# Practical 1

# Using Plotly for Interactive Data Visualization in Python[¶](#Using-Plotly-for-Interactive-Data-Visua)

Plotly is an open-source module of Python which is used for data visualization and supports various graphs like line charts, scatter plots, bar charts, histograms, area plot, etc. Here, we will see how to plot a basic chart with plotly and also how to make a plot interactive. But before starting you might be wondering why there is a need to learn plotly, so let’s have a look at it. ##Why Plotly? Plotly uses javascript behind the scenes and is used to make interactive plots where we can zoom in on the graph or add additional information like data on hover and many more things. Let’s see few more advantages of plotly – Plotly has hover tool capabilities that allow us to detect any outliers or anomalies in a large number of data points. It is visually attractive that can be accepted by a wide range of audiences. It allows us for the endless customization of our graphs that makes our plot more meaningful and understandable for others. ##Installation Plotly does not come built-in with Python. To install it type the below command in the terminal. pip install plotly plotly installation image widget Overview of Plotly Package Structure in Plotly, there are three main modules – plotly.plotly acts as the interface between the local machine and Plotly. It contains functions that require a response from Plotly’s server. plotly.graph\_objects module contains the objects (Figure, layout, data, and the definition of the plots like scatter plot, line chart) that are responsible for creating the plots. The Figure can be represented either as dict or instances of plotly.graph\_objects.Figure and these are serialized as JSON before it gets passed to plotly.js. Figures are represented as trees where the root node has three top layer attributes – data, layout, and frames and the named nodes called ‘attributes’. Note: plotly.express module can create the entire Figure at once. It uses the graph\_objects internally and returns the graph\_objects.Figure instance. Example:

In [1]:

import plotly.express as px

from plotly import graph\_objects as go

from plotly import express as px

# Creating the Figure instance

fig = px.line(x=[1, 2], y=[3, 4])

# printing the figure instance

print(fig)

Figure({

'data': [{'hovertemplate': 'x=%{x}<br>y=%{y}<extra></extra>',

'legendgroup': '',

'line': {'color': '#636efa', 'dash': 'solid'},

'mode': 'lines',

'name': '',

'orientation': 'v',

'showlegend': False,

'type': 'scatter',

'x': array([1, 2], dtype=int64),

'xaxis': 'x',

'y': array([3, 4], dtype=int64),

'yaxis': 'y'}],

'layout': {'legend': {'tracegroupgap': 0},

'margin': {'t': 60},

'template': '...',

'xaxis': {'anchor': 'y', 'domain': [0.0, 1.0], 'title': {'text': 'x'}},

'yaxis': {'anchor': 'x', 'domain': [0.0, 1.0], 'title': {'text': 'y'}}}

})

plotly.tools module contains various tools in the forms of the functions that can enhance the Plotly experience.

After going through the basics of plotly let’s see how to create some basic charts using plotly.

Line chart

A line chart is one of the simple plots where a line is drawn to shoe relation between the X-axis and Y-axis. It can be created using the px.line() method with each data position is represented as a vertex (which location is given by the x and y columns) of a polyline mark in 2D space.

Syntax:

Syntax: plotly.express.line(data\_frame=None, x=None, y=None, line\_group=None, color=None, line\_dash=None, hover\_name=None, hover\_data=None, title=None, template=None, width=None, height=None)

In [4]:

import plotly.express as px

# using the iris dataset

df = px.data.iris()

# plotting the line chart

fig = px.line(df, y="sepal\_width")

#plot(data, config={'displayModeBar'=True})

# showing the plot

fig.show()

In the above example, we can see that –

The labels to the x-axis and y-axis have given automatically by plotly.

The data of the x-axis and y-axis is shown on hover.

We can also select a part of the data according to our needs and can also zoom out.

Plotly also provides a set of tools (seen on the top right corner) to interact with every chart.

Poltly also allows us to save the graph locally in a static format.

Now let’s try to customize our graph a little.

Example 1: In this example, we will use the line dash parameter which is used to group the lines according to the dataframe column passed.

In [5]:

import plotly.express as px

# using the iris dataset

df = px.data.iris()

# plotting the line chart

fig = px.line(df, y="sepal\_width", line\_group='species')

# showing the plot

fig.show()

Example 2: In this example, we will group and color the data according to the species. We will also change the line format. For this we will use two attributes such – line\_dash and color.

In [6]:

import plotly.express as px

# using the iris dataset

df = px.data.iris()

# plotting the line chart

fig = px.line(df, y="sepal\_width", line\_dash='species',

color='species')

# showing the plot

fig.show()

Bar Chart

A bar chart is a pictorial representation of data that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. In other words, it is the pictorial representation of dataset. These data sets contain the numerical values of variables that represent the length or height. It can be created using the px.bar() method.

Syntax:

plotly.express.bar(data\_frame=None, x=None, y=None, color=None, facet\_row=None, facet\_col=None, facet\_col\_wrap=0, hover\_name=None, hover\_data=None, custom\_data=None, text=None, error\_x=None, error\_x\_minus=None, error\_y=None, error\_y\_minus=None, title=None, template=None, width=None, height=None, \*\*kwargs)

In [7]:

import plotly.express as px

# Loading the data

df = px.data.tips()

# Creating the bar chart

fig = px.bar(df, x='day', y="total\_bill")

fig.show()

In [8]:

df.head()

Out[8]:

|  | **total\_bill** | **tip** | **sex** | **smoker** | **day** | **time** | **size** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| **1** | 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| **2** | 21.01 | 3.50 | Male | No | Sun | Dinner | 3 |
| **3** | 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| **4** | 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |

Let’s try to customize this plot. Customizations that we will use –

color: Used to color the bars.

facet\_row: Divides the graph into rows according to the data passed

facet\_col: Divides the graph into columns according to the data passed

Example:

In [9]:

import plotly.express as px

# Loading the data

df = px.data.tips()

# Creating the bar chart

fig = px.bar(df, x='day', y="total\_bill", color='sex', facet\_row='time', facet\_col='sex')

fig.show()

Scatter Plot

A scatter plot is a set of dotted points to represent individual pieces of data in the horizontal and vertical axis. A graph in which the values of two variables are plotted along X-axis and Y-axis, the pattern of the resulting points reveals a correlation between them. it can be created using the px.scatter() method.

Syntax:

plotly.express.scatter(data\_frame=None, x=None, y=None, color=None, symbol=None, size=None, hover\_name=None, hover\_data=None, facet\_row=None, facet\_col=None, facet\_col\_wrap=0, opacity=None, title=None, template=None, width=None, height=None, \*\*kwargs)

In [10]:

import plotly.express as px

# using the dataset

df = px.data.tips()

# plotting the scatter chart

fig = px.scatter(df, x='total\_bill', y="tip")

# showing the plot

fig.show()

Let’s see various customizations available for this chart that we will use –

color: Color the points.

symbol: Gives a symbol to each point according to the data passed.

size: The size for each point.

In [11]:

import plotly.express as px

# using the dataset

df = px.data.tips()

# plotting the scatter chart

fig = px.scatter(df, x='total\_bill', y="tip", color='time',

symbol='sex', size='size', facet\_row='day',

facet\_col='time')

# showing the plot

fig.show()

Histogram

A histogram is basically used to represent data in the form of some groups. It is a type of bar plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency. It can be created using the px.histogram() method.

Syntax:

plotly.express.histogram(data\_frame=None, x=None, y=None, color=None, facet\_row=None, facet\_col=None, barnorm=None, histnorm=None, nbins=None, title=None, template=None, width=None, height=None)

In [12]:

import plotly.express as px

# using the dataset

df = px.data.tips()

# plotting the histogram

fig = px.histogram(df, x="total\_bill")

# showing the plot

fig.show()

Let’s customize the above graph. Customizations that we will be using are –

color: To color the bars

nbins: To set the number of bins

histnorm: Mode through which the bins are represented. Different values that can be passed using this argument are-

percent or probability: The output of histfunc for a given bin is divided by the sum of the output of histfunc for all bins.

density: The output of histfunc for a given bin is divided by the size of the bin.

probability density: The output of histfunc for a given bin is normalized such that it corresponds to the probability that a random

barmode: Can be either ‘group’, ‘overlay’ or ‘relative’.

group: Bars are stacked above zero for positive values and below zero for negative values

overlay: Bars are drawn on the top of each other

group: Bars are placed beside each other.

Example:

In [13]:

import plotly.express as px

# using the dataset

df = px.data.tips()

# plotting the histogram

fig = px.histogram(df, x="total\_bill", color='sex',

nbins=50, histnorm='percent',

barmode='group')

# showing the plot

fig.show()

Pie Chart

A pie chart is a circular statistical graphic, which is divided into slices to illustrate numerical proportions. It depicts a special chart that uses “pie slices”, where each sector shows the relative sizes of data. A circular chart cuts in a form of radii into segments describing relative frequencies or magnitude also known as a circle graph. It can be created using the px.pie() method.

Syntax:

plotly.express.pie(data\_frame=None, names=None, values=None, color=None, color\_discrete\_sequence=None, color\_discrete\_map={}, hover\_name=None, hover\_data=None, custom\_data=None, labels={}, title=None, template=None, width=None, height=None, opacity=None, hole=None)

In [14]:

import plotly.express as px

# Loading the iris dataset

df = px.data.tips()

fig = px.pie(df, values="total\_bill", names="day")

fig.show()

Let’s customize the above graph. Customizations that we will be using are –

color\_discrete\_sequence: Strings defining valid CSS colors

opacity: Opacity for markers. The value should be between 0 and 1

hole: Creates a hole in between to make it a donut chart. The value should be between 0 and 1

Example:

In [15]:

import plotly.express as px

# Loading the iris dataset

df = px.data.tips()

fig = px.pie(df, values="total\_bill", names="day",

color\_discrete\_sequence=px.colors.sequential.RdBu,

opacity=0.8, hole=0.5)

fig.show()

Box Plot

A Box Plot is also known as Whisker plot is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum. In the box plot, a box is created from the first quartile to the third quartile, a vertical line is also there which goes through the box at the median. Here x-axis denotes the data to be plotted while the y-axis shows the frequency distribution. It can be created using the px.box() method

Syntax:

plotly.express.box(data\_frame=None, x=None, y=None, color=None, facet\_row=None, facet\_col=None, title=None, template=None, width=None, height=None, \*\*kwargs)

In [16]:

import plotly.express as px

# using the dataset

df = px.data.tips()

# plotting the boxplot

fig = px.box(df, x="day", y="tip")

# showing the plot

fig.show()

Let’s see various customizations that can be used on boxplots –

color: used to assign color to marks

facet\_row: assign marks to facetted subplots in the vertical direction

facet\_col: assign marks to facetted subplots in the horizontal direction

boxmode: One of ‘group’ or ‘overlay’ In ‘overlay’ mode, boxes are on drawn top of one another. In ‘group’ mode, boxes are placed beside each other.

notched: If True, boxes are drawn with notches

Example:

In [17]:

import plotly.express as px

# using the dataset

df = px.data.tips()

# plotting the boxplot

fig = px.box(df, x="day", y="tip", color='sex',

facet\_row='time', boxmode='group',

notched=True)

# showing the plot

fig.show()

Violin Plot

Violin Plot is a method to visualize the distribution of numerical data of different variables. It is similar to Box Plot but with a rotated plot on each side, giving more information about the density estimate on the y-axis. The density is mirrored and flipped over and the resulting shape is filled in, creating an image resembling a violin. The advantage of a violin plot is that it can show nuances in the distribution that aren’t perceptible in a boxplot. On the other hand, the boxplot more clearly shows the outliers in the data. It can be created using the px.violin() method.

Syntax:

violin(data\_frame=None, x=None, y=None, color=None, facet\_row=None, facet\_col=None, facet\_col\_wrap=0, facet\_row\_spacing=None, facet\_col\_spacing=None, hover\_name=None, hover\_data=None, title=None, template=None, width=None, height=None, \*\*kwargs)

In [18]:

import plotly.express as px

# using the dataset

df = px.data.tips()

# plotting the violin plot

fig = px.violin(df, x="day", y="tip")

# showing the plot

fig.show()

For customizing the violin plot we will use the same customizations available for the box plot except the boxmode and notched which are not available for the violin plot. We will also the box parameter. Setting this parameter to True will show a box plot inside the violin plot.

Example:

In [19]:

import plotly.express as px

# using the dataset

df = px.data.tips()

# plotting the violin plot

fig = px.violin(df, x="day", y="tip", color='sex',

facet\_row='time', box=True)

# showing the plot

fig.show()

3D Scatter Plot

3D Scatter Plot can plot two-dimensional graphics that can be enhanced by mapping up to three additional variables while using the semantics of hue, size, and style parameters. All the parameter control visual semantic which are used to identify the different subsets. Using redundant semantics can be helpful for making graphics more accessible. It can be created using the scatter\_3d function of plotly.express class.

Syntax:

plotly.express.scatter\_3d(data\_frame=None, x=None, y=None, z=None, color=None, symbol=None, size=None, range\_x=None, range\_y=None, range\_z=None, title=None, template=None, width=None, height=None, \*\*kwargs)

In [20]:

import plotly.express as px

# data to be plotted

df = px.data.tips()

# plotting the figure

fig = px.scatter\_3d(df, x="total\_bill", y="sex", z="tip")

fig.show()

Customizing the 3D scatter plot. We will use the following customization –

color: Set the color of the markers

size: Set the size of the marker

symbol: Set the symbol of the plot

Example:

In [21]:

import plotly.express as px

# data to be plotted

df = px.data.tips()

# plotting the figure

fig = px.scatter\_3d(df, x="total\_bill", y="sex", z="tip", color='day',

size='total\_bill', symbol='time')

fig.show()

Adding interaction to the plot

Every graph created by the plotly provides various interactions, where we can select a part of the plot, we get information on hovering over the plot, and also a toolbar is also created with every plot that can many tasks like saving the plot locally or zooming in and out, etc. Besides all these plotly allows us to add more tools like dropdowns, buttons, sliders, etc. These can be created using the update menu attribute of the plot layout. Let’s see how to do all such things in detail.

Dropdown Menu

A drop-down menu is a part of the menu button which is displayed on a screen all the time. Every menu button is associated with a Menu widget that can display the choices for that menu button when clicked on it. In plotly, there are 4 possible methods to modify the charts by using update menu method.

restyle: modify data or data attributes

relayout: modify layout attributes

update: modify data and layout attributes

animate: start or pause an animation

Example:

In [25]:

import plotly.graph\_objects as px

import numpy as np

import pandas as pd

df = pd.read\_csv('D:/D Backup/Datasets/tips.csv')

plot = px.Figure(data=[px.Scatter(

x=df['day'],

y=df['tip'],

mode='markers',)

])

# Add dropdown

plot.update\_layout(

updatemenus=[

dict(buttons=list([

dict(

args=["type", "scatter"],

label="Scatter Plot",

method="restyle"

),

dict(

args=["type", "bar"],

label="Bar Chart",

method="restyle"

)

]),

direction="down",

),

]

)

plot.show()

Adding Buttons

In plotly, actions custom Buttons are used to quickly make actions directly from a record. Custom Buttons can be added to page layouts in CRM, Marketing, and Custom Apps. There are also 4 possible methods that can be applied in custom buttons:

restyle: modify data or data attributes

relayout: modify layout attributes

update: modify data and layout attributes

animate: start or pause an animation

In [29]:

import plotly.graph\_objects as px

import pandas as pd

# reading the database

data = pd.read\_csv("D:/D Backup/Datasets/tips.csv")

plot = px.Figure(data=[px.Scatter(

x=data['day'],

y=data['tip'],

mode='markers',)

])

# Add dropdown

plot.update\_layout(

updatemenus=[

dict(

type="buttons",

direction="left",

buttons=list([

dict(

args=["type", "scatter"],

label="Scatter Plot",

method="restyle"

),

dict(

args=["type", "bar"],

label="Bar Chart",

method="restyle"

),

dict(

args=["type", "histogram"],

label="Histogram",

method="restyle"

)

]),

),

]

)

plot.show()

Creating Sliders and Selectors to the Plot

In plotly, the range slider is a custom range-type input control. It allows selecting a value or a range of values between a specified minimum and maximum range. And the range selector is a tool for selecting ranges to display within the chart. It provides buttons to select pre-configured ranges in the chart. It also provides input boxes where the minimum and maximum dates can be manually input.

Example:

In [31]:

import plotly.graph\_objects as px

import plotly.express as go

import numpy as np

#df = go.data.tips()

df = pd.read\_csv("D:/D Backup/Datasets/tips.csv")

x = df['total\_bill']

y = df['tip']

plot = px.Figure(data=[px.Scatter(

x=x,

y=y,

mode='markers',)

])

plot.update\_layout(

xaxis=dict(

rangeselector=dict(

buttons=list([

dict(count=1,

step="day",

stepmode="backward"),

])

),

rangeslider=dict(

visible=True

),

)

)

plot.show()

In [ ]:

PRACTICAL 2

Data visualization allows us to quickly interpret the data and adjust different variables to see their effect

•Technology is increasingly making it easier for us to do so

Why visualize data?

o Observe the patterns

o Identify extreme values that could be anomalies

o Easy interpretation

#### Popular plotting libraries in Python[¶](#Popular-plotting-libraries-in-Python)

Python offers multiple graphing libraries that offers diverse features

• 1) matplotlib --> to create 2D graphs and plots •

2) pandas visualization --> easy to use interface, built on Matplotlib •

3) seaborn --> provides a high level interface for drawing attractive and informative statistical graphics •

4) ggplot --> based on R’s ggplot2, uses Grammar of Graphics •

5) plotly --> can create interactive plots

# Scatter Plot[¶](#Scatter-Plot)

# What is a scatter plot?[¶](#What-is-a-scatter-plot?)

A scatter plot is a set of points that represents the values obtained for two different variables plotted on a horizontal and vertical axes

When to use scatter plots?

Scatter plots are used to convey the relationship between two numerical variables

Scatter plots are sometimes called correlation plots because they show how two variables are correlated

In [1]:

import matplotlib.pyplot as plt

# create a figure and axis

fig, ax = plt.subplots()

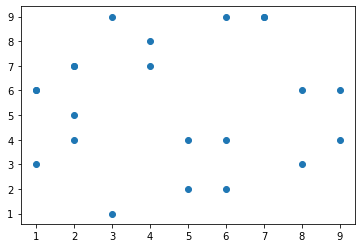
x = [2, 4, 6, 6, 9, 2, 7, 2, 6, 1, 8, 4, 5, 9, 1, 2, 3, 7, 5, 8, 1, 3]

y = [7, 8, 2, 4, 6, 4, 9, 5, 9, 3, 6, 7, 2, 4, 6, 7, 1, 9, 4, 3, 6, 9]

ax.scatter(x, y)

Out[1]:

<matplotlib.collections.PathCollection at 0x1108ae5c608>



In [2]:

import pandas as pd

iris = pd.read\_csv('D:/AnitaRJ/DATA SCIENCE/MScI\_DataSci\_Practicals/Practical7/iris.csv', names=['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width', 'class'])

print(iris.head())

sepal\_length sepal\_width petal\_length petal\_width class

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species

1 5.1 3.5 1.4 0.2 Iris-setosa

2 4.9 3.0 1.4 0.2 Iris-setosa

3 4.7 3.2 1.3 0.2 Iris-setosa

4 4.6 3.1 1.5 0.2 Iris-setosa

In [3]:

import matplotlib.pyplot as plt

# create a figure and axis

fig, ax = plt.subplots()

# scatter the sepal\_length against the sepal\_width

ax.scatter(iris['sepal\_length'], iris['sepal\_width'])

# set a title and labels

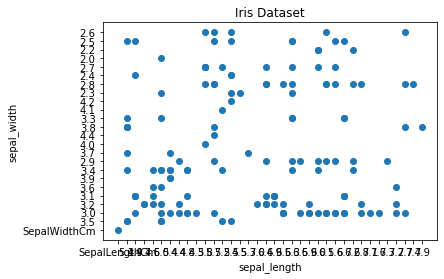
ax.set\_title('Iris Dataset')

ax.set\_xlabel('sepal\_length')

ax.set\_ylabel('sepal\_width')

Out[3]:

Text(0, 0.5, 'sepal\_width')



In [8]:

import pandas as pd

cars\_data=pd.read\_csv('D:/AnitaRJ/DATA SCIENCE/MScI\_DataSci\_Practicals/Practical7/Toyota.csv',index\_col=0)

cars\_data.head()

Out[8]:

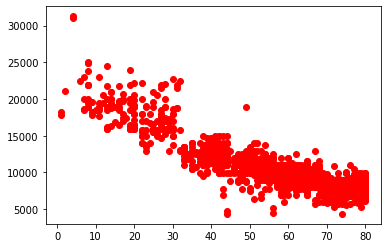
|  | **Price** | **Age** | **KM** | **FuelType** | **HP** | **MetColor** | **Automatic** | **CC** | **Doors** | **Weight** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 13500 | 23.0 | 46986 | Diesel | 90 | 1.0 | 0 | 2000 | three | 1165 |
| **1** | 13750 | 23.0 | 72937 | Diesel | 90 | 1.0 | 0 | 2000 | 3 | 1165 |
| **2** | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| **3** | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| **4** | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |

In [10]:

import matplotlib.pyplot as plt

plt.scatter(cars\_data['Age'],cars\_data['Price'], c='red')

plt.show()



# Line Chart[¶](#Line-Chart)

In Matplotlib we can create a line chart by calling the plot method. We can also plot multiple columns in one graph, by looping through the columns we want and plotting each column on the same axis.

In [8]:

import pandas as pd

import numpy as np

from matplotlib import pyplot as plt

x=range(1,6)

y=np.random.randint(1,20,5)

plt.plot(x,y)

plt.xticks(x)

plt.yticks(y)

Out[8]:

([<matplotlib.axis.YTick at 0x12989b8e448>,

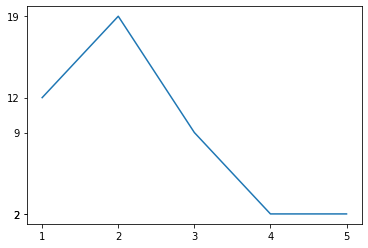
<matplotlib.axis.YTick at 0x129898e4f88>,

<matplotlib.axis.YTick at 0x12989b88408>,

<matplotlib.axis.YTick at 0x12989bb92c8>,

<matplotlib.axis.YTick at 0x12989bb99c8>],

<a list of 5 Text yticklabel objects>)



In [7]:

import matplotlib.pyplot as plt

# create a figure and axis

fig, ax = plt.subplots()

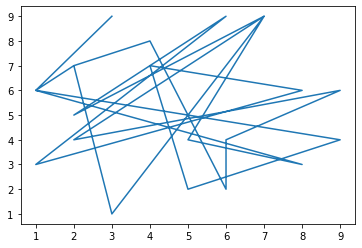
x = [2, 4, 6, 6, 9, 2, 7, 2, 6, 1, 8, 4, 5, 9, 1, 2, 3, 7, 5, 8, 1, 3]

y = [7, 8, 2, 4, 6, 4, 9, 5, 9, 3, 6, 7, 2, 4, 6, 7, 1, 9, 4, 3, 6, 9]

ax.plot(x,y)

Out[7]:

[<matplotlib.lines.Line2D at 0x12989b27088>]



In [35]:

import pandas as pd

df = pd.DataFrame({

'name':['john','mary','peter','jeff','bill','lisa','jose'],

'age':[23,78,22,19,45,33,20],

'gender':['M','F','M','M','M','F','M'],

'state':['california','dc','california','dc','california','texas','texas'],

'num\_children':[2,0,0,3,2,1,4],

'num\_pets':[5,1,0,5,2,2,3]

})

# From pandas to plot multiple plots on same figure

# gca stands for 'get current axis'

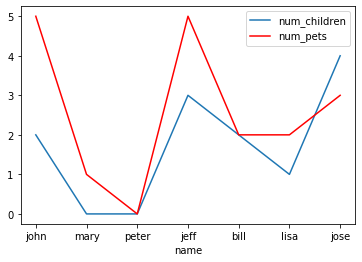
ax = plt.gca()

df.plot(kind='line',x='name',y='num\_children',ax=ax)

df.plot(kind='line',x='name',y='num\_pets', color='red',ax=ax)

Out[35]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x11090c193c8>



In [9]:

import pandas as pd

iris = pd.read\_csv('D:/AnitaRJ/DATA SCIENCE/MScI\_DataSci\_Practicals/Practical7/iris.csv', names=['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width', 'class'])

print(iris.head())

sepal\_length sepal\_width petal\_length petal\_width class

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species

1 5.1 3.5 1.4 0.2 Iris-setosa

2 4.9 3.0 1.4 0.2 Iris-setosa

3 4.7 3.2 1.3 0.2 Iris-setosa

4 4.6 3.1 1.5 0.2 Iris-setosa

In [5]:

# get columns to plot

columns = iris.columns.drop(['class'])

# create x data

x\_data = range(0, iris.shape[0])

# create figure and axis

fig, ax = plt.subplots()

# plot each column

for column in columns:

ax.plot(x\_data, iris[column], label=column)

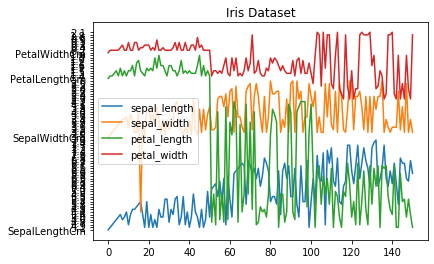
# set title and legend

ax.set\_title('Iris Dataset')

ax.legend()

Out[5]:

<matplotlib.legend.Legend at 0x1108aaf9888>



# Histogram[¶](#Histogram)

In Matplotlib we can create a Histogram using the hist method. If we pass it categorical data like the points column from the wine-review dataset it will automatically calculate how often each class occurs.

In [6]:

# create figure and axis

fig, ax = plt.subplots()

# plot histogram

ax.hist(iris['sepal\_length'])

# set title and labels

ax.set\_title('iris')

ax.set\_xlabel('sepal\_length')

ax.set\_ylabel('Frequency')

Out[6]:

Text(0, 0.5, 'Frequency')



# Bar Chart[¶](#Bar-Chart)

A bar chart can be created using the bar method. The bar-chart isn’t automatically calculating the frequency of a category so we are going to use pandas value\_counts function to do this. The bar-chart is useful for categorical data that doesn’t have a lot of different categories (less than 30) because else it can get quite messy.

In [8]:

wine\_reviews = pd.read\_csv('D:/AnitaRJ/DATA SCIENCE/MScI\_DataSci\_Practicals/Practical7/winemag-data-130k-v2.csv', index\_col=0)

wine\_reviews.head()

Out[8]:

|  | **country** | **description** | **designation** | **points** | **price** | **province** | **region\_1** | **region\_2** | **taster\_name** | **taster\_twitter\_handle** | **title** | **variety** | **winery** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Italy | Aromas include tropical fruit, broom, brimston... | Vulkà Bianco | 87 | NaN | Sicily & Sardinia | Etna | NaN | Kerin O’Keefe | @kerinokeefe | Nicosia 2013 Vulkà Bianco (Etna) | White Blend | Nicosia |
| **1** | Portugal | This is ripe and fruity, a wine that is smooth... | Avidagos | 87 | 15.0 | Douro | NaN | NaN | Roger Voss | @vossroger | Quinta dos Avidagos 2011 Avidagos Red (Douro) | Portuguese Red | Quinta dos Avidagos |
| **2** | US | Tart and snappy, the flavors of lime flesh and... | NaN | 87 | 14.0 | Oregon | Willamette Valley | Willamette Valley | Paul Gregutt | @paulgwine | Rainstorm 2013 Pinot Gris (Willamette Valley) | Pinot Gris | Rainstorm |
| **3** | US | Pineapple rind, lemon pith and orange blossom ... | Reserve Late Harvest | 87 | 13.0 | Michigan | Lake Michigan Shore | NaN | Alexander Peartree | NaN | St. Julian 2013 Reserve Late Harvest Riesling ... | Riesling | St. Julian |
| **4** | US | Much like the regular bottling from 2012, this... | Vintner's Reserve Wild Child Block | 87 | 65.0 | Oregon | Willamette Valley | Willamette Valley | Paul Gregutt | @paulgwine | Sweet Cheeks 2012 Vintner's Reserve Wild Child... | Pinot Noir | Sweet Cheeks |

In [9]:

#Bar Chart

# create a figure and axis

fig, ax = plt.subplots()

# count the occurrence of each class

data = wine\_reviews['points'].value\_counts()

# get x and y data

points = data.index

frequency = data.values

# create bar chart

ax.bar(points, frequency)

# set title and labels

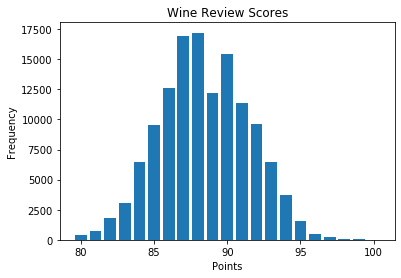
ax.set\_title('Wine Review Scores')

ax.set\_xlabel('Points')

ax.set\_ylabel('Frequency')

Out[9]:

Text(0, 0.5, 'Frequency')

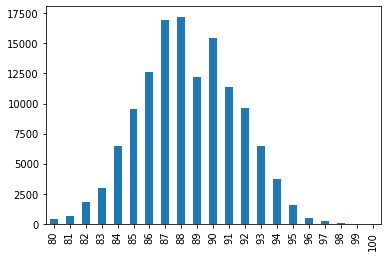


In [10]:

wine\_reviews['points'].value\_counts().sort\_index().plot.bar()

Out[10]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x1108ea89088>



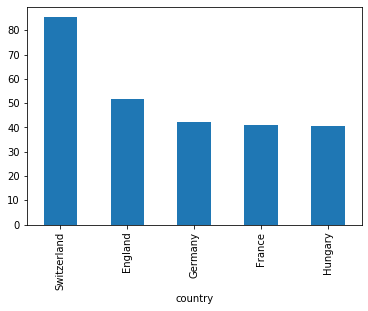
We can also plot other data then the number of occurrences.

In [12]:

wine\_reviews.groupby("country").price.mean().sort\_values(ascending=False)[:5].plot.bar()

Out[12]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x1108eb548c8>



Adding more charecteristics to bar graph

In [32]:

import numpy as np

import matplotlib.pyplot as plt

objects = ('Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp')

y\_pos = np.arange(len(objects))

performance = [10,8,6,4,2,1]

# Bar Chart

# X Axis positions as first parameter list, it can be floating point numbers also

# Y Values as 2nd parameter list

# Alpha is transparency,

# Align can be center or edge

# Color can be single value or a list of color codes, one for each bar.

plt.bar(y\_pos, performance, width=0.5, align='center', alpha=0.5, color=['r', 'r', 'g', 'g', 'b', 'b'])

# To define labels for x axis values.

plt.xticks(y\_pos, objects)

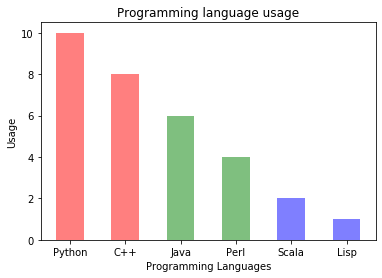
plt.ylabel('Usage')

plt.xlabel('Programming Languages')

plt.title('Programming language usage')

Out[32]:

Text(0.5, 1.0, 'Programming language usage')



In [26]:

# Importing the matplotlib library

import matplotlib.pyplot as plt

# Declaring the figure or the plot (y, x) or (width, height)

plt.figure(figsize = (12,7))

# Categorical data: Country names

countries = ['USA', 'Brazil', 'Russia', 'Spain', 'UK', 'India']

# Integer value interms of death counts

totalDeaths = [112596, 37312, 5971, 27136, 40597, 7449]

# Passing the parameters to the bar function, this is the main function which creates the bar plot

plt.bar(countries, totalDeaths, width= 0.9, align='center',color='cyan', edgecolor = 'red')

# This is the location for the annotated text

i = 1.0

j = 2000

# Annotating the bar plot with the values (total death count)

for i in range(len(countries)):

plt.annotate(totalDeaths[i], (-0.1 + i, totalDeaths[i] + j))

# Creating the legend of the bars in the plot

plt.legend(labels = ['Total Deaths'])

# Giving the tilte for the plot

plt.title("Bar plot representing the total deaths by top 6 countries due to coronavirus")

# Namimg the x and y axis

plt.xlabel('Countries')

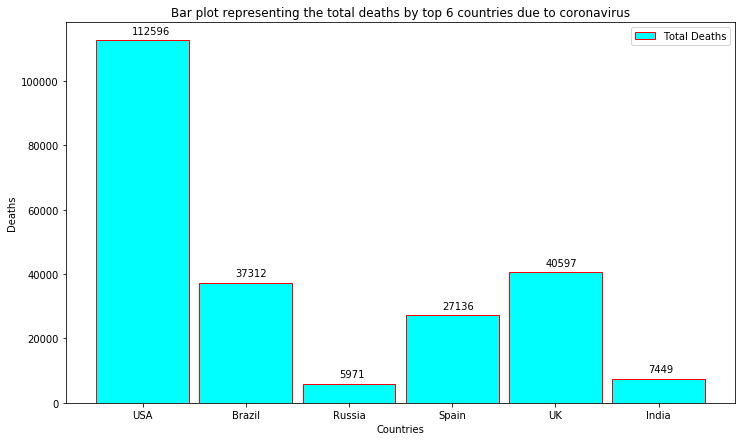
plt.ylabel('Deaths')

# Saving the plot as a 'png'

plt.savefig('1BarPlot.png')

# Displaying the bar plot

plt.show()



Horizontal bar plot

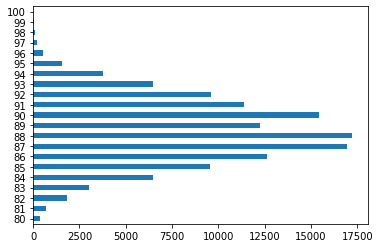
It’s also really simple to make a horizontal bar-chart using the plot.barh() method. By adding one extra character ‘h’, we can align the bars horizontally. Also, we can represent the bars in two or more different colors, this will increase the readability of the plots.

In [11]:

wine\_reviews['points'].value\_counts().sort\_index().plot.barh()

Out[11]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x1108c825248>



In [28]:

# Importing the matplotlib library

import matplotlib.pyplot as plt

# Declaring the figure or the plot (y, x) or (width, height)

plt.figure(figsize=[14, 10])

# Passing the parameters to the bar function, this is the main function which creates the bar plot

# For creating the horizontal make sure that you append 'h' to the bar function name

plt.barh(['USA', 'Brazil', 'Russia', 'Spain', 'UK'], [2026493, 710887, 476658, 288797, 287399], label = "Danger zone", color = 'r')

plt.barh(['India', 'Italy', 'Peru', 'Germany', 'Iran'], [265928, 235278, 199696, 186205, 173832], label = "Not safe zone", color = 'g')

# Creating the legend of the bars in the plot

plt.legend()

# Namimg the x and y axis

plt.xlabel('Total cases')

plt.ylabel('Countries')

# Giving the tilte for the plot

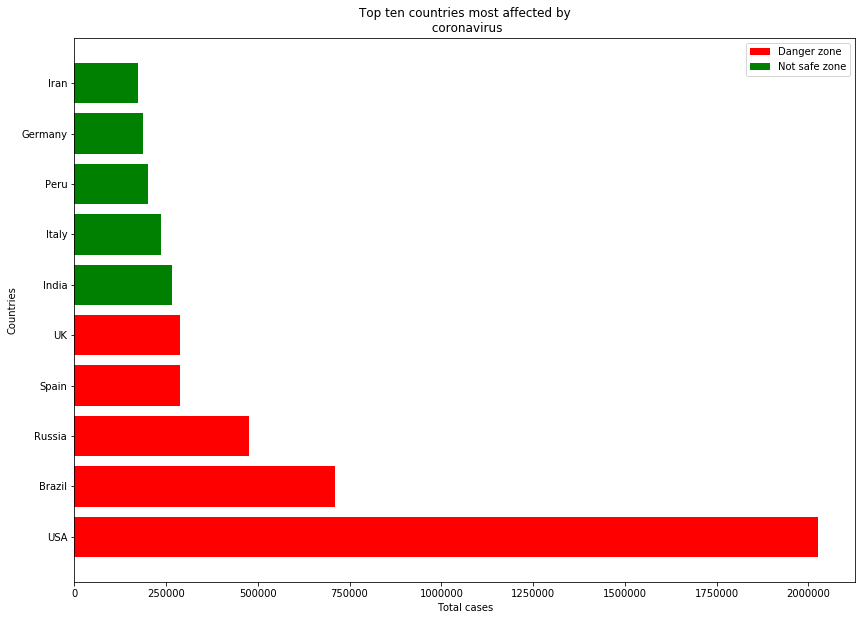
plt.title('Top ten countries most affected by\n coronavirus')

# Saving the plot as a 'png'

plt.savefig('2BarPlot.png')

# Displaying the bar plot

plt.show()



Stacking two bar plots on top of each other

At times you might want to stack two or more bar plots on top of each other. With the help of this, you can differentiate two separate quantities visually. To do this just follow.

In [37]:

import pandas as pd

df = pd.DataFrame({

'name':['john','mary','peter','jeff','bill','lisa','jose'],

'age':[23,78,22,19,45,33,20],

'gender':['M','F','M','M','M','F','M'],

'state':['california','dc','california','dc','california','texas','texas'],

'num\_children':[2,0,0,3,2,1,4],

'num\_pets':[5,1,0,5,2,2,3]

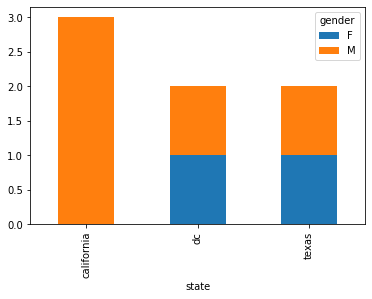
})

# From pandas to plot multiple plots on same figure

df.groupby(['state','gender']).size().unstack().plot(kind='bar', stacked=True)

Out[37]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x1109171ea88>

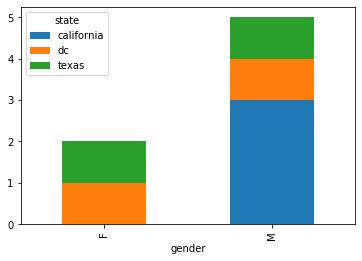


In [38]:

df.groupby(['gender','state']).size().unstack().plot(kind='bar',stacked=True)

Out[38]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x1108ee54508>



In [30]:

# Importing the matplotlib library

import matplotlib.pyplot as plt

# Declaring the figure or the plot (y, x) or (width, height)

plt.figure(figsize=[15, 5])

# Categorical data: Country names

countries = ['USA', 'Brazil', 'Russia', 'Spain', 'UK', 'India']

# Integer value interms of total cases

totalCases = (2026493, 710887, 476658, 288797, 287399, 265928)

# Integer value interms of death counts

totalDeaths = (113055, 37312, 5971, 27136, 40597, 7473)

# Plotting both the total death and the total cases in a single plot. Formula total cases - total deaths

for i in range(len(countries)):

plt.bar(countries[i], totalDeaths[i], bottom = totalCases[i] - totalDeaths[i], color='black')

plt.bar(countries[i], totalCases[i] - totalDeaths[i], color='red')

# Creating the legend of the bars in the plot

plt.legend(labels = ['Total Deaths','Total Cases'])

# Giving the tilte for the plot

plt.title("Bar plot representing the total deaths and total cases country wise")

# Namimg the x and y axis

plt.xlabel('Countries')

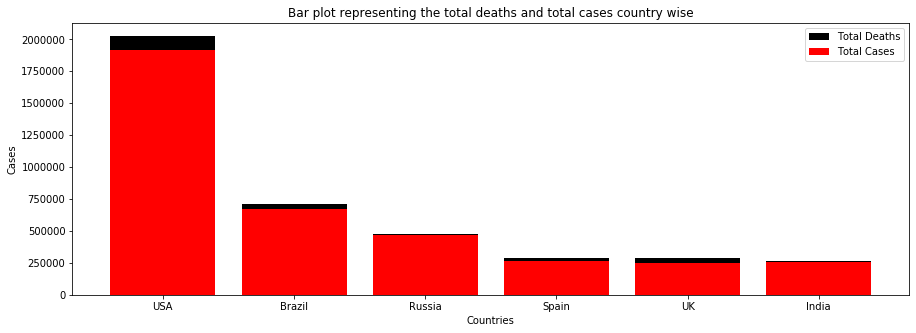
plt.ylabel('Cases')

# Saving the plot as a 'png'

plt.savefig('3BarPlot.png')

# Displaying the bar plot

plt.show()



In [ ]:

Plotting two or bar plot next to another (Grouping)

Often many-a-times you might want to group two or more plots just to represent two or more different quantities or whatever. Also in the below code, you can learn to override the name of the x-axis with the name of your choice.

In [34]:

import pandas as pd

from matplotlib import pyplot as plt

Data = {'Country': ['USA','Canada','Germany','UK','France'],

'GDP\_Per\_Capita': [45000,42000,52000,49000,47000],

'Income\_Per\_Capita': [4000,5000,7000,55000,60000]

}

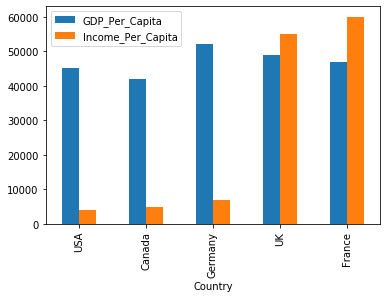
df = pd.DataFrame(Data)

# Multiple metrics in same chart

df.plot(x ='Country', y=['GDP\_Per\_Capita', 'Income\_Per\_Capita'], kind = 'bar')

Out[34]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x11090f8c048>



In [31]:

# Importing the matplotlib library

import numpy as np

import matplotlib.pyplot as plt

# Declaring the figure or the plot (y, x) or (width, height)

plt.figure(figsize=[15, 10])

# Data to be plotted

totalDeath = [113055, 37312, 5971, 7473, 33964]

totalRecovery = [773480, 325602, 230688, 129095, 166584]

activeCases = [1139958, 347973, 239999, 129360, 34730]

country = ['USA', 'Brazil', 'Russia', 'India', 'Italy']

# Using numpy to group 3 different data with bars

X = np.arange(len(totalDeath))

# Passing the parameters to the bar function, this is the main function which creates the bar plot

# Using X now to align the bars side by side

plt.bar(X, totalDeath, color = 'black', width = 0.25)

plt.bar(X + 0.25, totalRecovery, color = 'g', width = 0.25)

plt.bar(X + 0.5, activeCases, color = 'b', width = 0.25)

# Creating the legend of the bars in the plot

plt.legend(['Total Deaths', 'Total Recovery', 'Active Cases'])

# Overiding the x axis with the country names

plt.xticks([i + 0.25 for i in range(5)], country)

# Giving the tilte for the plot

plt.title("Bar plot representing the total deaths, total recovered cases and active cases country wise")

# Namimg the x and y axis

plt.xlabel('Countries')

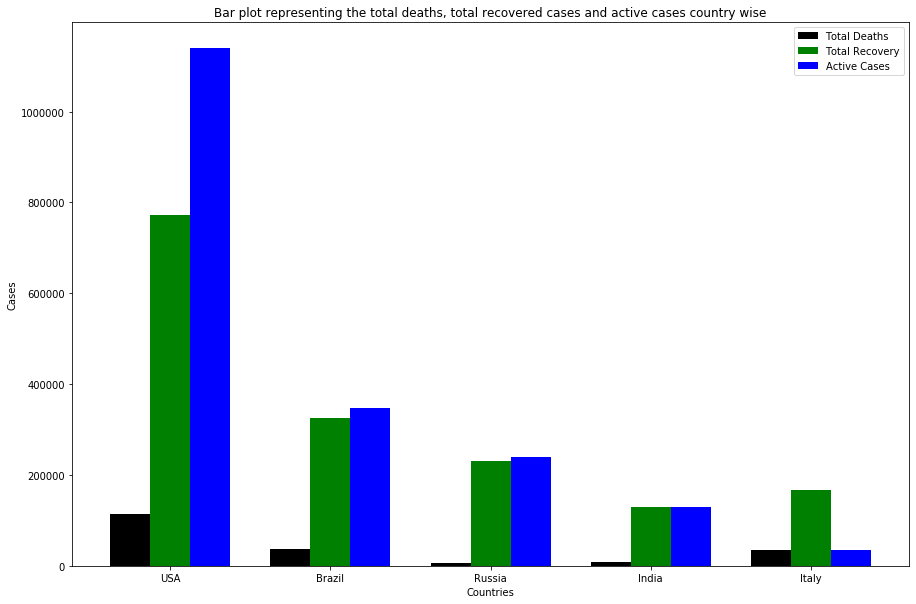
plt.ylabel('Cases')

# Saving the plot as a 'png'

plt.savefig('4BarPlot.png')

# Displaying the bar plot

plt.show()



# PRACTICAL 3

# Pie chart[¶](#Pie-chart)

A pie chart is a type of data visualization that is used to illustrate numerical proportions in data.

In [40]:

# Data Frame plotting

from pandas import DataFrame

import matplotlib.pyplot as plt

Data = {'Tasks': [300,500,700],

'Task Type' : ['Tasks Pending','Tasks Ongoing','Tasks Completed']

}

df = DataFrame(Data)

df.set\_index('Task Type', inplace=True)

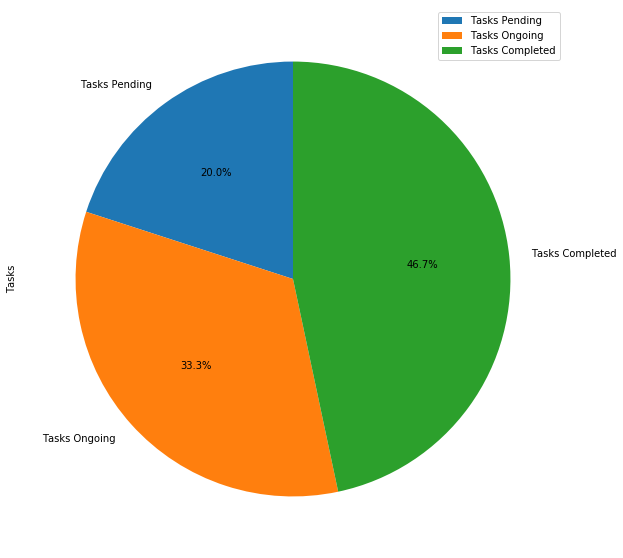
# autopct has extra % at the end as escape, as % is interpreted as formatting string begin by default.

# Only pie chart needs labels to be data frame index

df.plot.pie(y='Tasks', figsize=(10,10),autopct='%1.1f%%', startangle=90)

Out[40]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x1108eea4dc8>



We will plot this data on a pie chart with Matplotlib's ax.pie() method. The pie piece labels are defined as a list of strings, and the pie piece sizes are defined as a list of integers. The code section below builds a pie chart with four pie pieces, each pie piece labeled with a relative size auto-calculated to the nearest 10th of a percent.

In [41]:

import numpy as np

import matplotlib.pyplot as plt

# if using a Jupyter notebook, include:

%matplotlib inline

# Pie chart, where the slices will be ordered and plotted counter-clockwise:

labels = ['Civil', 'Electrical', 'Mechanical', 'Chemical']

sizes = [15, 50, 45, 10]

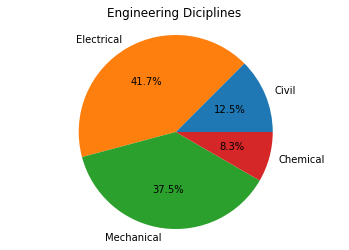
fig, ax = plt.subplots()

ax.pie(sizes, labels=labels, autopct='%1.1f%%')

ax.axis('equal') # Equal aspect ratio ensures the pie chart is circular.

ax.set\_title('Engineering Diciplines')

plt.show()



Pie pieces can be highlighted by "exploding" them out. Exploded pie pieces are applied to a Matplotlib pie chart by supplying the explode= keyword argument to the ax.pie() method. shadow=True and startangle= are two additional keyword arguments that can be passed to the ax.pie() method to control the angle and rotation of the pieces on a pie chart.

The code section below creates a pie chart with the pie pieces separated and the "Chemical" piece exploded out.

In [42]:

import numpy as np

import matplotlib.pyplot as plt

# if using a Jupyter notebook, include:

%matplotlib inline

# Pie chart, where the slices will be ordered and plotted counter-clockwise

labels = ['Civil', 'Electrical', 'Mechanical', 'Chemical']

sizes = [15, 30, 45, 10]

# Explode out the 'Chemical' pie piece by offsetting it a greater amount

explode = (0.1, 0.1, 0.1, 0.4)

fig, ax = plt.subplots()

ax.pie(sizes,

explode=explode,

labels=labels,

autopct='%1.1f%%',

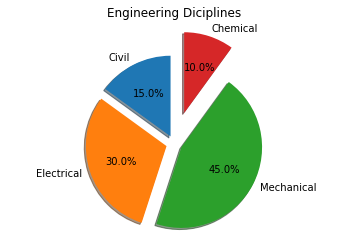
shadow=True,

startangle=90)

ax.axis('equal') # Equal aspect ratio ensures the pie chart is circular.

ax.set\_title('Engineering Diciplines')

plt.show()



# Subplots[¶](#Subplots)

In [43]:

plt.figure(figsize=(20,10))

plt.subplot(2,2,1)

plt.bar(range(1,6), np.random.randint(1,20,5))

plt.title("2,2,1")

plt.subplot(2,2,2)

plt.bar(range(1,6), np.random.randint(1,20,5))

plt.title("2,2,2")

plt.subplot(2,2,3)

# s is the size of dot

plt.scatter(range(1,6), np.random.randint(1,20,5), s=100, color="r")

plt.title("2,2,3")

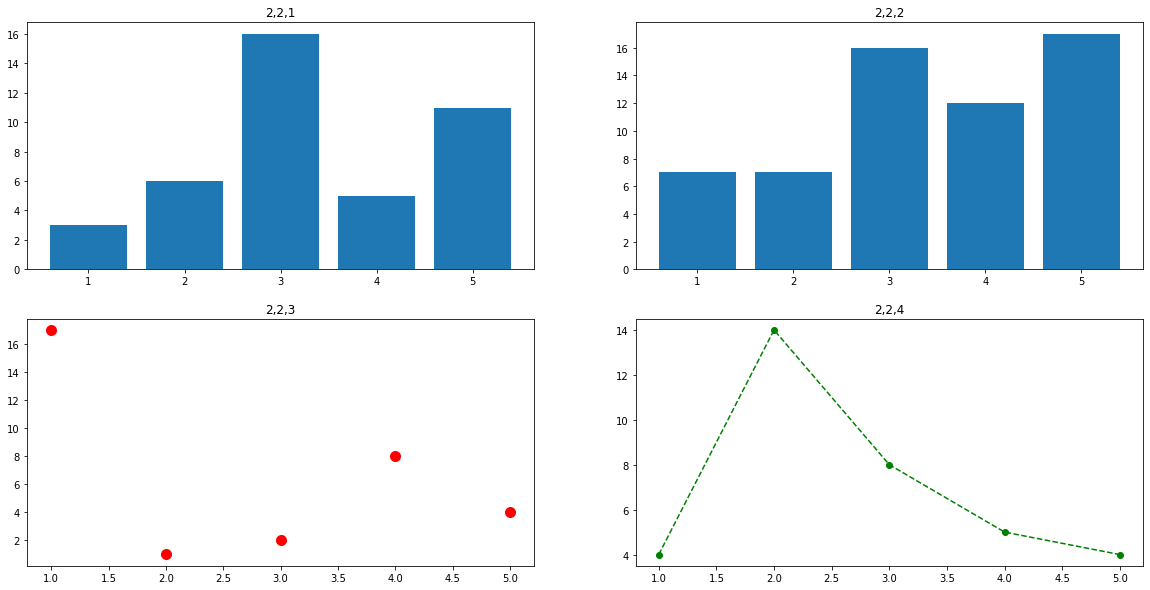
plt.subplot(2,2,4)

plt.plot(range(1,6), np.random.randint(1,20,5), marker='o', color='g', linestyle='--')

plt.title("2,2,4")

Out[43]:

Text(0.5, 1.0, '2,2,4')



In [44]:

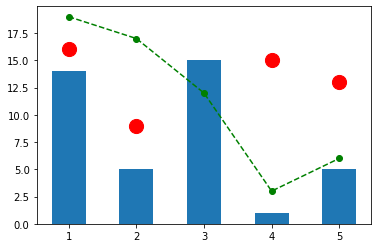
plt.bar(range(1,6), np.random.randint(1,20,5), width=0.5)

plt.scatter(range(1,6), np.random.randint(1,20,5), s=200, color="r")

plt.plot(range(1,6), np.random.randint(1,20,5), marker='o', color='g', linestyle='--')

Out[44]:

[<matplotlib.lines.Line2D at 0x1108ee23c88>]



# Seaborn[¶](#Seaborn)

• Seaborn is a Python data visualization library based on matplotlib • It provides a high level interface for drawing attractive and informative statistical graphics

In [18]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import os

In [19]:

os.chdir('D:/AnitaRJ/DATA SCIENCE/MScI\_DataSci\_Practicals/Practical7')

cars\_data=pd.read\_csv('Toyota.csv',index\_col=0,na\_values=["??","????"])

cars\_data.size

Out[19]:

14360

In [16]:

cars\_data.dropna(axis=0,inplace=True)

cars\_data.size

Out[16]:

10960

In [22]:

cars\_data=pd.read\_csv('Toyota.csv')

cars\_data.head()

Out[22]:

|  | **Unnamed: 0** | **Price** | **Age** | **KM** | **FuelType** | **HP** | **MetColor** | **Automatic** | **CC** | **Doors** | **Weight** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 13500 | 23.0 | 46986 | Diesel | 90 | 1.0 | 0 | 2000 | three | 1165 |
| **1** | 1 | 13750 | 23.0 | 72937 | Diesel | 90 | 1.0 | 0 | 2000 | 3 | 1165 |
| **2** | 2 | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| **3** | 3 | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| **4** | 4 | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |

In [23]:

cars\_data=pd.read\_csv('Toyota.csv',index\_col=0)

cars\_data.head()

Out[23]:

|  | **Price** | **Age** | **KM** | **FuelType** | **HP** | **MetColor** | **Automatic** | **CC** | **Doors** | **Weight** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 13500 | 23.0 | 46986 | Diesel | 90 | 1.0 | 0 | 2000 | three | 1165 |
| **1** | 13750 | 23.0 | 72937 | Diesel | 90 | 1.0 | 0 | 2000 | 3 | 1165 |
| **2** | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| **3** | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| **4** | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |

# Scatter plot[¶](#Scatter-plot)

Scatter plot of Price vs Age with default arguments

In [25]:

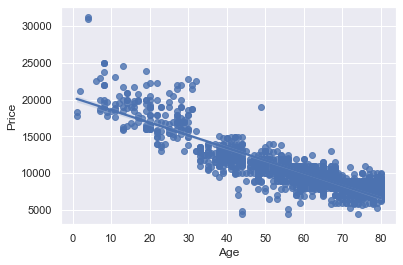
sns.set(style="darkgrid")

sns.regplot(x=cars\_data['Age'],y=cars\_data['Price'])

#It estimates and plots a regression model relating the x and y variables

Out[25]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x21787092b08>



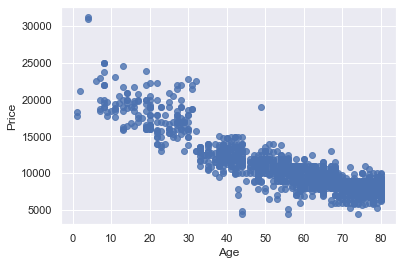
In [27]:

#Scatter plot of Price vs Age without the regression fit line

sns.regplot(x=cars\_data['Age'],y=cars\_data['Price'],fit\_reg=False)

Out[27]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x217870e4408>



In [29]:

#Scatter plot of Price vs Age by customizing the appearance of markers

sns.regplot(x=cars\_data['Age'], y=cars\_data['Price'], marker="\*", fit\_reg=False)

Out[29]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x217871da588>



In [30]:

# Scatter plot of Price vs Age by FuelType

#Using hue parameter, including another variable to show the fuel types categories with different colors

sns.lmplot(x='Age', y='Price', data=cars\_data, fit\_reg=False, hue='FuelType', legend=True, palette="Set1")

Out[30]:

<seaborn.axisgrid.FacetGrid at 0x2178722bbc8>



# Histogram[¶](#Histogram)

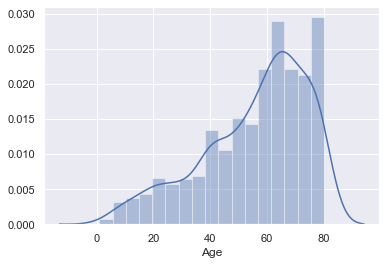
Histogram with default kernel density estimate

In [31]:

sns.distplot(cars\_data['Age'])

Out[31]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x217872a9b88>



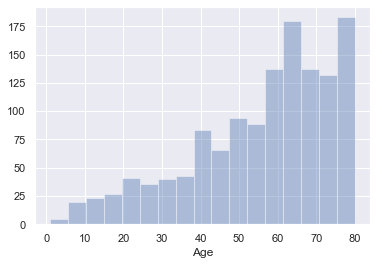
In [32]:

#Histogram without kernel density estimate

sns.distplot(cars\_data['Age'],kde=False)

Out[32]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x217871da488>



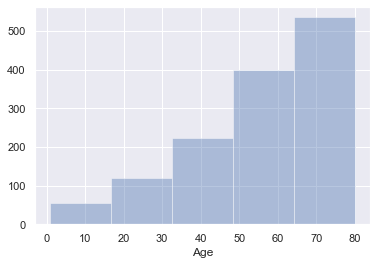
In [33]:

#Histogram with fixed no. of bins

sns.distplot(cars\_data['Age'],kde=False, bins=5)

Out[33]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x217873c21c8>



# Bar plot[¶](#Bar-plot)

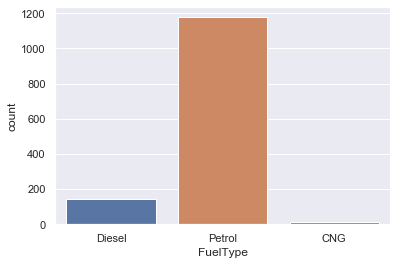
Frequency distribution of fuel type of the cars

In [34]:

sns.countplot(x="FuelType", data=cars\_data)

Out[34]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x21787434908>



In [35]:

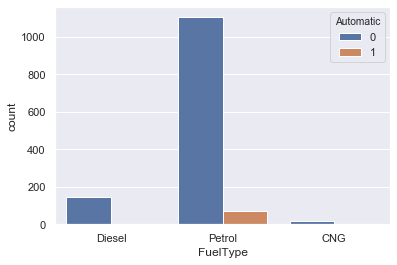
###Grouped bar plot

#Grouped bar plot of FuelType and Automatic

sns.countplot(x="FuelType", data=cars\_data, hue="Automatic")

Out[35]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x217874ab308>



In [37]:

pd.crosstab(index=cars\_data['Automatic'], columns=cars\_data['FuelType'],dropna=True)

Out[37]:

| **FuelType** | **CNG** | **Diesel** | **Petrol** |
| --- | --- | --- | --- |
| **Automatic** |  |  |  |
| **0** | 15 | 144 | 1104 |
| **1** | 0 | 0 | 73 |

# Box and whiskers plot[¶](#Box-and-whiskers-plot)

Box and whiskers plot for numerical vs categorical variable

A Box Plot is also known as Whisker plot is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum. In the box plot, a box is created from the first quartile to the third quartile, a vertical line is also there which goes through the box at the median. Here x-axis denotes the data to be plotted while the y-axis shows the frequency distribution.

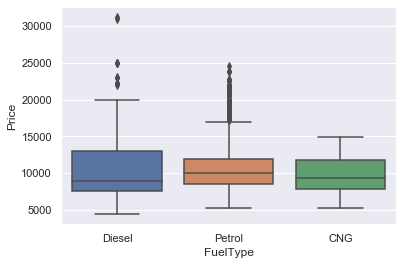
Price of the cars for various fuel types

In [38]:

sns.boxplot(x=cars\_data['FuelType'],y=cars\_data["Price"])

Out[38]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x2178750e4c8>



# Grouped box and whiskers plot[¶](#Grouped-box-and-whiskers-plot)

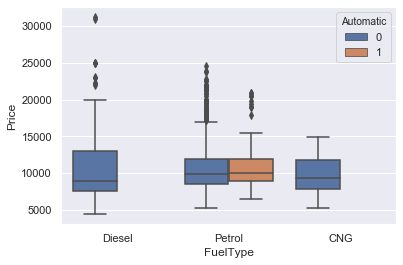
Grouped box and whiskers plot of Price vs FuelType and Automatic

In [39]:

sns.boxplot(x="FuelType", y=cars\_data["Price"],hue="Automatic",data=cars\_data)

Out[39]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x21787267dc8>



# Box[¶](#Box)

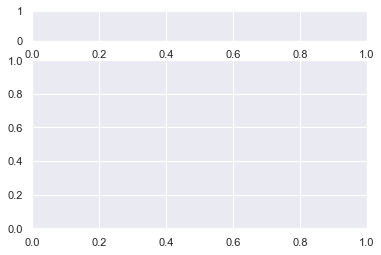
whiskers plot and Histogram

Let’s plot box whiskers plot and histogram on the same window

Split the plotting window into 2 parts

In [40]:

f,(ax\_box,ax\_hist)=plt.subplots(2,gridspec\_kw={"height\_ratios": (.15, .85)})



### Now, add create two plots[¶](#Now,-add-create-two-plots)

In [43]:

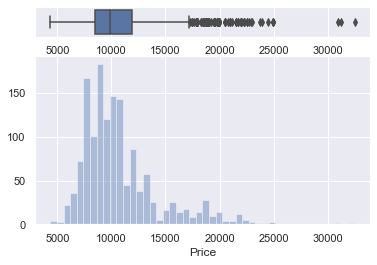
f,(ax\_box,ax\_hist)=plt.subplots(2,gridspec\_kw={"height\_ratios": (.15, .85)})

sns.boxplot(cars\_data['Price'],ax=ax\_box)

sns.distplot(cars\_data['Price'],ax=ax\_hist,kde=False)

Out[43]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x217887d0d88>



# Pairwise plots[¶](#Pairwise-plots)

It is used to plot pairwise relationships in a dataset

Creates scatterplots for joint relationships and histograms for univariate distributions

In [48]:

sns.pairplot(cars\_data,kind="scatter",hue="FuelType",diag\_kws={'bw': 0.1})

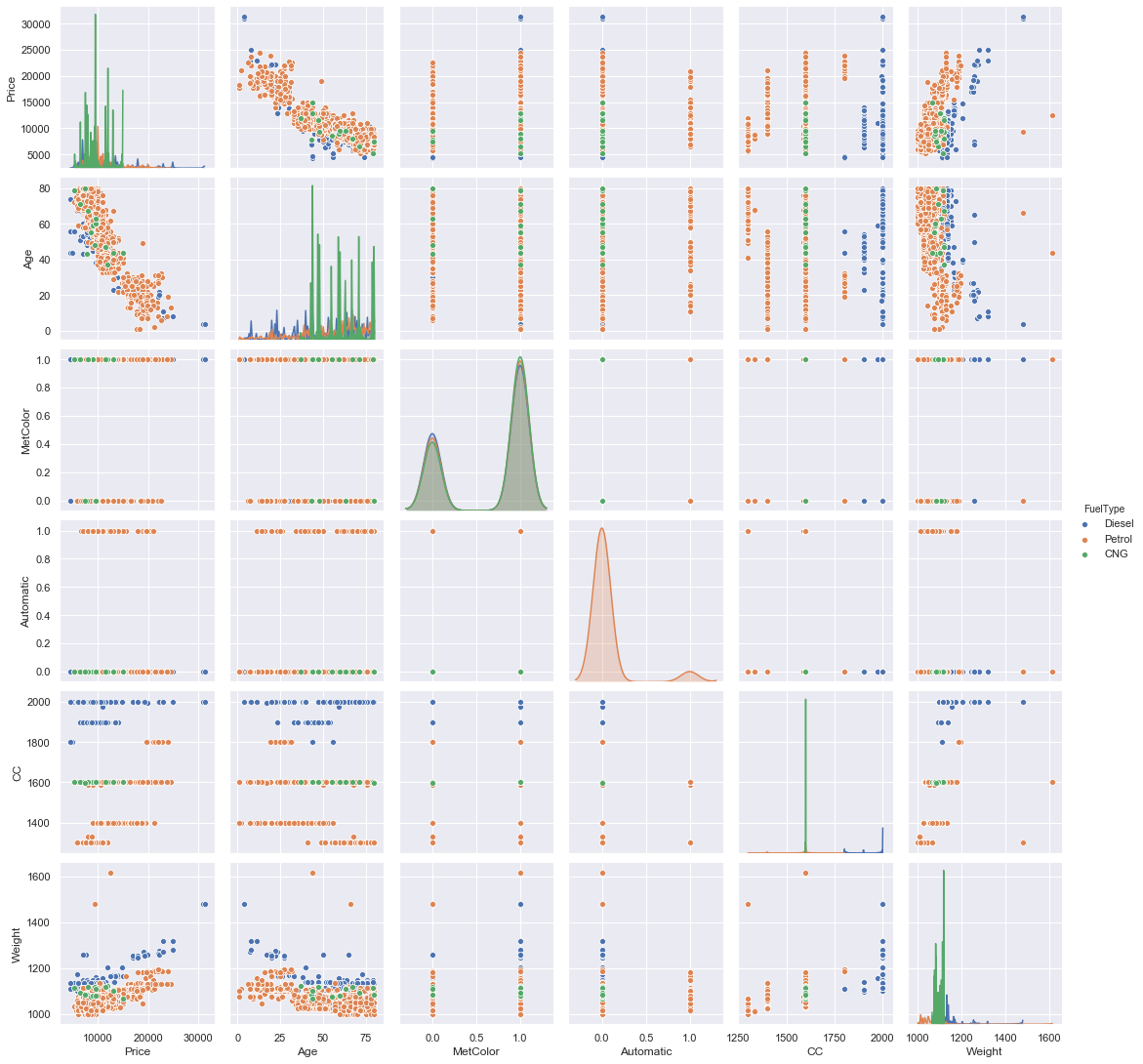
plt.show()

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:288: UserWarning: Data must have variance to compute a kernel density estimate.

warnings.warn(msg, UserWarning)

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:288: UserWarning: Data must have variance to compute a kernel density estimate.

warnings.warn(msg, UserWarning)



# Heatmap[¶](#Heatmap)

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. I

In [16]:

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import os

In [17]:

data=np.random.randint(1,100,size=(10,10))

print("The data to be plotted: \n")

print(data)

The data to be plotted:

[[ 7 46 51 1 51 45 57 23 57 82]

[85 83 92 48 26 62 8 76 28 94]

[63 4 14 54 80 28 48 74 67 4]

[28 61 19 25 29 38 99 54 65 82]

[31 92 58 93 43 51 94 37 32 50]

[59 36 1 23 64 16 32 42 56 2]

[83 12 98 91 55 22 63 79 53 21]

[32 61 32 83 38 87 44 10 35 60]

[76 55 99 58 13 21 43 93 56 31]

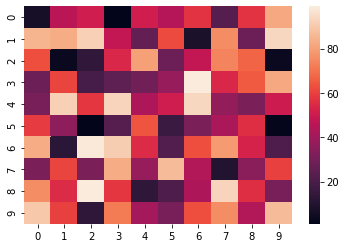
[90 60 13 72 41 31 63 76 44 87]]

In [18]:

#Plotting Heatmap

hm=sns.heatmap(data=data)

plt.show()



If we set the vmin value to 30 and the vmax value to 70, then only the cells with values between 30 and 70 will be displayed. This is called anchoring the colormap.

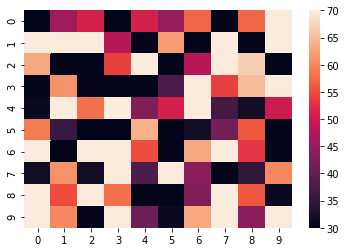
In [22]:

hm = sns.heatmap(data=data,

vmin='30',

vmax='70')

plt.show()



Choosing the colormap

In this, we will be looking at the cmap parameter. Matplotlib provides us with multiple colormaps, you can look at all of them here. Centering the cmap to 0 by passing the center parameter as 0.

In [46]:

# setting the parameter values

cmap = "tab20"

center = 0

# setting the parameter values

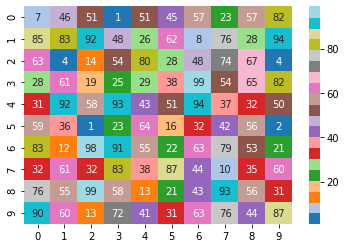
annot = True

# plotting the heatmap

hm = sns.heatmap(data=data, cmap=cmap, annot=annot)

# displaying the plotted heatmap

plt.show()



In [ ]:

Practical 4

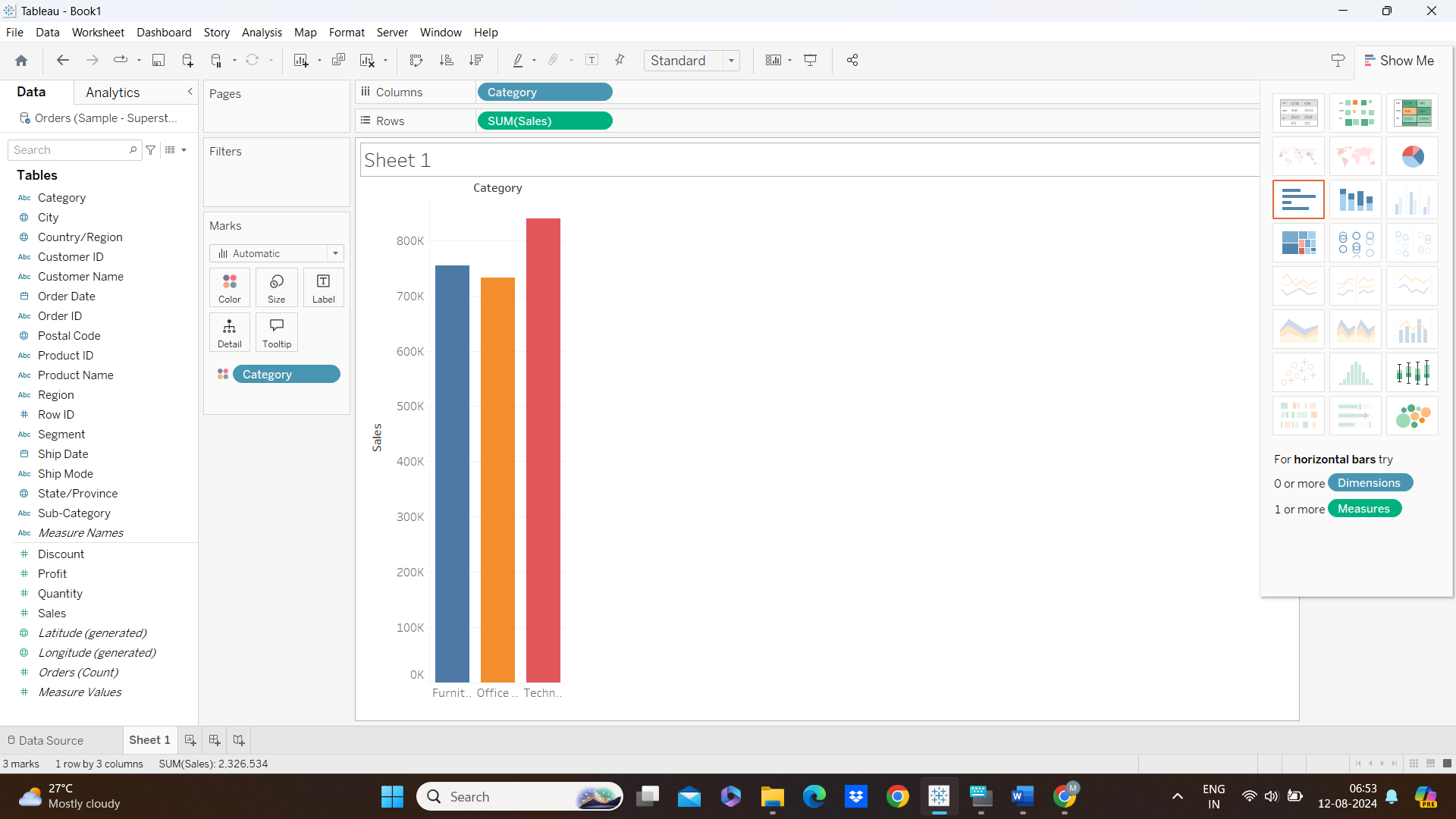
Tableau Practical

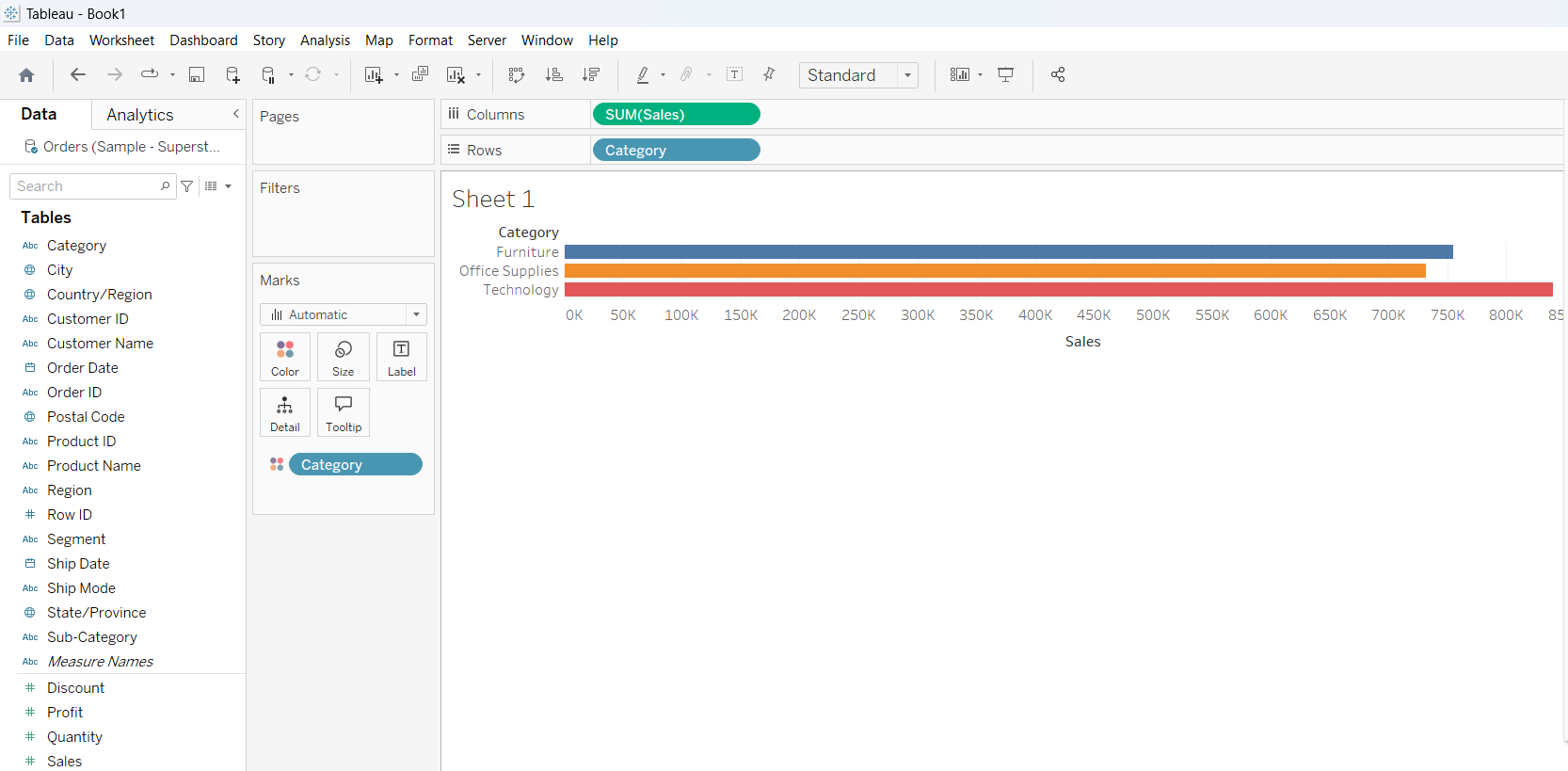
Step 1:

Upload Sample-Superstore excel file

Step 2:

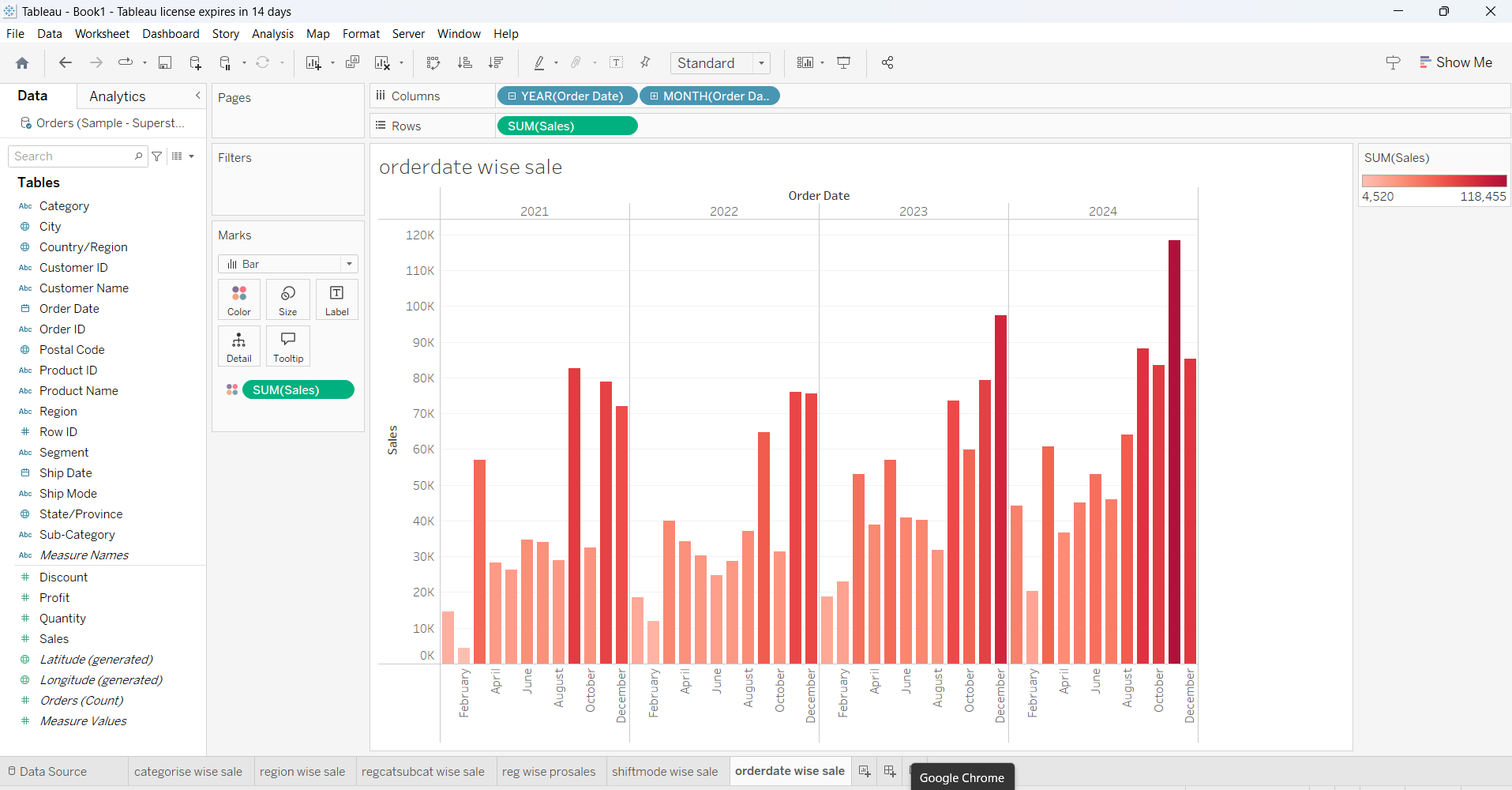
Categories wise sale generate vertical and horizontal bar chart:





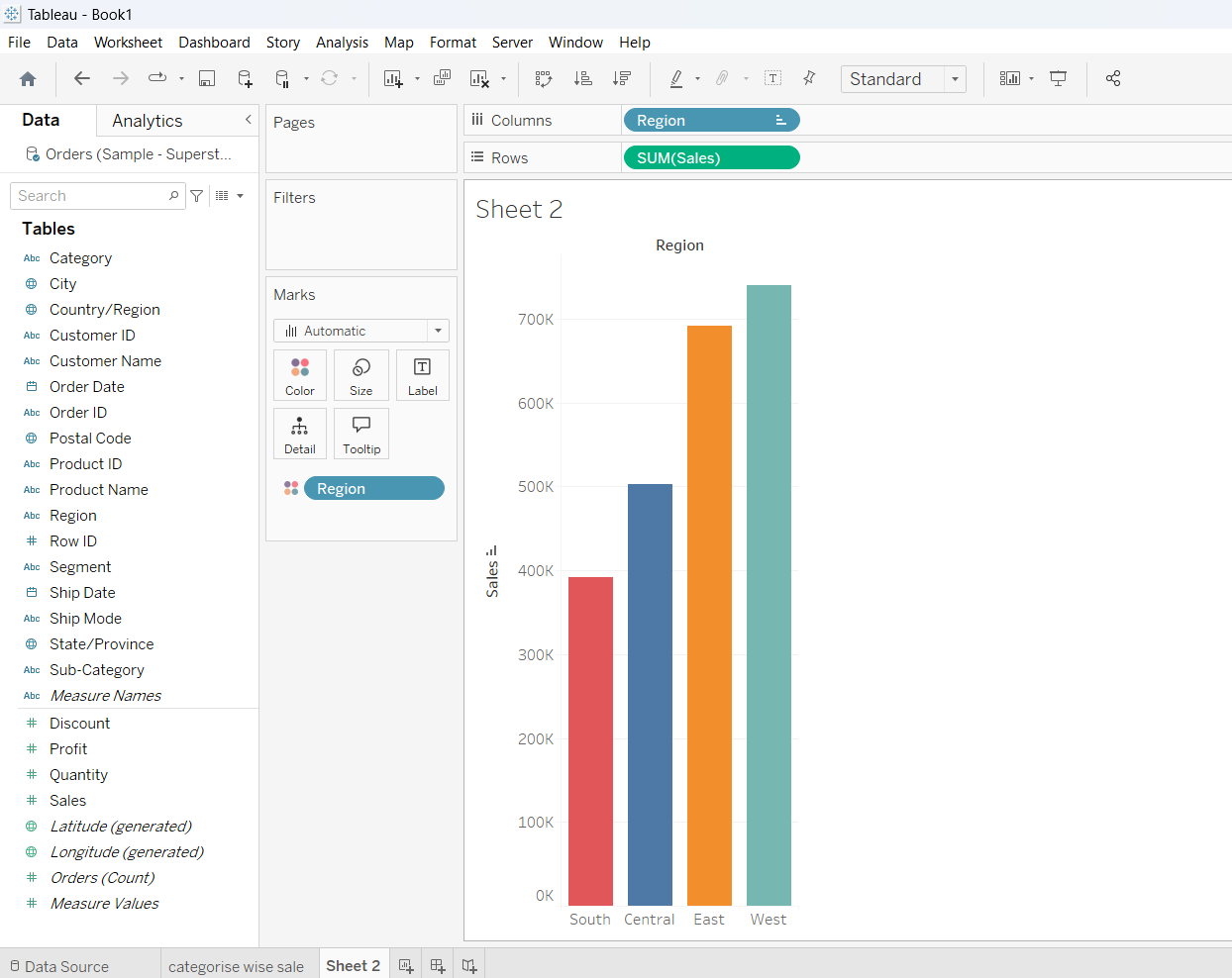
Step 3:

Order date wise sale:



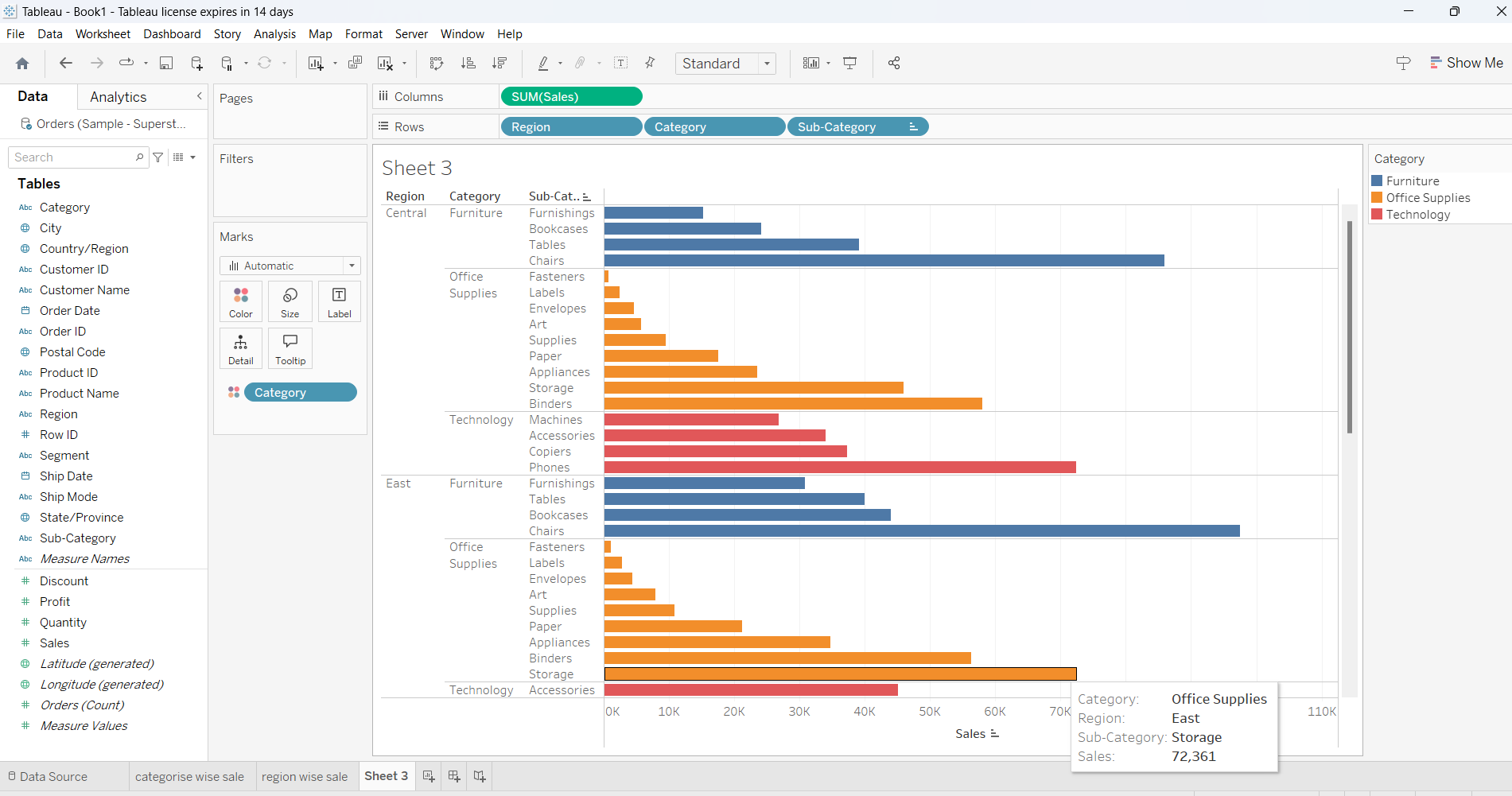
Step 4:

Region wise sale:



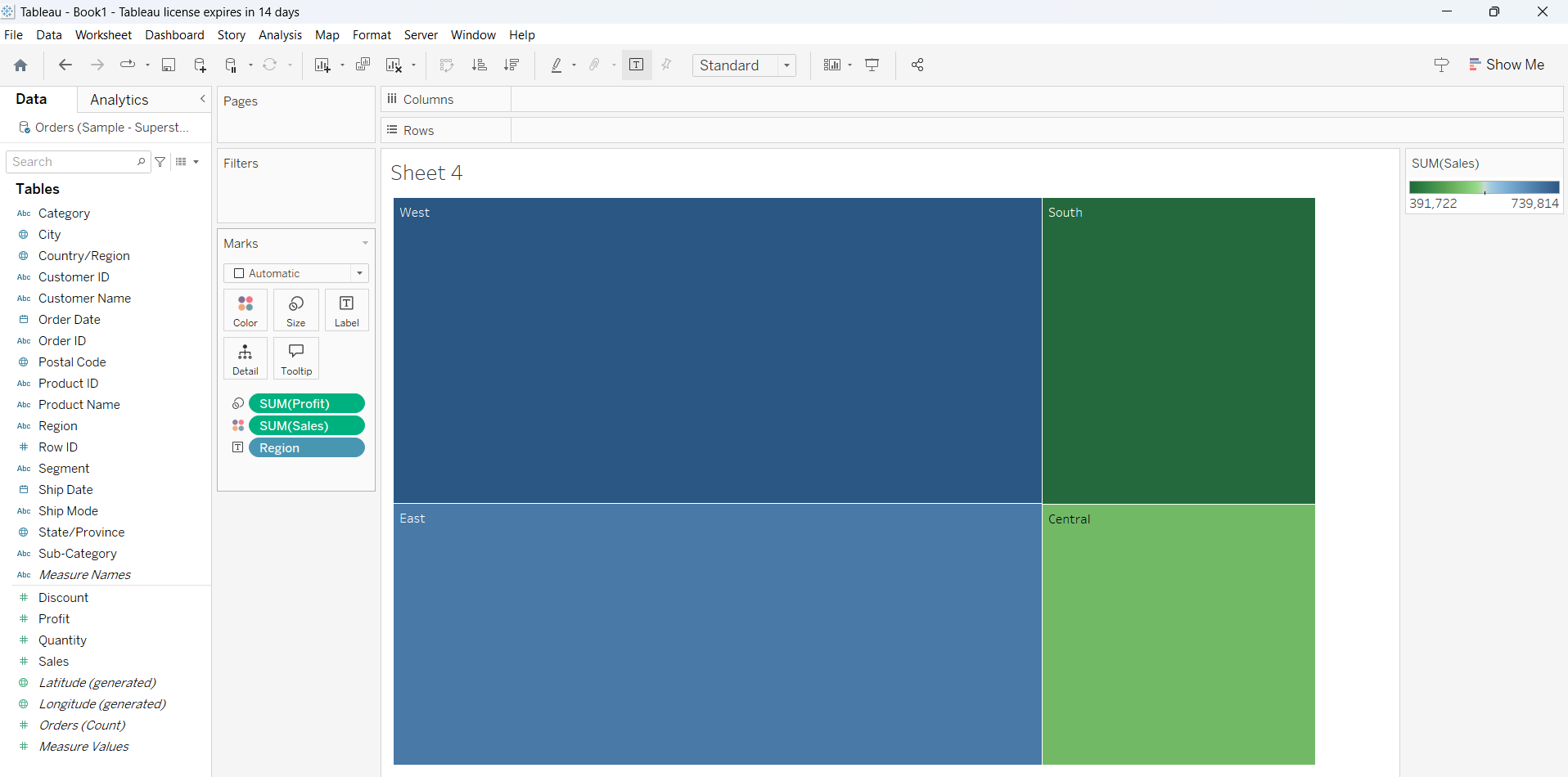
Step 5:

Region category and sub category sales:



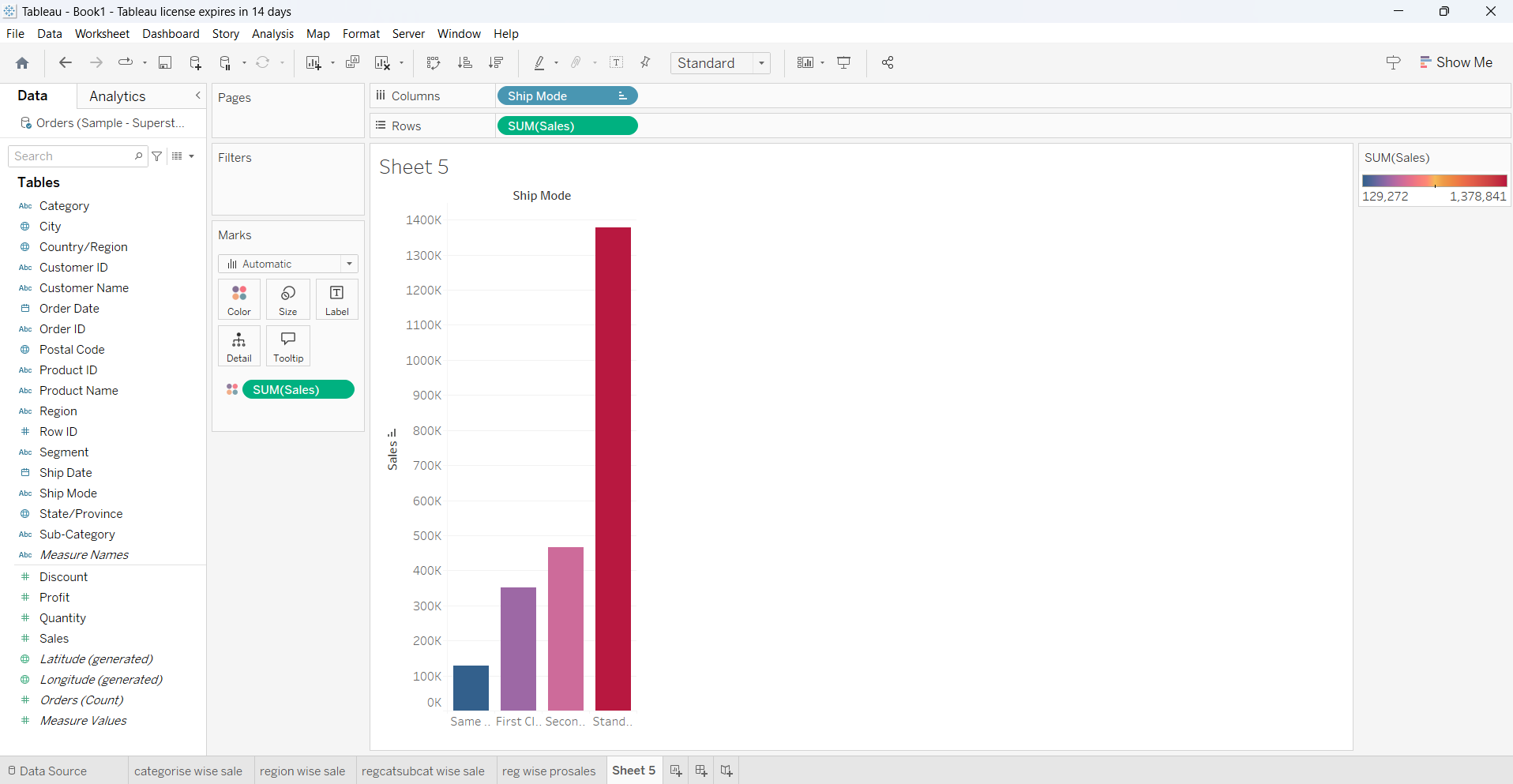
Step 6:

Generate tree map for region wise profit and sale:



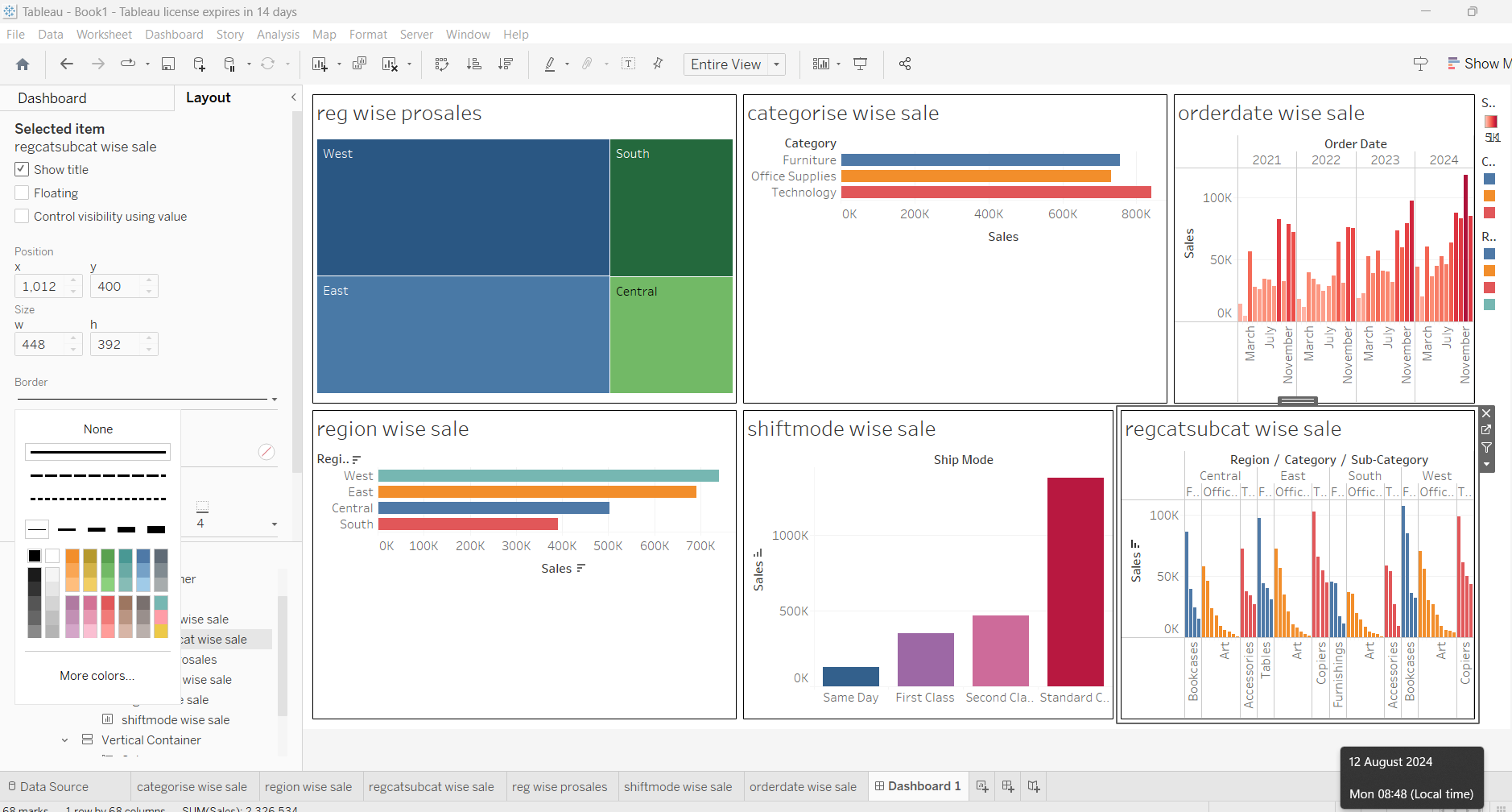
Step 7:

Ship Mode wise sale:



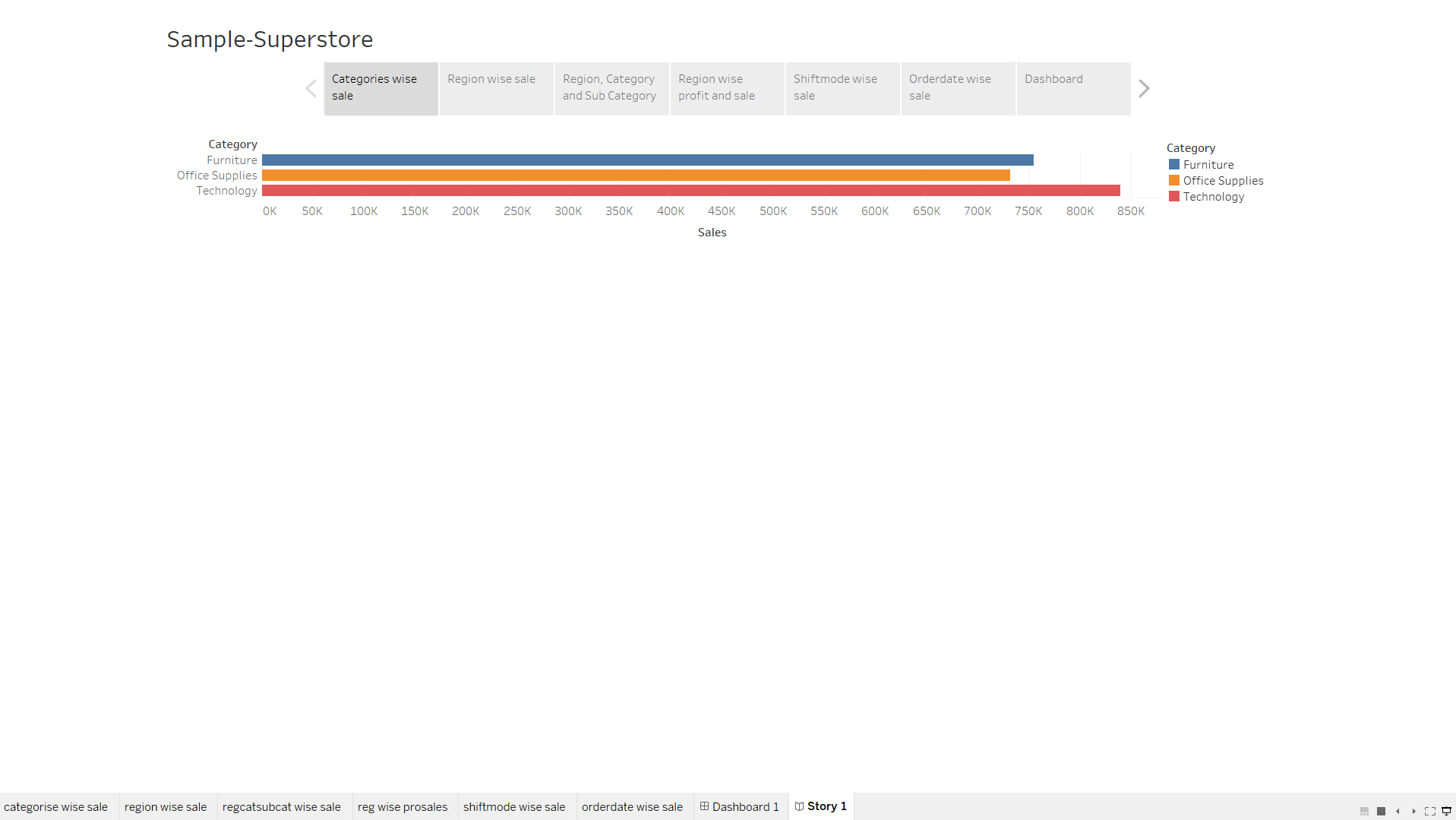
Step 8:

Create dashboard of above sheets:



Step 9:

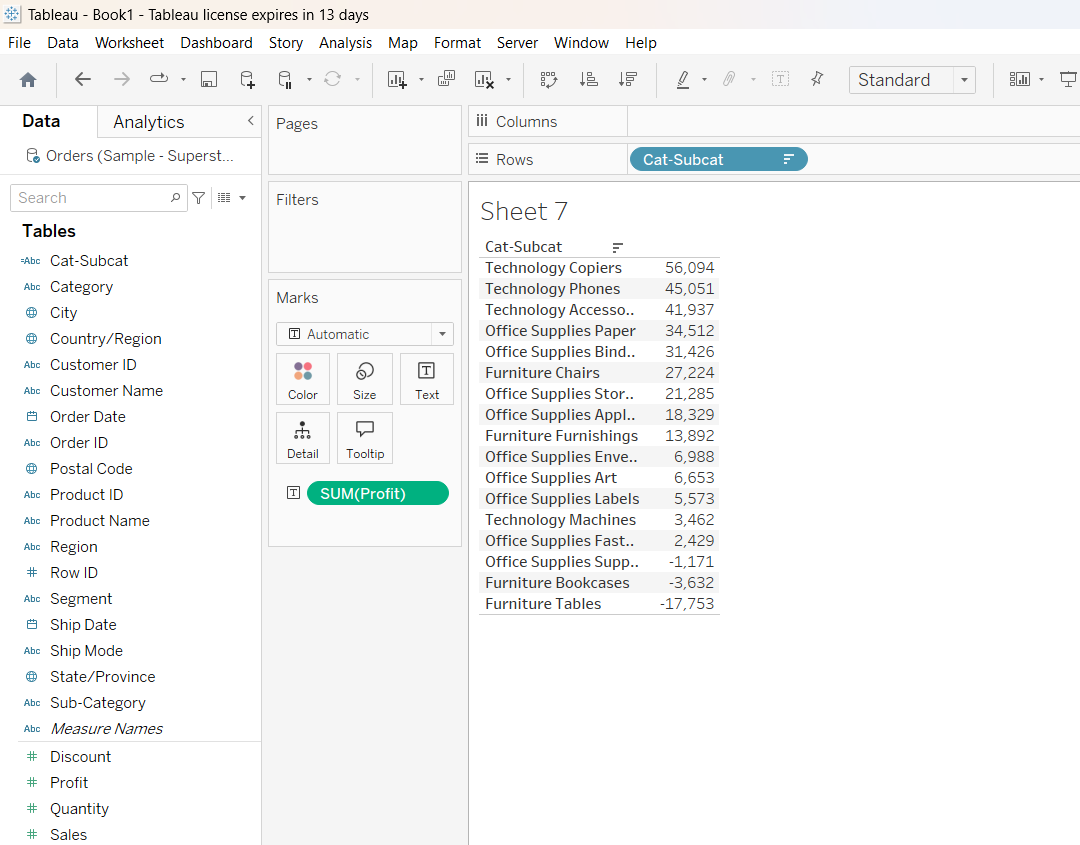
Generate tableau story:



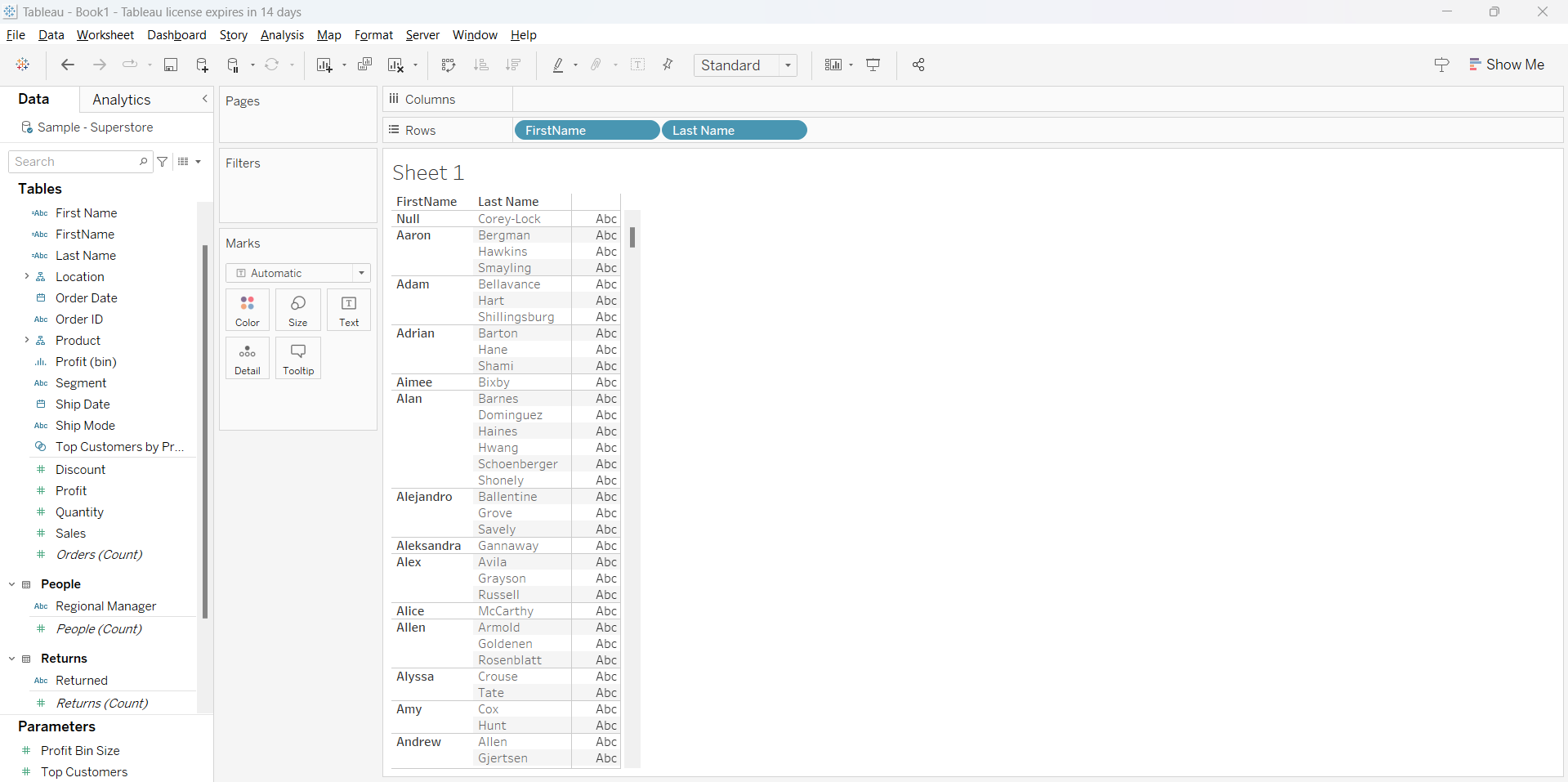
Calculation

1. String calculation

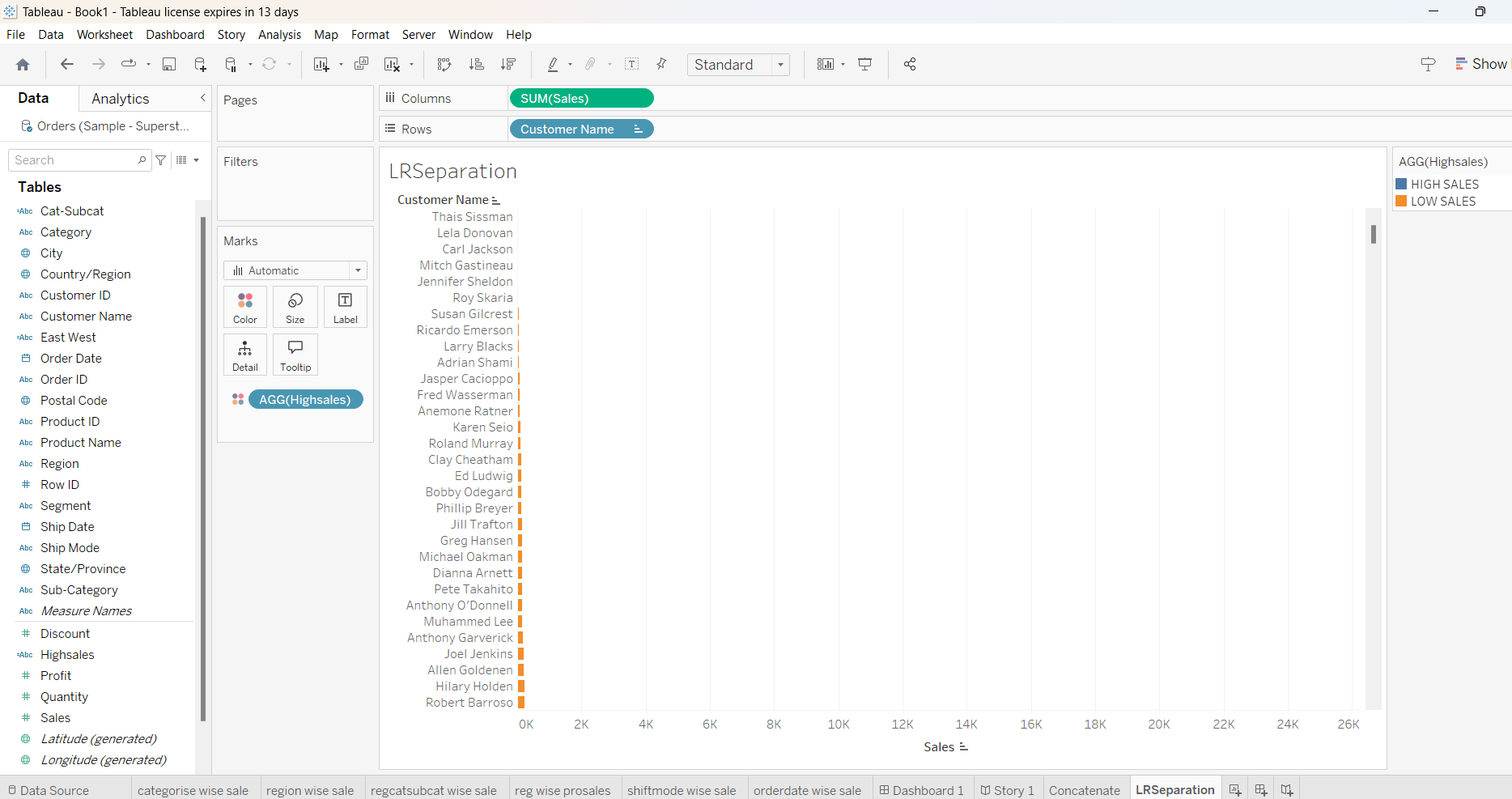
Concatenation:



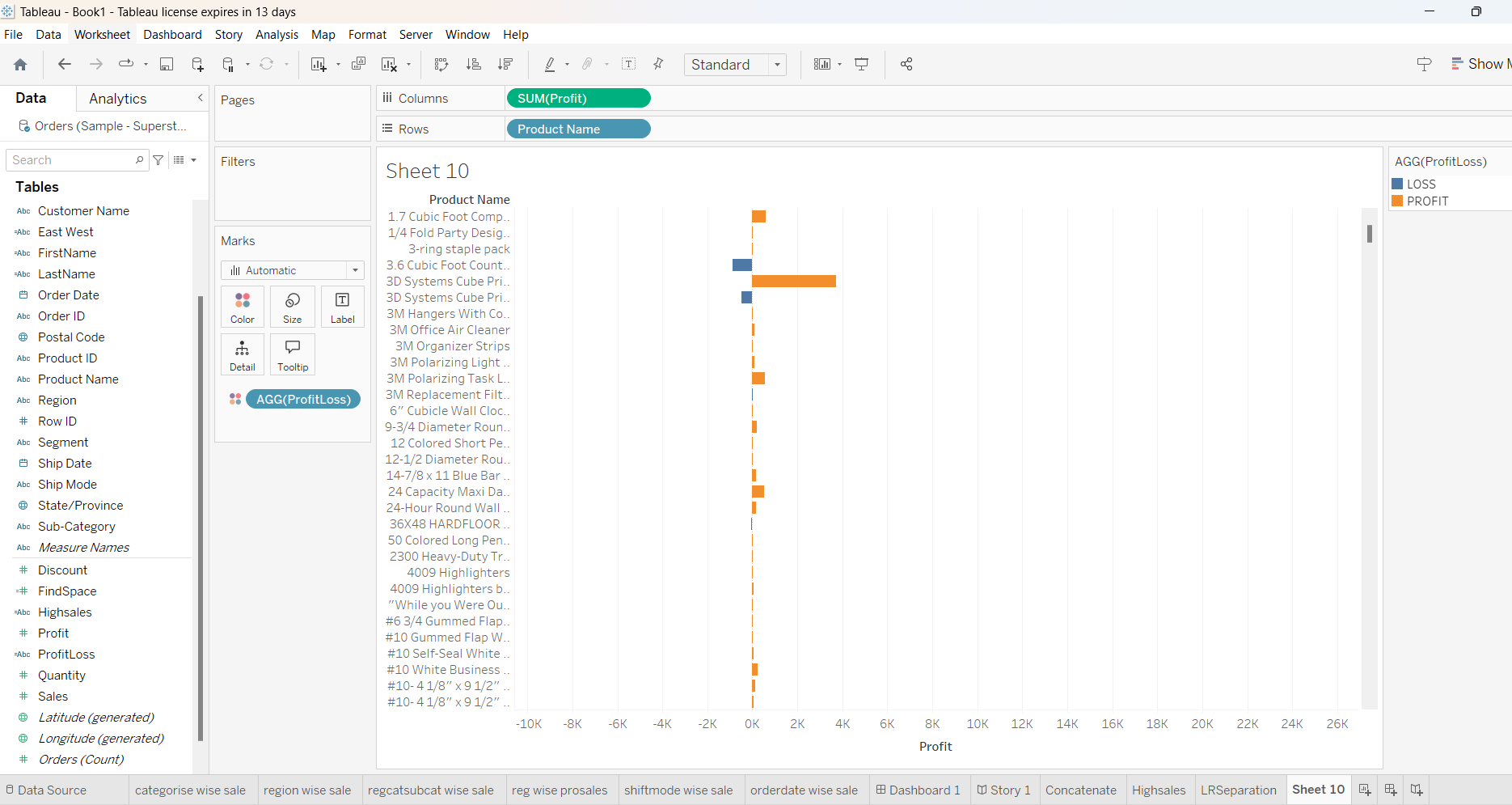
Left-side and Right-side separation:

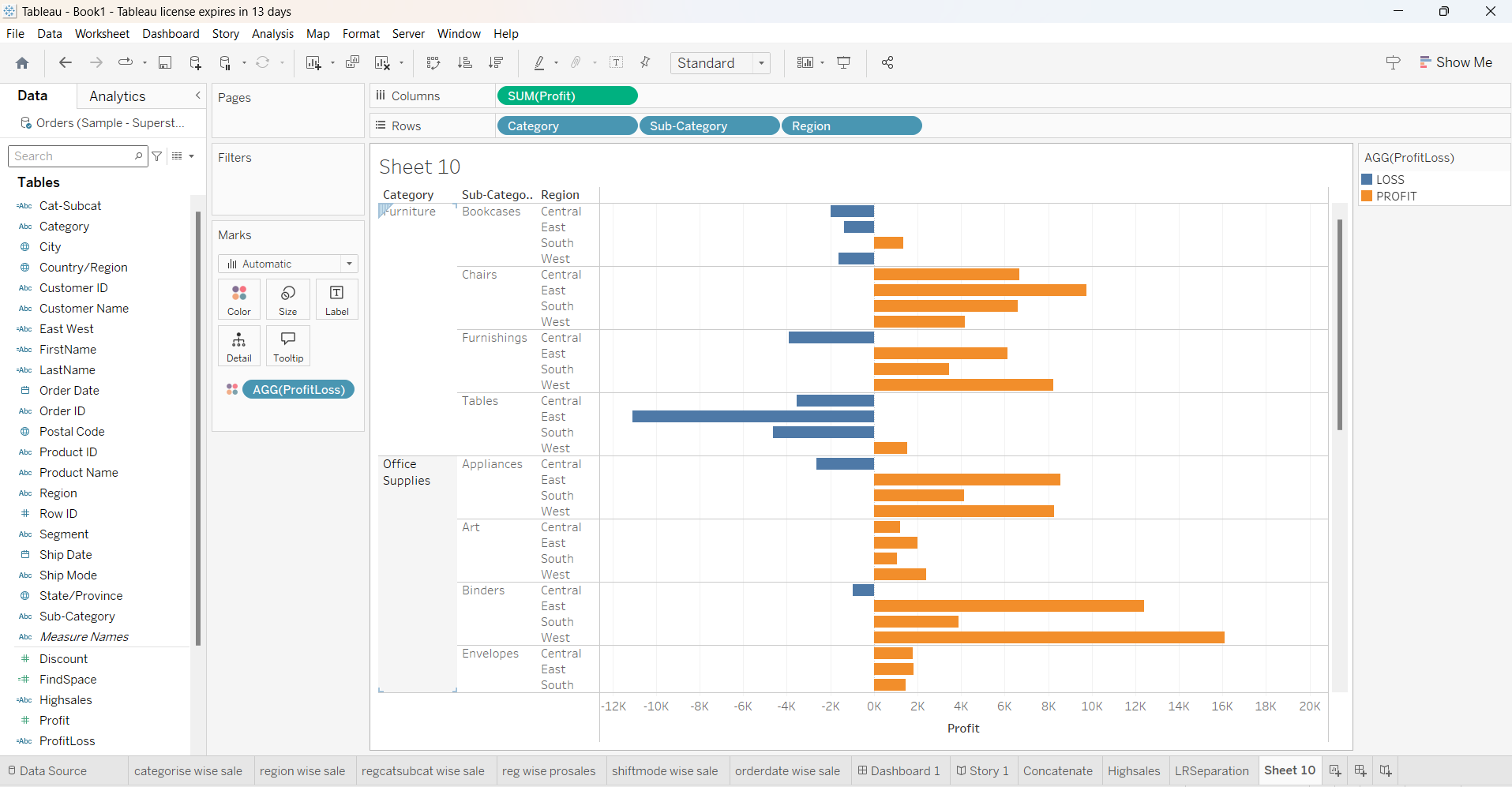


2. Calculated field to generate the high sale:

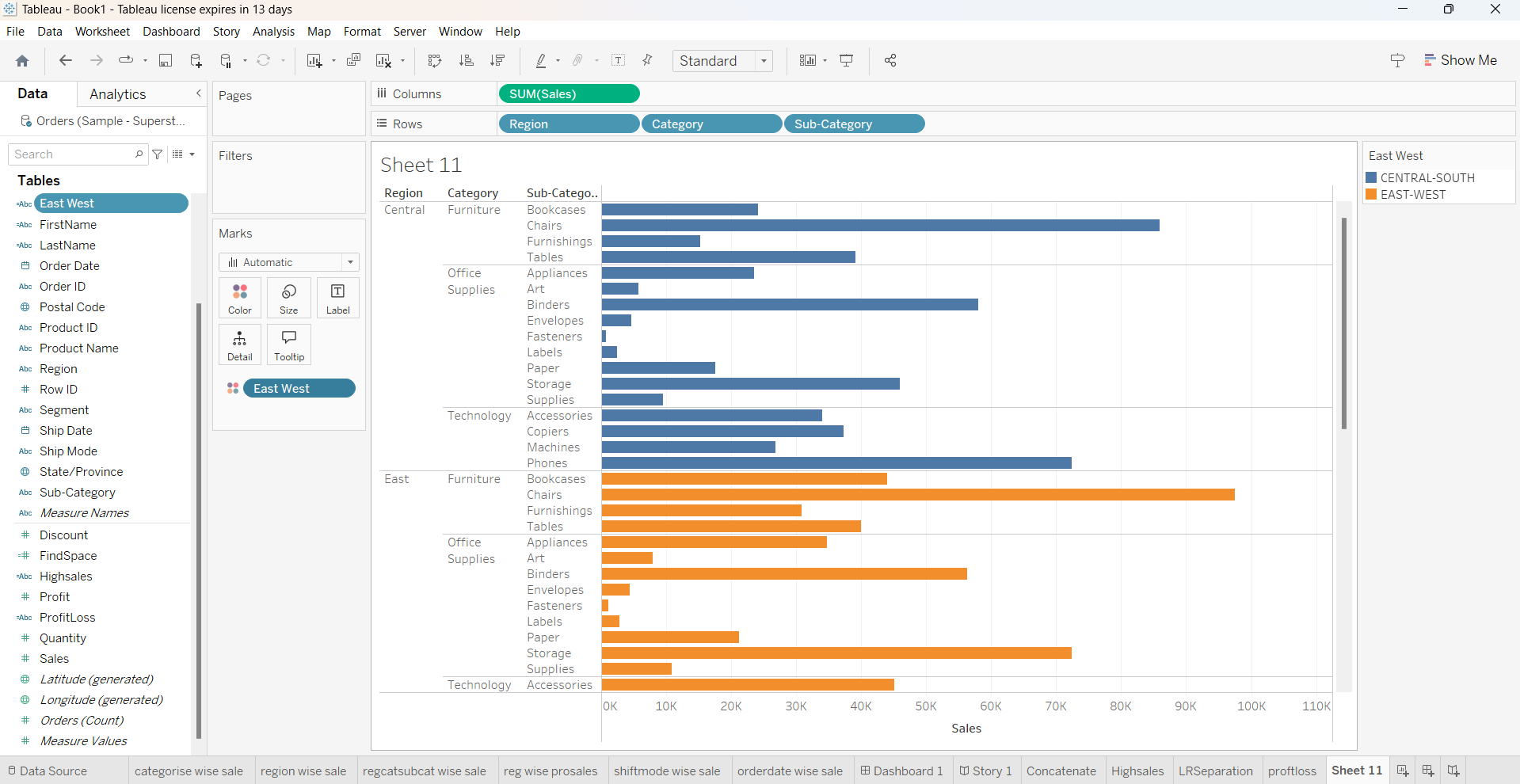


3. Calculated field to generate profit and loss:





4. Region wise separation:



5. Selection of top and bottom 10 customers by using filter and indexing:

