PROGRAMMING FOR AI

Lecture 5

Libraries & Visualization

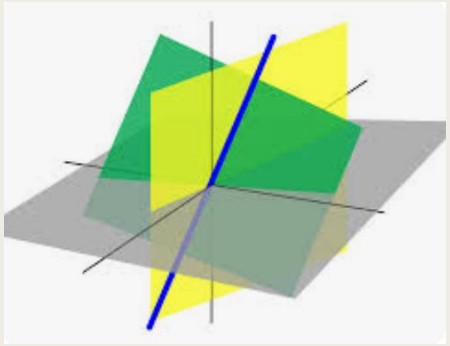
Instructor: Zafar Iqbal

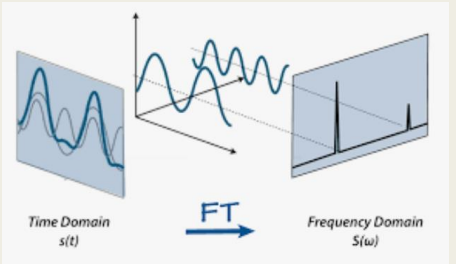
Agenda

- NumPy: Numerical computing
- Pandas: Data manipulation
- Matplotlib/Seaborn: Data visualization

What is NumPy?

- A **Python library** used for working with **arrays.**
- Provides functions for:
  - **Linear algebra**
  - **Fourier transform**
  - **Matrices**
- Created in **2005 by Travis Oliphant.**
- **Open-source** (free to use).
- **Stands for "Numerical Python".**
- Primarily **Python**, but performance-critical parts are in **C or C++.**





What is Numpy

- NumPy is a Python library.
- NumPy is used for working with arrays.
- NumPy is short for "Numerical Python“
- Example
  - *Create a NumPy array:*

```
import numpy
arr = numpy.array([1, 2, 3, 4, 5])
print(arr)
```

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

Create a NumPy ndarray Object

- NumPy is used to work with arrays. The array object in NumPy is called ndarray.
- We can create a NumPy ndarray object by using the array() function.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

2-D Arrays

- An array that has 1-D arrays as its elements is called a 2-D array.
- These are often used to represent matrix or 2nd order tensors.

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)
```

3-D arrays

- An array that has 2-D arrays (matrices) as its elements is called 3-D array.
- These are often used to represent a 3rd order tensor.

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

Number of Dimensions

- NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Higher Dimensional Arrays

- Create an array with 5 dimensions and verify that it has 5 dimensions:

```
import numpy as np

arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)
print('number of dimensions :', arr.ndim)
```

NumPy Data Types

- Below is a list of all data types in NumPy and the characters used to represent them.
  - *i - integer*
  - *b - boolean*
  - *u - unsigned integer*
  - *f - float*
  - *c - complex float*
  - *m - timedelta*
  - *M - datetime*
  - *O - object*
  - *S - string*
  - *U - unicode string*
  - *V - fixed chunk of memory for other type (void)*

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr.dtype)
```

Creating Arrays With a Defined Data Type

```
import numpy as np

arr = np.array([1, 2, 3, 4], dtype='S')

print(arr)
print(arr.dtype)
```

copy

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

Get the Shape of an Array

- NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

```
import numpy as np

arr =
np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(arr.shape)
```

Random Data Distribution

- Generate a 1-D array containing 100 values, where each value has to be 3, 5, 7 or 9.

```
from numpy import random

x = random.choice([3, 5, 7, 9],
p=[0.1, 0.3, 0.6, 0.0], size=(100))

print(x)
```

Pandas

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- Relevant data is very important in data science.
- Pandas gives you answers about the data. Like:
- Is there a correlation between two or more columns?
  - *What is average value?*
  - *Max value?*
  - *Min value?*
- Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

Pandas as pd

```
import pandas as pd

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pd.DataFrame(mydataset)

print(myvar)
```

Pandas Series

- What is a Series?
- A Pandas Series is like a column in a table.
- It is a one-dimensional array holding data of any type.

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

- Create Labels
- With the index argument, you can name your own labels.

```
import pandas as pd

a = [1, 7, 2]

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

print(myvar)
```

Pandas DataFrames

- A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

Locate Row

- As you can see from the result above, the DataFrame is like a table with rows and columns.
- Pandas use the loc attribute to return one or more specified row(s)

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

#refer to the row index:
print(df.loc[0])
```

Pandas

```
import pandas

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pandas.DataFrame(mydataset)

print(myvar)
```

```
import pandas as pd

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pd.DataFrame(mydataset)

print(myvar)
```

Pandas Read CSV

- Read CSV Files
- A simple way to store big data sets is to use CSV files (comma separated files).
- CSV files contains plain text and is a well known format that can be read by everyone including Pandas.
- In our examples we will be using a CSV file called 'data.csv'.

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df)
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

Matplotlib/Seaborn

- **Seaborn** and **Matplotlib** are both popular Python libraries used for data visualization
- Key Differences

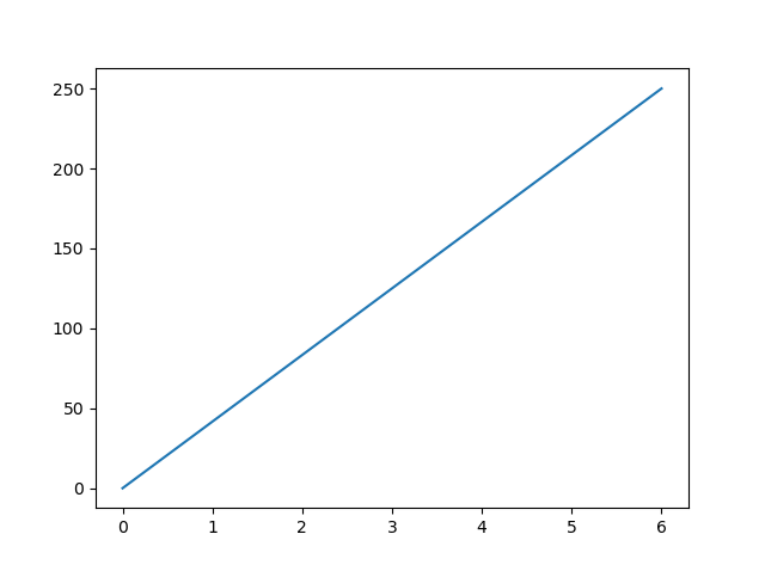
Aspect	Matplotlib	Seaborn
Plot Types	General-purpose: line, scatter, bar, pie, 3D, etc.	Specialized: boxplots, violin plots, pair plots, heatmaps, categorical plots.
Dependency	Standalone library (depends on NumPy)	Built on Matplotlib; requires Matplotlib as a dependency.
Aesthetics	Basic default styles; requires manual styling for professional look.	Attractive default themes and color palettes optimized for readability.

Matplotlib/Seaborn

- Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias
- Seaborn is imported under the sns alias (import seaborn as sns)

- Draw a line in a diagram from position (0,0) to position (6,250)

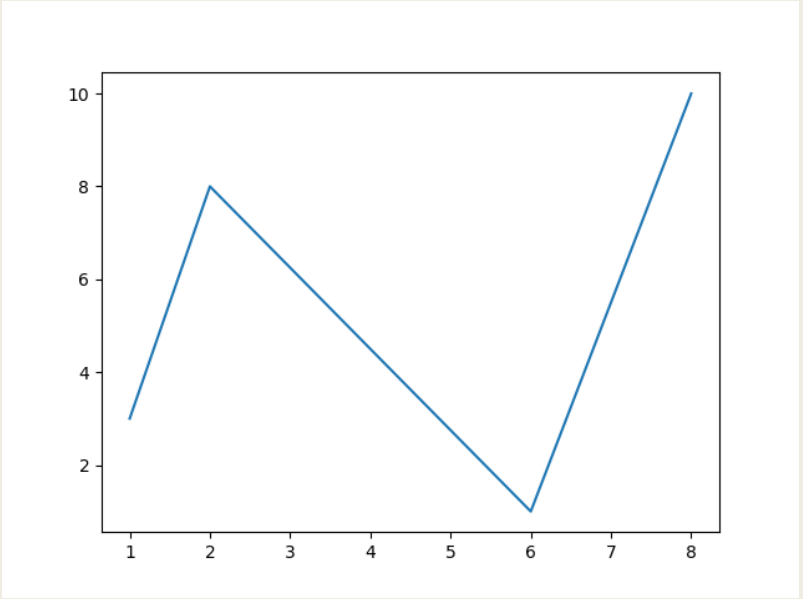
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

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



Multiple Points

- You can plot as many points as you like, just make sure you have the same number of points in both axis.
- Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10)

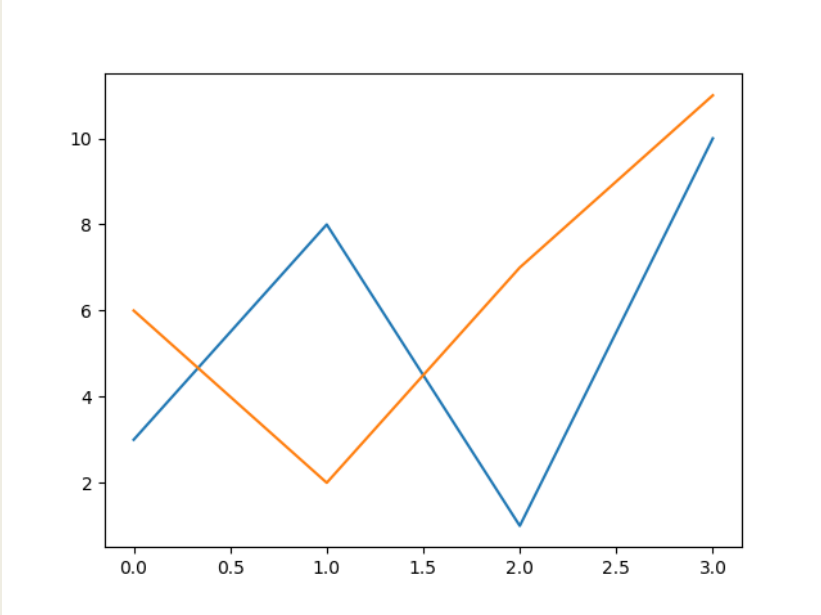
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

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



Multiple Lines

- You can plot as many lines as you like by simply adding more plt.plot() functions
- Draw two lines by specifying a plt.plot() function for each line

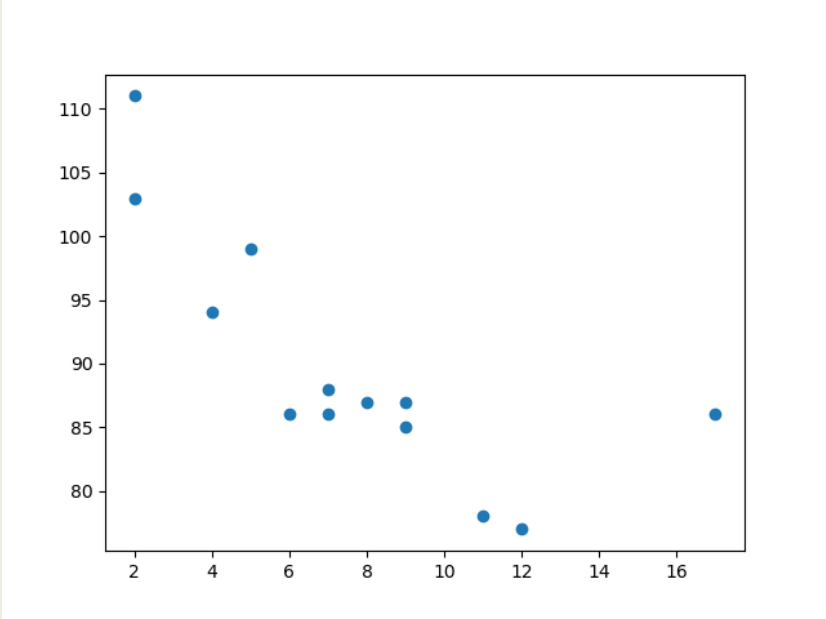
```
import matplotlib.pyplot as plt
import numpy as np

x1 = np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])
x2 = np.array([0, 1, 2, 3])
y2 = np.array([6, 2, 7, 11])

plt.plot(x1, y1, x2, y2)
plt.show()
```



Scatter Plots

- The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis

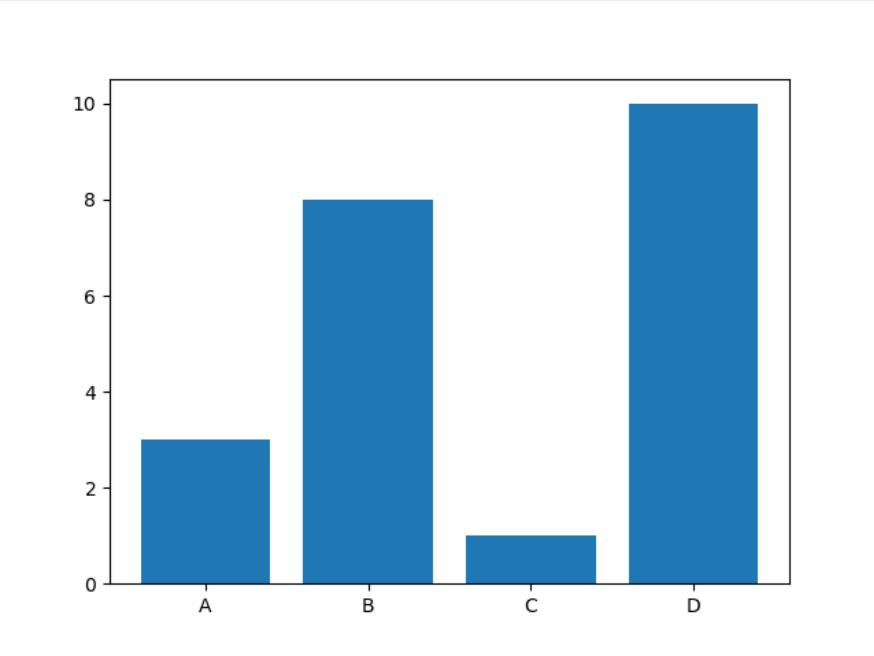
```
import matplotlib.pyplot as plt
import numpy as np

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)
plt.show()
```



Creating Bars

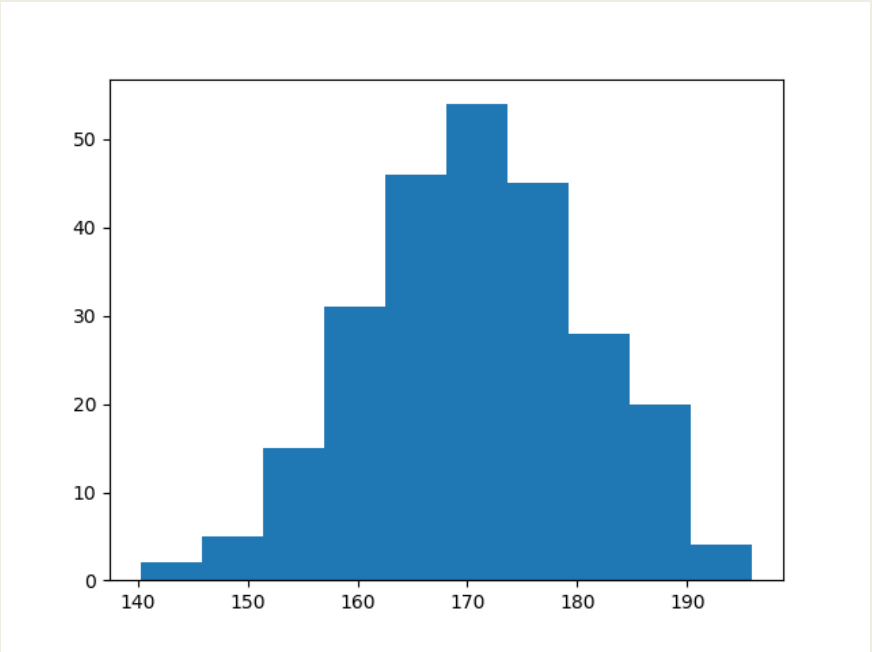
- With Pyplot, you can use the bar() function to draw bar graphs

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

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

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



Matplotlib Histograms

- A histogram is a graph showing *frequency* distributions.
- You can read from the histogram that there are approximately:
- 2 people from 140 to 145cm
  5 people from 145 to 150cm
  15 people from 151 to 156cm
  31 people from 157 to 162cm
  46 people from 163 to 168cm
  53 people from 168 to 173cm
  45 people from 173 to 178cm
  28 people from 179 to 184cm
  21 people from 185 to 190cm
  4 people from 190 to 195cm



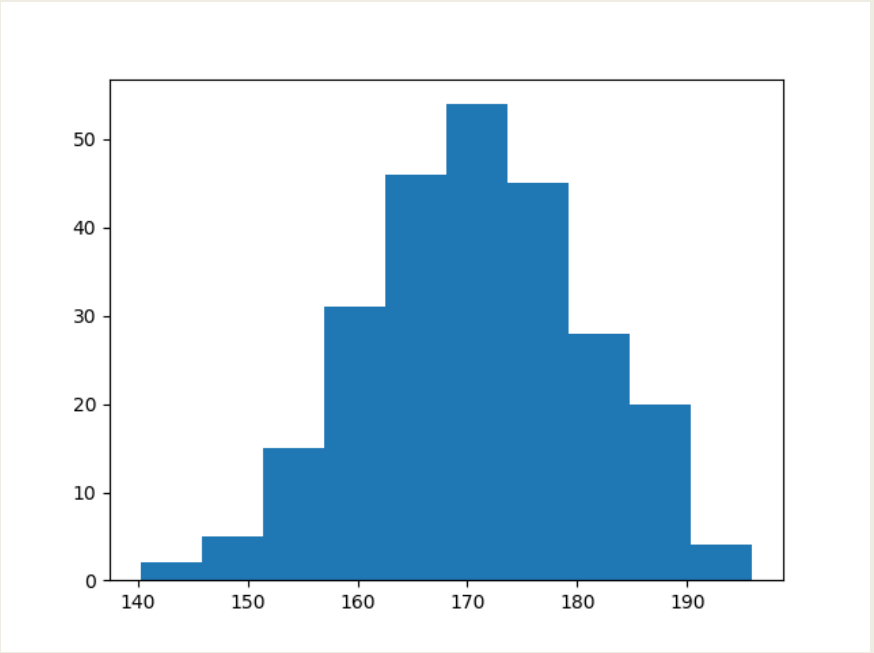
Matplotlib Histograms

- This will generate a *random* result, and could look like this

```
import numpy as np

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

print(x)
```

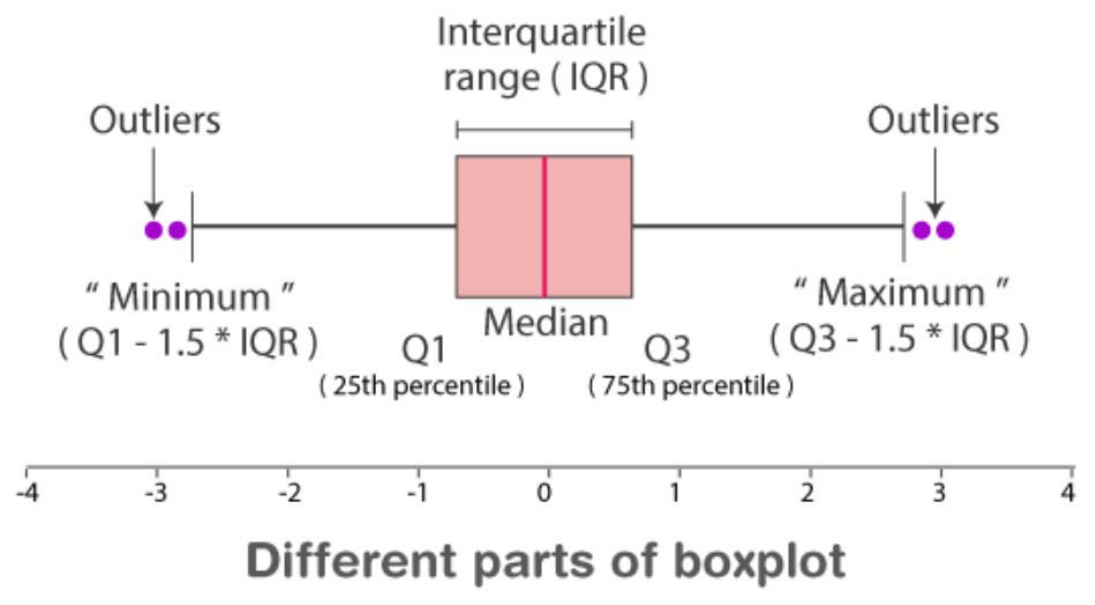
```
[167.62255766 175.32495609 152.84661337 165.50264047 163.17457988
 162.29867872 172.83638413 168.67303667 164.57361342 180.81120541
 170.57782187 167.53075749 176.15356275 176.95378312 158.4125473
 187.8842668  159.03730075 166.69284332 160.73882029 152.22378865
 164.01255164 163.95288674 176.58146832 173.19849526 169.40206527
 166.88861903 149.90348576 148.39039643 177.90349066 166.72462233
 177.44776004 170.93335636 173.26312881 174.76534435 162.28791953
```

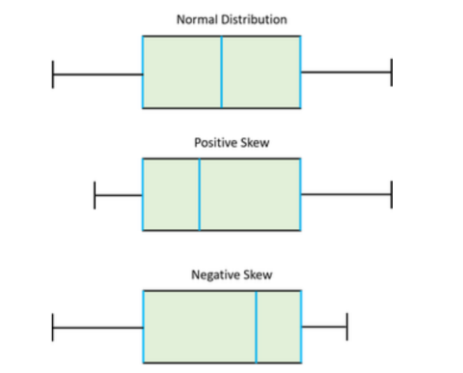
```
import matplotlib.pyplot as plt
import numpy as np

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

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



boxplots

- To display the data distribution in a standardized way using 5 summary
  - *minimum,*
  - *Q1 (First Quartile),*
  - *median,*
  - *Q3(third Quartile),*
  - *and maximum,*
- It is called a **Box plot** also termed as **whisker plot.**





Example:

Find the maximum, minimum, median, first quartile, third quartile for the given data set: 23, 42, 12, 10, 15, 14, 9.

Solution:

- Given: 23, 42, 12, 10, 15, 14, 9.
- Arrange the given dataset in ascending order.
- 9, 10, 12, 14, 15, 23, 42

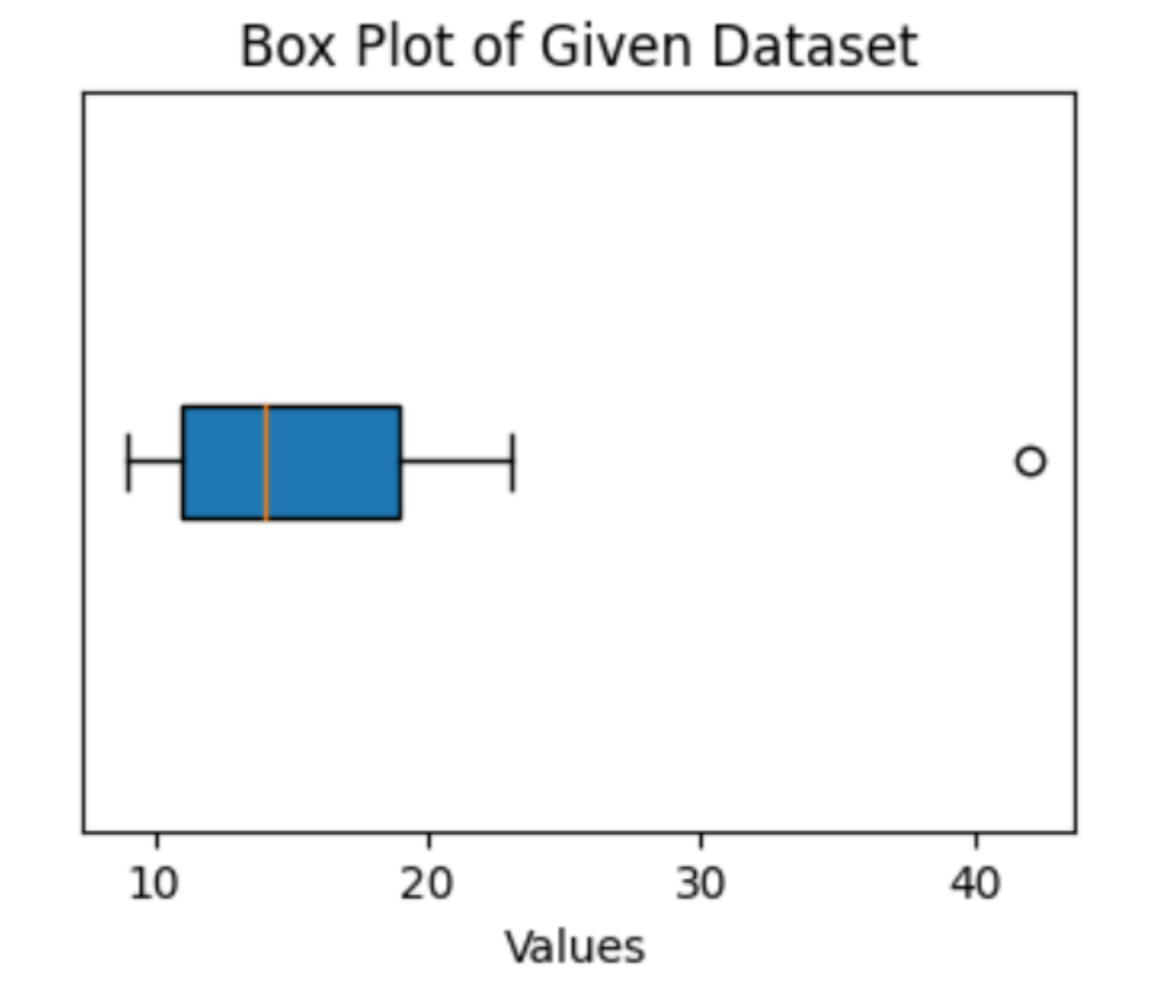
Hence,

- Minimum = 9
- Maximum = 42
- Median = 14
- First Quartile = 10 (Middle value of 9, 10, 12 is 10)
- Third Quartile = 23 (Middle value of 15, 23, 42 is 23).

boxplots

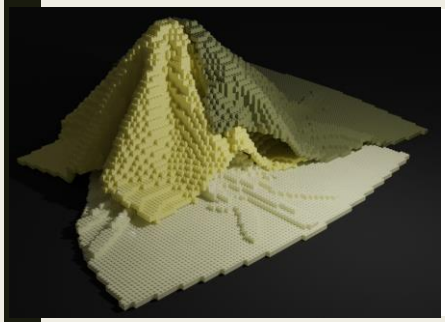
```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Dataset
data = [23, 42, 12, 10, 15, 14, 9]

# Create a box plot
plt.figure(figsize=(4, 3))
plt.boxplot(data, vert=False, patch_artist=True)
plt.title('Box Plot of Given Dataset')
plt.xlabel('Values')
plt.yticks([])  # Hide y-axis labels since it's a single box

# Save the plot
plt.savefig('box_plot.png')
```

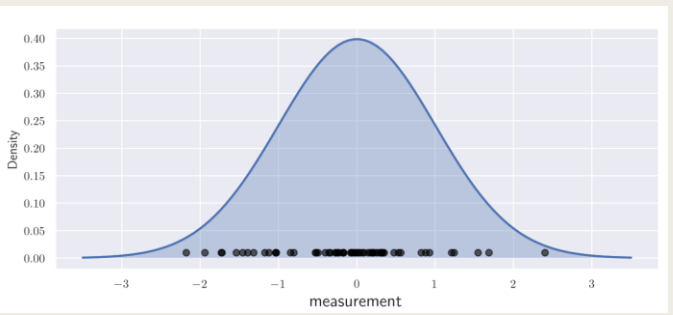
boxplots

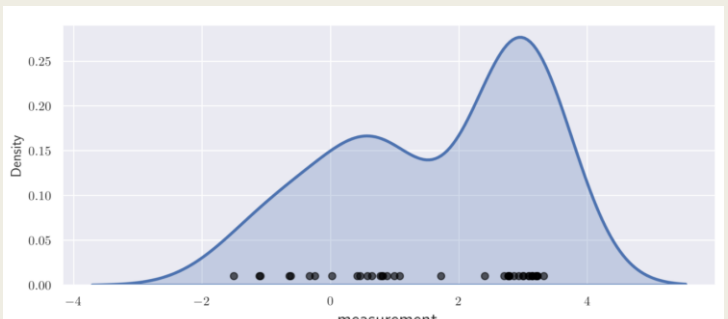


Kernel Density Plot (KDE plot)

- Visualization technique used to estimate the probability density function (PDF) of a continuous random variable
- To understanding KDE is to think of it as a function made up of building blocks
- Similar to how different objects are made up of Lego bricks.
- The distinctive feature of KDE is that it employs only one type of brick, known as the kernel (‘one brick to rule them all‘).
- The key property of this brick is the ability to shift and stretch/shrink. Each datapoint is given a brick, and KDE is the sum of all bricks.



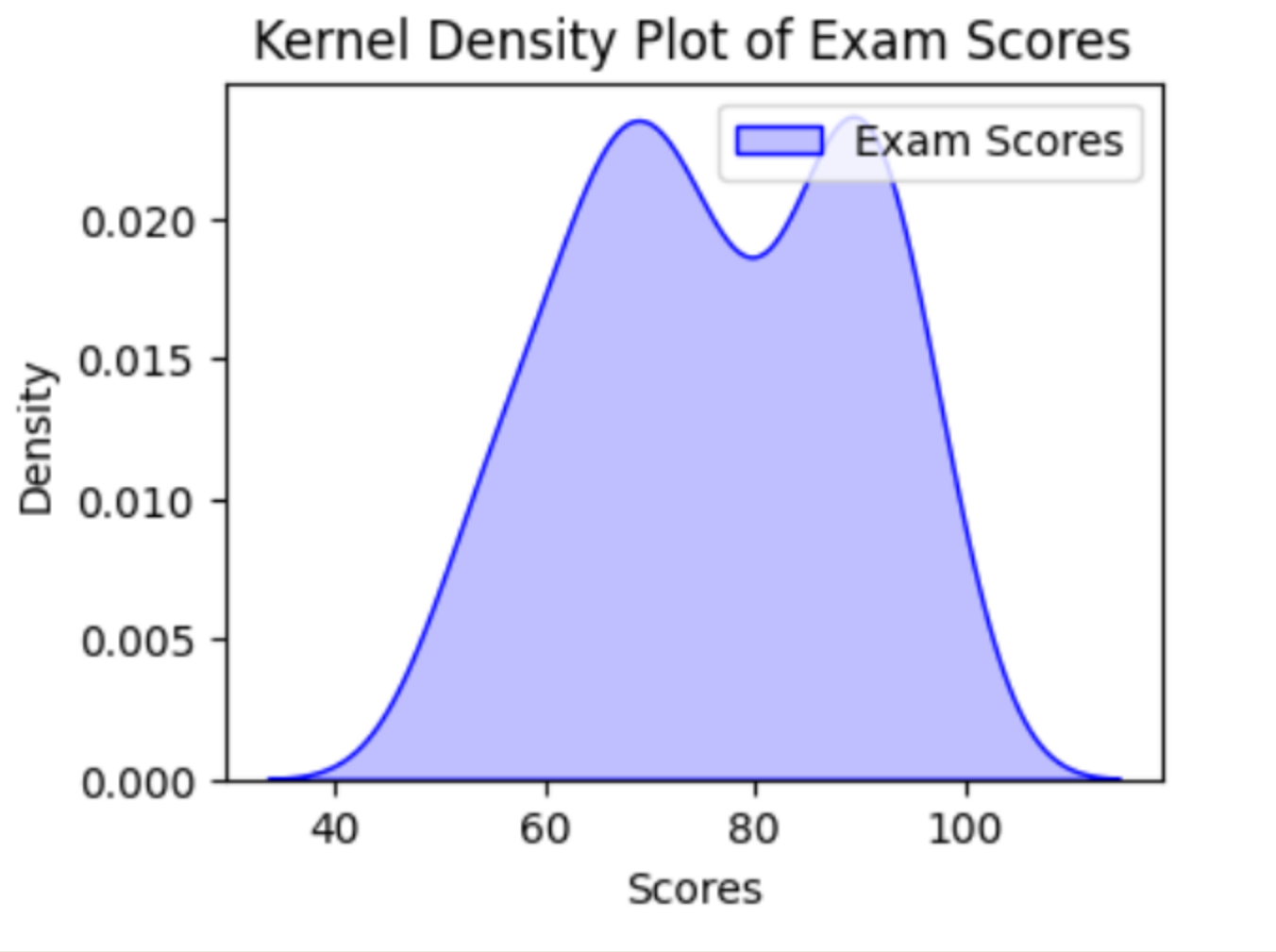




Kernel Density Plot (KDE plot)

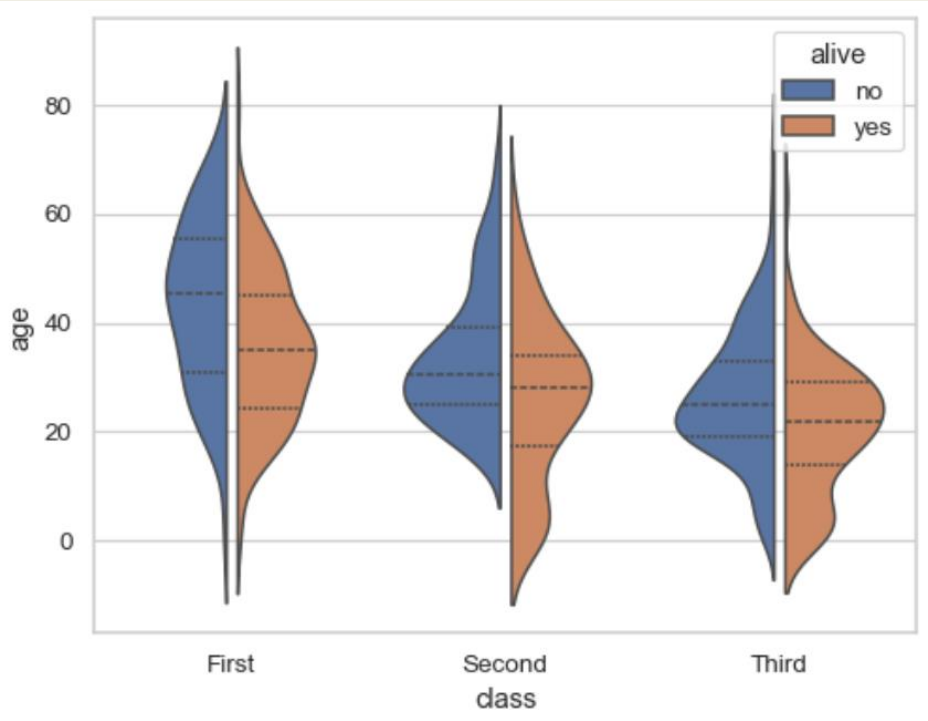
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Generate sample data: exam scores
np.random.seed(42)
scores = np.concatenate([np.random.normal(70, 10, 50), np.random.normal(90, 5, 30)])

# Create a KDE plot
plt.figure(figsize=(4, 3))
sns.kdeplot(scores, color='blue', fill=True, label='Exam Scores')
plt.title('Kernel Density Plot of Exam Scores')
plt.xlabel('Scores')
plt.ylabel('Density')
plt.legend()
# Save the plot
plt.savefig('kde_plot.png')
```

Kernel Density Plot (KDE plot)



violin plots

- A visualization combining a box plot and kernel density plot.
- Displays the distribution of data across different categories or along a continuous axis.
- Shaped like a violin, with wider sections indicating higher data density.



violin plots

- **Exam Scores for Two Classes**
- Class A: [65, 70, 72, 75, 78, 80, 82, 85]
- Class B: [60, 62, 65, 68, 70, 72, 75, 90]
- Compare the distribution of exam scores between Class A and Class B using a violin plot. What can you infer about the spread and density of scores in each class?

violin plots

- **Exam Scores for Two Classes**
- Class A: [65, 70, 72, 75, 78, 80, 82, 85]
- Class B: [60, 62, 65, 68, 70, 72, 75, 90]
- Compare the distribution of exam scores between Class A and Class B using a violin plot. What can you infer about the spread and density of scores in each class?

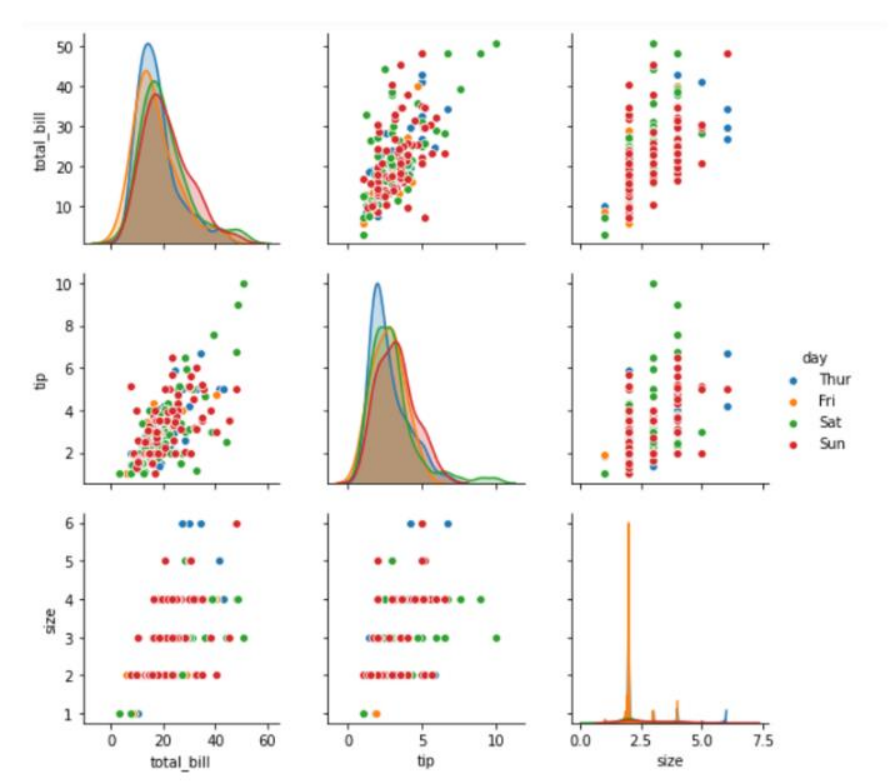
violin plots

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

# Sample dataset
class_a_scores = [65, 70, 72, 75, 78, 80, 82, 85]
class_b_scores = [60, 62, 65, 68, 70, 72, 75, 90]

# Combine data into a format for violin plot
data = {'Scores': class_a_scores + class_b_scores, 'Class': ['A'] * len(class_a_scores) +
        ['B'] * len(class_b_scores)}
data_df = pd.DataFrame(data)  # Create DataFrame directly

# Create a violin plot
plt.figure(figsize=(4, 3))  # Small figure size for slide
sns.violinplot(x='Class', y='Scores', data=data_df)
plt.title('Violin Plot of Exam Scores by Class')
plt.xlabel('Class')
plt.ylabel('Scores')

# Save the plot
plt.savefig('violin_plot.png')
```

violin plots

pair plots

- This method is used for visualizing relationships between multiple variables in a dataset.
- By creating a grid of scatter plots it helps to identify how different features interact with each other to identify patterns, correlations and trends in data.



heatmaps

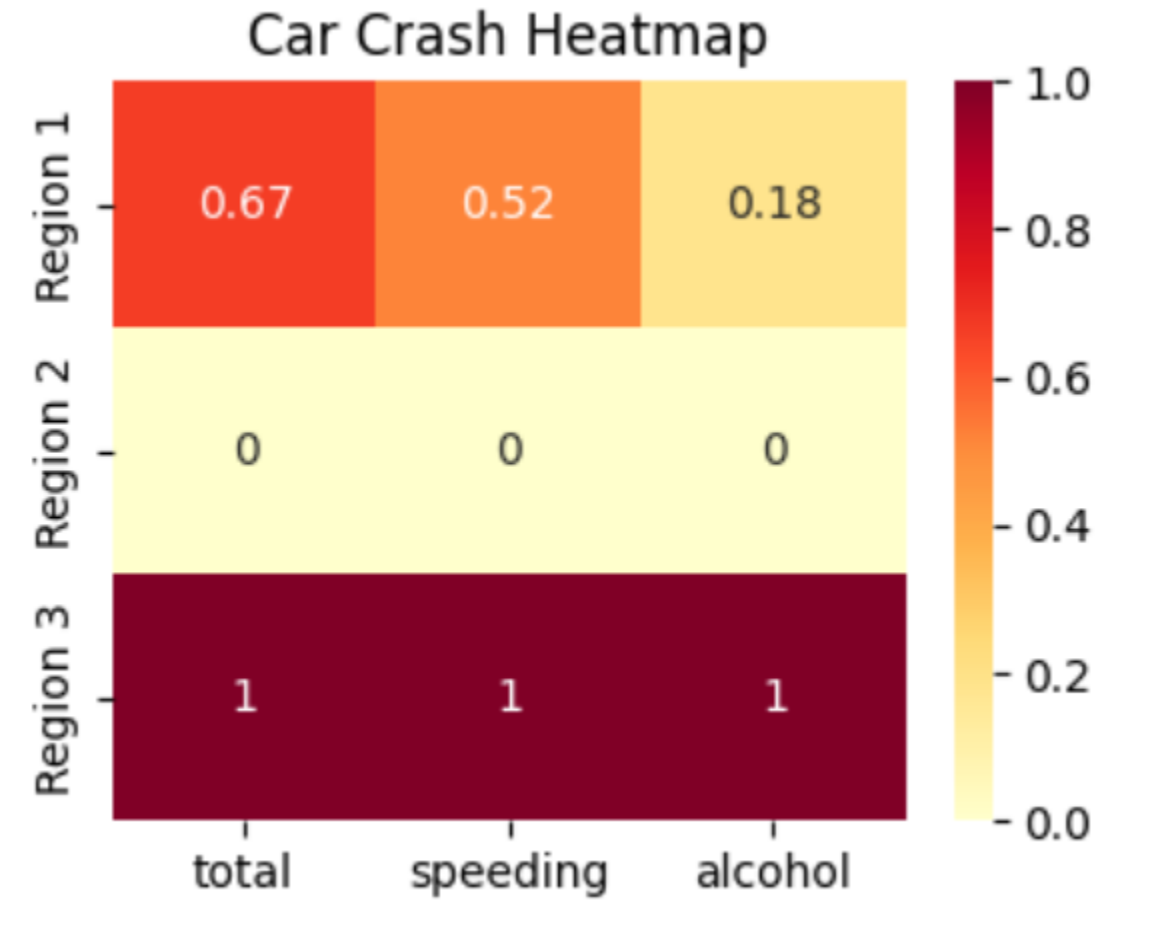
- A table-style data visualization type.
- Each numeric data point is depicted using a selected color scale.
- Colors represent the magnitude of data points within the dataset.
- Main purpose: Illustrate potential hot and cold spots requiring special attention.
  - ***Example:*** *The car crashes dataset in Seaborn contains statistics*

	total	speeding	alcohol
Region 1	18.5	6.2	5.1
Region 2	15.3	4.8	4.9
Region 3	20.1	7.5	6.0

heatmaps

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

# Small dataset
data = {'total': [18.5, 15.3, 20.1], 'speeding': [6.2, 4.8, 7.5], 'alcohol': [5.1, 4.9, 6.0]}
df = pd.DataFrame(data, index=['Region 1', 'Region 2', 'Region 3'])

# Normalize data
df_normalized = pd.DataFrame(MinMaxScaler().fit_transform(df), columns=df.columns, index=df.index)

# Create heatmap
plt.figure(figsize=(4, 3))
sns.heatmap(df_normalized, annot=True, cmap='YlOrRd')
plt.title('Car Crash Heatmap')
plt.savefig('small_car_crash_heatmap.png')
```

heatmaps


Car Crash Heatmap
Region 1
Region 2
Region 3
0.67
0.52
0.18
0
0
0
1
1
1
total
speeding
alcohol
1.0
0.8
0.6
0.4
0.2
0.0