```
!pip install faker
!pip install radar
```

```
Collecting faker
  Downloading Faker-19.2.0-py3-none-any.whl (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 6.3 MB/s eta 0:00:00
Requirement already satisfied: python-dateutil>=2.4 in /usr/local/lib/python3.10/dist-packages (from faker) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.4->faker) (1.16.0)
Installing collected packages: faker
Successfully installed faker-19.2.0
Collecting radar
  Downloading radar-0.3.tar.gz (4.5 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: radar
  Building wheel for radar (setup.py) ... done
  Created wheel for radar: filename=radar-0.3-py3-none-any.whl size=4987 sha256=12806fee2f2c69403ca3b54ca404e3b73797108e08aed0a33ac
  Stored in directory: /root/.cache/pip/wheels/b5/73/ec/597d19e0d44507094a8ce35e41b21f9e71bb599ba37491d4e0
Successfully built radar
Installing collected packages: radar
Successfully installed radar-0.3
```

```
from faker import Faker
fake = Faker()
```

```
import datetime
import math
import pandas as pd
import random
#for date & Time
import radar
```

```
def generateData(n):
  listdata = []
  start = datetime.datetime(2019, 8, 1)
  end = datetime.datetime(2019, 8, 30)
  delta = end - start
  for _ in range(n):
    date = radar.random_datetime(start='2019-08-1', stop='2019-08-30').strftime("%Y-%m-%d")
    price = round(random.uniform(900, 1000), 4)
    listdata.append([date, price])
  df = pd.DataFrame(listdata, columns = ['Date', 'Price'])
  df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d')
  df = df.groupby(by='Date').mean()

  return df
```

1. Line Chart

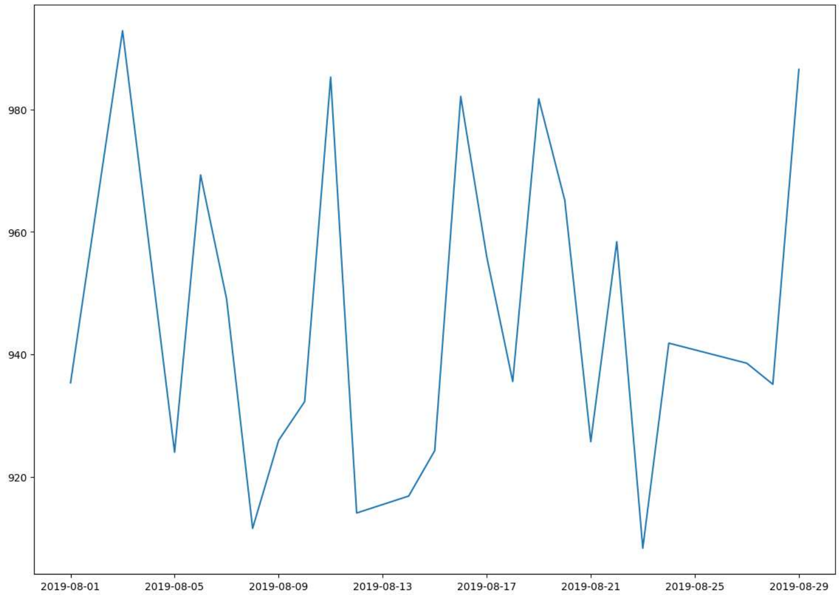
```
df = generateData(50)
df.head(10)
```

	Price
Date	
2019-08-01	935.364000
2019-08-03	992.821500
2019-08-05	924.037950
2019-08-06	969.289933
2019-08-07	949.042217
2019-08-08	911.613500
2019-08-09	925.978100
2019-08-10	932.300200
2019-08-11	985.251900
2019-08-12	914.124400

```
df.to_csv(r'stock.csv')
```

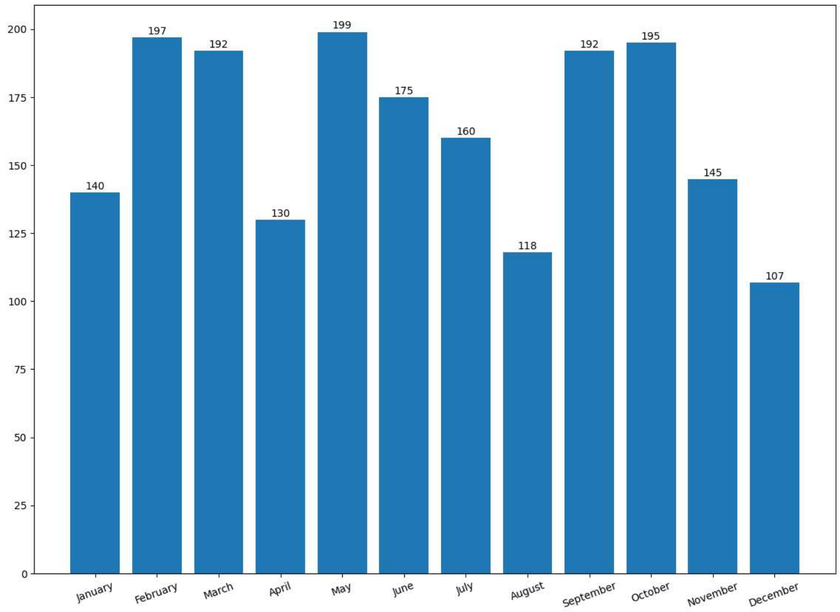
```
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (14, 10)
plt.plot(df)
```

```
[<matplotlib.lines.Line2D at 0x78a102a48a90>]
```

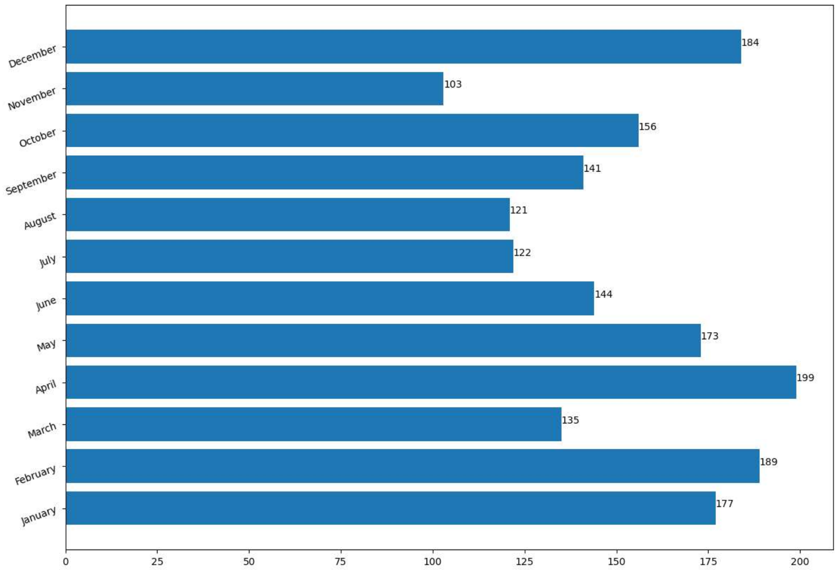


2.Bar Chart (Plots)

```
# Let us import the required libraries
import numpy as np
import calendar
import matplotlib.pyplot as plt

# Step 1: Set up the data. Remember range stoping parameter is exclusive. Meaning if you generate range from (1, 13), the last item 13 is
months = list(range(1, 13))
sold_quantity = [round(random.uniform(100, 200)) for x in range(1, 13)]

# Step 2: Specify the layout of the figure and allocate space.
figure, axis = plt.subplots()

# Step 3: In the X-axis, we would like to display the name of the months.
plt.xticks(months, calendar.month_name[1:13], rotation=20)

# Step 4: Plot the graph
plot = axis.bar(months, sold_quantity)

# Step 5: This step can be optinal depending upon if you are interested in displaying the data vaue on the head of the bar.
# It visually gives more meaning to show actual number of sold iteams on the bar itself.
for rectangle in plot:
  height = rectangle.get_height()
  axis.text(rectangle.get_x() + rectangle.get_width() /2., 1.002 * height, '%d' % int(height), ha='center', va = 'bottom')

# Step 6: Display the graph on the screen.
plt.show()
```



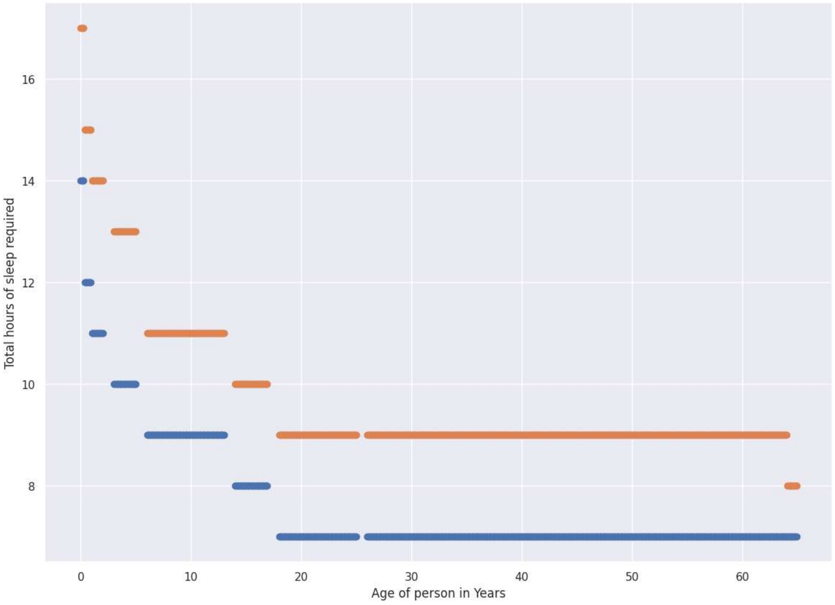
```
# Step 1: Set up the data. Remember range stoping parameter is exclusive. Meaning if you generate range from (1, 13), the last item 13 is
months = list(range(1, 13))
sold_quantity = [round(random.uniform(100, 200)) for x in range(1, 13)]

# Step 2: Specify the layout of the figure and allocate space.
figure, axis = plt.subplots()

# Step 3: In the X-axis, we would like to display the name of the months.
plt.yticks(months, calendar.month_name[1:13], rotation=20)

# Step 4: Plot the graph
plot = axis.barh(months, sold_quantity)

# Step 5: This step can be optinal depending upon if you are interested in displaying the data vaue on the head of the bar.
# It visually gives more meaning to show actual number of sold iteams on the bar itself.
for rectangle in plot:
  width = rectangle.get_width()
  axis.text(width + 2.5, rectangle.get_y() + 0.38, '%d' % int(width), ha='center', va = 'bottom')

# Step 6: Display the graph on the screen.
plt.show()
```



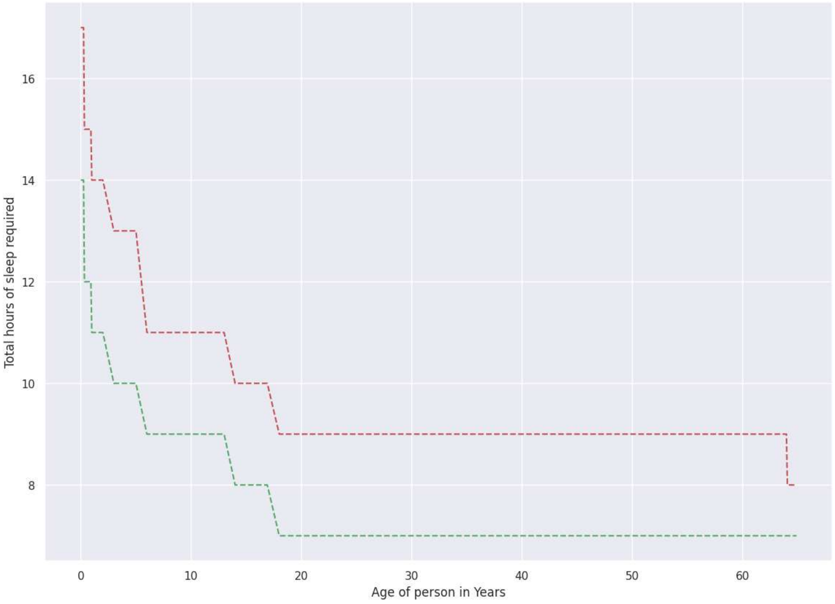
3.Scatter Plots

```
age = list(range(0, 65))
sleep = []

classBless = ['newborns(0-3)', 'infants(4-11)', 'toddlers(12-24)', 'preschoolers(36-60)', 'school-aged-children(72-156)', 'teenagers(168-
headers_cols = ['age','min_recommended', 'max_recommended', 'may_be_appropriate_min', 'may_be_appropriate_max', 'min_not_recommended', 'r

# Newborn (0-3)
for i in range(0, 4):
  min_recommended = 14
  max_recommended = 17
  may_be_appropriate_min = 11
  may_be_appropriate_max = 13
  min_not_recommended = 11
  max_not_recommended = 19
  sleep.append([i, min_recommended, max_recommended, may_be_appropriate_min, may_be_appropriate_max, min_not_recommended, max_not_recomme

# infants(4-11)
for i in range(4, 12):
  min_recommended = 12
  max_recommended = 15
  may_be_appropriate_min = 10
  may_be_appropriate_max = 11
  min_not_recommended = 10
  max_not_recommended = 18
  sleep.append([i, min_recommended, max_recommended, may_be_appropriate_min, may_be_appropriate_max, min_not_recommended, max_not_recomme

# toddlers(12-24)
for i in range(12, 25):
  min_recommended = 11
  max_recommended = 14
  may_be_appropriate_min = 9
  may_be_appropriate_max = 10
  min_not_recommended = 9
  max_not_recommended = 16
  sleep.append([i, min_recommended, max_recommended, may_be_appropriate_min, may_be_appropriate_max, min_not_recommended, max_not_recomme

# preschoolers(36-60)
for i in range(36, 61):
  min_recommended = 10
  max_recommended = 13
  may_be_appropriate_min = 8
  may_be_appropriate_max = 9
  min_not_recommended = 8
  max_not_recommended = 14
  sleep.append([i, min_recommended, max_recommended, may_be_appropriate_min, may_be_appropriate_max, min_not_recommended, max_not_recomme

# school-aged-children(72-156)
for i in range(72, 157):
  min_recommended = 9
  max_recommended = 11
  may_be_appropriate_min = 7
  may_be_appropriate_max = 8
  min_not_recommended = 7
  max_not_recommended = 12
  sleep.append([i, min_recommended, max_recommended, may_be_appropriate_min, may_be_appropriate_max, min_not_recommended, max_not_recomme

# teenagers(168-204)
for i in range(168, 204):
  min_recommended = 8
  max_recommended = 10
  may_be_appropriate_min = 7
  may_be_appropriate_max = 11
  min_not_recommended = 7
  max_not_recommended = 11
  sleep.append([i, min_recommended, max_recommended, may_be_appropriate_min, may_be_appropriate_max, min_not_recommended, max_not_recomme

# young-adults(216-300)
for i in range(216, 301):
  min_recommended = 7
  max_recommended = 9
  may_be_appropriate_min = 6
  may_be_appropriate_max = 11
  min_not_recommended = 6
  max_not_recommended = 11
  sleep.append([i, min_recommended, max_recommended, may_be_appropriate_min, may_be_appropriate_max, min_not_recommended, max_not_recomme

# adults(312-768)
for i in range(312, 769):
  min_recommended = 7
  max_recommended = 9
  may_be_appropriate_min = 6
  may_be_appropriate_max = 10
  min_not_recommended = 6
  max_not_recommended = 10
  sleep.append([i, min_recommended, max_recommended, may_be_appropriate_min, may_be_appropriate_max, min_not_recommended, max_not_recomme

# older-adults(>=780)
for i in range(769, 780):
  min_recommended = 7
  max_recommended = 8
  may_be_appropriate_min = 5
  may_be_appropriate_max = 6
  min_not_recommended = 5
  max_not_recommended = 9
  sleep.append([i, min_recommended, max_recommended, may_be_appropriate_min, may_be_appropriate_max, min_not_recommended, max_not_recomme

sleepDf = pd.DataFrame(sleep, columns=headers_cols)
sleepDf.head(10)
sleepDf.to_csv(r'sleep_vs_age.csv')
```

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set()

# A regular scatter plot
plt.scatter(x=sleepDf["age"]/12., y=sleepDf["min_recommended"])
plt.scatter(x=sleepDf["age"]/12., y=sleepDf['max_recommended'])
plt.xlabel('Age of person in Years')
plt.ylabel('Total hours of sleep required')
plt.show()
```

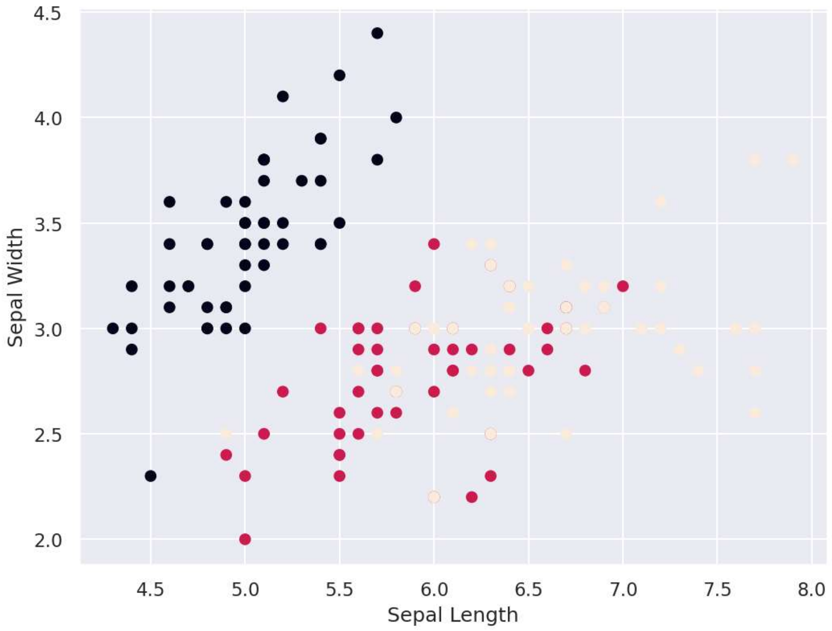


```
# Line plot
plt.plot(sleepDf['age']/12., sleepDf['min_recommended'], 'g--')
plt.plot(sleepDf['age']/12., sleepDf['max_recommended'], 'r--')
plt.xlabel('Age of person in Years')
plt.ylabel('Total hours of sleep required')
plt.show()
```



```
import seaborn as sns
import matplotlib.pyplot as plt
```

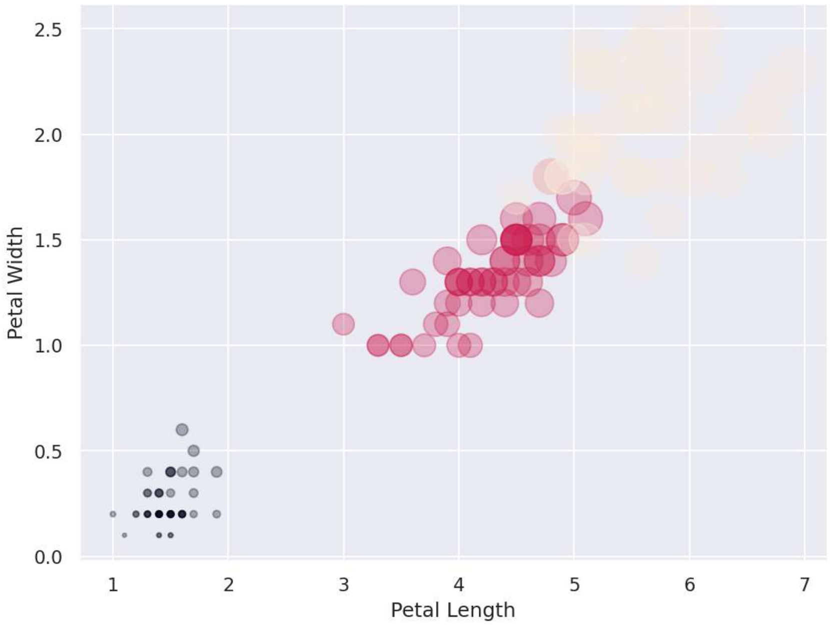
```
# Set some default parameters of matplotlib
plt.rcParams['figure.figsize'] = (8, 6)
plt.rcParams['figure.dpi'] = 150

# Use style froms seaborn. Try to comment the next line and see the difference in graph
sns.set()

# Load the Iris dataset
df = sns.load_dataset('iris')

df['species'] =  df['species'].map({'setosa': 0, "versicolor": 1, "virginica": 2})

# A regular scatter plot
plt.scatter(x=df["sepal_length"], y=df["sepal_width"], c = df.species)

# Create labels for axises
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')

# Display the plot on the screen
plt.show()
```



3a.Bubble plot

```
# Load the Iris dataset
df = sns.load_dataset('iris')

df['species'] =  df['species'].map({'setosa': 0, "versicolor": 1, "virginica": 2})

# Create bubble plot
plt.scatter(df.petal_length, df.petal_width,
            s=50*df.petal_length*df.petal_width,
            c=df.species,
            alpha=0.3
            )

# Create labels for axises
```

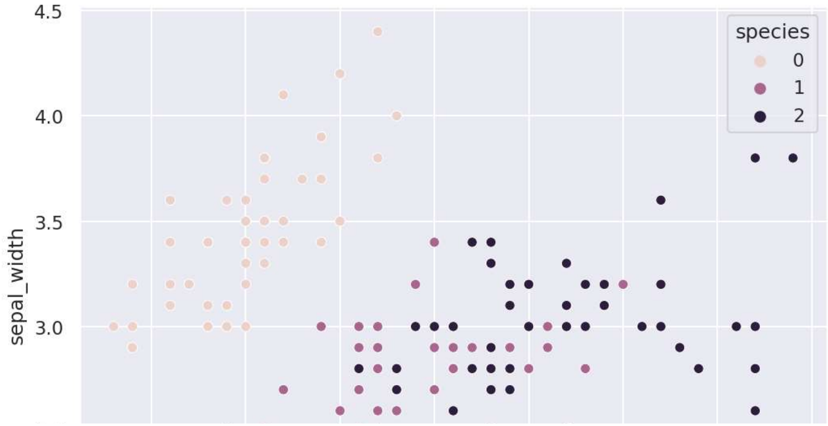
```
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
```



3b.Scatter plot using seaborn

```
df = sns.load_dataset('iris')

df['species'] =  df['species'].map({'setosa': 0, "versicolor": 1, "virginica": 2})
sns.scatterplot(x=df["sepal_length"], y=df["sepal_width"], hue=df.species, data=df)
```

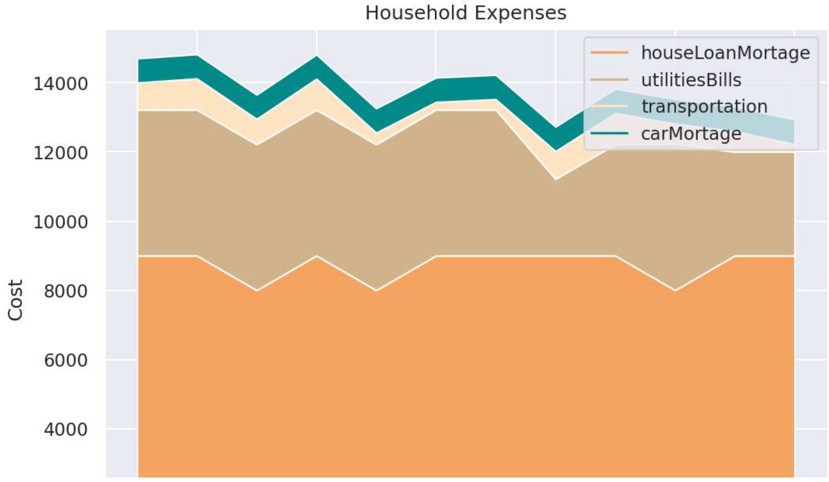
```
<Axes: xlabel='sepal_length', ylabel='sepal_width'>
```



4. Area plot and Stacked Plot

```
houseLoanMortage = [9000, 9000, 8000, 9000,
                    8000, 9000, 9000, 9000,
                    9000, 8000, 9000, 9000]
utilitiesBills = [4218, 4218, 4218, 4218,
                  4218, 4218, 4219, 2218,
                  3218, 4233, 3000, 3000]
transportation = [782, 900, 732, 892,
                  334, 222, 300, 800,
                  900, 582, 596, 222]
carMortage = [700, 701, 702, 703,
              704, 705, 706, 707,
              708, 709, 710, 711]
```

```
import matplotlib.pyplot as plt
import seaborn as sns

months= [x for x in range(1,13)]

sns.set()
plt.plot([],[], color='sandybrown', label='houseLoanMortage')
plt.plot([],[], color='tan', label='utilitiesBills')
plt.plot([],[], color='bisque', label='transportation')
plt.plot([],[], color='darkcyan', label='carMortage')

plt.stackplot(months, houseLoanMortage, utilitiesBills, transportation, carMortage, colors=['sandybrown', 'tan', 'bisque', 'darkcyan'])
plt.legend()

plt.title('Household Expenses')
plt.xlabel('Months of the year')
plt.ylabel('Cost')

plt.show()
```

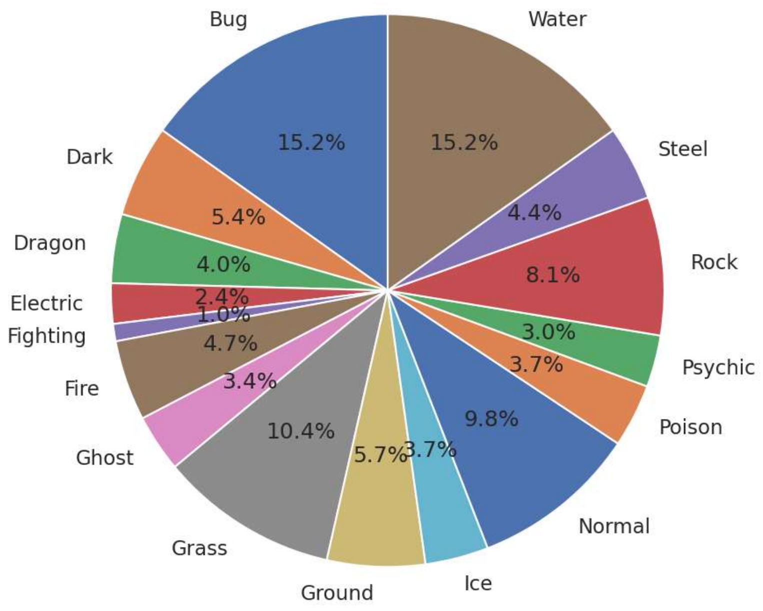


5. Pie Chart

```
# Create URL to JSON file (alternatively this can be a filepath)
url = 'https://raw.githubusercontent.com/hmcuesta/PDA_Book/master/Chapter3/pokemonByType.csv'

# Load the first sheet of the JSON file into a data frame
pokemon = pd.read_csv(url, index_col='type')

pokemon
```

	amount
type	
Bug	45
Dark	16
Dragon	12
Electric	7
Fighting	3
Fire	14
Ghost	10
Grass	31
Ground	17
Ice	11
Normal	29
Poison	11
Psychic	9
Rock	24
Steel	13
Water	45

```
import matplotlib.pyplot as plt

plt.pie(pokemon['amount'], labels=pokemon.index, shadow=False, startangle=90, autopct='%1.1f%%',)
plt.axis('equal')
plt.show()
```



Do you know you can directly use Pandas library to create pie chart? Check the one liner below.

```
pokemon.plot.pie(y="amount", figsize=(20, 10))
```

```
<Axes: ylabel='amount'>
```

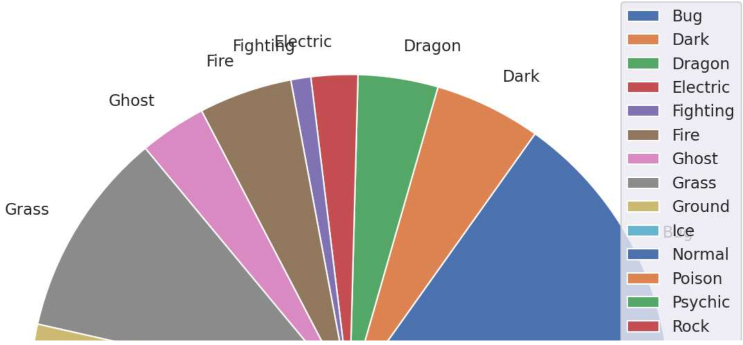
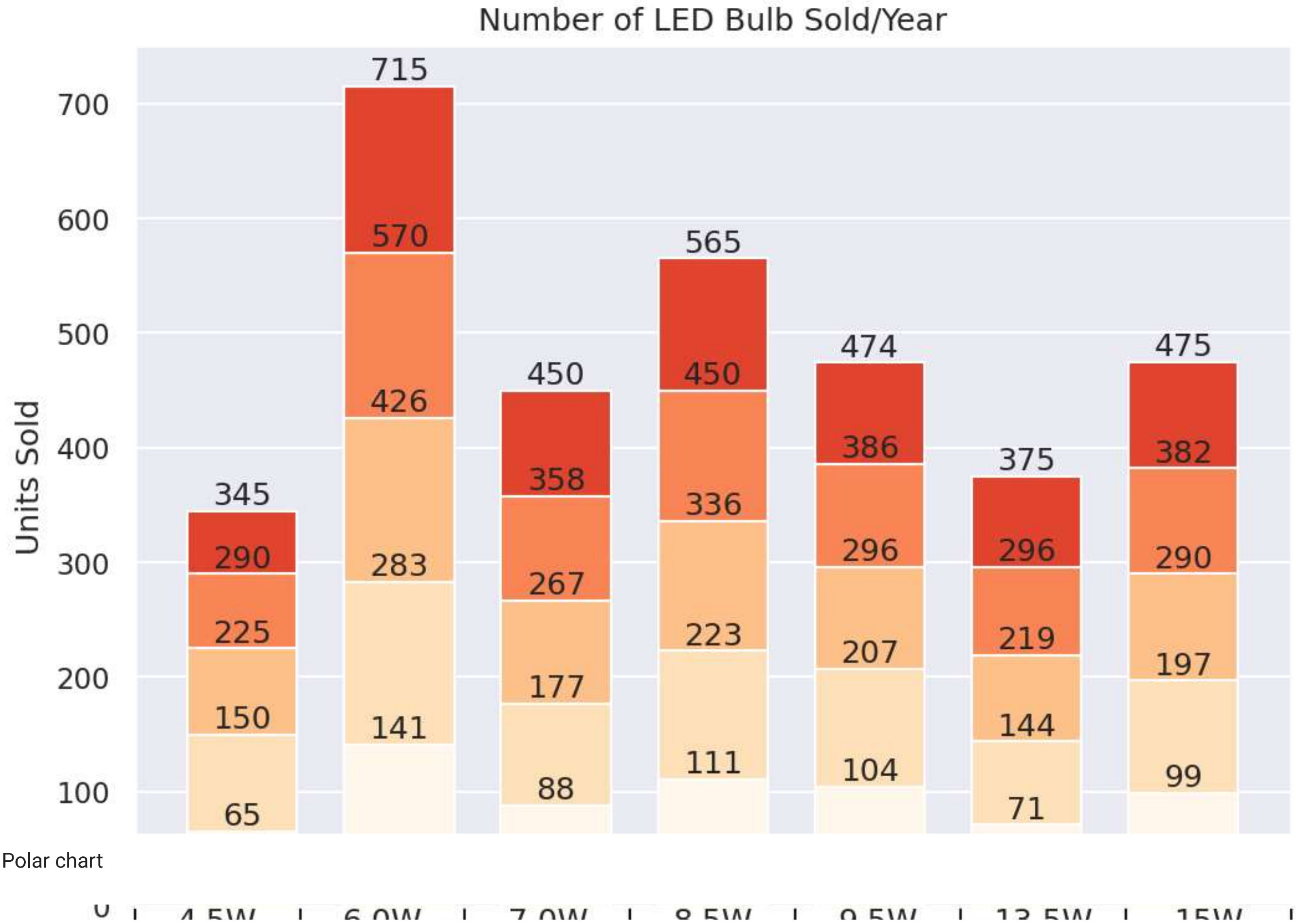


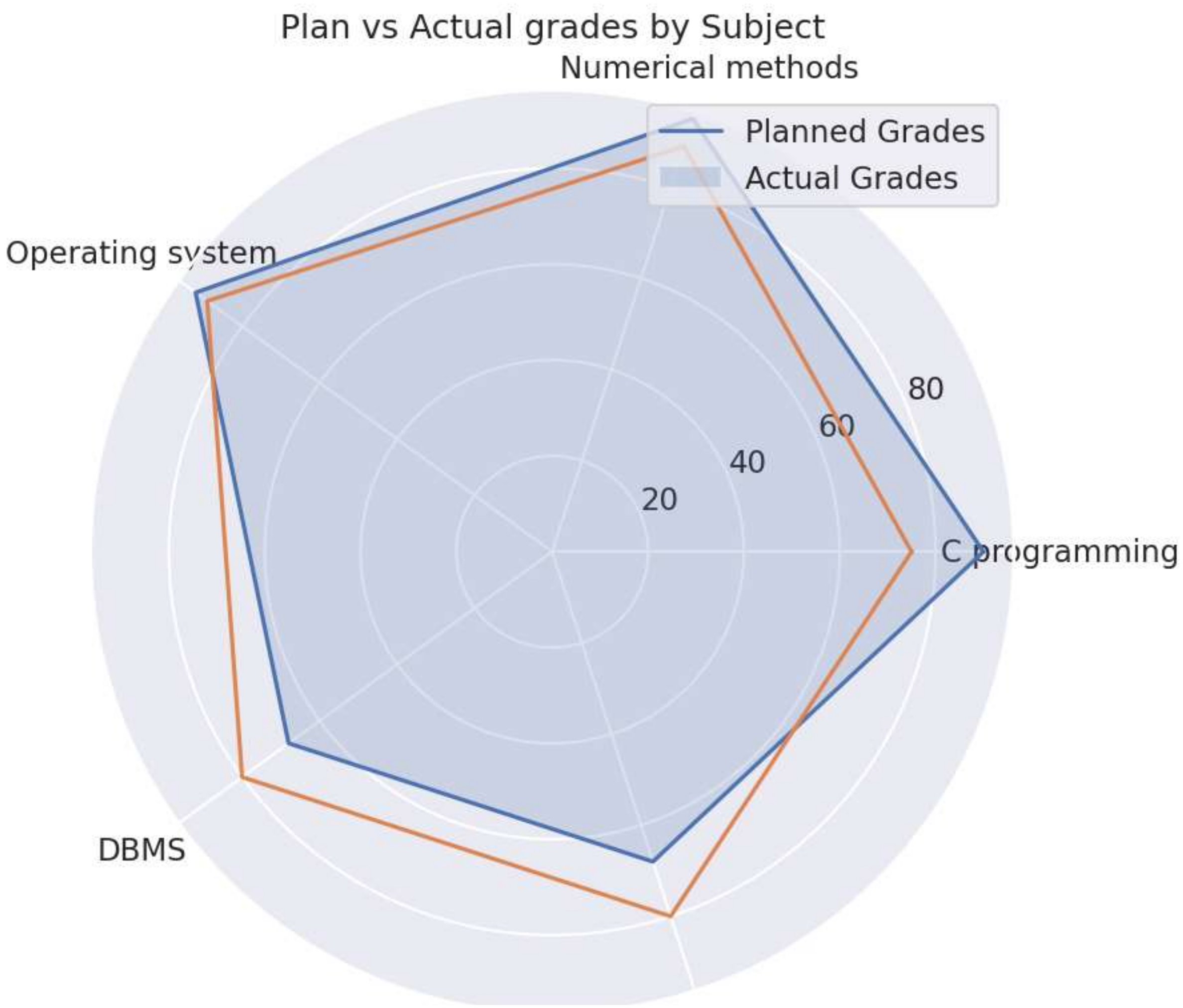
Table Chart

```
# Years under consideration
years = ["2010", "2011", "2012", "2013", "2014"]

# Available watt
columns = ['4.5W', '6.0W', '7.0W','8.5W','9.5W','13.5W','15W']
unitsSold = [
             [65, 141, 88, 111, 104, 71, 99],
             [85, 142, 89, 112, 103, 73, 98],
             [75, 143, 90, 113, 89, 75, 93],
             [65, 144, 91, 114, 90, 77, 92],
             [55, 145, 92, 115, 88, 79, 93],
            ]

# Define the range and scale for the y axis
values = np.arange(0, 600, 100)


colors = plt.cm.OrRd(np.linspace(0, 0.7, len(years)))
index = np.arange(len(columns)) + 0.3
bar_width = 0.7

y_offset = np.zeros(len(columns))
fig, ax = plt.subplots()

cell_text = []

n_rows = len(unitsSold)
for row in range(n_rows):
    plot = plt.bar(index, unitsSold[row], bar_width, bottom=y_offset,
                   color=colors[row])
    y_offset = y_offset + unitsSold[row]
    cell_text.append(['%1.1f' % (x) for x in y_offset])
    i=0
# Each iteration of this for loop, labels each bar with corresponding value for the given year
    for rect in plot:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2, y_offset[i],'%d'
                % int(y_offset[i]),
                ha='center', va='bottom')
        i = i+1

# Add a table to the bottom of the axes
the_table = plt.table(cellText=cell_text, rowLabels=years,
                rowColours=colors, colLabels=columns, loc='bottom')
plt.ylabel("Units Sold")
plt.xticks([])
plt.title('Number of LED Bulb Sold/Year')
plt.show()
```



6. Polar chart

```
# Let us assume you have 5 courses in your academic year.
subjects = ["C programming", "Numerical methods", "Operating system", "DBMS", "Computer Networks"]

# And you planned to obtained following grades in each subject
plannedGrade = [90, 95, 92, 68, 68, 90]

# However, after your final examination, this is the grade you got
actualGrade = [75, 89, 89, 80, 80, 75]


# 1. Import required libraries
import numpy as np
import matplotlib.pyplot as plt

# 2. Prepare the data set.
# 3. Set up theta

theta = np.linspace(0, 2 * np.pi, len(plannedGrade))

# 4. Initialize the plot by figure size and polar projection
plt.figure(figsize = (10,6))
plt.subplot(polar=True)

# 5. Get the grid lines to align with each of the subject names.
(lines,labels) = plt.thetagrids(range(0,360, int(360/len(subjects))),
                                                (subjects))

# 6. We use plot method to plot the graph. And fill the area under it.
plt.plot(theta, plannedGrade)
plt.fill(theta, plannedGrade, 'b', alpha=0.2)

# 7. Now, we plot the actual grade obtained
plt.plot(theta, actualGrade)

# 8. Finally, we add a legend and a nice comprehensible title to the plot.
plt.legend(labels=('Planned Grades','Actual Grades'),loc=1)
plt.title("Plan vs Actual grades by Subject")

# 9. Lastly, we show the plot on the screen.
plt.show()
```

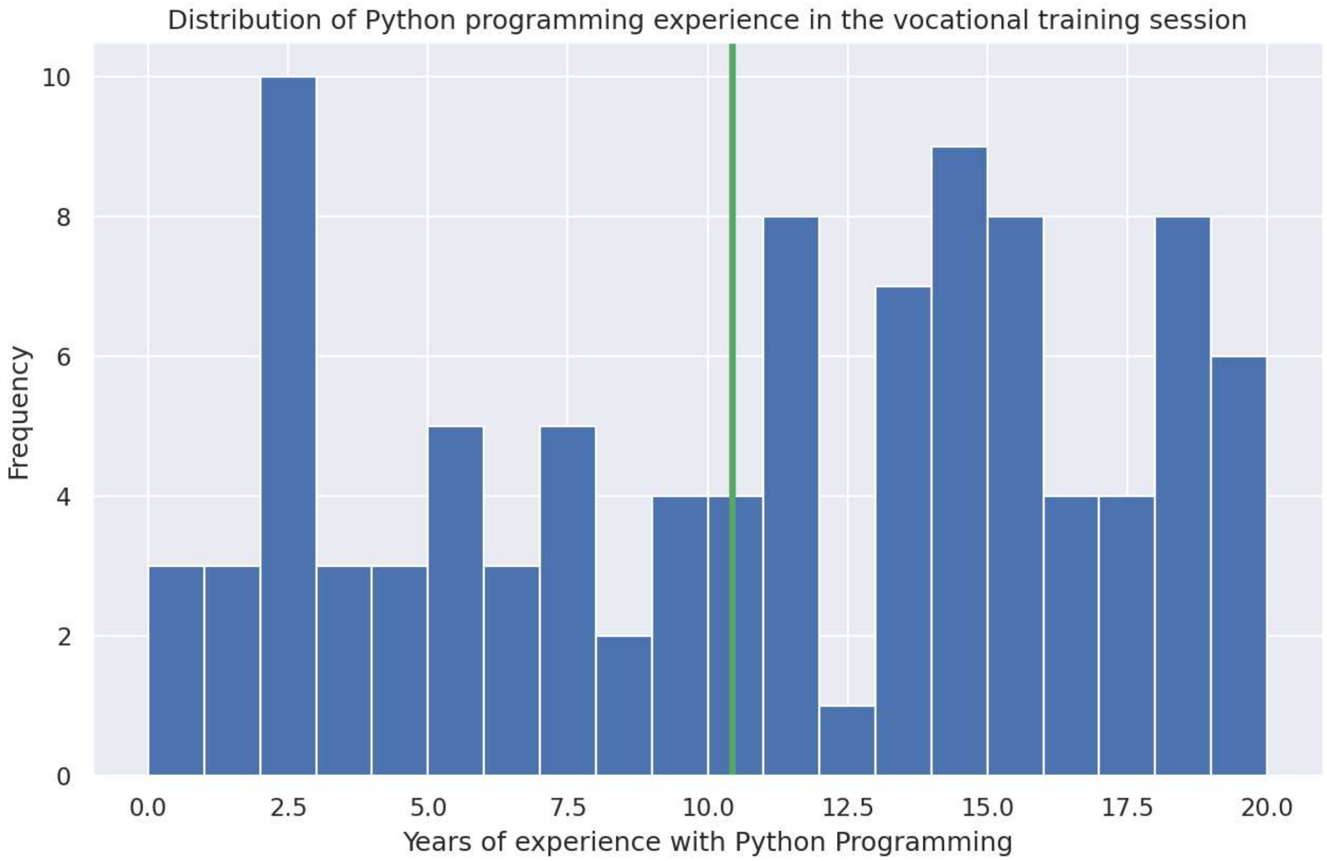


7. Histogram

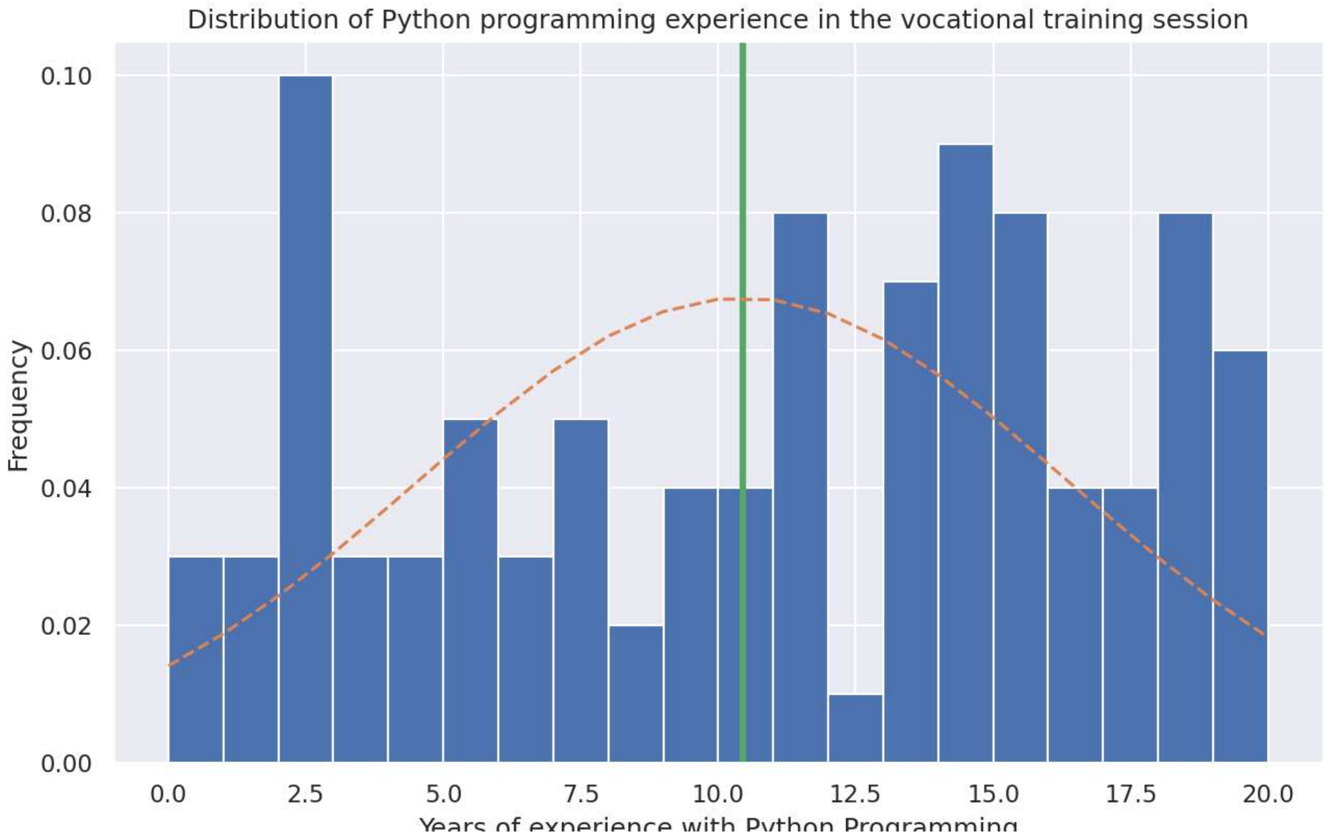
```
import numpy as np
import matplotlib.pyplot as plt

# 1. Create data set
yearsOfExperience = np.array([10, 16, 14,  5, 10, 11, 16, 14,  3, 14, 13, 19,  2,  5,  7,  3, 20,
       11, 11, 14,  2, 20, 15, 11,  1, 15, 15, 15,  2,  9, 18,  1, 17, 18,
       13,  9, 20, 13, 17, 13, 15, 17, 10,  2, 11,  8,  5, 19,  2,  4,  9,
       17, 16, 13, 18,  5,  7, 18, 15, 20,  2,  7,  0,  4, 14,  1, 14, 18,
        8, 11, 12,  2,  9,  7, 11,  2,  6, 15,  2, 14, 13,  4,  6, 15,  3,
        6, 10,  2, 11,  0, 18,  0, 13, 16, 18,  5, 14,  7, 14, 18])
yearsOfExperience
```

```
     array([10, 16, 14,  5, 10, 11, 16, 14,  3, 14, 13, 19,  2,  5,  7,  3, 20,
            11, 11, 14,  2, 20, 15, 11,  1, 15, 15, 15,  2,  9, 18,  1, 17, 18,
            13,  9, 20, 13, 17, 13, 15, 17, 10,  2, 11,  8,  5, 19,  2,  4,  9,
            17, 16, 13, 18,  5,  7, 18, 15, 20,  2,  7,  0,  4, 14,  1, 14, 18,
             8, 11, 12,  2,  9,  7, 11,  2,  6, 15,  2, 14, 13,  4,  6, 15,  3,
             6, 10,  2, 11,  0, 18,  0, 13, 16, 18,  5, 14,  7, 14, 18])
```

```
plt.figure(figsize = (10,6))

# 2. Plot the distribution of group experience
nbins = 20
n, bins, patches = plt.hist(yearsOfExperience, bins=nbins)

# 3. Add labels to the axis and title
plt.xlabel("Years of experience with Python Programming")
plt.ylabel("Frequency")
plt.title("Distribution of Python programming experience in the vocational training session")

# 4. Draw a green vertical line in the graph at the average experience:
plt.axvline(x=yearsOfExperience.mean(), linewidth=3, color = 'g')

# 5. Display the plot
plt.show()
```



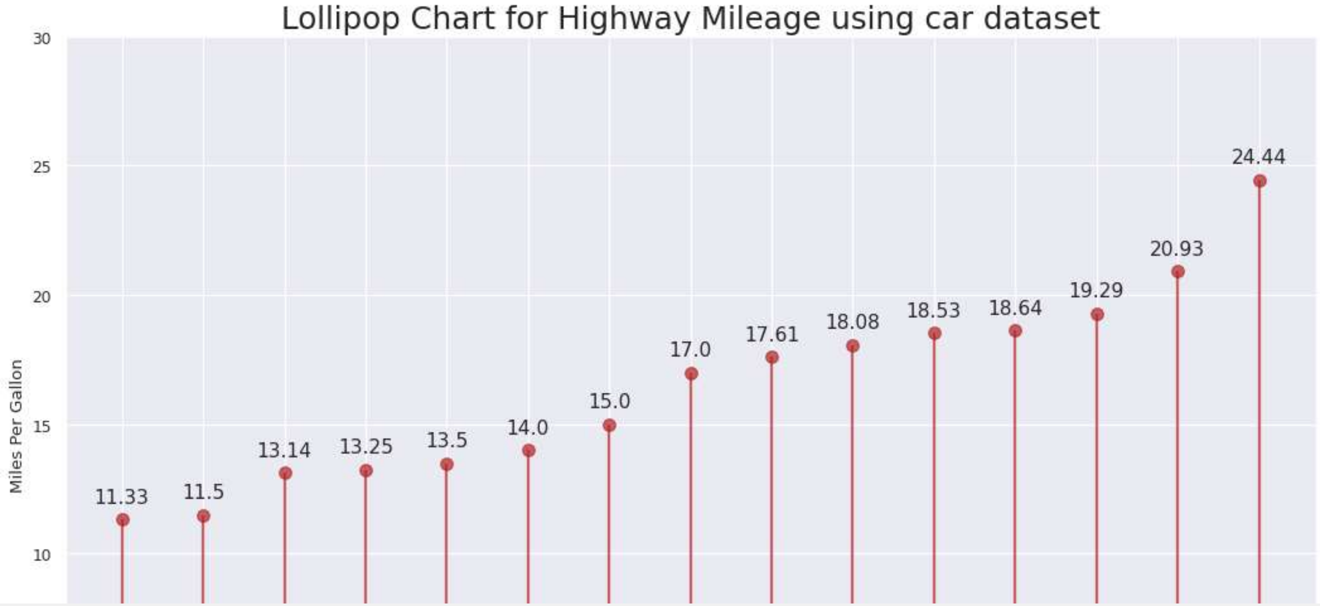
```
plt.figure(figsize = (10,6))

# 2. Plot the distribution of group experience
nbins = 20
n, bins, patches = plt.hist(yearsOfExperience, bins=nbins, density=1)

# 3. Add labels to the axis and title
plt.xlabel("Years of experience with Python Programming")
plt.ylabel("Frequency")
plt.title("Distribution of Python programming experience in the vocational training session")

# 4. Draw a green vertical line in the graph at the average experience:
plt.axvline(x=yearsOfExperience.mean(), linewidth=3, color = 'g')

# 5. Compute mean and standard deviation of the dataset.
mu = yearsOfExperience.mean()
sigma = yearsOfExperience.std()

# 6. Adding a best-fit line for normal distribution.
y = ((1 / (np.sqrt(2 * np.pi) * sigma)) * np.exp(-0.5 * (1 / sigma * (bins - mu))**2))

# 7. Plot the normal distribution
plt.plot(bins, y, '--')

# 8. Display the plot
plt.show()
```



8. Lollipop chart

```
# 1. Read the dataset

carDF = pd.read_csv('https://raw.githubusercontent.com/PacktPublishing/hands-on-exploratory-data-analysis-with-python/master/Chapter%202/

# 2. Group by manufacturer and take average milage
processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())

# 3. Sort the values by cty and reset index
processedDF.sort_values('cty', inplace=True)
processedDF.reset_index(inplace=True)

# 4. Plot the graph
fig, ax = plt.subplots(figsize=(16,10), dpi= 80)
ax.vlines(x=processedDF.index, ymin=0, ymax=processedDF.cty, color='firebrick', alpha=0.7, linewidth=2)
ax.scatter(x=processedDF.index, y=processedDF.cty, s=75, color='firebrick', alpha=0.7)

# 5. Annotate Title
ax.set_title('Lollipop Chart for Highway Mileage using car dataset', fontdict={'size':22})

# 6. Anotate labels and xticks, ylim
ax.set_ylabel('Miles Per Gallon')
ax.set_xticks(processedDF.index)
ax.set_xticklabels(processedDF.manufacturer.str.upper(), rotation=65, fontdict={'horizontalalignment': 'right', 'size':12})
ax.set_ylim(0, 30)

# 7. Write the values in the plot
for row in processedDF.itertuples():
    ax.text(row.Index, row.cty+.5, s=round(row.cty, 2), horizontalalignment= 'center', verticalalignment='bottom', fontsize=14)

# 8. Display the plot on the screen
plt.show()
```

```
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
<ipython-input-33-cd0eb2f09b41>:6: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  processedDF = carDF[['cty','manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
```



✓ 0s completed at 10:25 AM