Pandas & MatPlotLib

- Mostly Pandas & Numpy together is used for data cleaning & analysis.

[ ]:
```
!pip install pandas
```

[7]:
```
# Loading simple dataset
import pandas
mydataset = {
  'cities': ["New York", "Los Angeles", "Chicago"],
  'population': [8537673, 3976322, 2695598]
}

df = pandas.DataFrame(mydataset)

print(df)
```

```
        cities  population
0     New York     8537673
1  Los Angeles     3976322
2      Chicago     2695598
```

[10]:
```
# Pandas Series from a list
pd.Series([1, 7, 2])
```

[10]:
```
0    1
1    7
2    2
dtype: int64
```

[13]:
```
# Labelling the df
a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

```
x    1
y    7
z    2
dtype: int64
```

[16]:
```
# simple Pandas Series from a dictionary

temperatures = {"Monday": 25, "Tuesday": 28, "Wednesday": 30, "Thursday": 27, "Friday": 26}

temperature_series = pd.Series(temperatures, index=["Monday", "Wednesday", "Friday"])

print(temperature_series)
```

```
Monday       25
Wednesday    30
Friday       26
dtype: int64
```

[25]:
```
#refer to the row index:
nutrition_data = {
    "calories": [320, 360, 310],
    "duration": [40, 45, 35]
}

nutrition_df = pd.DataFrame(nutrition_data)

print(nutrition_df)
# refer to the 0th index
print(nutrition_df.loc[0])
```

```
   calories  duration
0       320        40
1       360        45
2       310        35
calories    320
duration     40
Name: 0, dtype: int64
```

[39]:
```
#Read data
df = pd.read_csv('https://people.sc.fsu.edu/~jburkardt/data/csv/addresses.csv')
df
```

[39]:

	John	Doe	120 jefferson st.	Riverside	NJ	08075
0	Jack	McGinnis	220 hobo Av.	Phila	PA	9119
1	John "Da Man"	Repici	120 Jefferson St.	Riverside	NJ	8075
2	Stephen	Tyler	7452 Terrace "At the Plaza" road	SomeTown	SD	91234
3	NaN	Blankman	NaN	SomeTown	SD	298
4	Joan "the bone", Anne	Jet	9th, at Terrace plc	Desert City	CO	123

[40]:
```
# Read Json
df = pd.read_json('./lerna.json')
df
```

[40]:

	version	loglevel	npmClient	useWorkspaces	command
publish	6.2.4	info	npm	True	{'conventionalCommits': True}
version	6.2.4	info	npm	True	{'push': False, 'conventionalCommits': True, '...

[44]:
```
# few functions
df.describe()
df.info()
df.index
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2 entries, publish to version
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   version         2 non-null      object
 1   loglevel        2 non-null      object
```

```
 2   npmClient      2 non-null      object
 3   useWorkspaces  2 non-null      bool
 4   command        2 non-null      object
dtypes: bool(1), object(4)
memory usage: 190.0+ bytes
```

[44]:
```
Index(['publish', 'version'], dtype='object')
```

[3]:
```python
import pandas as pd

# Creating a DataFrame from a dictionary
data = {
    "Name": ["Alice", "Bob", "Charlie", "David", "Eve"],
    "Age": [25, 30, 35, 40, 45],
    "Gender": ["Female", "Male", "Male", "Male", "Female"],
    "Occupation": ["Engineer", "Doctor", "Artist", "Lawyer", "Teacher"]
}
df = pd.DataFrame(data)

# Displaying the DataFrame
print("Original DataFrame:")
print(df)
print()

# Basic DataFrame attributes
print("Shape of DataFrame:", df.shape)
print("Number of rows:", len(df))
print("Column names:", df.columns)
print("Data types of columns:")
print(df.dtypes)
print()

# Accessing columns
print("Accessing columns:")
print(df["Name"])  # Accessing a single column
print(df[["Name", "Age"]])  # Accessing multiple columns
print()

# Accessing rows
print("Accessing rows using iloc:")
print(df.iloc[0])  # Accessing the first row
print(df.iloc[1:3])  # Accessing rows 2 to 3
print()

# Accessing rows based on condition
print("Accessing rows based on condition:")
print(df[df["Age"] > 30])  # Selecting rows where Age > 30
print()

# Adding a new column
df["City"] = ["New York", "Los Angeles", "Chicago", "Boston", "Seattle"]
print("DataFrame with a new column 'City':")
print(df)
print()

# Dropping columns
df.drop(columns=["Occupation"], inplace=True)
print("DataFrame after dropping 'Occupation' column:")
print(df)
print()

# Renaming columns
df.rename(columns={"Name": "Full Name", "Gender": "Sex"}, inplace=True)
print("DataFrame after renaming columns:")
print(df)
print()

# Sorting DataFrame
print("DataFrame sorted by Age in ascending order:")
print(df.sort_values(by="Age"))
print()


# Handling missing values
missing_data = {
    "Name": ["Frank", None, "Hannah"],
    "Age": [50, None, 55],
    "Gender": ["Male", "Female", None],
    "City": [None, "Miami", "Denver"]
}
missing_df = pd.DataFrame(missing_data)
print("DataFrame with missing values:")
print(missing_df)
print("Handling missing values:")
print(missing_df.dropna())  # Dropping rows with any missing values
print(missing_df.fillna({"Name": "Unknown", "Age": missing_df["Age"].mean(), "Gender": "Unknown", "City": "Unknown"}))  # Filling missing values
print()

# Concatenating DataFrames
concatenated_df = pd.concat([df, missing_df])
print("Concatenated DataFrame:")
print(concatenated_df)
print()

# Merging DataFrames
city_population = {
    "City": ["New York", "Los Angeles", "Chicago", "Boston", "Seattle"],
    "Population": [8537673, 3976322, 2695598, 692600, 724745]
}
city_pop_df = pd.DataFrame(city_population)
merged_df = pd.merge(df, city_pop_df, on="City", how="left")
print("Merged DataFrame:")
print(merged_df)
print()

# Exporting DataFrame to CSV
df.to_csv("data.csv", index=False)
print("DataFrame exported to 'data.csv'.")

# Reading DataFrame from CSV
read_df = pd.read_csv("data.csv")
print("DataFrame read from 'data.csv':")
print(read_df)
```

```
Original DataFrame:
      Name  Age  Gender Occupation
0    Alice   25  Female   Engineer
1      Bob   30    Male     Doctor
2  Charlie   35    Male     Artist
```

```
3    David   40    Male     Lawyer
4      Eve   45  Female    Teacher

Shape of DataFrame: (5, 4)
Number of rows: 5
Column names: Index(['Name', 'Age', 'Gender', 'Occupation'], dtype='object')
Data types of columns:
Name          object
Age            int64
Gender        object
Occupation    object
dtype: object

Accessing columns:
0      Alice
1        Bob
2    Charlie
3      David
4        Eve
Name: Name, dtype: object
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   35
3    David   40
4      Eve   45

Accessing rows using iloc:
Name             Alice
Age                 25
Gender          Female
Occupation    Engineer
Name: 0, dtype: object
      Name  Age Gender Occupation
1      Bob   30   Male     Doctor
2  Charlie   35   Male     Artist

Accessing rows based on condition:
      Name  Age  Gender Occupation
2  Charlie   35    Male     Artist
3    David   40    Male     Lawyer
4      Eve   45  Female    Teacher

DataFrame with a new column 'City':
      Name  Age  Gender Occupation         City
0    Alice   25  Female   Engineer     New York
1      Bob   30    Male     Doctor  Los Angeles
2  Charlie   35    Male     Artist      Chicago
3    David   40    Male     Lawyer       Boston
4      Eve   45  Female    Teacher      Seattle

DataFrame after dropping 'Occupation' column:
      Name  Age  Gender         City
0    Alice   25  Female     New York
1      Bob   30    Male  Los Angeles
2  Charlie   35    Male      Chicago
3    David   40    Male       Boston
4      Eve   45  Female      Seattle

DataFrame after renaming columns:
  Full Name  Age     Sex         City
0     Alice   25  Female     New York
1       Bob   30    Male  Los Angeles
2   Charlie   35    Male      Chicago
3     David   40    Male       Boston
4       Eve   45  Female      Seattle

DataFrame sorted by Age in ascending order:
  Full Name  Age     Sex         City
0     Alice   25  Female     New York
1       Bob   30    Male  Los Angeles
2   Charlie   35    Male      Chicago
3     David   40    Male       Boston
4       Eve   45  Female      Seattle

DataFrame with missing values:
     Name   Age  Gender    City
0   Frank  50.0    Male    None
1    None   NaN  Female   Miami
2  Hannah  55.0    None  Denver
Handling missing values:
Empty DataFrame
Columns: [Name, Age, Gender, City]
Index: []
      Name   Age   Gender     City
0    Frank  50.0     Male  Unknown
1  Unknown  52.5   Female    Miami
2   Hannah  55.0  Unknown   Denver

Concatenated DataFrame:
  Full Name   Age     Sex         City    Name  Gender
0     Alice  25.0  Female     New York     NaN     NaN
1       Bob  30.0    Male  Los Angeles     NaN     NaN
2   Charlie  35.0    Male      Chicago     NaN     NaN
3     David  40.0    Male       Boston     NaN     NaN
4       Eve  45.0  Female      Seattle     NaN     NaN
0       NaN  50.0     NaN         None   Frank    Male
1       NaN   NaN     NaN        Miami    None  Female
2       NaN  55.0     NaN       Denver  Hannah    None

Merged DataFrame:
  Full Name  Age     Sex         City  Population
0     Alice   25  Female     New York     8537673
1       Bob   30    Male  Los Angeles     3976322
2   Charlie   35    Male      Chicago     2695598
3     David   40    Male       Boston      692600
4       Eve   45  Female      Seattle      724745

DataFrame exported to 'data.csv'.
DataFrame read from 'data.csv':
  Full Name  Age     Sex         City
0     Alice   25  Female     New York
1       Bob   30    Male  Los Angeles
2   Charlie   35    Male      Chicago
3     David   40    Male       Boston
4       Eve   45  Female      Seattle
```

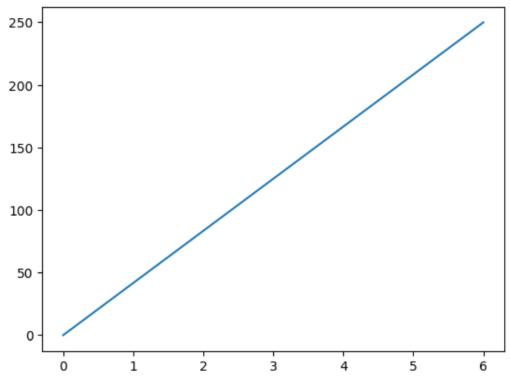
Matplotlib

[ ]:
```
!pip install matplotlib
```

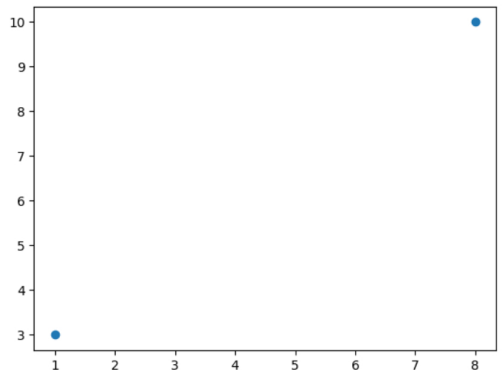
Plotting

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

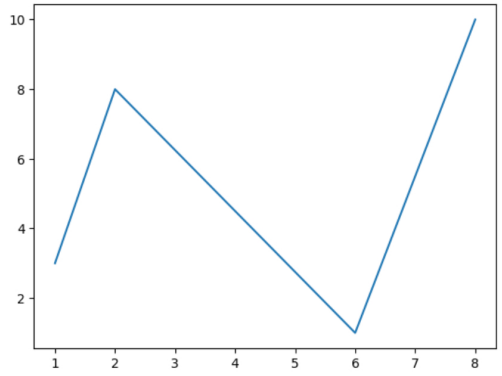
```
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
#X, Y Axis
plt.plot(xpoints, ypoints)
plt.show()
```



[3]:
```
#rINGS
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints, 'o')
plt.show()
```

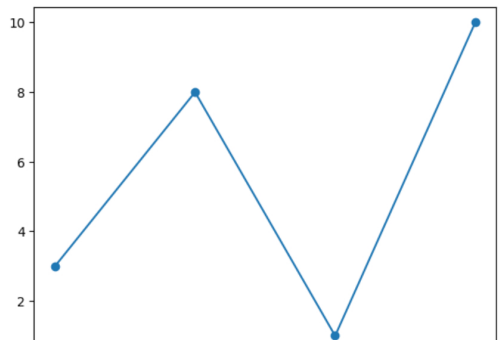


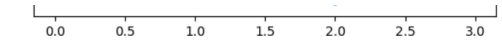
[5]:
```
# Same number of pts in both the axis
xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

plt.plot(xpoints, ypoints)
plt.show()
```



Markers

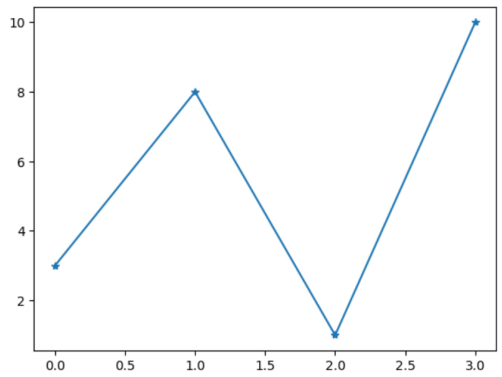
[7]:
```
# Marking the point with a circle
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o')
plt.show()
```



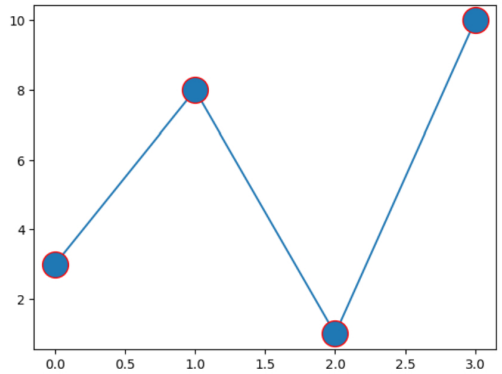


[13]:
```
plt.plot(ypoints, marker = '*')
```

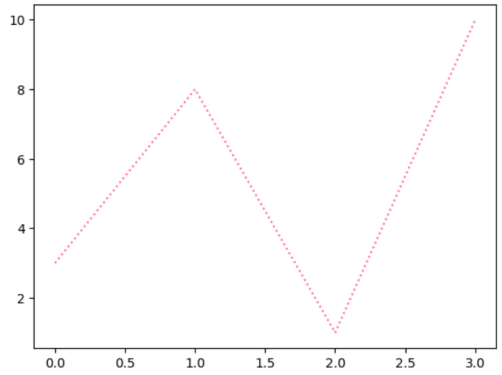
[13]:
```
[<matplotlib.lines.Line2D at 0x1a920713280>]
```



[21]:
```
ypoints = np.array([3, 8, 1, 10])
# setting size of the markers
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r')
plt.show()
```

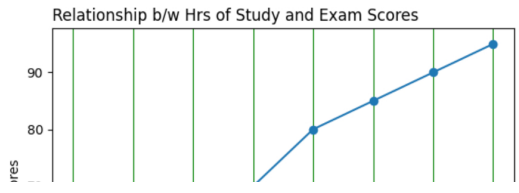


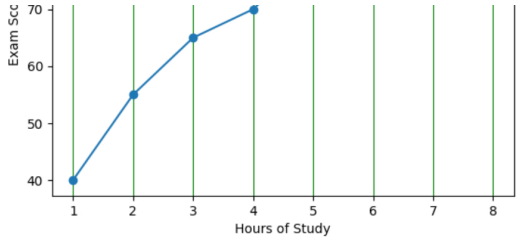
[27]:
```
#linestyle -> dotted, dashed
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linestyle = 'dotted', c = 'hotpink')
plt.show()
```



Labels

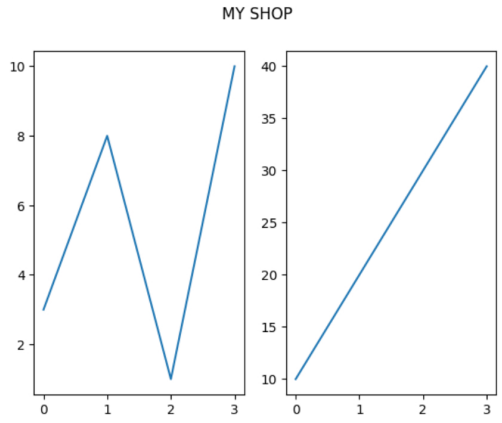
[43]:
```
# Labels & title & Grids
study_hours = np.array([1, 2, 3, 4, 5, 6, 7, 8])

exam_scores = np.array([40, 55, 65, 70, 80, 85, 90, 95])

plt.plot(study_hours, exam_scores, marker='o', linestyle='-')

font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.xlabel("Hours of Study")
plt.ylabel("Exam Scores")
plt.title("Relationship b/w Hrs of Study and Exam Scores", loc = 'left')

plt.grid(True, axis = 'x', color = 'green')
plt.show()
```





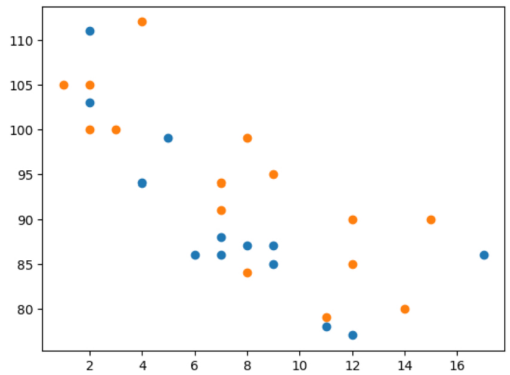
Subplot() -> Multiple Plots in single figure

[52]:
```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

#1 row , 2 cols , 1st plot
plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

#1 row , 2 cols , 2nd plot
plt.subplot(1, 2, 2)
plt.plot(x,y)

plt.suptitle("MY SHOP")

plt.show()
```



Scatter Plot

[58]:
```
#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()

# Color of each dot.
# Title can be set
# Axis names can be set
# All configurations can be availed here
```

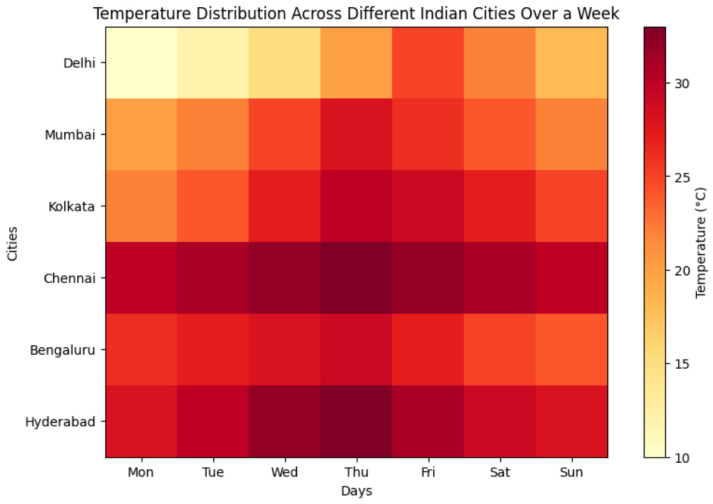


ColorMap

- List of colors, where each color has a value that ranges from 0 to 100

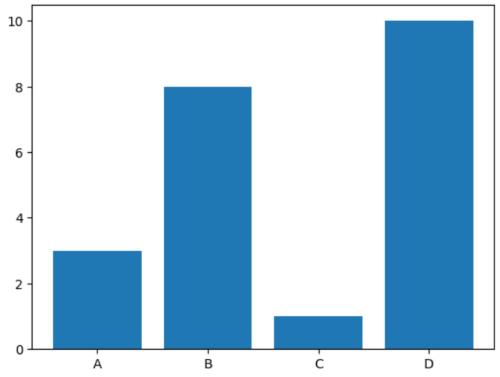
[65]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define Indian city names
cities = ['Delhi', 'Mumbai', 'Kolkata', 'Chennai', 'Bengaluru', 'Hyderabad']

# Define days of the week
```

```
days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

# Simulate temperature data for each city over the week
# These temperature ranges are approximate and vary based on the season
temperature_data = np.array([
    [10, 12, 15, 20, 25, 22, 18],  # Delhi
    [20, 22, 25, 28, 26, 24, 22],  # Mumbai
    [22, 24, 27, 30, 29, 27, 25],  # Kolkata
    [30, 31, 32, 33, 32, 31, 30],  # Chennai
    [26, 27, 28, 29, 27, 25, 24],  # Bengaluru
    [28, 30, 32, 33, 31, 29, 28]   # Hyderabad
])

# Create the heatmap
plt.figure(figsize=(10, 6))
heatmap = plt.imshow(temperature_data, cmap='YlOrRd')

# Add a colorbar
plt.colorbar(heatmap, label='Temperature (°C)')

# Set labels for x and y axis
plt.xticks(np.arange(len(days)), days)
plt.yticks(np.arange(len(cities)), cities)

# Add title and labels
plt.title('Temperature Distribution Across Different Indian Cities Over a Week')
plt.xlabel('Days')
plt.ylabel('Cities')

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

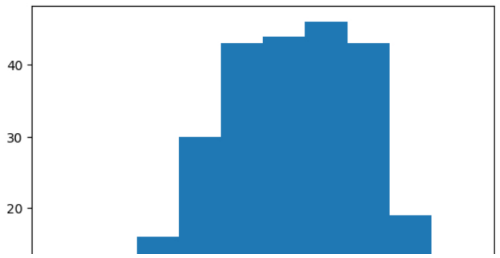


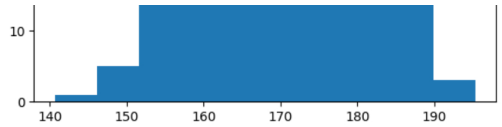
Bar Chart:

[66]:
```
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```



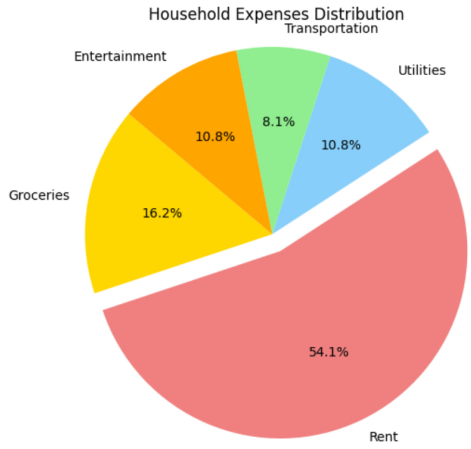
[83]:
```
# Histogram
# frequency distributions
# showing the number of observations within each given interval

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```





PIE CHART

[87]:
```
# Define expense categories and their corresponding amounts (in USD)
categories = ['Groceries', 'Rent', 'Utilities', 'Transportation', 'Entertainment']
expenses = [300, 1000, 200, 150, 200]

# Define colors for each category
colors = ['gold', 'lightcoral', 'lightskyblue', 'lightgreen', 'orange']

# Explode the slice corresponding to 'Rent'
explode = (0, 0.1, 0, 0, 0)  # "explode" the 2nd slice (Rent)

# Create a pie chart
plt.figure(figsize=(8, 6))
plt.pie(expenses, explode=explode, labels=categories, colors=colors, autopct='%1.1f%%', startangle=140)

# Add a title
plt.title('Household Expenses Distribution')

# Equal aspect ratio ensures that pie is drawn as a circle
plt.axis('equal')

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



[ ]:

[ ]: