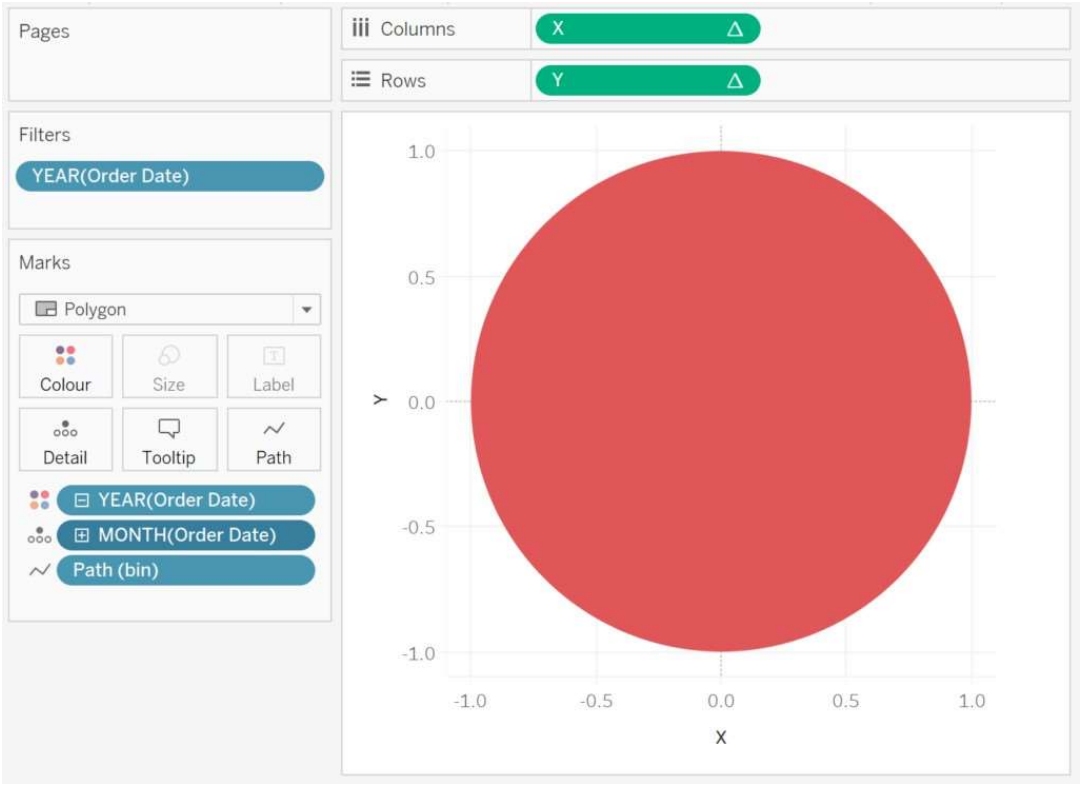
Create a worksheet

Worksheet

We will now build our worksheet:

- Change the **Mark Type** to **Polygon**
- Drag **Order Date** onto the **Filters Shelf**, and filter to **2015** and **2016**
- Drag **Order Date** onto the **Colour Mark**; right-click on this pill and make sure that this is a **Discrete Year**
- Drag **Order Date** onto the **Detail Mark**; right-click on this pill and make sure it is a **Discrete Month**
- Drag **Path (bin)** onto the **Columns Shelf**
    - Right-click on this pill and ensure that **Show Missing Values** is select
    - Drag this pill onto the **Path Mark**

- Drag **X** onto the **Columns Shelf**
    - Right-click on this pill, go to **Compute Using** and select **Path (bin)**
- Drag **Y** onto the **Rows Shelf**
    - Right-click on this pill, go to **Compute Using** and select **Path (bin)**

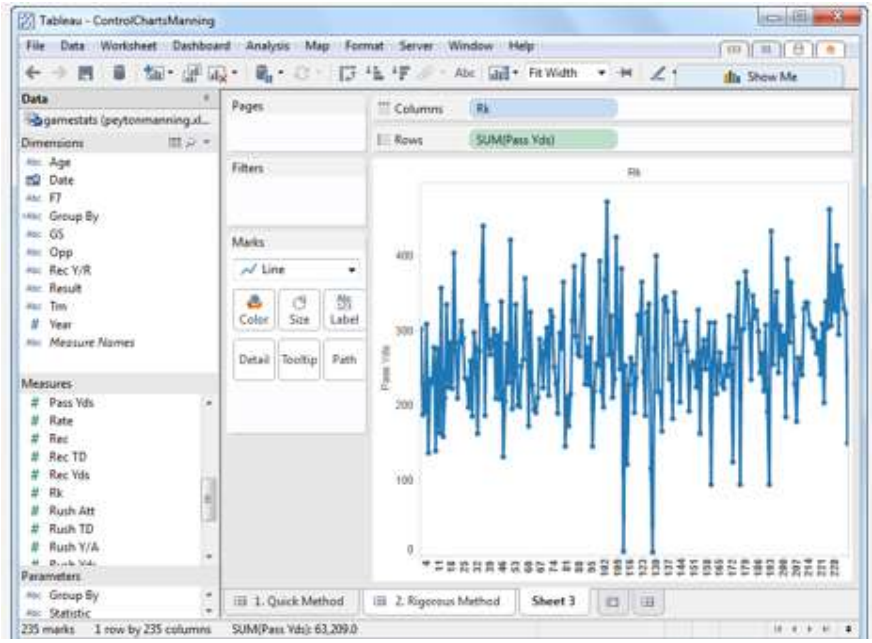


Task 5: 2 D: Range chart, Box-and- whisker plots, univariate scatter plots, histograms word cloud.

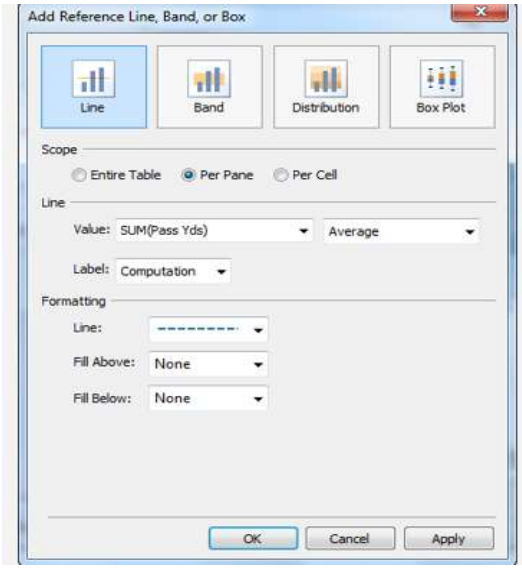
Aim : To Visualize Range chart, Box-and- whisker plots, univariate scatter plots, histograms, word cloud

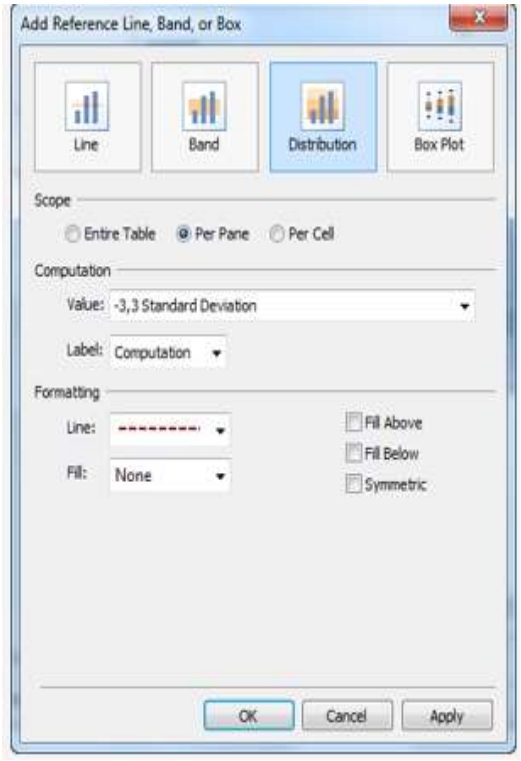
Range Charts:

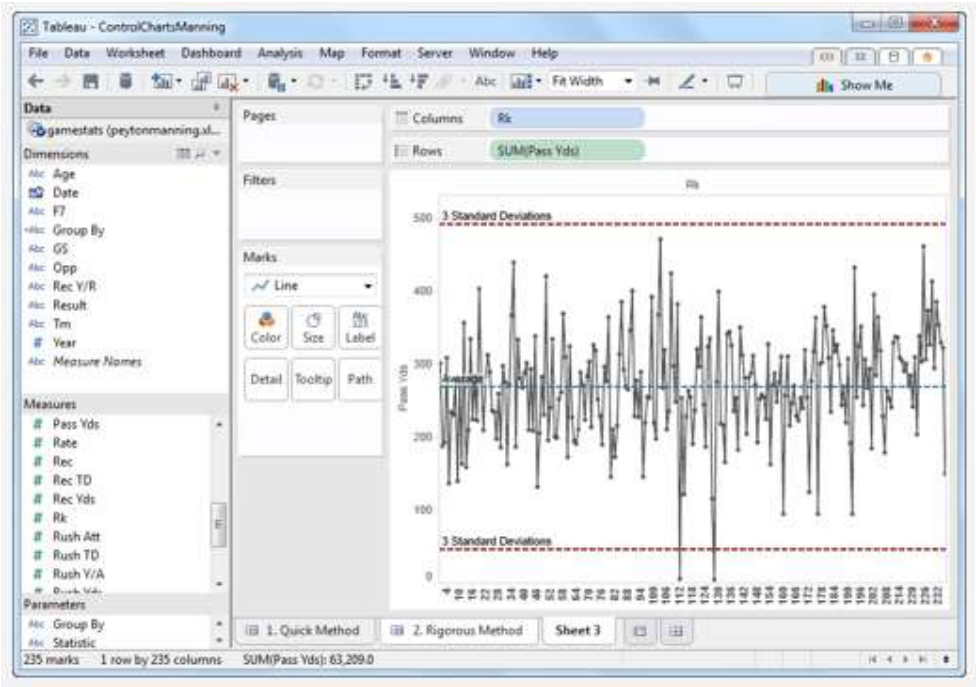
- How-to Create a Control Chart in Tableau
  - It's done using two different methods - the "quick method" and the "rigorous method". The difference between the two is how the control limits are calculated. The "quick method" uses what's called a "global measure of dispersion", or the standard deviation of all of the points. The "rigorous method" uses a "local measure of dispersion" called sigma(x), which is derived from the differences between successive data points.
  - Quick Method
    Step 1: Create a simple timeline with a discrete (blue pill) version of the game number (Rk) on the Columns shelf, and SUM(Pass Yards) field on the Rows shelf, fit to width



Step 2: Right-click on the y-axis, select "Add Reference Line" and add an average line byfilling out the resulting dialog box as shown below, clicking OK when you're done:

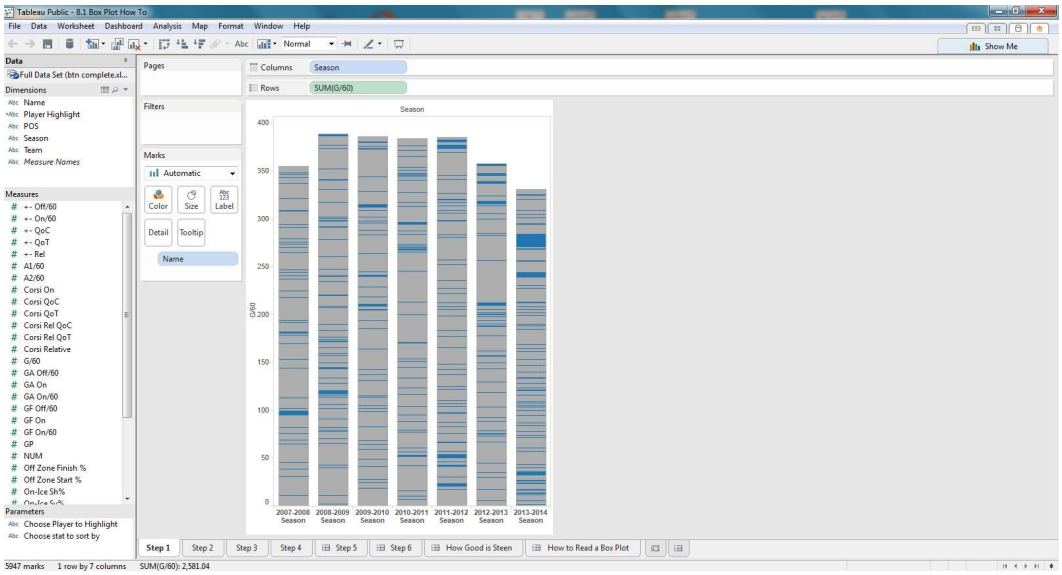

Add Reference Line, Band, or Box
Line
Band
Distribution
Box Plot
Scope
Entire Table
Per Pane
Per Cell
Line
Value:
SUM(Pass Yds)
Average
Label:
Computation
Formatting
Line:
Fill Above:
None
Fill Below:
None
OK
Cancel
Apply


Add Reference Line, Band, or Box
Line
Band
Distribution
Box Plot
Scope
Entire Table
Per Pane
Per Cell
Computation
Value:
-3,3 Standard Deviation
Label:
Computation
Formatting
Line:
Fill:
None
Fill Above
Fill Below
Symmetric
OK
Cancel
Apply



Box and Whisker plots:

- Box-and-whisker plots are typically used to demonstrate distribution of observations.



- Click on “Show Me” and click on the box plot down in the lower right corner and voilà!

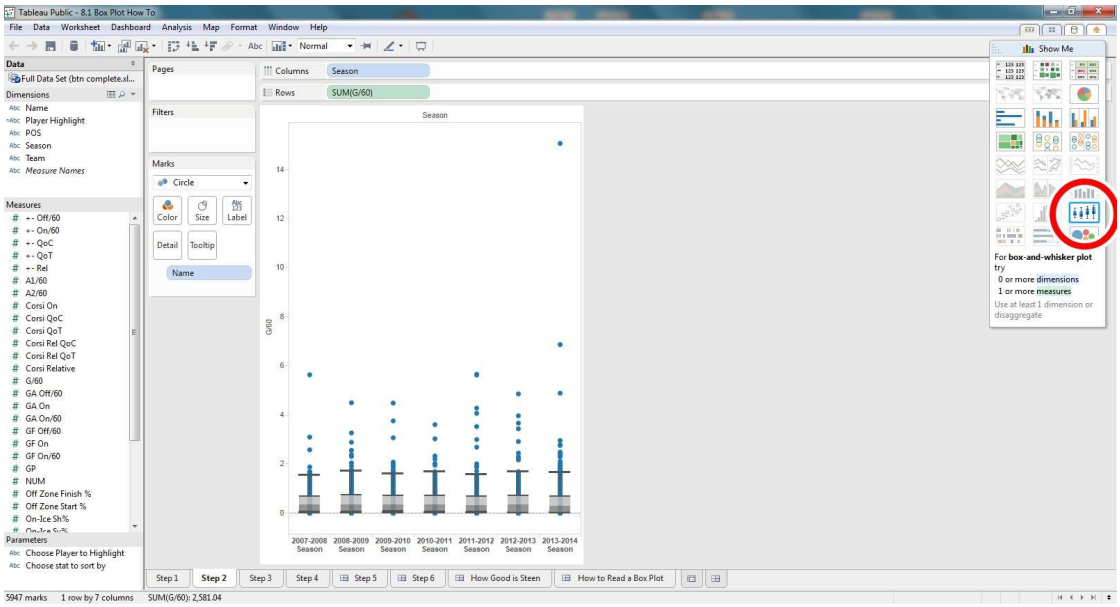


Tableau Public - 8.1 Box Plot How To
Columns
Season
Rows
SUM(G/60)
Show Me
For box-and-whisker plot
try
0 or more dimensions
1 or more measures
Use at least 1 dimension or disaggregate
Step 1
Step 2
Step 3
Step 4
Step 5
Step 6
How Good is Steen
How to Read a Box Plot

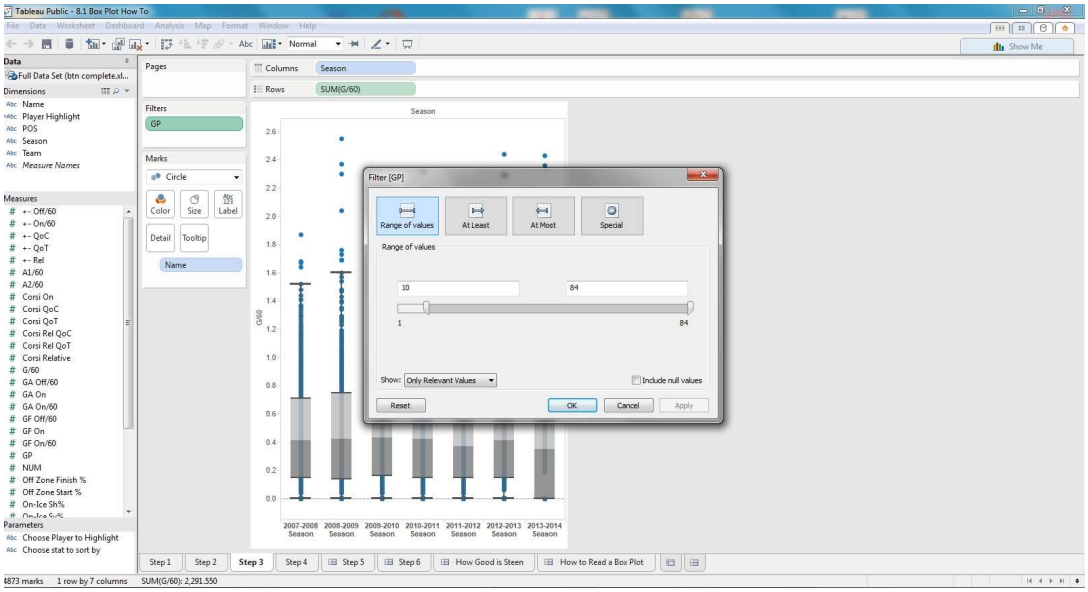
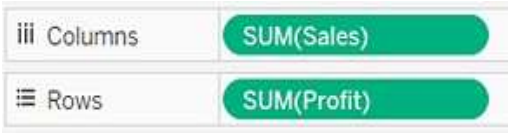


Tableau Public - 8.1 Box Plot How To
Filter [GP]
Range of values
At Least
At Most
Special
Reset
OK
Cancel
Apply
Step 1
Step 2
Step 3
Step 4
Step 5
Step 6
How Good is Steen
How to Read a Box Plot

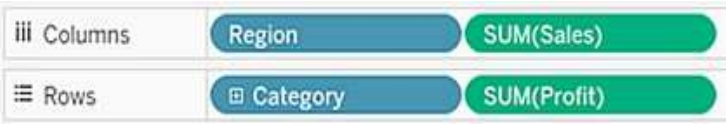
Univariate Scatter plots:

- Use scatter plots to visualize relationships between numerical variables.
- In Tableau, you create a scatter plot by placing at least one measure on the Columns shelf and at least one measure on the Rows shelf. If these shelves contain both dimensions and measures, Tableau places the measures as the innermost fields, which means that measures are always to the right of any dimensions that you have also placed on these shelves. The word "innermost" in this case refers to the table structure.

Creates Simple Scatter Plot



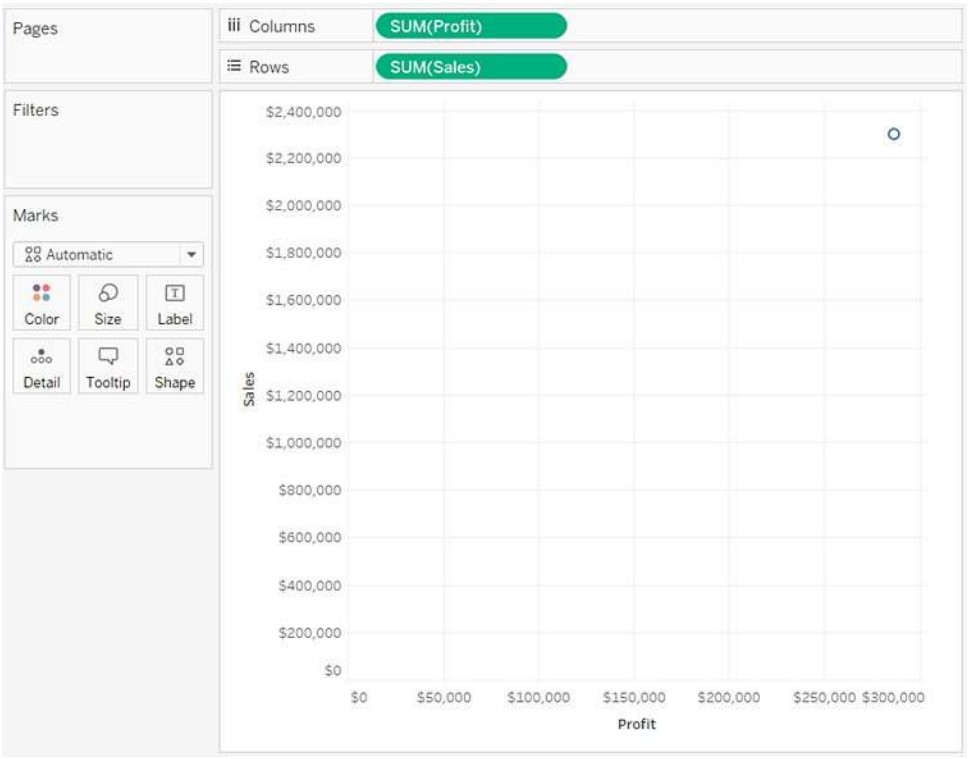
Creates Matrix of Scatter Plots



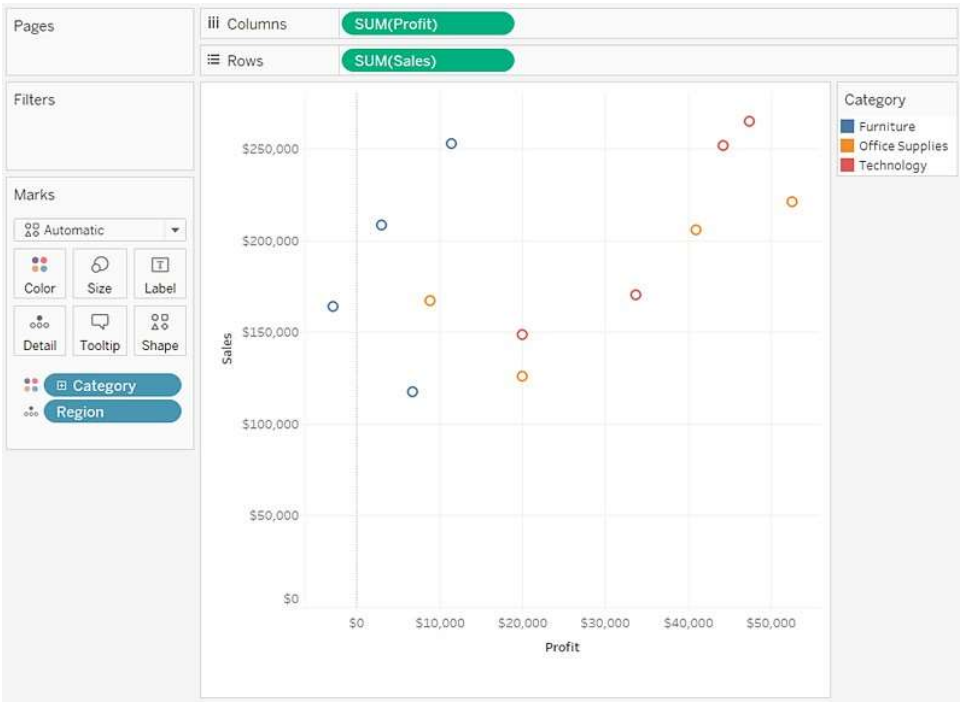
To use scatter plots and trend lines to compare sales to profit, follow these steps:

1. Open the **Sample - Superstore** data source.

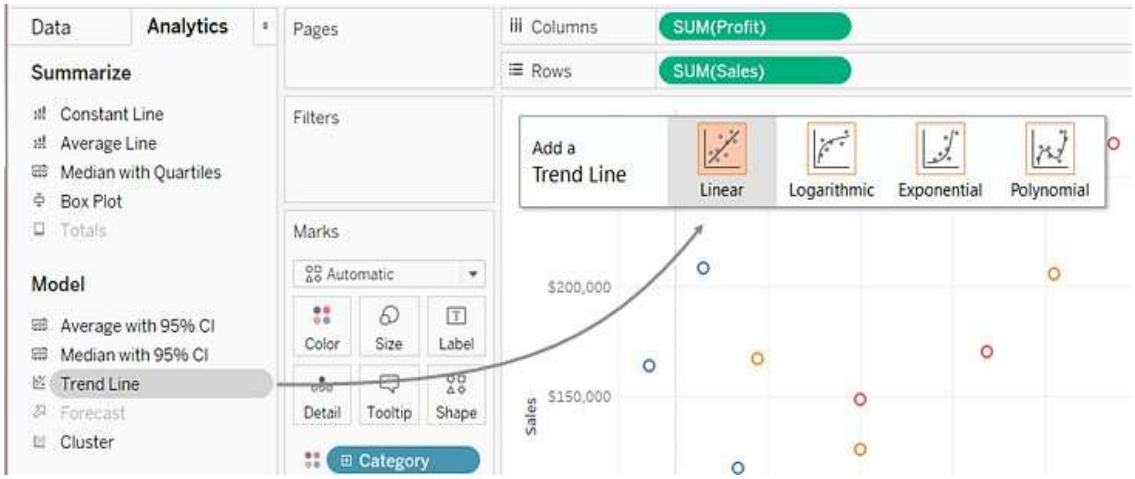
2. Drag the **Profit** measure to **Columns**.
   Tableau aggregates the measure as a sum and creates a horizontal axis.
3. Drag the **Sales** measure to **Rows**.
   Tableau aggregates the measure as a sum and creates a vertical axis.
   Measures can consist of continuous numerical data. When you plot one number against another, youare comparing two numbers; the resulting chart is analogous to a Cartesian chart, with x and y coordinates.
   Now you have a one-mark scatter plot:



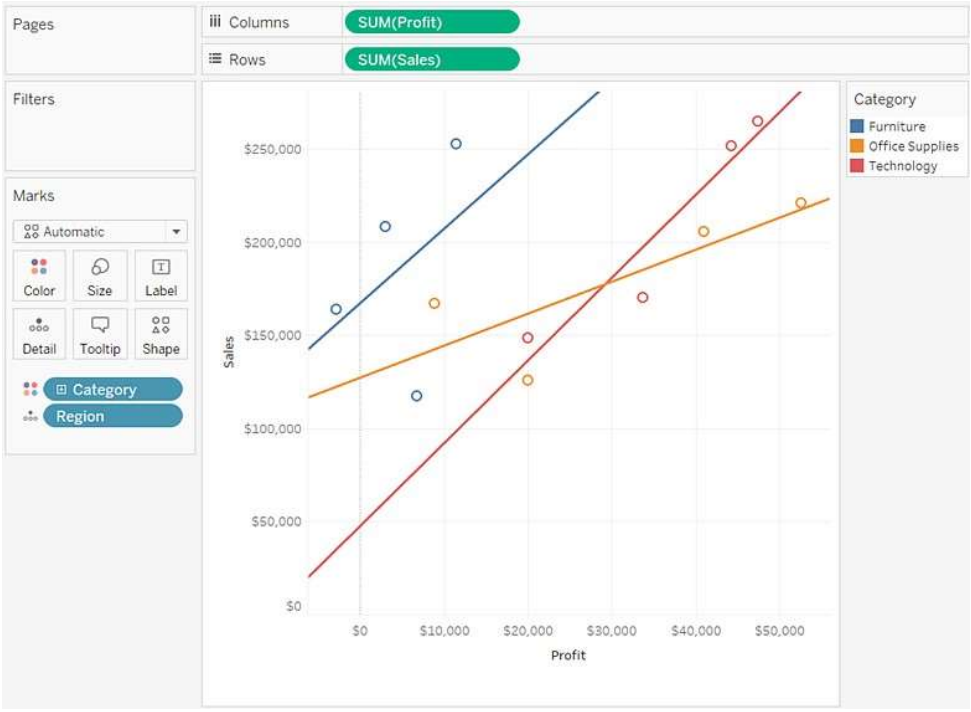
1. Drag the **Region** dimension to **Detail** on the **Marks** card.



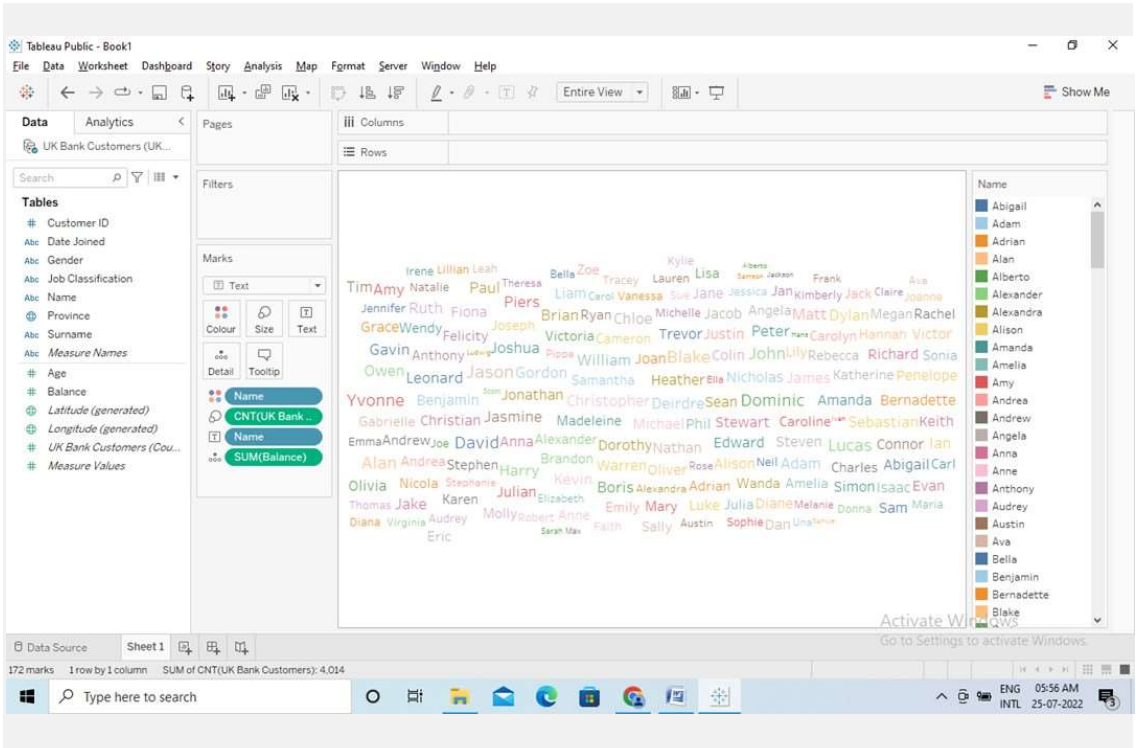
1. To add trend lines, from the **Analytics** pane, drag the **Trend Line** model to the view, and then drop iton the model type.



A trend line can provide a statistical definition of the relationship between two numerical values. Toadd trend lines to a view, both axes must contain a field that can be interpreted as a number—by definition, that is always the case with a scatter plot. Tableau adds three linear trend lines—one for each color that you are using to distinguish the three categories.



Histograms word Cloud:



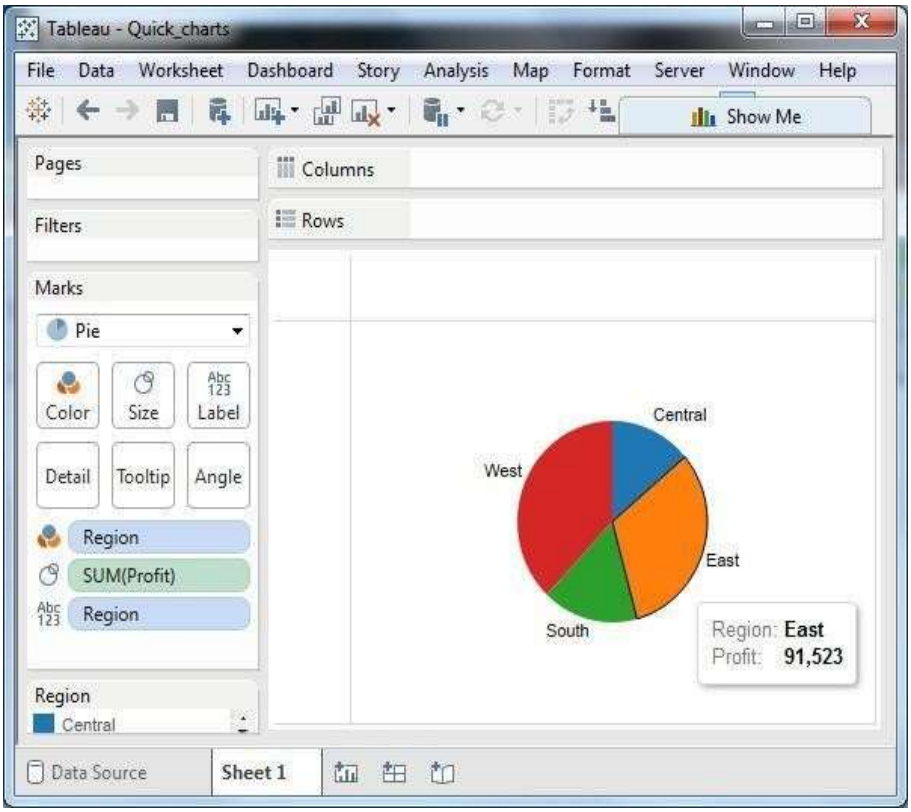
Task 6: 2 D: Pie chart, waffle chart, stacked bar chart, back-to-back bar chart, tree map .

Aim: To visualize 2D data using Pie chart, waffle chart, stacked bar chart, back-to-back bar chart, tree map .

Pie Chart:

A pie chart represents data as slices of a circle with different sizes and colors. The slices are labeled and thenumbers corresponding to each slice is also represented in the chart. You can select the pie chart option from the Marks card to create a pie chart.

Choose one dimension and one measure to create a simple pie chart. For example, take the dimension named region with the measure named profit. Drop the Region dimension in the colors and label marks. Drop the Profit measure into the size mark. Choose the chart type as Pie. The following chart appears which shows the 4 regions in different colors.



Waffle Chart:

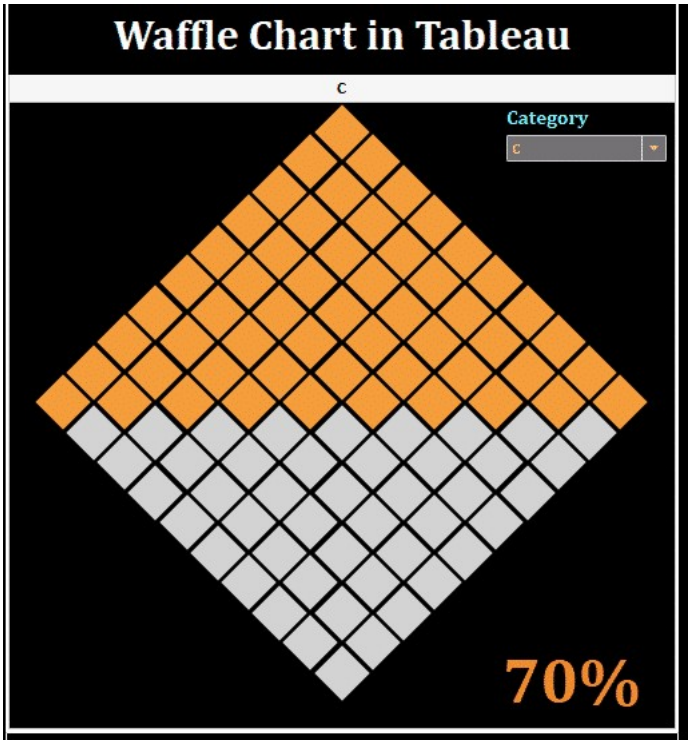
To create a Waffle chart that shows % value by category, follow these steps:
1. Drag **"Path"** on Detail marks. (*Path column has Value 1- 100)
2. Drag the **Y** Measure to Columns and compute using **"Path"**

3. Drag the **X** Measure to Rows and compute using **"Path"** . Double Click on **X** Axis dialog box, on theGeneral tab, Select **Reversed** option to reverse the order of values on the axis

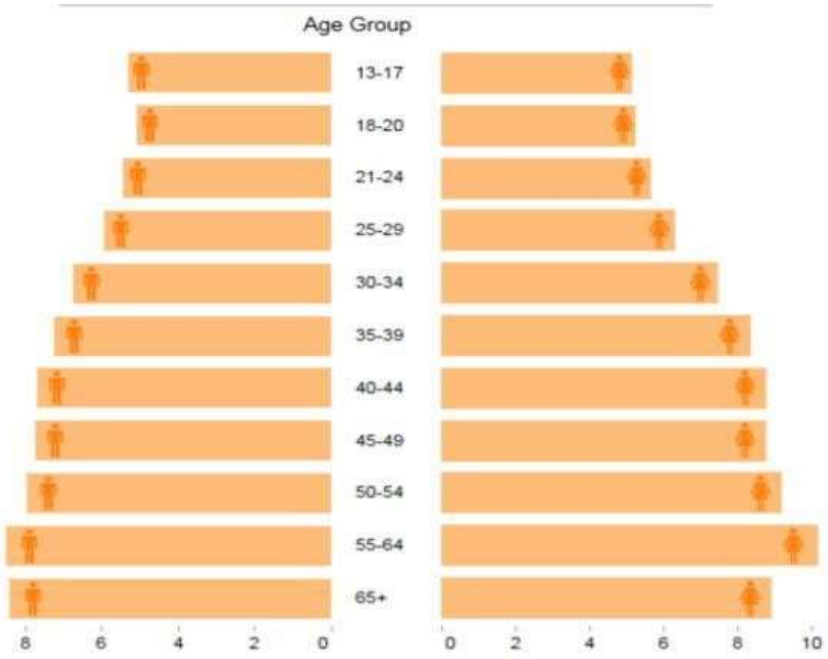
4. Drag the **Color** Boolean Field onto **Color Marks** and compute using **"Path"**

5. Add **Value** on **Text Shelf** and **Category** on **Filters**

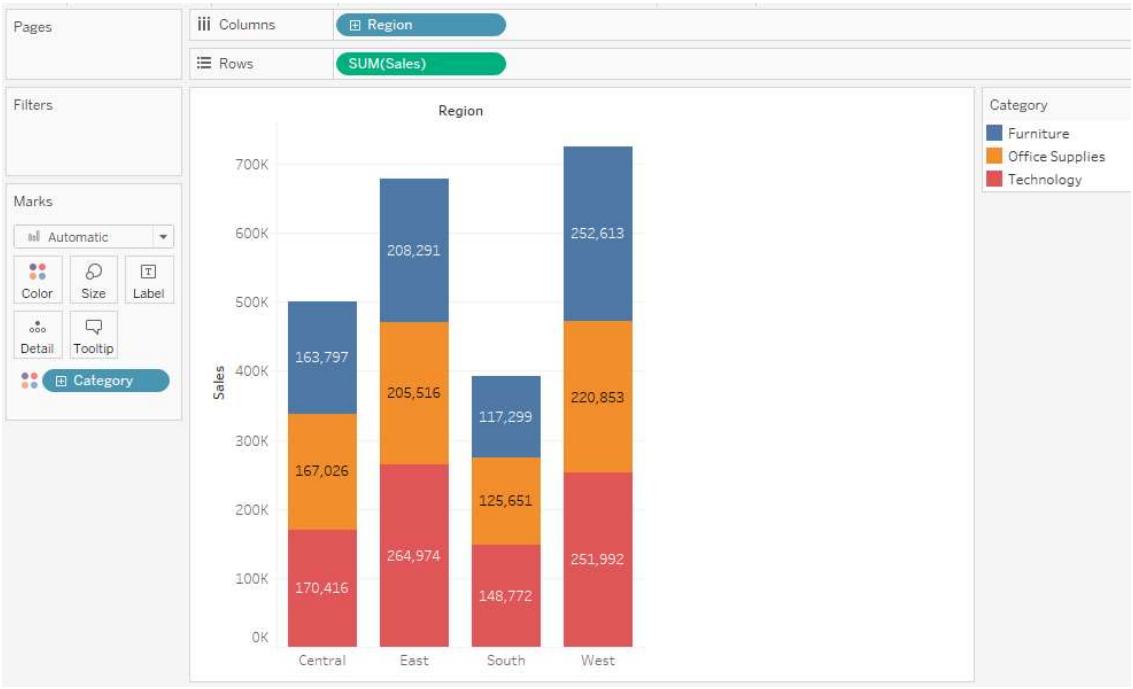
6. Change the Marks to Shape and select your desired filled Shape like" ■ "," ● "," ◆ " etc7.Add some formatting by specifying the font, style, size, and color as per your Choice.



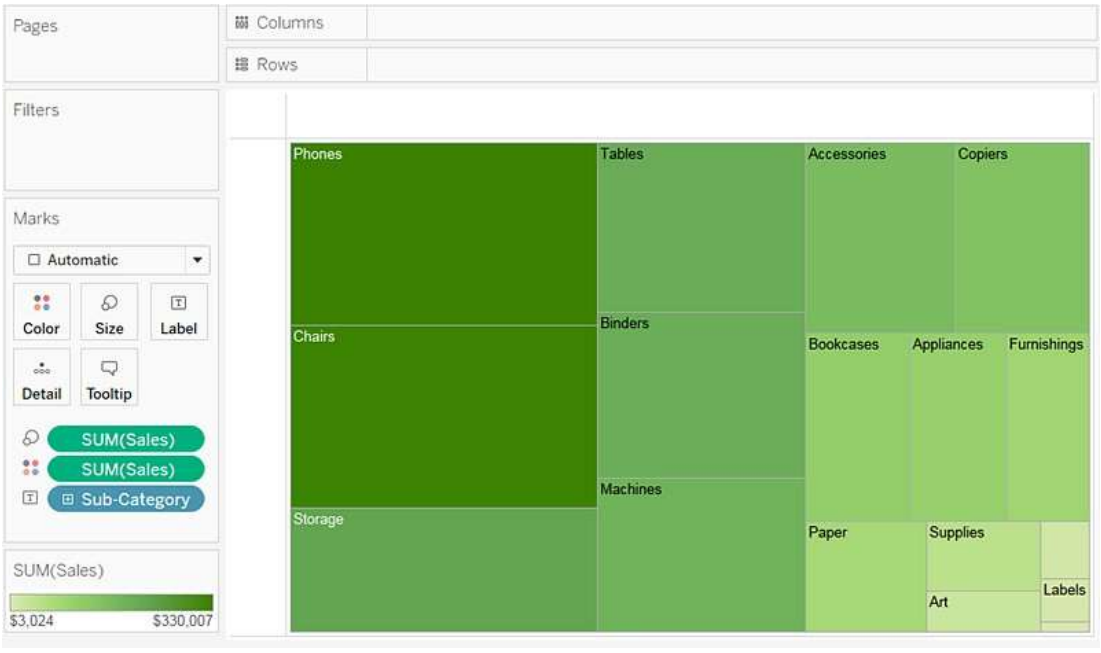
Back-to-Back Bar Chart:



Stacked Bar Chart:



Tree Map:

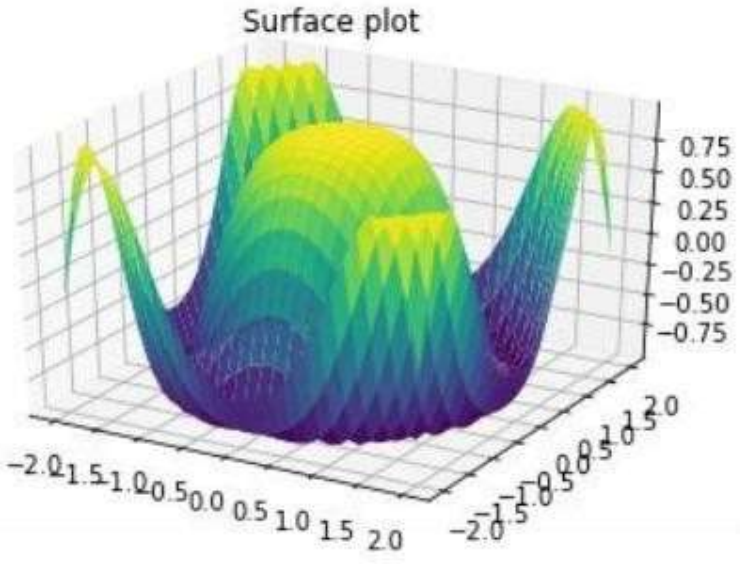


TASK 7: Experiment: Surfaces, Contours, Hidden surfaces.

Aim: To visualize 3D Surface plotting in Python using Matplotlib

Surface plot shows a functional relationship between a designated dependent variable (Y), and two independent variables (X and Z). The plot is a companion plot to the contour plot. Asurface plot is like a wireframe plot, but each face of the wireframe is a filled polygon. This can aid perception of the topology of the surface being visualized. The plot_surface() function x,y and z as arguments.

```
from mpl_toolkits import
mplot3dimport numpy as
np
import matplotlib.pyplot as plt
x = np.outer(np.linspace(-2, 2, 30),
np.ones(30))y = x.copy().T # transpose
z = np.cos(x ** 2
+ y ** 2)fig =
plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(x, y, z,cmap='viridis',
edgecolor='none')ax.set_title('Surface plot')
plt.show()
```



Contours

Contour plots (sometimes called Level Plots) are a way to show a three-dimensional surface on a two-dimensional plane. It graphs two predictor variables X Y on the y-axis and a

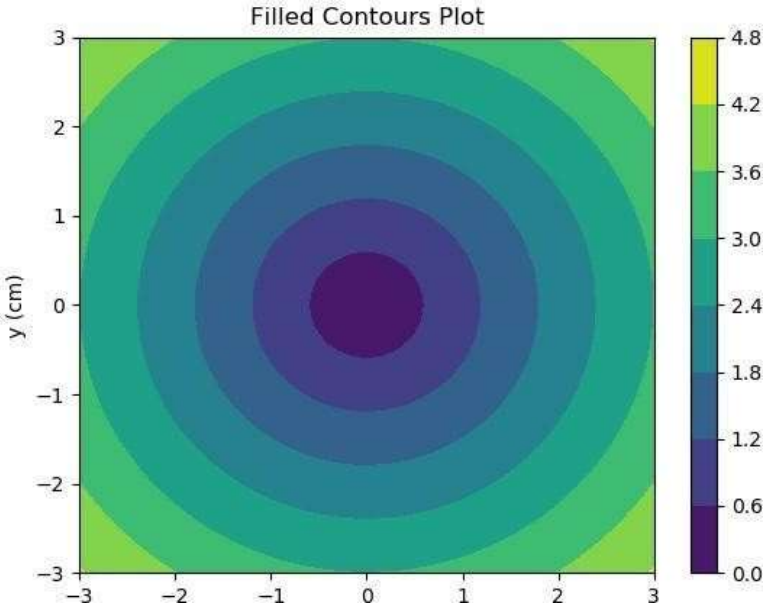
response variable Z as contours. These contours are sometimes called the z-slices or the iso-response values.

A contour plot is appropriate if you want to see how Value Z changes as a function of two inputs X and Y, such that Z = f(X,Y). A contour line or isoline of a function of two variables is a curve along whichthe function has a constant value.

The independent variables x and y are usually restricted to a regular grid called meshgrid. The numpy.meshgrid creates a rectangular grid out of an array of x values and an array of y values.

Matplotlib API contains contour() and contourf() functions that draw contour lines and filled contours,respectively. Both functions need three parameters x,y and z.

```
import numpy as np

import  matplotlib.pyplot  as  plt

xlist = np.linspace(-3.0, .0,100)

ylist = np.linspace(-3.0, 3.0, 100)

X, Y = np.meshgrid(xlist, ylist)

Z = np.sqrt(X**2 + Y**2)

fig,ax=plt.subplots(1,1)

cp = ax.contourf(X, Y, Z)

fig.colorbar(cp) # Add a colorbar to a plotax.set_title('Filled Contours Plot')

#ax.set_xlabel('x (cm)')

ax.set_ylabel(

'y (cm)')

plt.show()
```


Filled Contours Plot
y (cm)
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
4.8
4.2
3.6
3.0
2.4
1.8
1.2
0.6
0.0

Hidden Surfaces:

In addition to import matplotlib.pyplot as plt and calling plt.show(), to create a 3D plot in matplotlib, you need to:

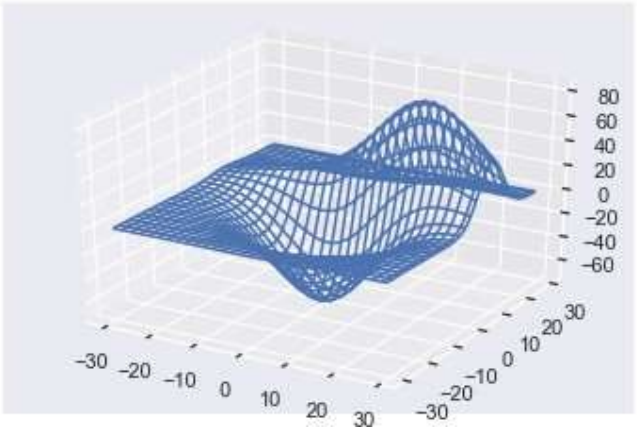
Import the Axes3D object

Initialize your Figure and Axes3D object

Get some 3D data

Plot it using Axes notation

```
# Standard import
import matplotlib.pyplot as plt
# Import 3D Axes
from mpl_toolkits.mplot3d import axes3d
# Set up Figure and 3D Axes
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Get some data
```

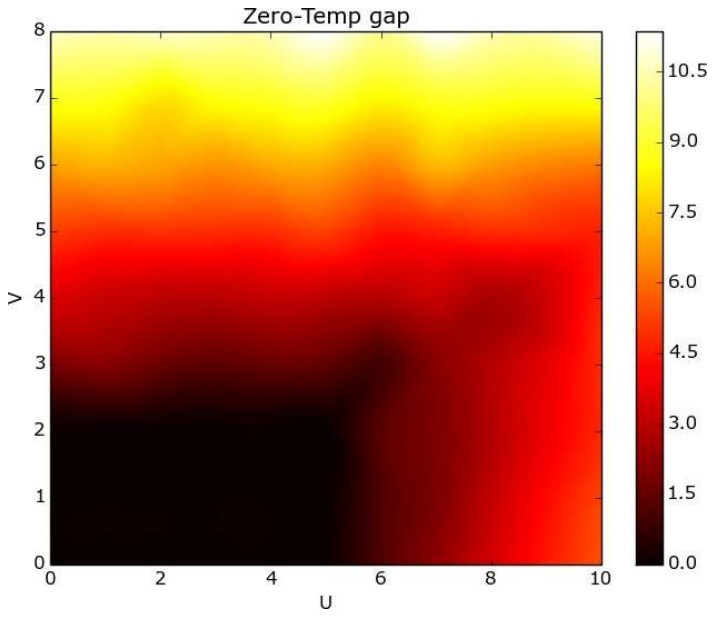


Task 8:3-D: pm3d coloring, 3Dmapping.

Aim: To Visualize pm3d coloring, 3Dmapping.

Pm3d coloring:

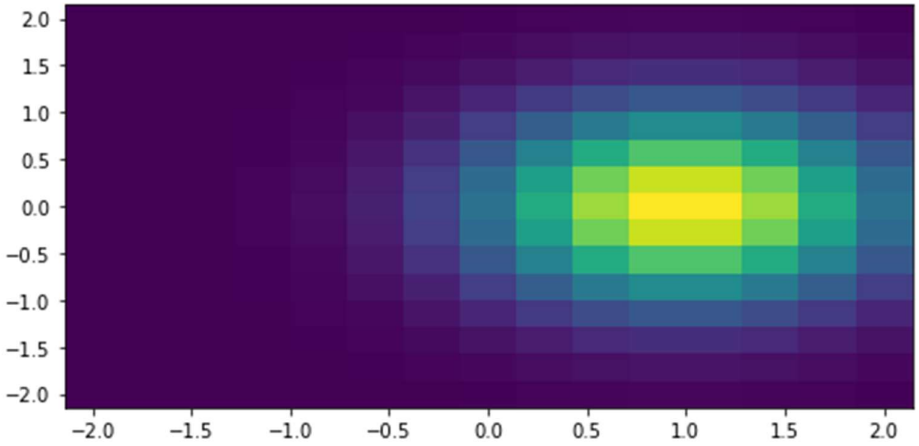
```
import numpy as np
import matplotlib.pyplot as pl
from scipy.interpolate import griddata

z=np.loadtxt("data.dat",usecols=[0,1,2],unpack=True)
x=np.arange(0,10.1,0.10)
y=np.arange(0,8.1,0.10)
zi=griddata((z[0],z[1]),z[2],(x[None,:], y[:,None]), method='cubic')
pl.imshow(zi,origin="lower",vmin=0,cmap="hot",aspect="auto",extent=[0,10,0,8])
pl.xlabel("U")
pl.ylabel("V")
pl.title("Zero-Temp gap ")
pl.colorbar()
pl.savefig("outAF.pdf")
```



3D mapping

```
from matplotlib import pyplot as plt, cm, colors
import numpy as np
plt.rcParams["figure.figsize"] = [7.00, 3.50]
plt.rcParams["figure.autolayout"] = True
side = np.linspace(-2, 2, 15)
X, Y = np.meshgrid(side, side)
```

```
Z = np.exp(-((X - 1) ** 2 + Y ** 2))
plt.pcolormesh(X, Y, Z, shading='auto')
plt.show()
```



TASK 9: Program on multi-dimensional data visualization.

Visualizing Structured Multi-Dimensional Data

Let's get cracking instead of me droning on about theory and concepts. We will use the **Wine Quality Data Set** available from the UCI Machine Learning Repository. This data actually consists of two datasets depicting various attributes of red and white variants of the Portuguese *"Vinho Verde"* wine.

We'll start by loading up the following necessary dependencies for our analyses.

```
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib as mpl
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

We will mainly be using `matplotlib` and `seaborn` as our visualization frameworks here but you are free to check out and try the same with any other framework of your choice. Let's take a look at the data after some basic data pre-processing steps.

```
path1='C:/Users/GRIET/Desktop/data/winequality-red.csv'
path2='C:/Users/GRIET/Desktop/data/winequality-white.csv'

red_wine=pd.read_csv(path1,sep=';')
white_wine=pd.read_csv(path2,sep=';')

# store wine type as an attribute
red_wine['wine_type'] = 'red'
white_wine['wine_type'] = 'white'

# bucket wine quality scores into qualitative quality labels
red_wine['quality_label'] = red_wine['quality'].apply(lambda value: 'low'
                                        if value <= 5 else 'medium'
                                          if value <= 7 else 'high')
red_wine['quality_label'] = pd.Categorical(red_wine['quality_label'],
                                categories=['low', 'medium', 'high'])
white_wine['quality_label'] = white_wine['quality'].apply(lambda value: 'low'
                                        if value <= 5 else 'medium'
                                          if value <= 7 else 'high')
white_wine['quality_label'] = pd.Categorical(white_wine['quality_label'],
                                categories=['low', 'medium', 'high'])
```

```
# merge red and white wine datasets
wines = pd.concat([red_wine, white_wine])

# re-shuffle records just to randomize data points
```

wines = wines.sample(frac=1, random_state=42).reset_index(drop=True)

We create a single data frame wines by merging both the datasets pertaining to red and white wine samples. We also create a new categorical variable quality_label based on the quality attribute of wine samples. Let's take a peek at the data now.
wines.head()

```
In [15]: wines.head()
```

Out[15]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	wine_type	quality_label
0	7.0	0.17	0.74	12.8	0.045	24.0	126.0	0.99420	3.26	0.38	12.2	8	white	high
1	7.7	0.64	0.21	2.2	0.077	32.0	133.0	0.99560	3.27	0.45	9.9	5	red	low
2	6.8	0.39	0.34	7.4	0.020	38.0	133.0	0.99212	3.18	0.44	12.0	7	white	medium
3	6.3	0.28	0.47	11.2	0.040	61.0	183.0	0.99592	3.12	0.51	9.5	6	white	medium
4	7.4	0.35	0.20	13.9	0.054	63.0	229.0	0.99888	3.11	0.50	8.9	6	white	medium

It is quite evident that we have several numeric and categorical attributes for wine samples. Each observation belongs to a red or white wine sample and the attributes are specific attributes or properties measured and obtained from physicochemical tests.

```
subset_attributes = ['residual sugar', 'total sulfur dioxide', 'sulphates',
                     'alcohol', 'volatile acidity', 'quality']
rs =  round(red_wine[subset_attributes].describe(),2)
ws = round(white_wine[subset_attributes].describe(),2)
pd.concat([rs, ws], axis=1, keys=['Red Wine Statistics', 'White Wine Statistics'])
```

Out[16]:

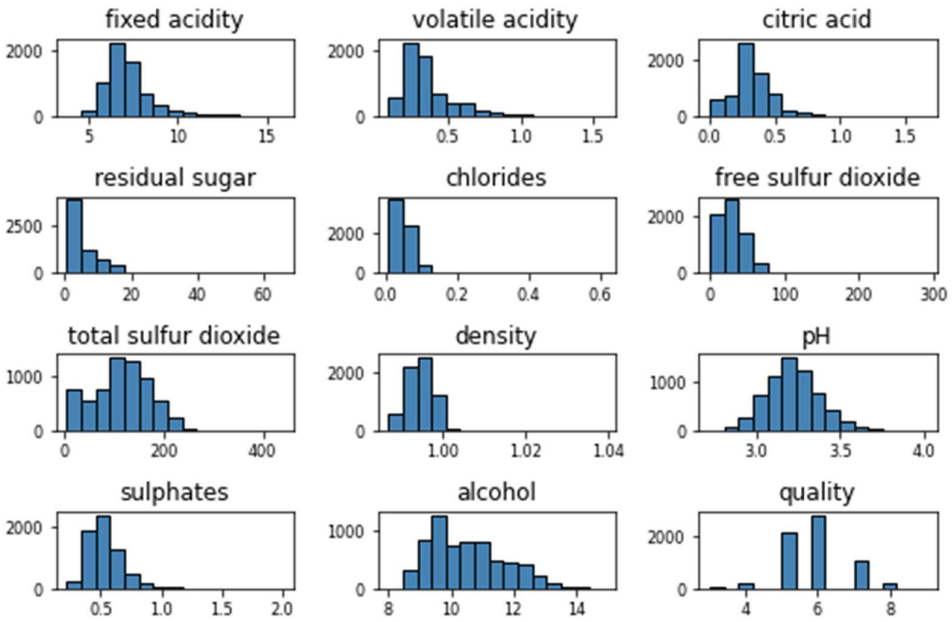
	Red Wine Statistics						White Wine Statistics					
	residual sugar	total sulfur dioxide	sulphates	alcohol	volatile acidity	quality	residual sugar	total sulfur dioxide	sulphates	alcohol	volatile acidity	quality
count	1599.00	1599.00	1599.00	1599.00	1599.00	1599.00	4898.00	4898.00	4898.00	4898.00	4898.00	4898.00
mean	2.54	46.47	0.66	10.42	0.53	5.64	6.39	138.36	0.49	10.51	0.28	5.88
std	1.41	32.90	0.17	1.07	0.18	0.81	5.07	42.50	0.11	1.23	0.10	0.89
min	0.90	6.00	0.33	8.40	0.12	3.00	0.60	9.00	0.22	8.00	0.08	3.00
25%	1.90	22.00	0.55	9.50	0.39	5.00	1.70	108.00	0.41	9.50	0.21	5.00
50%	2.20	38.00	0.62	10.20	0.52	6.00	5.20	134.00	0.47	10.40	0.26	6.00
75%	2.60	62.00	0.73	11.10	0.64	6.00	9.90	167.00	0.55	11.40	0.32	6.00
max	15.50	289.00	2.00	14.90	1.58	8.00	65.80	440.00	1.08	14.20	1.10	9.00

Visualizing data in One Dimension (1-D)

One of the quickest and most effective ways to visualize all numeric data and their distributions, is to leverage *histograms* using pandas

```
wines.hist(bins=15, color='steelblue', edgecolor='black', linewidth=1.0,
           xlabelsize=8, ylabelsize=8, grid=False)
```

```
plt.tight_layout(rect=(0, 0, 1.2, 1.2))
```



The plots above give a good idea about the basic data distribution of any of the attributes.

Let's drill down to *visualizing one of the continuous, numeric attributes.* Essentially a *histogram* or a *density plot* works quite well in understanding how the data is distributed for that attribute.

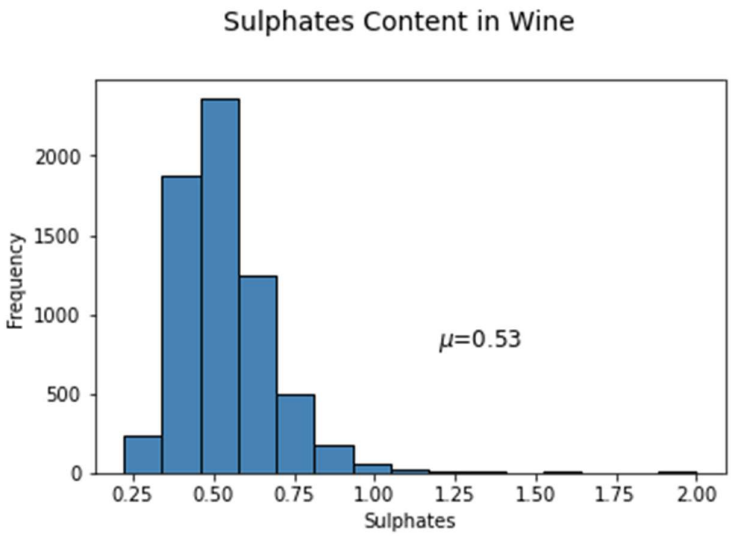
Histogram

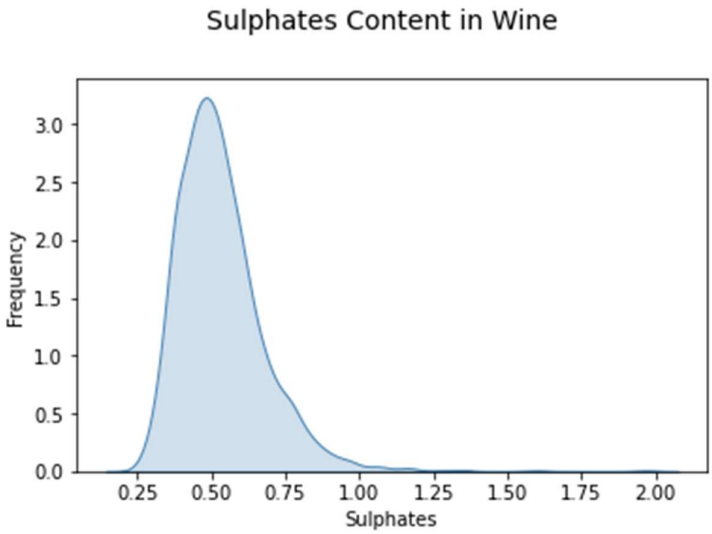
```
fig = plt.figure(figsize = (6,4))
title = fig.suptitle("Sulphates Content in Wine", fontsize=14)
fig.subplots_adjust(top=0.85, wspace=0.3)
ax = fig.add_subplot(1,1, 1)
ax.set_xlabel("Sulphates")
ax.set_ylabel("Frequency")
ax.text(1.2, 800, r'$\mu$='+str(round(wines['sulphates'].mean(),2)),
        fontsize=12)
freq, bins, patches = ax.hist(wines['sulphates'], color='steelblue', bins=15,
                              edgecolor='black', linewidth=1)
```

Density Plot

```
fig = plt.figure(figsize = (6, 4))

title = fig.suptitle("Sulphates Content in Wine", fontsize=14)
```

```
fig.subplots_adjust(top=0.85, wspace=0.3)

ax1 = fig.add_subplot(1,1, 1)

ax1.set_xlabel("Sulphates")

ax1.set_ylabel("Frequency")

sns.kdeplot(wines['sulphates'], ax=ax1, shade=True, color='steelblue')
```

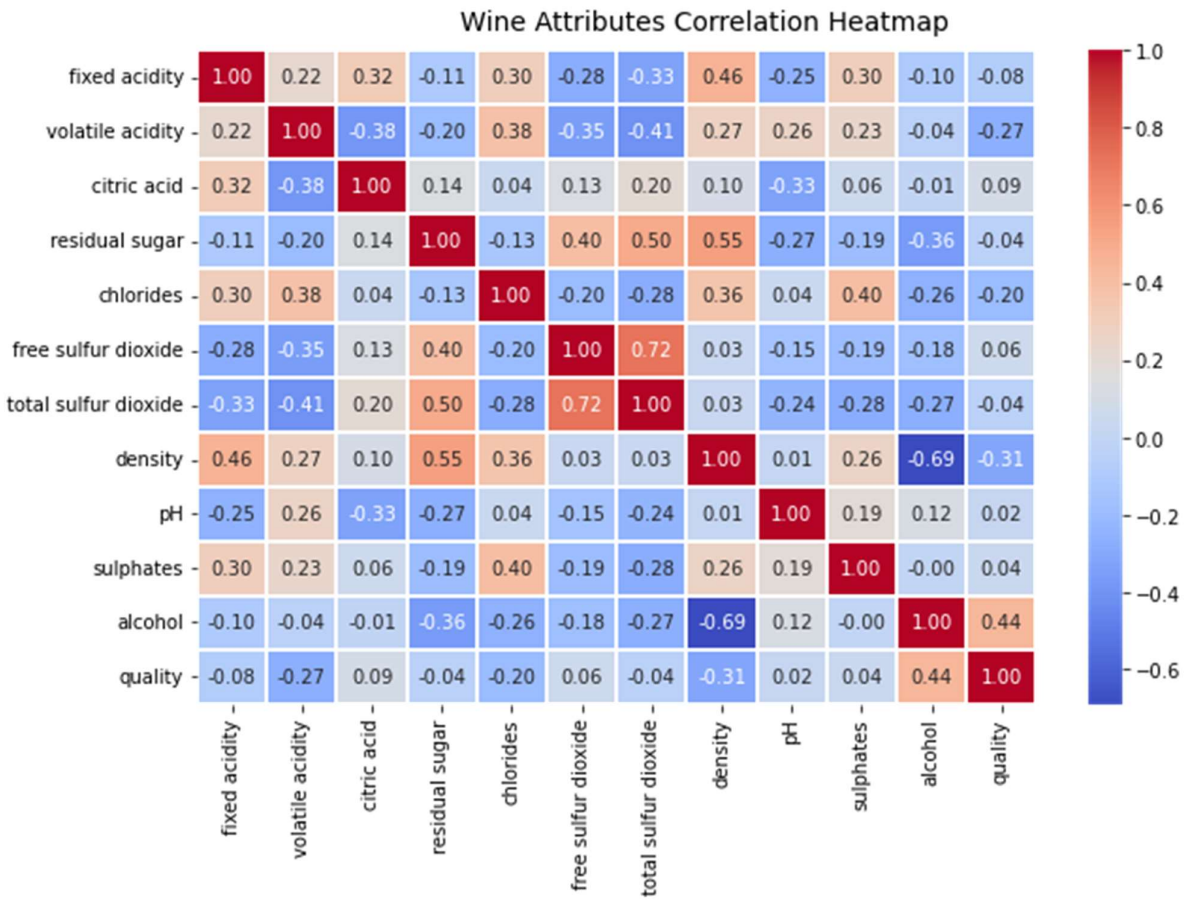




Visualizing data in Two Dimensions (2-D)

One of the best ways to check out potential relationships or correlations amongst the different data attributes is to leverage a *pair-wise correlation matrix* and depict it as a *heatmap*.

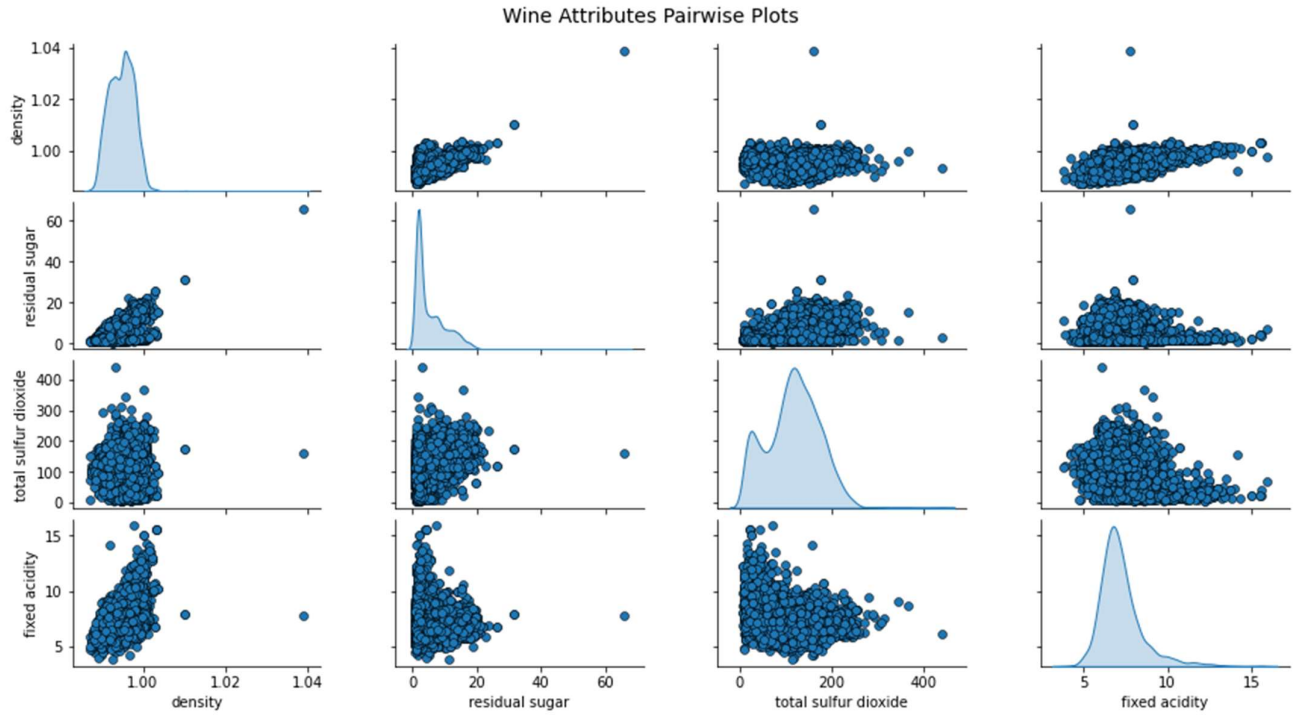
```
# Correlation Matrix Heatmap

f, ax = plt.subplots(figsize=(10, 6))

corr = wines.corr()

hm = sns.heatmap(round(corr,2), annot=True, ax=ax, cmap="coolwarm",fmt='.2f',

                 linewidths=.05)

f.subplots_adjust(top=0.93)

t= f.suptitle('Wine Attributes Correlation Heatmap', fontsize=14)
```



The gradients in the heatmap vary based on the strength of the correlation and you can clearly see it is very easy to spot potential attributes having strong correlations amongst themselves. Another way to visualize the same is to use *pair-wise scatter plots* amongst attributes of interest.

```
# Pair-wise Scatter Plots
```

```
cols = ['density', 'residual sugar', 'total sulfur dioxide', 'fixed acidity']
pp = sns.pairplot(wines[cols], size=1.8, aspect=1.8,
                  plot_kws=dict(edgecolor="k", linewidth=0.5),
                  diag_kind="kde", diag_kws=dict(shade=True))

fig = pp.fig

fig.subplots_adjust(top=0.93, wspace=0.3)

t = fig.suptitle('Wine Attributes Pairwise Plots', fontsize=14)
```

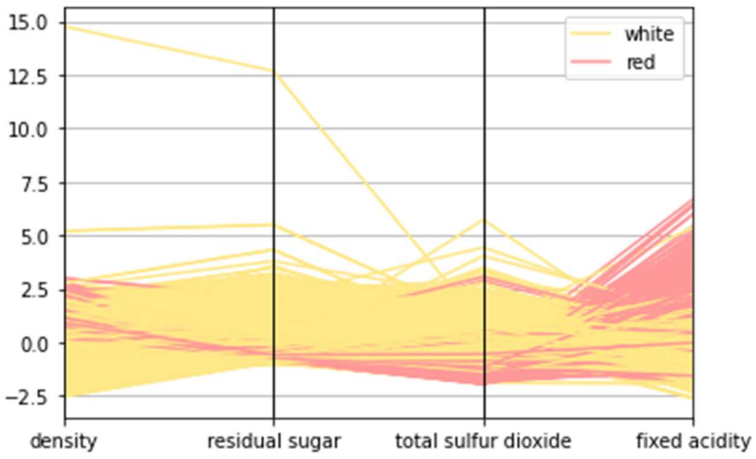


Based on the above plot, you can see that scatter plots are also a decent way of observing potential relationships or patterns in two-dimensions for data attributes.

Another way of visualizing multivariate data for multiple attributes together is to use *parallel coordinates*.

```
# Scaling attribute values to avoid few outiers

cols = ['density', 'residual sugar', 'total sulfur dioxide', 'fixed acidity']

subset_df = wines[cols]
```

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()

scaled_df = ss.fit_transform(subset_df)

scaled_df = pd.DataFrame(scaled_df, columns=cols)

final_df = pd.concat([scaled_df, wines['wine_type']], axis=1)

final_df.head()


# plot parallel coordinates

from pandas.plotting import parallel_coordinates

pc = parallel_coordinates(final_df, 'wine_type', color=('#FFE888', '#FF9999'))
```

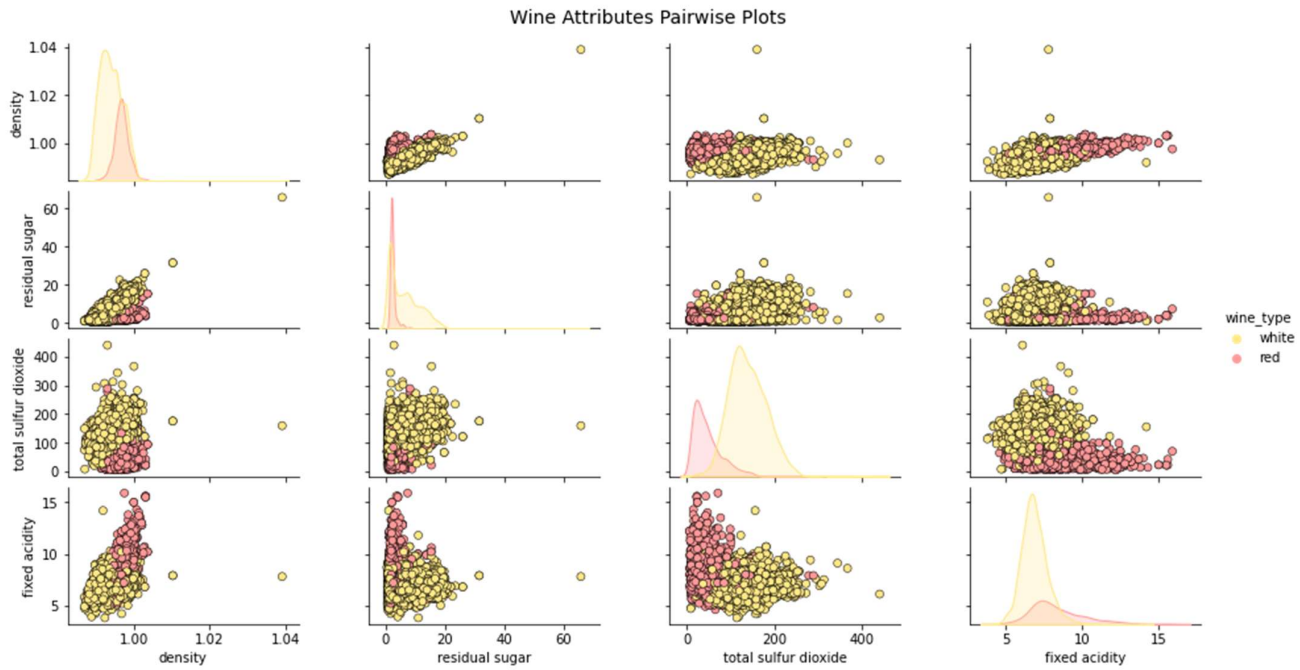


Visualizing data in Three Dimensions (3-D)

Considering three attributes or dimensions in the data, we can visualize them by considering a *pair-wise scatter plot* and introducing the notion of *color* or *hue* to separate out values in a categorical dimension.

```
# Scatter Plot with Hue for visualizing data in 3-D

cols = ['density', 'residual sugar', 'total sulfur dioxide', 'fixed acidity', 'wine_type']

pp = sns.pairplot(wines[cols], hue='wine_type', size=1.8, aspect=1.8,

                  palette={"red": "#FF9999", "white": "#FFE888"},
```

```
            plot_kws=dict(edgecolor="black", linewidth=0.5))
fig = pp.fig
fig.subplots_adjust(top=0.93, wspace=0.3)
t = fig.suptitle('Wine Attributes Pairwise Plots', fontsize=14)
```

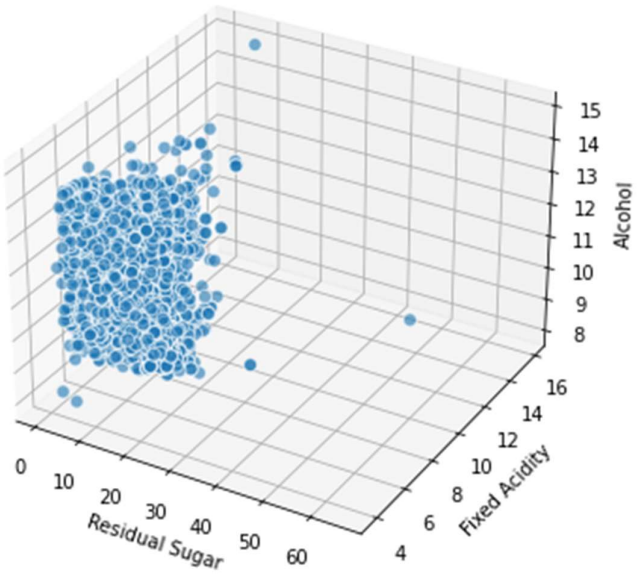


The above plot enables you to check out correlations and patterns and also compare around wine groups. Like we can clearly see total sulfur dioxide and residual sugar is higher for *white wine* as compared to *red*.

Let's look at strategies for ***visualizing three continuous, numeric attributes***. One way would be to have two dimensions represented as the regular ***length*** (*x*-axis)and ***breadth*** (*y*-axis) and also take the notion of ***depth*** (*z*-axis) for the third dimension.

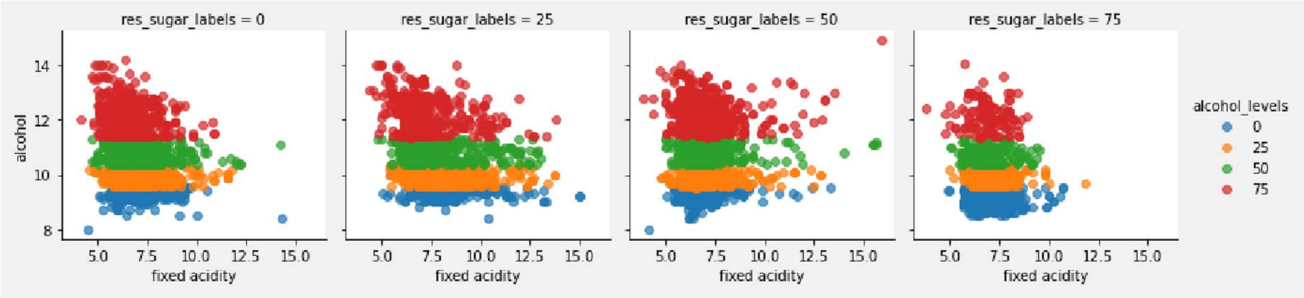
```
# Visualizing 3-D numeric data with Scatter Plots

# length, breadth and depth

fig = plt.figure(figsize=(8, 6))

ax = fig.add_subplot(111, projection='3d')


xs = wines['residual sugar']

ys = wines['fixed acidity']

zs = wines['alcohol']

ax.scatter(xs, ys, zs, s=50, alpha=0.6, edgecolors='w')


ax.set_xlabel('Residual Sugar')
```

```
ax.set_ylabel('Fixed Acidity')

ax.set_zlabel('Alcohol')
```



A better option would be to use the notion of ***faceting*** as the third dimension (essentially ***subplots***) where each subplot indicates a specific bin from our third variable (dimension). Do remember you need to create your bins manually if you are using the scatterplot functionality from **matplotlib** as opposed to **seaborn.**

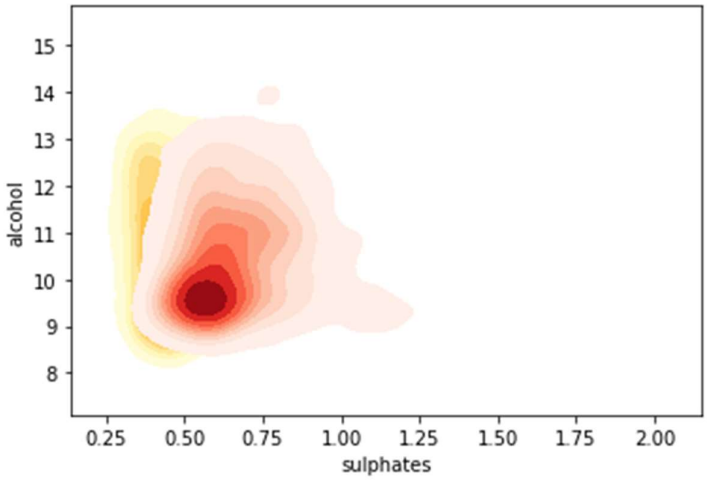
```
# example depicting representing 3-D continous data

# using color and facets

quantile_list = [0, .25, .5, .75, 1.]

quantile_labels = ['0', '25', '50', '75']

wines['res_sugar_labels'] = pd.qcut(wines['residual sugar'],

                        q=quantile_list, labels=quantile_labels)

wines['alcohol_levels'] = pd.qcut(wines['alcohol'],

                        q=quantile_list, labels=quantile_labels)

g = sns.FacetGrid(wines, col="res_sugar_labels",

            hue='alcohol_levels')

g.map(plt.scatter, "fixed acidity", "alcohol", alpha=.7)

g.add_legend();
```



you can also use a *kernel density plot* to understand the data in three dimensions.

 # Visualizing 3-D mix data using kernel density plots

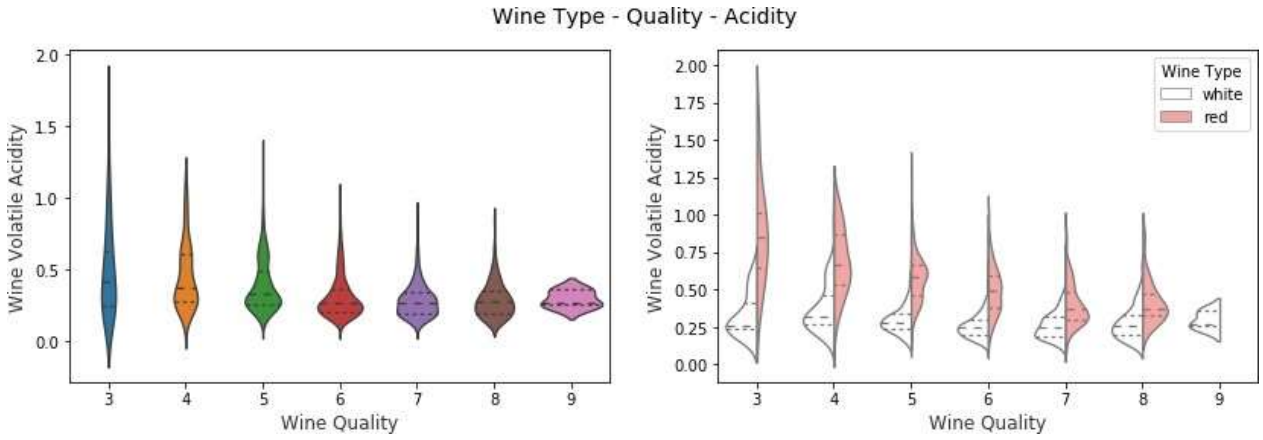
```
# leveraging the concepts of hue for categorical dimension
ax = sns.kdeplot(white_wine['sulphates'], white_wine['alcohol'],
                 cmap="YlOrBr", shade=True, shade_lowest=False)
ax = sns.kdeplot(red_wine['sulphates'], red_wine['alcohol'],
                 cmap="Reds", shade=True, shade_lowest=False)
```



 # Visualizing 3-D mix data using violin plots

```
# leveraging the concepts of hue and axes for > 1 categorical dimensions
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 4))
f.suptitle('Wine Type - Quality - Acidity', fontsize=14)


sns.violinplot(x="quality", y="volatile acidity",
               data=wines, inner="quart", linewidth=1.3,ax=ax1)
```

```
ax1.set_xlabel("Wine Quality",size = 12,alpha=0.8)

ax1.set_ylabel("Wine Volatile Acidity",size = 12,alpha=0.8)


sns.violinplot(x="quality", y="volatile acidity", hue="wine_type",

        data=wines, split=True, inner="quart", linewidth=1.3,

        palette={"red": "#FF9999", "white": "white"}, ax=ax2)

ax2.set_xlabel("Wine Quality",size = 12,alpha=0.8)

ax2.set_ylabel("Wine Volatile Acidity",size = 12,alpha=0.8)

l = plt.legend(loc='upper right', title='Wine Type')
```

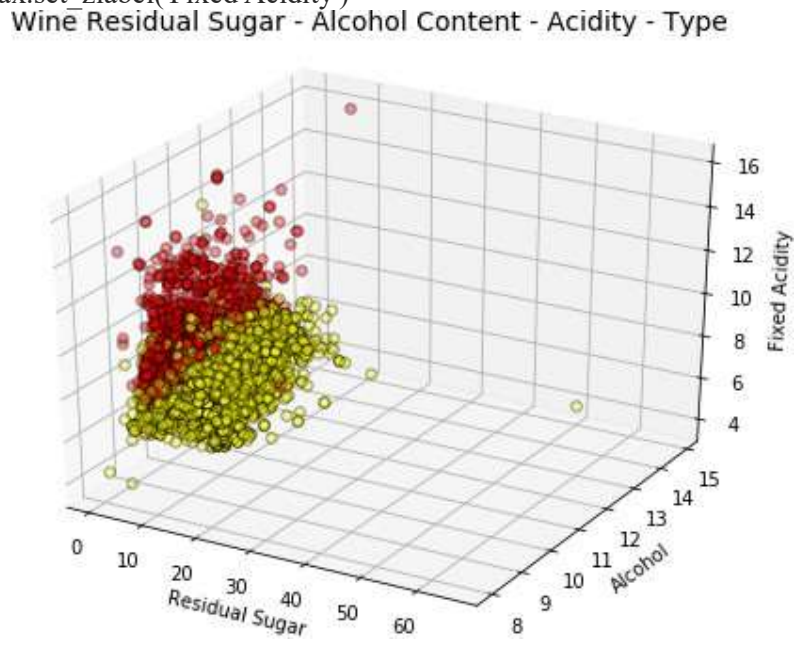


Visualizing data in Four Dimensions (4-D)

Based on our discussion earlier, we leverage various components of the charts visualize multiple dimensions. One way to visualize data in four dimensions is to use *depth* and *hue* as specific data dimensions in a conventional plot like a *scatter plot*.

```
# Visualizing 4-D mix data using scatter plots
# leveraging the concepts of hue and depth
fig = plt.figure(figsize=(8, 6))
t = fig.suptitle('Wine Residual Sugar - Alcohol Content - Acidity - Type', fontsize=14)
ax = fig.add_subplot(111, projection='3d')

xs = list(wines['residual sugar'])
ys = list(wines['alcohol'])
zs = list(wines['fixed acidity'])
```

```
data_points = [(x, y, z) for x, y, z in zip(xs, ys, zs)]
colors = ['red' if wt == 'red' else 'yellow' for wt in list(wines['wine_type'])]

for data, color in zip(data_points, colors):
    x, y, z = data
    ax.scatter(x, y, z, alpha=0.4, c=color, edgecolors='none', s=30)

ax.set_xlabel('Residual Sugar')
ax.set_ylabel('Alcohol')
ax.set_zlabel('Fixed Acidity')
```

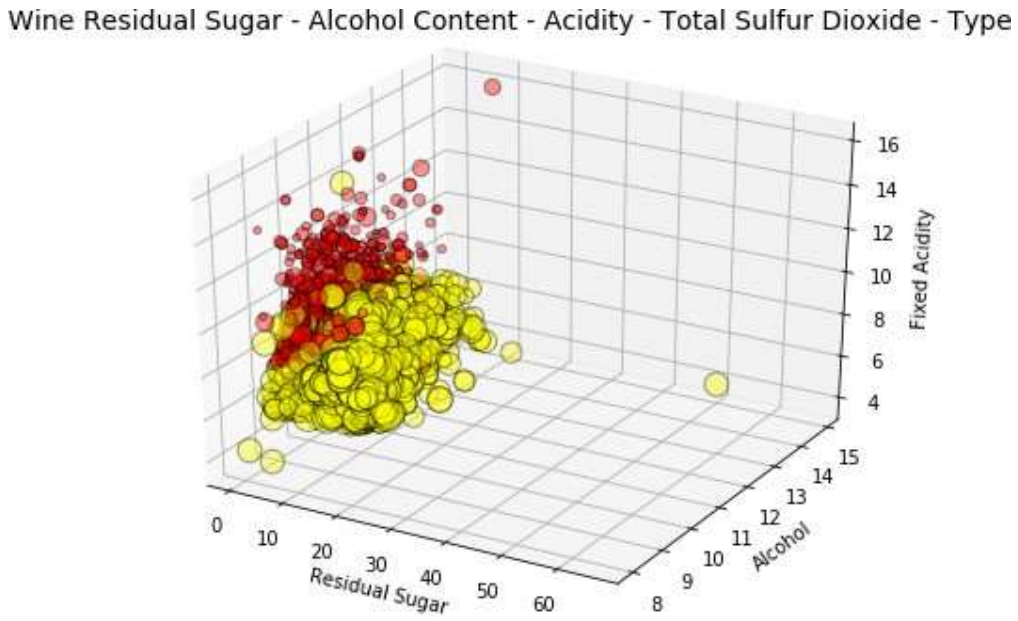


Visualizing data in Five Dimensions (5-D)

Once again following a similar strategy as we followed in the previous section, to visualize data in five dimensions, we leverage various plotting components. Let's use *depth*, *hue* and *size* to represent three of the data dimensions besides *regular axes* representing the other two dimensions. Since we use the notion of size, we will be basically plotting a three dimensional *bubble chart*.

```
# Visualizing 5-D mix data using bubble charts
# leveraging the concepts of hue, size and depth
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
t = fig.suptitle('Wine Residual Sugar - Alcohol Content - Acidity - Total Sulfur Dioxide - Type',
fontsize=14)

xs = list(wines['residual sugar'])
ys = list(wines['alcohol'])
zs = list(wines['fixed acidity'])
data_points = [(x, y, z) for x, y, z in zip(xs, ys, zs)]

ss = list(wines['total sulfur dioxide'])
colors = ['red' if wt == 'red' else 'yellow' for wt in list(wines['wine_type'])]
```

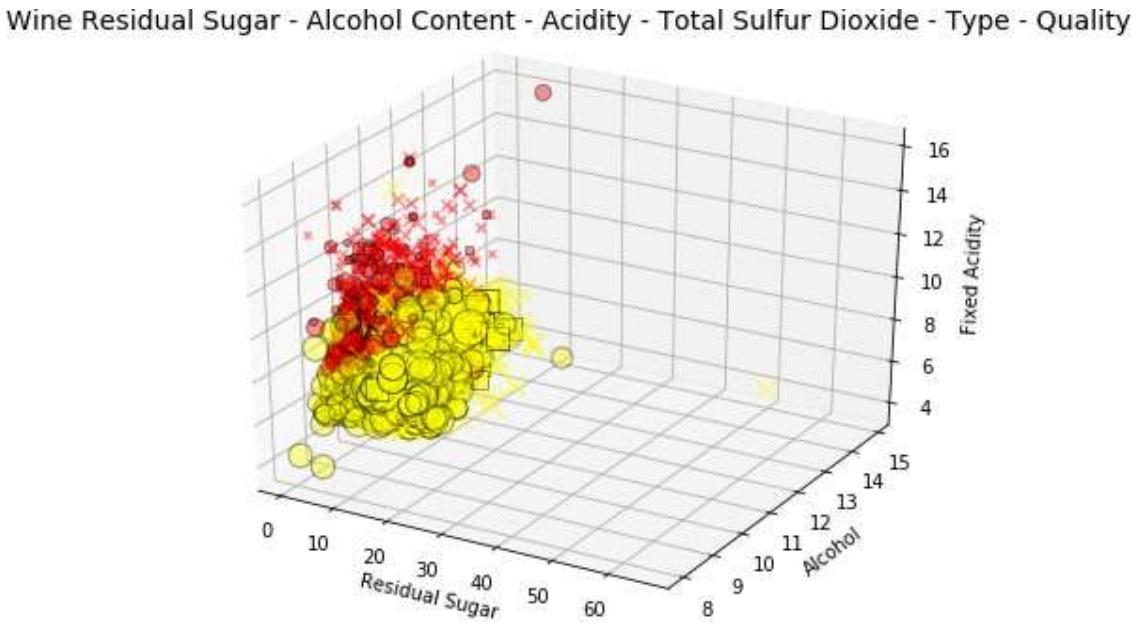
```
for data, color, size in zip(data_points, colors, ss):
    x, y, z = data
    ax.scatter(x, y, z, alpha=0.4, c=color, edgecolors='none', s=size)

ax.set_xlabel('Residual Sugar')
ax.set_ylabel('Alcohol')
ax.set_zlabel('Fixed Acidity')
```



Visualizing data in Six Dimensions (6-D)

Now that we are having fun (I hope!), let's add another data dimension in our visualizations. We will leverage ***depth***, ***hue***, ***size*** and ***shape*** besides our ***regular two axes*** to depict all the six data dimensions.

```
# Visualizing 6-D mix data using scatter charts
# leveraging the concepts of hue, size, depth and shape
fig = plt.figure(figsize=(8, 6))
t = fig.suptitle('Wine Residual Sugar - Alcohol Content - Acidity - Total Sulfur Dioxide - Type - Quality', fontsize=14)
ax = fig.add_subplot(111, projection='3d')
xs = list(wines['residual sugar'])
ys = list(wines['alcohol'])
zs = list(wines['fixed acidity'])
data_points = [(x, y, z) for x, y, z in zip(xs, ys, zs)]
ss = list(wines['total sulfur dioxide'])
colors = ['red' if wt == 'red' else 'yellow' for wt in list(wines['wine_type'])]
markers = [',' if q == 'high' else 'x' if q == 'medium' else 'o' for q in list(wines['quality_label'])]
for data, color, size, mark in zip(data_points, colors, ss, markers):
    x, y, z = data
    ax.scatter(x, y, z, alpha=0.4, c=color, edgecolors='none', s=size, marker=mark)
ax.set_xlabel('Residual Sugar')
ax.set_ylabel('Alcohol')
ax.set_zlabel('Fixed Acidity')
```

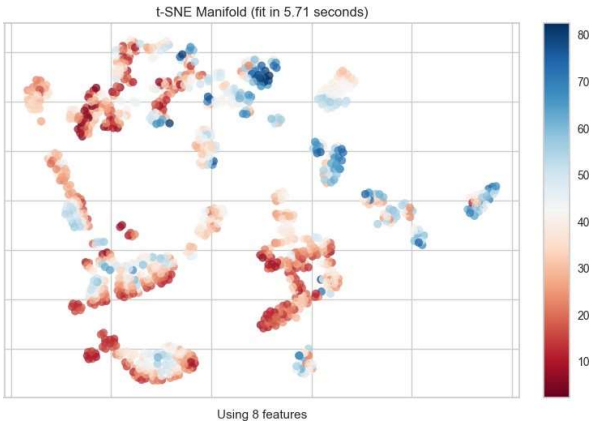

Wine Residual Sugar - Alcohol Content - Acidity - Total Sulfur Dioxide - Type - Quality
Residual Sugar
0
10
20
30
40
50
60
Alcohol
8
9
10
11
12
13
14
15
Fixed Acidity
4
6
8
10
12
14
16

TASK 10: Program on manifold visualization.

Manifold Visualization

The Manifold visualizer provides high dimensional visualization using manifold learning to embed instances described by many dimensions into 2, thus allowing the creation of a scatter plot that shows latentstructures in data. Unlike decomposition methods such as PCA and SVD, manifolds generally use nearest-neighbors approaches to embedding, allowing them to capture non-linear structures that would

Visualizer	**Manifold**
Quick Method	**manifold_embedding ()**
Models	Classification, Regression
Workflow	Feature Engineering



The Manifold visualizer allows access to all currently available scikit-learn manifold implementations by specifying the manifold as a string to the visualizer. The currently implemented default manifolds are as follows:

Manifold	Description
"lle"	Locally Linear Embedding (LLE) uses many local linear decompositions to preserve globally non-linear structures.
"ltsa"	LTSA LLE: local tangent space alignment is similar to LLE in that it uses locality to preserve neighborhood distances.
"hessian"	Hessian LLE an LLE regularization method that applies a hessian-based quadratic form at each neighborhood
"modified"	Modified LLE applies a regularization parameter to LLE.

"isomap"	Isomap seeks a lower dimensional embedding that maintains geometric distances between each instance.

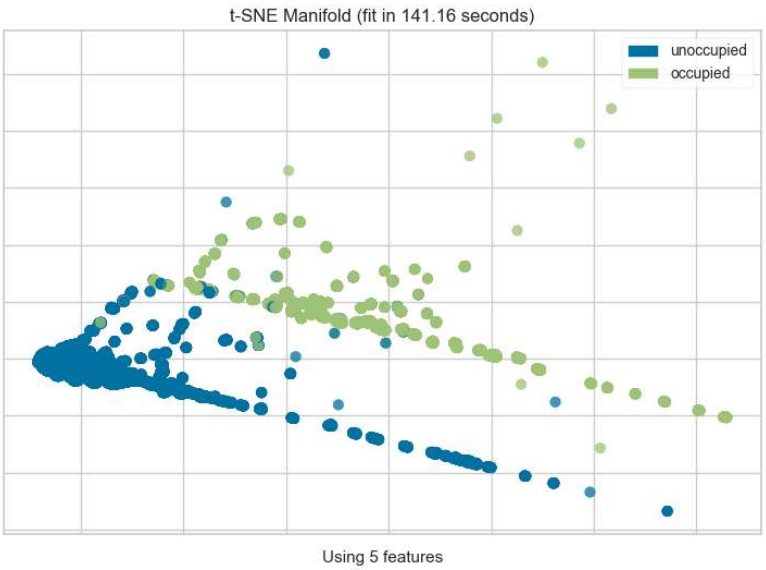
"mds"	MDS: multi-dimensional scaling uses similarity to plot points that are near to each other close in the embedding.
"spectral"	Spectral Embedding a discrete approximation of the low dimensional manifold using a graph representation.
"tsne"	t-SNE: converts the similarity of points into probabilities then uses those probabilities to create an embedding.

Each manifold algorithm produces a different embedding and takes advantage of different properties of the underlying data. Generally speaking, it requires multiple attempts on new data to determine the manifold that works best for the structures latent in your data. Note however, that different manifold algorithms have different time, complexity, and resource requirements.

Manifolds can be used on many types of problems, and the color used in the scatter plot can describe the target instance. In an unsupervised or clustering problem, a single color is used to show structure and overlap. In a classification problem discrete colors are used for each class. In a regression problem, a color map can be used to describe points as a heat map of their regression values.

Discrete Target
In a classification or clustering problem, the instances can be described by discrete labels - the classes or categories in the supervised problem, or the clusters they belong to in the unsupervised version. The manifold visualizes this by assigning a color to each label and showing the labels in a legend.

```
from yellowbrick.features import Manifold

from yellowbrick.datasets import load_occupancy

# Load the classification dataset

X, y = load_occupancy()

classes = ["unoccupied", "occupied"]

# Instantiate the visualizer

viz = Manifold(manifold="tsne", classes=classes)

viz.fit_transform(X, y) # Fit the data to the visualizer

viz.show()              # Finalize and render the figure
```



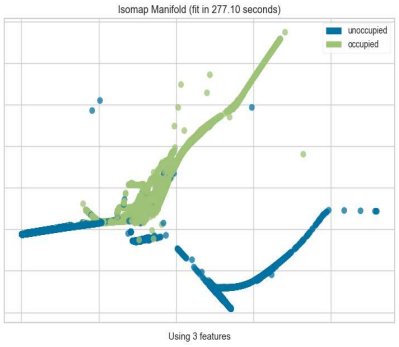
The visualization also displays the amount of time it takes to generate the embedding; as you can see, this can take a long time even for relatively small datasets. One tip is scale your data using the StandardScalar; another is to sample your instances (e.g. using train_test_split to preserve class stratification) or to filter features to decrease sparsity in the dataset.

One common mechanism is to use SelectKBest to select the features that have a statistical correlation with the target dataset. For example, we can use the f_classif score to find the 3 best features in our occupancy dataset.

```
from sklearn.pipeline import Pipeline

from sklearn.feature_selection import f_classif, SelectKBest

from yellowbrick.features import Manifold

from yellowbrick.datasets import load_occupancy

# Load the classification dataset

X, y = load_occupancy()

classes = ["unoccupied", "occupied"]

# Create a pipeline

model = Pipeline([    ("selectk", SelectKBest(k=3, score_func=f_classif)),

   ("viz", Manifold(manifold="isomap", n_neighbors=10, classes=classes)),])
```

```
model.fit_transform(X, y)          # Fit the data to the model
```

model.named_steps['viz'].show() # Finalize and render the figure



Task 11: Program on graph data visualization.

A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph.

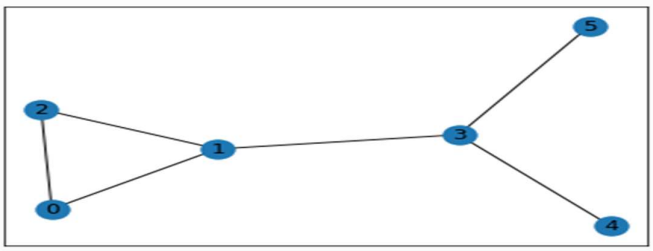
In this tutorial we are going to visualize undirected Graphs in Python with the help of networkx library.

Installation:

To install this module type the below command in the terminal.

pip install networkx

```
# First networkx library is imported
# along with matplotlib
import networkx as nx
import matplotlib.pyplot as plt

# Defining a Class
class GraphVisualization:

        def __init_(self):

                # visual is a list which stores all
                # the set of edges that constitutes a
                # graph
                self.visual = []

        # addEdge function inputs the vertices of an
        # edge and appends it to the visual list
        def addEdge(self, a, b):
                temp = [a, b]
                self.visual.append(temp)
        # In visualize function G is an object of
        # class Graph given by networkx G.add_edges_from(visual)
        # creates a graph with a given list
        # nx.draw_networkx(G) - plots the graph
        # plt.show() - displays the graph
        def visualize(self):
                G = nx.Graph()
                G.add_edges_from(self.visual)
                nx.draw_networkx(G)
                plt.show()
# Driver code
G = GraphVisualization()
G.addEdge(0, 2)
G.addEdge(1, 2)
G.addEdge(1, 3)
G.addEdge(5, 3)
G.addEdge(3, 4)
G.addEdge(1, 0)
G.visualize()
```


5
2
3
1
0
4

TASK 12: Program on Annotation.

matplotlib.pyplot.annotate
matplotlib.pyplot.annotate(*text*, *xy*, **args*, ***kwargs*)

Annotate the point *xy* with text *text*.

In the simplest form, the text is placed at *xy*. Optionally, the text can be displayed in another position *xytext*. An arrow pointing from the text to the annotated point *xy* can then be added by defining *arrowprops*.

Parameters
text*str*
The text of the annotation.
xy*(float, float)*
The point *(x, y)* to annotate. The coordinate system is determined by *xycoords*.
xytext*(float, float), default: xy*
The position *(x, y)* to place the text at. The coordinate system is determined by *textcoords*.
xycoords*str or Artist or Transform or callable or (float, float), default: 'data'*
The coordinate system that *xy* is given in.
The following types of values are supported:

- One of the following strings:

Value	Description
'figure points'	Points from the lower left of the figure
'figure pixels'	Pixels from the lower left of the figure
'figure fraction'	Fraction of figure from lower left
'subfigure points'	Points from the lower left of the subfigure
'subfigure pixels'	Pixels from the lower left of the subfigure
'subfigure fraction'	Fraction of subfigure from lower left
'axes points'	Points from lower left corner of axes
'axes pixels'	Pixels from lower left corner of axes
'axes fraction'	Fraction of axes from lower left
'data'	Use the coordinate system of the object being annotated (default)
'polar'	*(theta, r)* if not native 'data' coordinates

 # Implementation of matplotlib.pyplot.annotate function

 # Implementation of matplotlib.pyplot.annotate function

```
import numpy as np
import matplotlib. pyplot as plt
x = np.arange(0, 10, 0.005)
y = np.exp(-x / 3.) * np.sin(3 * np.pi * x)
fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_xlim(0, 10)
ax.set_ylim(-1, 1)
```

```
  # Setting up the parameters
xdata, ydata = 5, 0
xdisplay, ydisplay = ax.transData.transform((xdata, ydata))
  bbox = dict(boxstyle ="round", fc ="0.8")

 arrowprops = dict(arrowstyle = "->“,  connectionstyle = "angle, angleA = 0, angleB = 90,\
   rad = 10")

offset = 72 #
Annotation
    ax.annotate('data = (%.1f, %.1f)'%(xdata, ydata),  (xdata, ydata), xytext =(-2 * offset, offset),
            textcoords ='offset points',  bbox = bbox, arrowprops = arrowprops)

  disp = ax.annotate('display = (%.1f, %.1f)'%(xdisplay, ydisplay),  (xdisplay, ydisplay), xytext =(0.5 *
    offset, -offset), xycoords ='figure pixels', textcoords ='offset points', bbox = bbox, arrowprops =
    arrowprops)
  # To display the annotation
    plt.show()
```

Output: