Using Python

1.2 Exercises Charts_DSC640 - Week 1 & 2

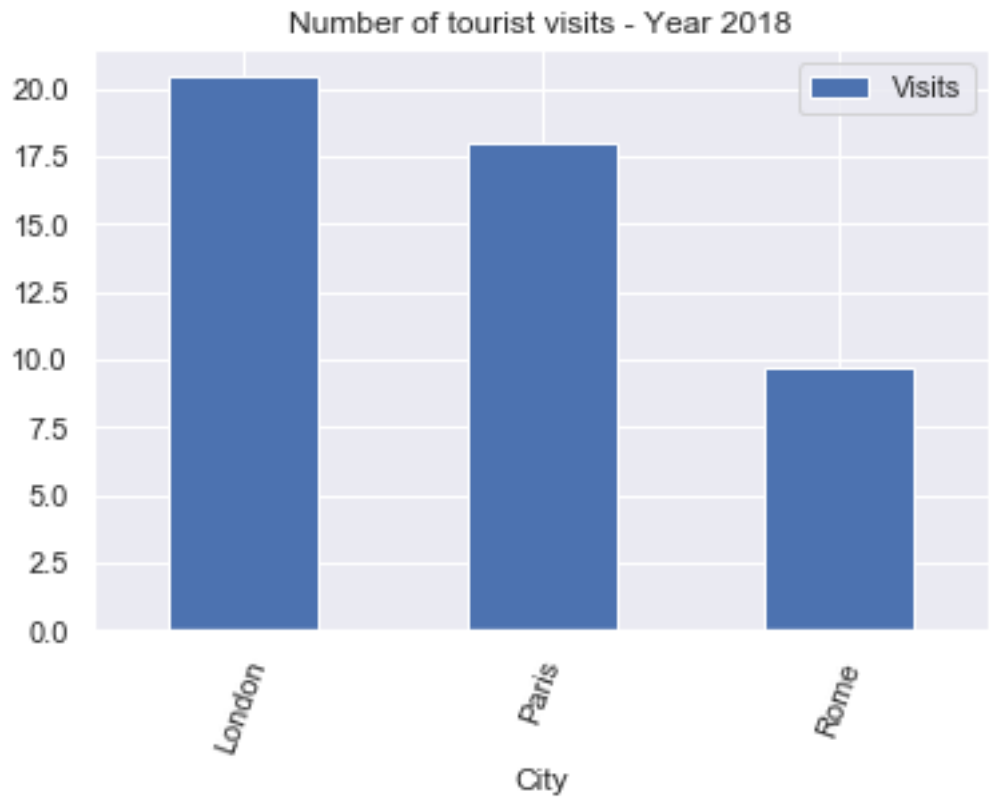
You need to submit 3 bar charts, 3 stacked bar charts, 3 pie charts, 3 donut charts, and 3 line charts using Tableau or PowerBI, Python and R using the data below (or your own datasets). You can also submit using D3 if you choose – but it is not required. You can choose which library to use in Python or R, documentation is provided to help you decide and as you start to play around in the libraries, you will decide which you prefer. Python

A bar chart

A bar chart drawn using the function pyplot.bar() can be further customized using various parameters of the bar() function. Bars can be aligned left side to an X-axis tick by specifying 'left' as the value for the align parameter. The value 'center' will make the bar align central to the tick. If the bar needs to be right aligned negative value need to be specified for the width parameter.

In [34]:

```
import pandas as pd

import matplotlib.pyplot as plot


# A python dictionary

data = {"City":["London", "Paris", "Rome"],

        "Visits":[20.42,17.95,9.7]

        };

# Dictionary loaded into a DataFrame

df = pd.DataFrame(data=data);


# Draw a vertical bar chart

df.plot.bar(x="City", y="Visits", rot=70, title="Number of tourist visits - Year 2018"
);

plot.show(block=True);
```



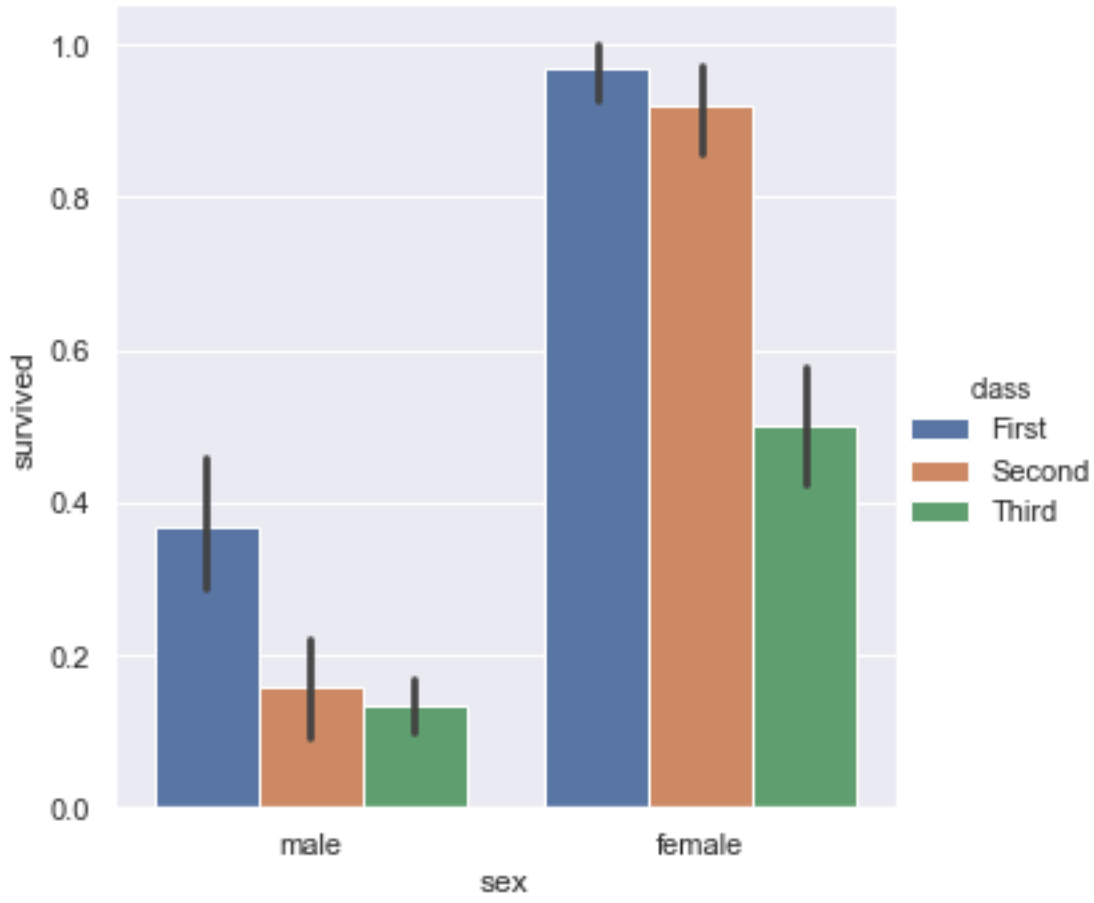
Stacked bar plots

https://seaborn.pydata.org/tutorial/categorical.html#distributions-of-observations-within-categories

A familiar style of plot that accomplishes this goal is a bar plot. In seaborn, the barplot() function operates on a full dataset and applies a function to obtain the estimate (taking the mean by default). When there are multiple observations in each category, it also uses bootstrapping to compute a confidence interval around the estimate, which is plotted using error bars:

In [35]:

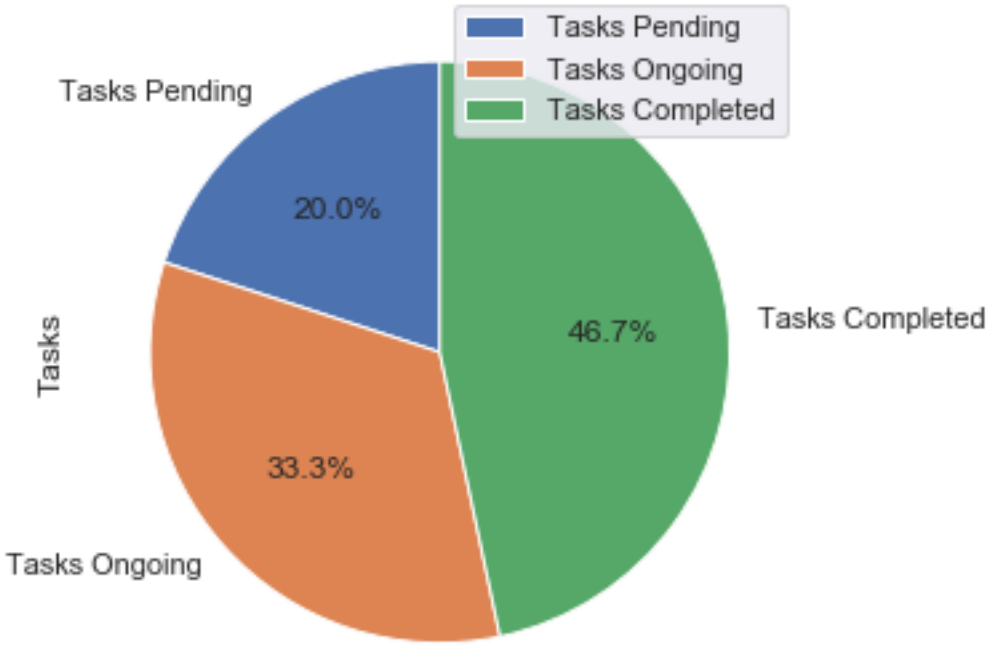
```
import seaborn as sns

import matplotlib.pyplot as plt


#sns.set(style="ticks", color_codes=True)


titanic = sns.load_dataset("titanic")

sns.catplot(x="sex", y="survived", hue="class", kind="bar", data=titanic);
```



Pie Chart

In [36]:

```
from pandas import DataFrame

import matplotlib.pyplot as plt


data = {'Tasks': [300,500,700]}

df = DataFrame(data,columns=['Tasks'],index = ['Tasks Pending','Tasks Ongoing','Tasks Completed'])


df.plot.pie(y='Tasks',figsize=(5, 5),autopct='%1.1f%%', startangle=90)

plt.show()
```



Donut Chart

A doughnut chart displays value data as percentages of the whole. Like pie chart, categories are represented by individual slices in a doughnut chart. If you have several parts of something one, you can demonstrate each item in one pie chart, however, sometimes you want to demonstrate the changes of those parts and doughnut chart will help you to do this. Besides, Doughnut charts can give you a great perspective of the relative changes. Unlike a stacked bar chart or pie chart, with a doughnut chart, you can get a birds-eye view of the relative change in each item of the series.

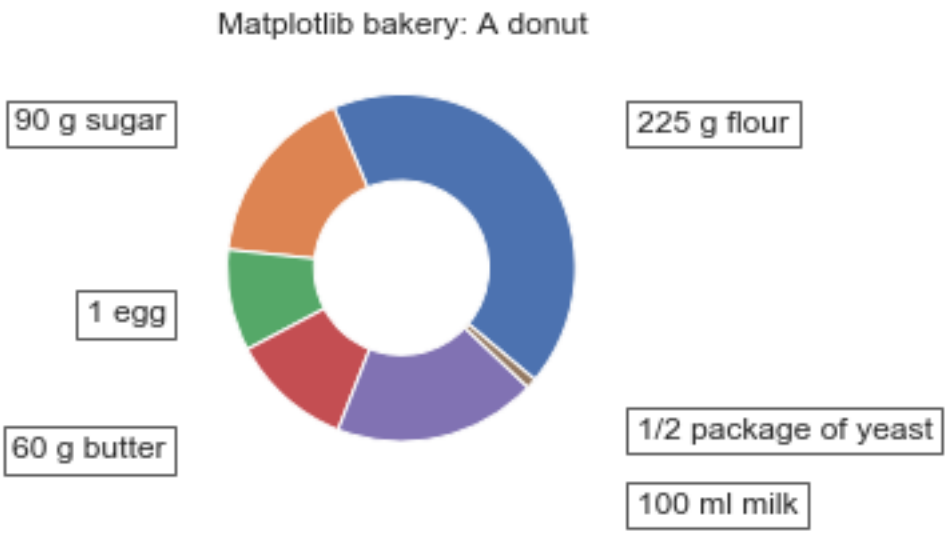
There are two types of doughnut charts: doughnut and exploded doughnut. A doughnut chart displays category groups, series groups, and values series as doughnut slices. The size of the slice is determined by the series value as a percentage of the total of all values. An exploded doughnut chart is identical to a doughnut chart except that the slices are moved away from the center of the doughnut. This results in space between the doughnut slices

https://www.edrawsoft.com/doughnut-chart-solutions.html

In [37]:

```
import numpy as np

import matplotlib.pyplot as plt          # This is also used for plotting


fig, ax = plt.subplots(figsize=(6, 3), subplot_kw=dict(aspect="equal"))


recipe = ["225 g flour",
```

```
          "90 g sugar",
          "1 egg",
          "60 g butter",
          "100 ml milk",
          "1/2 package of yeast"]

data = [225, 90, 50, 60, 100, 5]

wedges, texts = ax.pie(data, wedgeprops=dict(width=0.5), startangle=-40)

bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
kw = dict(arrowprops=dict(arrowstyle="-"),
          bbox=bbox_props, zorder=0, va="center")

for i, p in enumerate(wedges):
    ang = (p.theta2 - p.theta1)/2. + p.theta1
    y = np.sin(np.deg2rad(ang))
    x = np.cos(np.deg2rad(ang))
    horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
    connectionstyle = "angle,angleA=0,angleB={}".format(ang)
    kw["arrowprops"].update({"connectionstyle": connectionstyle})
    ax.annotate(recipe[i], xy=(x, y), xytext=(1.35*np.sign(x), 1.4*y),
                horizontalalignment=horizontalalignment, **kw)

ax.set_title("Matplotlib bakery: A donut")

plt.show()
```



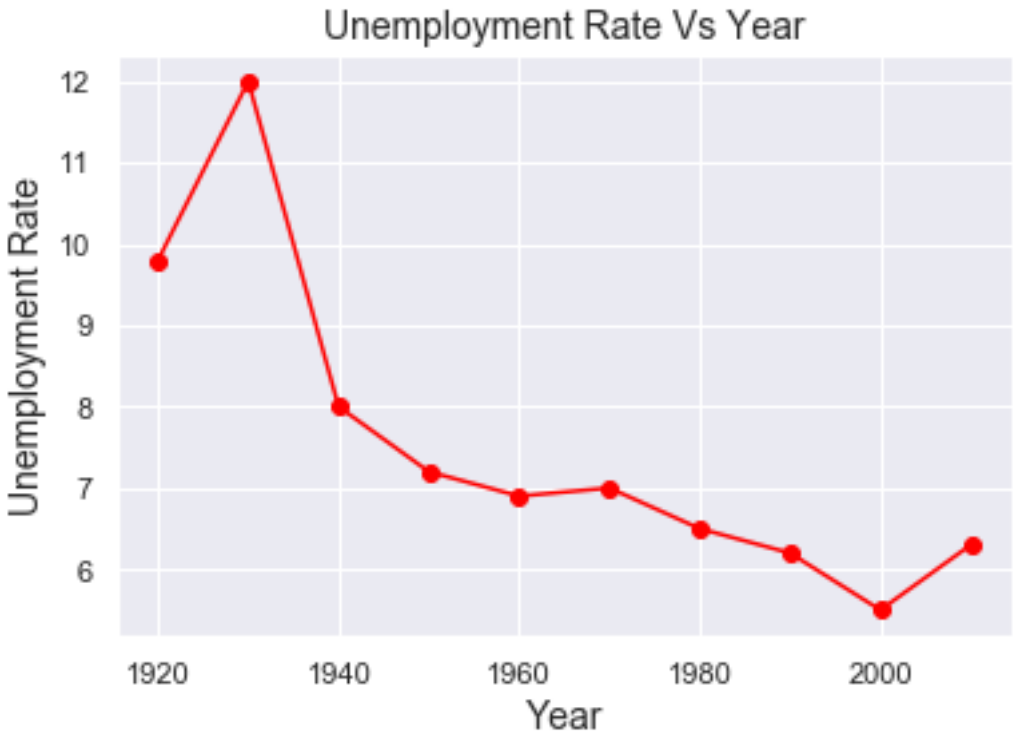
3 Line Charts Using pandas

In [38]:

```
from pandas import DataFrame

import matplotlib.pyplot as plt


data = {'Year': [1920,1930,1940,1950,1960,1970,1980,1990,2000,2010],

        'Unemployment_Rate': [9.8,12,8,7.2,6.9,7,6.5,6.2,5.5,6.3]

       }


df = DataFrame(data,columns=['Year','Unemployment_Rate'])

df.plot(x ='Year', y='Unemployment_Rate', kind = 'line')

plt.show()
```

In [39]:

```
import matplotlib.pyplot as plt


Year = [1920,1930,1940,1950,1960,1970,1980,1990,2000,2010]

Unemployment_Rate = [9.8,12,8,7.2,6.9,7,6.5,6.2,5.5,6.3]


plt.plot(Year, Unemployment_Rate, color='red', marker='o')

plt.title('Unemployment Rate Vs Year', fontsize=14)

plt.xlabel('Year', fontsize=14)

plt.ylabel('Unemployment Rate', fontsize=14)

plt.grid(True)

plt.show()
```



Line Chart: Drawing A Line Chart Using Pandas DataFrame

A line chart plots a set of (x, y) values in a two-dimensional plane and connects those data points through straight lines. A line chart is one of the most commonly used charts to understand the relationship, trend of one variable with another.

Drawing a Line chart using pandas DataFrame in Python: The DataFrame class has a plot member through which several graphs for visualization can be plotted. A line chart or line graph is one among them.

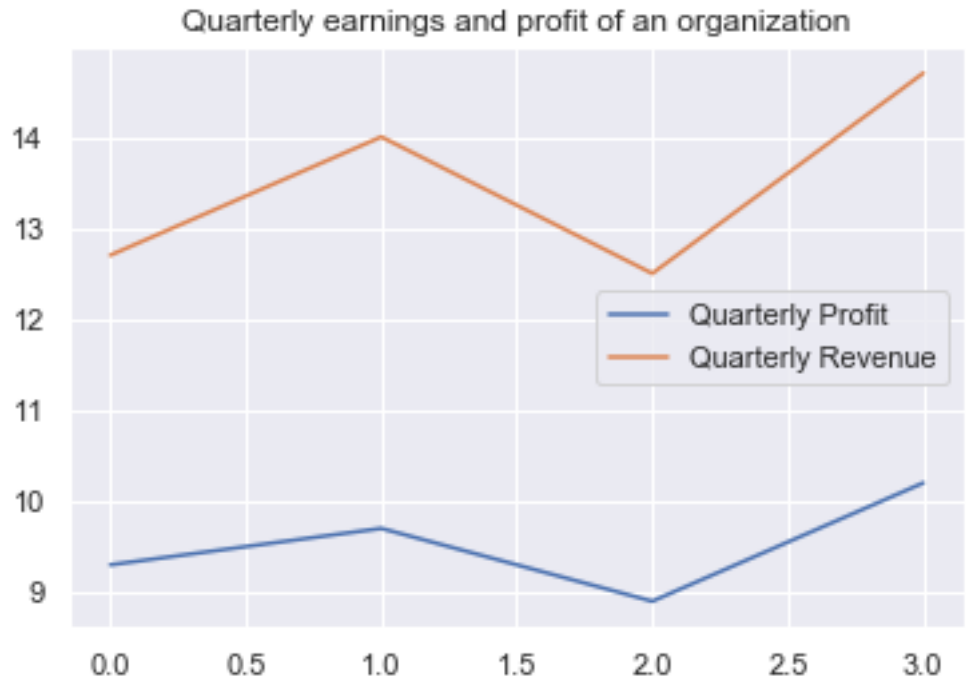
Calling the line() method on the plot instance draws a line chart. If the column name for X-axis is not specified, the method takes the index of the column as the X-axis, which is of the pattern 0, 1, 2, 3 and so on. After drawing the X-axis from the index of the DataFrame or using the specified column, the subsequent numeric columns are plotted as lines against the X-axis.

https://pythontic.com/pandas/dataframe-plotting/line%20chart

In [40]:

```
# Python program to plot a line chart for a pandas DataFrame


import pandas as pd

import matplotlib.pyplot as plot


# Earnings data for 4 quarters as a Python Dictionary
```

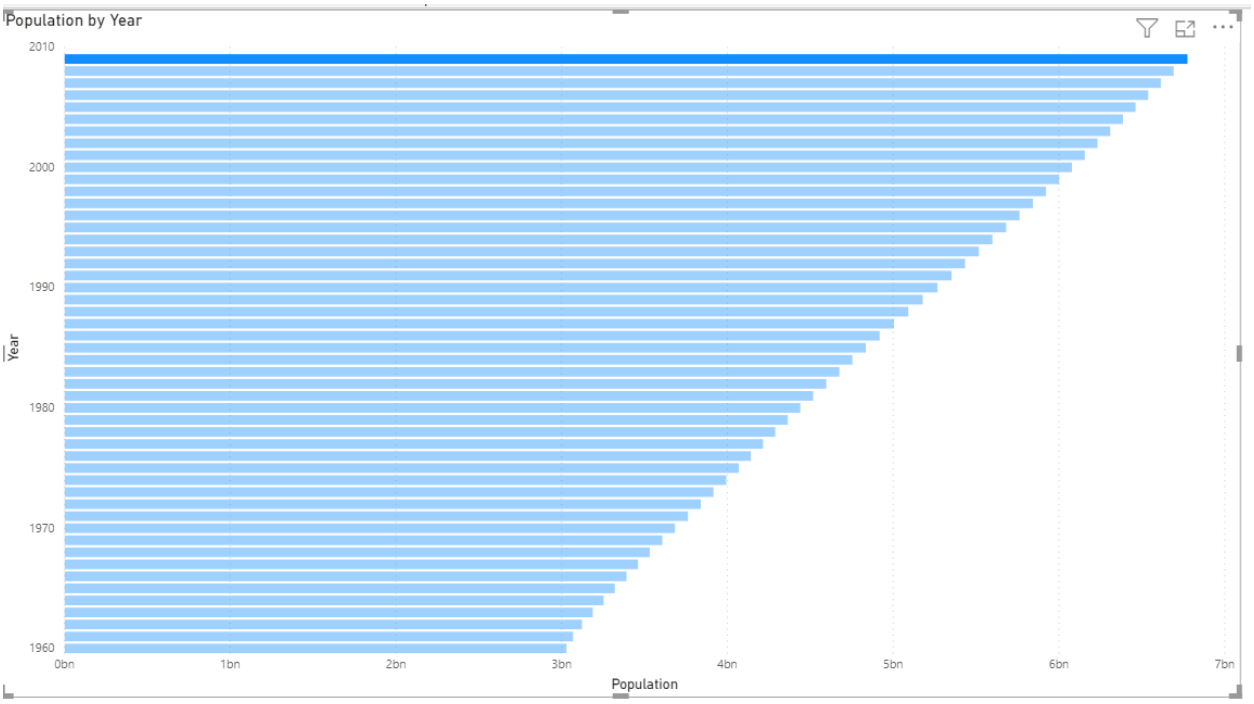
```
earningsData = {"Quarterly Profit": [9.3, 9.7, 8.9, 10.2],
                "Quarterly Revenue": [12.7, 14.0, 12.5, 14.7]
               };


df = pd.DataFrame(data=earningsData);


# Draw a line chart
df.plot.line(title="Quarterly earnings and profit of an organization");
plot.show(block=True);
```



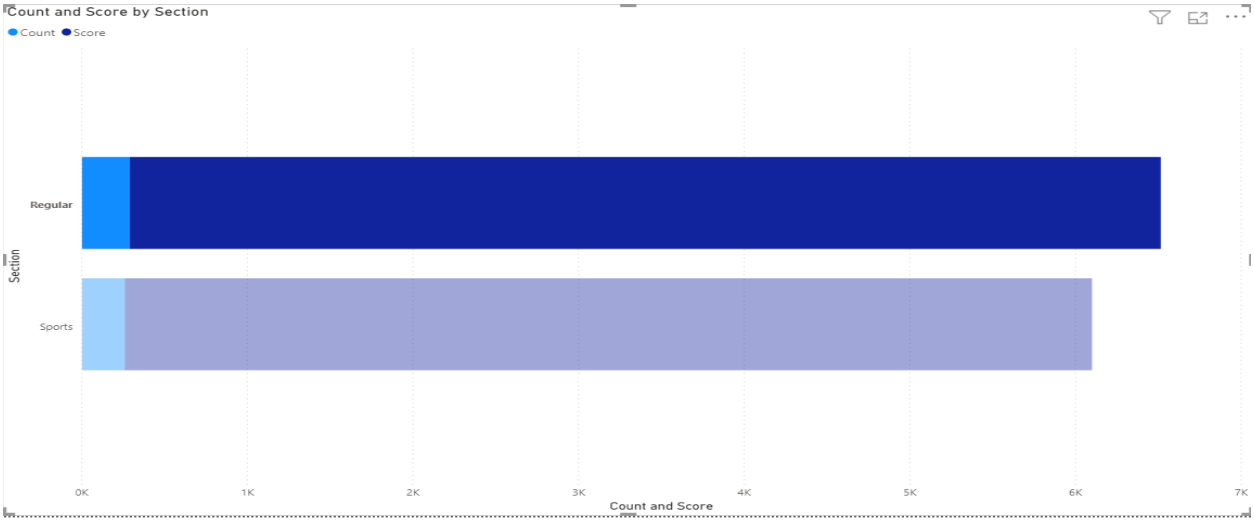
The End

Using BI

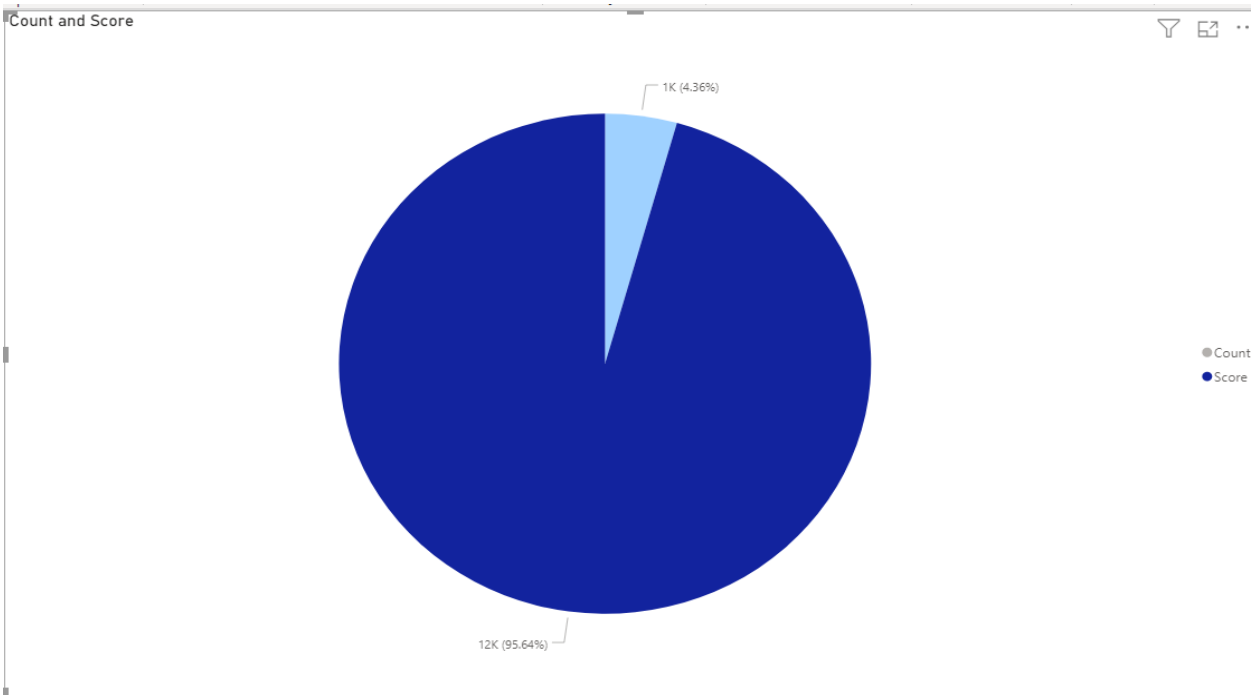
Bar Char:



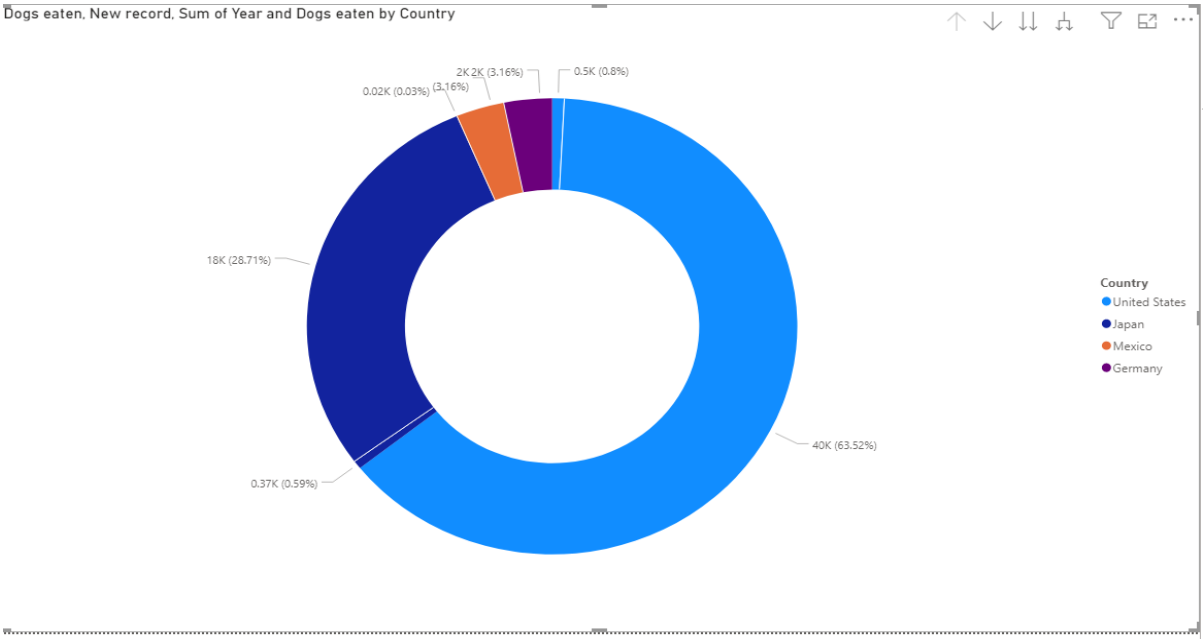
Stacked Bar Chart



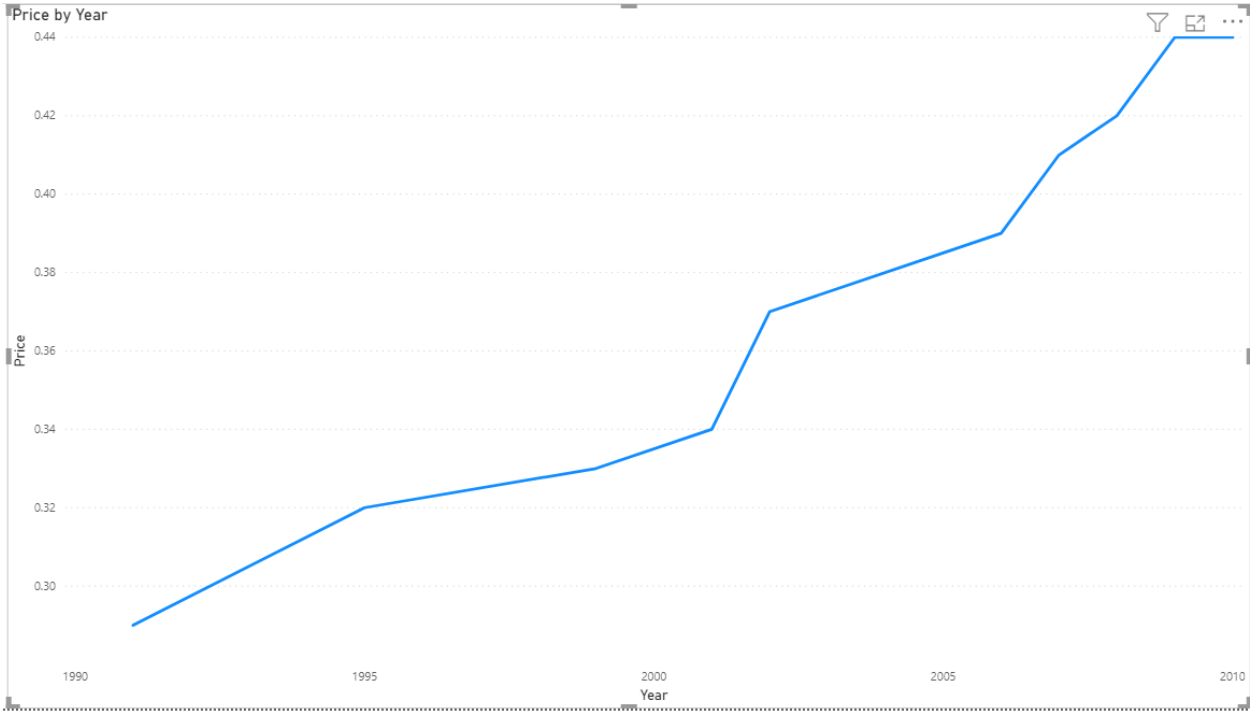
Pie Chart:


Count and Score
1K (4.36%)
12K (95.64%)
Count
Score

Donut Chart:


Dogs eaten, New record, Sum of Year and Dogs eaten by Country
2K 2K (3.16%)
0.5K (0.8%)
0.02K (0.03%)
(3.16%)
18K (28.71%)
40K (63.52%)
0.37K (0.59%)
Country
United States
Japan
Mexico
Germany

Line Chart:



Using R

1.2 Exercises

Soukhna Wade

6/14/2020

```
{r setup, include=FALSE} knitr::opts_chunk$set(echo = TRUE)
```

R Markdown

Basic Bar Chart - https://plotly.com/r/bar-charts/

library(plotly)

fig <- plot_ly(x = c("giraffes", "orangutans", "monkeys"), y = c(20, 14, 23), name = "SF Zoo", type = "bar")

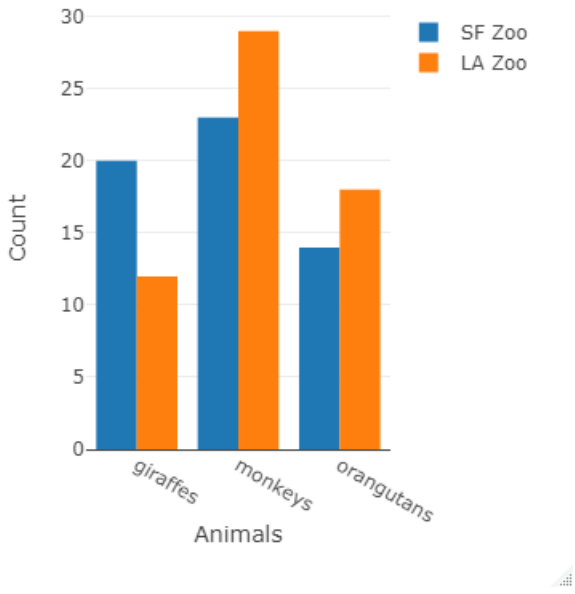
Grouped Bar Chart

library(plotly)

Animals <- c("giraffes", "orangutans", "monkeys") SF_Zoo <- c(20, 14, 23) LA_Zoo <- c(12, 18, 29) data <- data.frame(Animals, SF_Zoo, LA_Zoo)

fig <- plot_ly(data, x = ~Animals, y = ~SF_Zoo, type = 'bar', name = 'SF Zoo') fig <- fig %>% add_trace(y = ~LA_Zoo, name = 'LA Zoo') fig <- fig %>% layout(yaxis = list(title = 'Count'), barmode = 'group')

fig



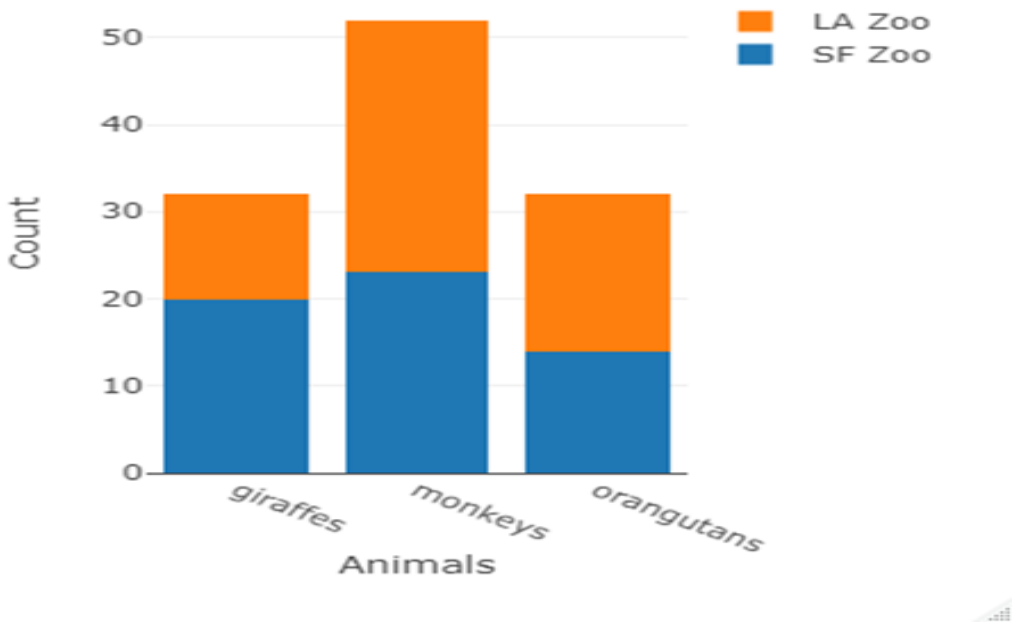
Stacked Bar Chart

library(plotly)

Animals <- c("giraffes", "orangutans", "monkeys") SF_Zoo <- c(20, 14, 23) LA_Zoo <- c(12, 18, 29) data <- data.frame(Animals, SF_Zoo, LA_Zoo)

fig <- plot_ly(data, x = ~Animals, y = ~SF_Zoo, type = ‘bar’, name = ‘SF Zoo’) fig <- fig %>% add_trace(y = ~LA_Zoo, name = ‘LA Zoo’) fig <- fig %>% layout(yaxis = list(title = ‘Count’), barmode = ‘stack’)

fig



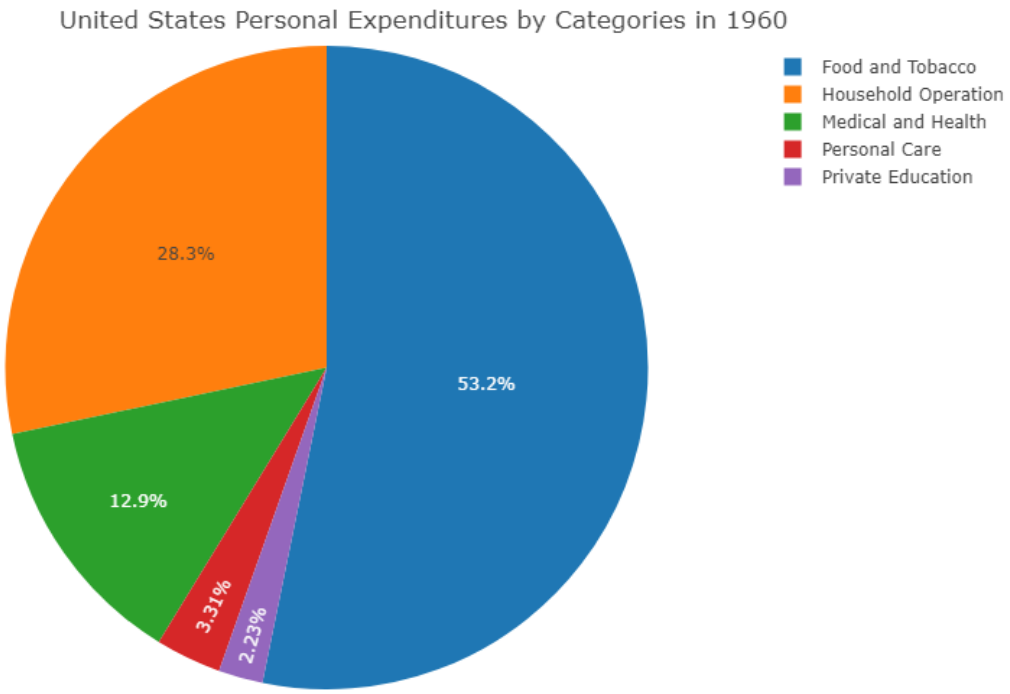
Basic Pie Chart

library(ggplot2) library(plotly)

USPersonalExpenditure <- data.frame(“Categorie”=rownames(USPersonalExpenditure), USPersonalExpenditure) data <- USPersonalExpenditure[,c(‘Categorie’, ‘X1960’)]

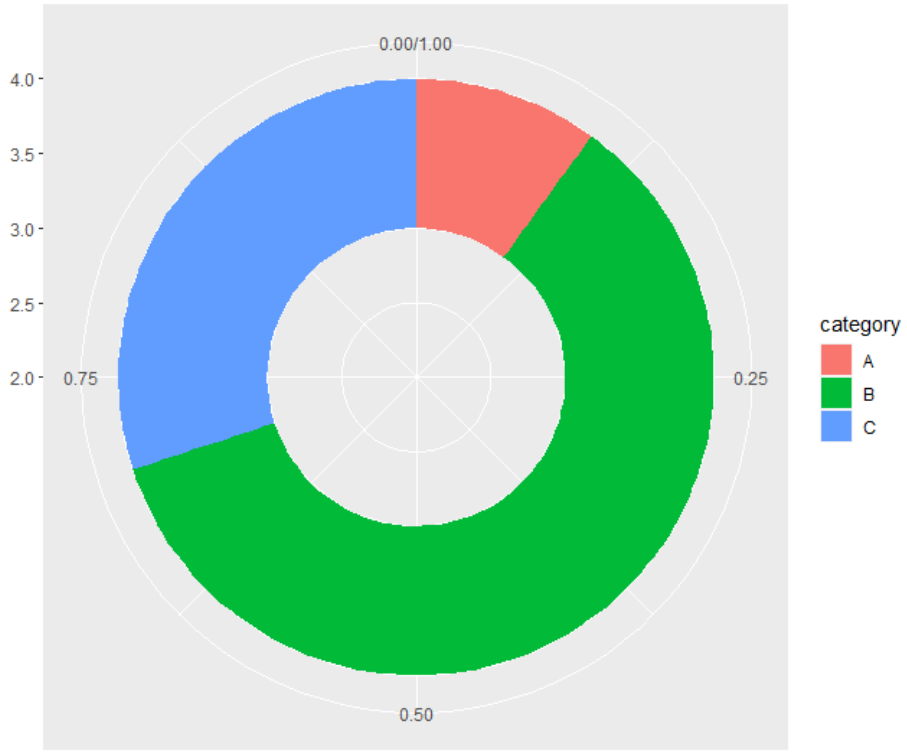
fig <- plot_ly(data, labels = ~Categorie, values = ~X1960, type = ‘pie’) fig <- fig %>% layout(title = ‘United States Personal Expenditures by Categories in 1960’, xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE), yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE))

fig




```
# load library

library(ggplot2)

# Create test data.

data <- data.frame(

  category=c("A", "B", "C"),

  count=c(10, 60, 30)

)


# Compute percentages
```

```
data$fraction = data$count / sum(data$count)


# Compute the cumulative percentages (top of each rectangle)

data$ymax = cumsum(data$fraction)


# Compute the bottom of each rectangle

data$ymin = c(0, head(data$ymax, n=-1))

# Make the plot

ggplot(data, aes(ymax=ymax, ymin=ymin, xmax=4, xmin=3, fill=category)) +

 geom_rect() +

 coord_polar(theta="y") + # Try to remove that to understand how the chart is built initially

 xlim(c(2, 4)) # Try to remove that to see how to make a pie chart
```



```
# Create the data for the line chart.
v <- c(7,12,28,3,41)
# Give the chart file a name.
png(file = "line_chart.jpg")
# Plot the bar chart.
plot(v,type = "o")
# Save the file.
dev.off()
```