

Imports

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import stats
import math
from fractions import Fraction
import statistics
```

Lecture 1

How to check the type in code

```
print(type(a)) # Output: <class 'int'>
print(type(b)) # Output: <class
'float'>
```

How to convert (Casting)

```
c = int(10.9) # Converts to 10 (drops decimal)
d = float(5) # Converts to 5.0
```

STEP 1: Always write this line
first from fractions import Fraction

```
# STEP 2: Define your fractions
f1 = Fraction(3, 4) # Represents
3/4 f2 = Fraction(1, 2) #
Represents 1/2
```

```
# STEP 3: Perform Math
print(f1 + f2) # Adds them (Result: 5/4)
print(f1 * f2) # Multiplies them (Result:
3/8)
```

```
# 1. Inputting a String (Text)
name = input("Please enter your name:
") print("Hello", name)
```

Lecture 2&3

```
# 1. Create a simple Vector (1D  
Array)  
vector = np.array([1, 2, 3, 4, 5])  
print(vector)  
  
# 2. Create a Matrix (2D Array - Rows and  
Columns)  
matrix = np.array( [[1, 2, 3],  
                   [4, 5, 6]] )  
print(matrix)  
  
# 3. Create an array full of Zeros (e.g., 3 rows, 4  
columns) zeros = np.zeros((3, 4))  
print(zeros)  
  
# 4. Create an array full of  
Ones ones = np.ones((2, 3))  
print(ones)  
  
# 5. Create a range of numbers(Start,  
Stop,Step)  
range_arr = np.arange(0, 10, 2)  
print(range_arr)  
  
# 6. Create evenly spaced numbers (Start, Stop, How many  
numbers)  
linspace_arr = np.linspace(0, 10, 5)  
print(linspace_arr)  
  
# 7. Check Dimensions (Rows, Columns)  
print(arr.shape) # Output: (2, 3) -> 2 Rows, 3  
Cols  
  
# 8. Check Total number of elements  
print(arr.size) # Output: 6  
  
# 9. Check Number of Dimensions (Axes)  
print(arr.ndim) # Output: 2 (because it is  
2D)  
  
# 10. Check Data Type (Integer, Float,  
etc.) print(arr.dtype)  
Example 1:  
  
data = np.array([10, 20, 30, 40, 50])
```

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

```
# --- For 1D Vectors ---  
print(data[0])    # Get first item  
(10) print(data[-1])  # Get last  
item (50)  
print(data[1:4])  # Get items from index 1 to 3 (20, 30, 40)
```

```
# --- For 2D Matrices ---  
# Syntax is: [Row_Index, Column_Index]  
print(matrix[0, 1]) # Row 0, Column 1 (Value: 2)  
print(matrix[1, :]) # Get ALL of Row 1 (4, 5, 6)  
print(matrix[:, 0]) # Get ALL of Column 0 (1, 4)
```

Example 2:

```
arr = np.array([1, 2, 3, 4, 5, 6]) # This has 6 elements
```

```
# 1. Reshape to 2 Rows, 3 Columns  
# Note: 2 * 3 must equal 6 (total  
elements) new_shape = arr.reshape(2,  
3) print(new_shape)
```

```
# 2. Reshape to 3 Rows, 2 Columns  
print(arr.reshape(3, 2))
```

```
# 3. Flatten (Convert Matrix back to 1D  
Vector) flat = new_shape.flatten()  
print(flat)
```

Lecture 4

```
# Create dataset
data = np.array([10, 20, 20, 30, 40, 50, 100])

# (Average)

nums = np.array([1,2,3])
print(np.average(nums)) # result: 2.0
print(np.average(nums, weights=[0.2,0.2,0.6])) # result: 2.4

# 1. Mean
print("Mean:",np.mean(data))

# 2. Median (Middle Value)
print("Median:", np.median(data))

# 3. Mode (Most Frequent
Value) # We use Scipy for this
mode_result =
stats.mode(data)
print("Mode:", mode_result.mode[0])

# 4. Range (Max - Min)
# 'ptp' stands for Peak-to-Peak
print("Range:", np.ptp(data))

# 5. Standard Deviation
(Volatility/Spread) print("Std Dev:",
np.std(data))

# 6. Variance (Square of Std Dev)
print("Variance:", np.var(data))

# 7. Percentiles (Quartiles)
print("25th Percentile (Q1):", np.percentile(data,
25)) print("75th Percentile (Q3):",
np.percentile(data, 75))
```

```
# Example: Hours studied vs Grades
hours = np.array([1, 2, 3, 4, 5])
grades = np.array([50, 55, 65, 70, 85])
```

```
# Calculate Correlation Coefficient
# Returns a matrix. The correlation value is at [0,1]
correlation_matrix = np.corrcoef(hours, grades)
print(correlation_matrix)

print("Correlation:", correlation_matrix[0,
1]) # If close to 1: Strong Positive
Relationship # If close to -1: Strong
Negative Relationship # If close to 0: No
Relationship
```

```
# Example: Group by 'City' and find the mean of other
columns grouped = df.groupby('City').mean()
print(grouped)
```

```
# Example: Group by 'City' and count the entries
counts = df.groupby('City').count()
print(counts)
```

```
data = [10, 20, 20, 30, 40, 50, 100]
```

```
# 1. Standard Deviation (Average distance from the
mean) std_dev = statistics.stdev(data)
print("Standard Deviation:", std_dev)
```

```
# 2. Variance (Standard Deviation squared)
var_val = statistics.variance(data)
print("Variance:", var_val)
```

```
# 3. Range (Max - Min)
data_range = max(data) - min(data)
print("Range:", data_range)
```

```
data = [10, 20, 30, 40, 50, 60, 70, 80]
```

```
# 1. Calculate Q1 (25%) and Q3 (75%)
# We use numpy for this because 'statistics' module doesn't do it
easily q1 = np.percentile(data, 25)
q3 = np.percentile(data, 75)

print("Q1 (25th percentile):",
q1) print("Q3 (75th
```

```
percentile):", q3)

# 2. Calculate IQR (Interquartile Range) iqr = q3 - q1
print("IQR:", iqr)
```

```
# Example Data: Hours studied vs Exam
Score hours = [1, 2, 3, 4, 5]
scores = [50, 55, 65, 70, 85]

# 1. Calculate Pearson Correlation (r)
# Returns two values: (Correlation Coefficient, P-value) corr, p_value = stats.pearsonr(hours, scores)
print(f"Correlation: {corr}") # Logic check:
# If close to 1: Strong Positive # If close to -1: Strong Negative # If close to 0: No relationship
```

```
# Group A: Scores of students who studied group_a = [80, 85, 90, 75, 88]

# Group B: Scores of students who didn't study group_b = [50, 60, 55, 65, 58]

# Perform Independent T-Test
t_stat, p_val = stats.ttest_ind(group_a, group_b)

print(f"T-statistic: {t_stat}") print(f"P-value: {p_val}")

# Interpretation logic (Write this in your answer):
if p_val < 0.05:
    print("Result: Significant Difference (Reject Null Hypothesis)") else:
    print("Result: No Significant Difference (Fail to Reject Null Hypothesis)")
```

```
data = [1, 1, 2, 3, 100] # This has a big outlier (100)

# Calculate Skew
skew_val =
stats.skew(data)
print("Skewness:",
```

```
skew_val)

# Interpretation:
# > 0 : Positive Skew (Tail is on the Right) -> Like Income
data # < 0 : Negative Skew (Tail is on the Left) -> Like Age
at death # = 0 : Normal Distribution (Symmetric)

# 1. Create a List
# Square brackets []
fruits = ["Apple", "Banana", "Cherry"]

# 2. Add Items
fruits.append("Orange") # Adds to the
end fruits.insert(1, "Mango") #
Inserts at index 1

# 3. Remove Items
fruits.remove("Banana") # Removes by
name popped = fruits.pop() # Removes
the last item

# 4. Slicing (Getting parts of the list)
numbers = [10, 20, 30, 40, 50, 60]
print(numbers[0]) # First item
(10) print(numbers[-1]) # Last
item (60)
print(numbers[1:4]) # From index 1 to 3 (20, 30, 40)

# 5. Basic Math on Lists
print(len(numbers))
print(max(numbers))
print(min(numbers))
print(sum(numbers))
```

Lecture 5

```
# 1. Create a Series (1D data like a list, but with  
index) # You can define your own index labels (a,  
b, c...)  
s = pd.Series([10, 20, 30], index=['a', 'b',  
'c']) print(s)  
  
# 2. Create a DataFrame from a Dictionary (Most  
Common) data = {  
    'Name': ['Ali', 'Ahmed',  
    'Sara'], 'Age': [25, 30, 22],  
    'City': ['Baghdad', 'Erbil', 'Basra']  
}  
  
df = pd.DataFrame(data)  
print(df)  
  
# 3. Create a DataFrame from a List of Lists  
data_list = [['Ali', 25], ['Ahmed', 30], ['Sara',  
22]]  
df2 = pd.DataFrame(data_list, columns=['Name', 'Age'])  
print(df2)
```

```
# 1. Read data from a CSV file  
# Ensure the file name is  
correct! df =  
pd.read_csv('filename.csv')  
  
# 2. Save your results to a new CSV file  
# index=False prevents saving the row numbers (0, 1,  
2...) df.to_csv('new_results.csv', index=False)
```

```
# 1. View top and bottom  
rows print(df.head())      #  
First 5 rows print(df.tail())  #  
Last 5 rows  
  
# 2. Check structure (Rows, Columns)  
print(df.shape)  # Output example: (100,  
5)
```

```
# 3. Check Data Types & Missing info  
print(df.info()) # Shows columns, counts, and types (int, float, object)
```

```
# 4. Statistical Summary (Count, Mean, Std, Min, Max)  
print(df.describe()) # Only for numerical columns
```

```
# 5. Get Column Names  
print(df.columns)
```

```
# 1. Select a single column  
print(df['Name'])
```

```
# 2. Select multiple columns (Double Brackets!)  
print(df[['Name', 'City']])
```

```
# 3. Select by Position (iloc) -> Uses  
Integers # Row 0, Column 1  
print(df.iloc[0, 1])
```

```
# 4. Select by Label (loc) -> Uses  
Names # Row with index 0, Column  
named 'City' print(df.loc[0, 'City'])
```

```
# 1. Simple Filter (One Condition)  
# Shows only rows where Age is greater than  
25 high_age = df[df['Age'] > 25]  
print(high_age)
```

```
# 2. Multiple Conditions (Use & for AND, | for OR)  
# IMPORTANT: Use parentheses () around each  
condition filtered = df[(df['Age'] > 20) & (df['City'] ==  
'Baghdad')] print(filtered)
```

```
# 1. Check for missing values (NaN)
print(df.isnull().sum()) # Counts missing values in each column

# 2. Drop rows containing ANY missing
values df_clean = df.dropna()

# 3. Fill missing values
(Imputation) # Fill with a specific
number (e.g., 0) df['Age'].fillna(0,
inplace=True)

# Fill with the Mean (Average) of the column
mean_age = df['Age'].mean()
df['Age'].fillna(mean_age, inplace=True)
```

```
# 1. Check how many duplicates
exist print(df.duplicated().sum())
# 2. Remove duplicates
# keep='first' keeps the first occurrence and deletes the rest
df.drop_duplicates(inplace=True)
```

```
# 1. Sort Data
# ascending=False means Big to Small (Descending)
df_sorted = df.sort_values(by='Age',
ascending=False)
print(df_sorted)
```

```
# 2. Rename Columns
# Change 'Age' to 'Years'
df.rename(columns={'Age': 'Years'},
inplace=True)
```

```
# 1. Group by a column and calculate
Mean # This finds the average Age for
each City
print(df.groupby('City')['Age'].mean())
```

```
# 2. Group by a column and Count
# This counts how many rows (people) are in each City
print(df.groupby('City')['Name'].count())
```

9. numpy.histogram() — Data Distribution

Scenario:

You collected 100 random test scores between 0 and 100.

Task:

Use `numpy.histogram()` and `matplotlib.pyplot.hist()` to plot score distribution. Describe whether the distribution is normal, skewed, or uniform.

```
np.random.seed(42)

scores = np.random.randint(0, 101, 100)

frequencies, bin_edges = np.histogram(scores, bins=10, range=(0, 100))

plt.hist(scores, bins=bin_edges)

plt.title("Distribution of 100 Random Test Scores")
plt.xlabel("Test Score")
plt.ylabel("Number of Students (Frequency)")
```

11. numpy.random.randint(), numpy.random.choice(), numpy.random.uniform()

Scenario:

You are simulating 50 random product prices between \$10–\$100. Then, randomly pick 5 products to apply a discount.

Task:

1. Generate prices using `numpy.random.randint(10, 100, 50)`.
2. Use `numpy.random.choice()` to select 5 random items.
3. Apply a 10% discount to them and print the new list.

```
# توليد أسعار 50 منتجًا بشكل عشوائي بين 10 و 100
# نستخدم 101 لأن الرقم الأخير في الدالة غير مشمول (exclusive)
prices = np.random.randint(10, 101, 50)

print("--- (القائمة الأصلية (أول 10 عناصر ---")
print(prices[0:10]) # طباعة عينة فقط للتوضيح

# اختيار 5 منتجات عشوائية لتطبيق الخصم
# من 0 إلى 49 (Indices) "هنا نختار "أرقام المواقع
# تعني عدم تكرار اختيار نفس المنتج مرتين
selected_indices = np.random.choice(50, 5, replace=False)
```

```
print("\n--- مواقع المنتجات المختارة للخصم ---")
print(selected_indices)

# لكي تقبل الفواصل بعد الخصم (float) نقوم بتحويل المصفوفة إلى أرقام عشرية
prices = prices.astype(float)

# تطبيق خصم 10 % على المنتجات المختارة
السعر الجديد = السعر القديم * 0.90
#
prices[selected_indices] = prices[selected_indices] * 0.90

print("\n--- (القائمة الجديدة بعد الخصم (أول 10 عناصر ---")
print(prices[:10])

# طباعة المنتجات التي تم تعديليها بالتحديد للتأكد
print("\n--- أسعار المنتجات الخمسة بعد الخصم ---")
print(prices[selected_indices])
```

Lecture 6

```
# 1. Define Data
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
# 2. Plot
plt.plot(xpoints,
          ypoints) plt.show()
# --- Plotting without X-axis ---
# If you only give one array, it assumes X is 0, 1, 2,
3... ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints)
plt.show()
```

```
# 1. Add Markers (Circles 'o', Stars '*', Squares 's')
plt.plot(ypoints, marker='o')
# 2. Format String 'marker|line|color'
# 'o:r' means Circle marker, Dotted line, Red color
plt.plot(ypoints, 'o:r')
# 3. Line Styles (ls)
# 'dotted', 'dashed', 'solid'
plt.plot(ypoints, ls='dashed', color='green', linewidth=20)
plt.show()
```

```
x = np.array([80, 85, 90, 95, 100])
y = np.array([240, 250, 260, 270, 280])
```

```
# 1. Create Font Dictionaries (to change text
style)
```

```
# 2. Plot Labels with Fonts
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
```

```
# 3. Add Grid Lines
plt.grid(color='green', linestyle='--', linewidth=0.5)
plt.plot(x,
          plt.show())
```

```

x = [0, 1, 2, 3, 4, 5]
y1 = [0, 2, 4, 6, 8, 10]
y2 = [0, 3, 6, 9, 12, 15]

# 1. Plot both lines with labels
plt.plot(y1, label="y = 2x")
plt.plot(y2, label="y = 3x")

# 2. Add Legend Inside
# (Basic) # plt.legend()

# 3. Add Legend at Specific Location
# (bbox_to_anchor) # (1.05, 1) puts it Outside Top-
# Right plt.legend(bbox_to_anchor=(1.05, 1),
# loc='upper left')

plt.show()

```

```

# Group 1
x = np.array([5,7,8,7,2,17,2,9])
y = np.array([99,86,87,88,111,86,103,87])
plt.scatter(x, y, color='hotpink')

# Group 2
x = np.array([2,2,8,1,15,8,12,9])
y = np.array([100,105,84,105,90,99,90,95])
plt.scatter(x, y,
            color='#88c999') plt.show()

# --- Advanced Scatter with ColorMap ---
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])
plt.scatter(x, y, c=colors, cmap='viridis') # 'c' maps values to
colors plt.colorbar() # Shows the color scale bar on the side

```

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

# 1. Vertical Bars
plt.bar(x, y, color='red', width=0.1)
plt.show()

# 2. Horizontal Bars (barh)
# Use 'height' instead of 'width'
plt.barh(x, y, color='green',
height=0.5) plt.show()
```

```
# Generate 250 numbers centered at 170 with std
dev 10
x = np.random.normal(170, 10, 250)
```

```
# Plot
Histogram
plt.hist(x)
plt.show()
```

```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
```

```
# 'explode' separates a slice (0.2 moves the first slice
out)
myexplode = [0.2, 0, 0, 0]
```

```
plt.pie(y, labels=mylabels, explode=myexplode,
shadow=True)
plt.legend(title="Four Fruits")
plt.show()
```

```

y1 = []
y2 = []
x = range(-100, 100, 10)
for i in x: y1.append(i**2)
for i in x: y2.append(-i**2)
plt.plot(x, y1)
plt.plot(x, y2)
plt.xlabel("X")
plt.ylabel("Y")
# plt.ylim(-2000, 2000)
# plt.axhline(0)
# plt.axvline(0)
# plt.savefig("test.png")
plt.show()

```

```

import matplotlib.pyplot as plt

# x axis values
x = [1,2,3,4,5,6]
# corresponding y axis values
y = [2,4,1,5,2,6]

# plotting the points
plt.plot(x, y, color='green', linestyle='dashed', linewidth = 3,
         marker='o', markerfacecolor='blue', markersize=12)

# setting x and y axis range
plt.ylim(1,8)
plt.xlim(1,8)

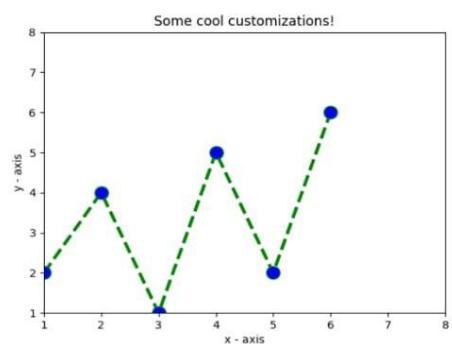
# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('Some cool customizations!')

# function to show the plot
plt.show()

```

Customization of Plots



```

import matplotlib.pyplot as plt

# heights of bars
height = [10, 24, 36, 40, 5]

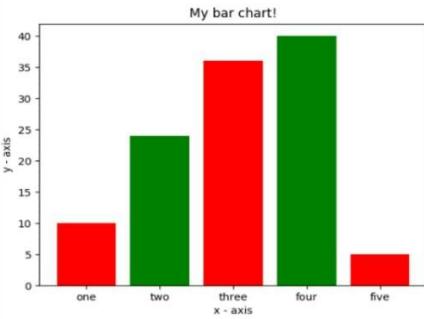
# labels for bars
names = ['one','two','three','four','five']

# plotting a bar chart
c1=['red', 'green']
c2=['b', 'g'] # we can use this for color
plt.bar(left, height, width=0.8, color=c1)

# naming the x-axis
plt.xlabel('x - axis')
# naming the y-axis
plt.ylabel('y - axis')
# plot title
plt.title('My bar chart!')

# function to show the plot
plt.show()

```



- Here, we use `plt.bar()` function to plot a bar chart.
- you can also give some name to x-axis coordinates by defining `tick_labels`

```

import matplotlib.pyplot as plt

# frequencies
ages=[2,5,70,40,30,45,50,45,43,40,44,60,7,13,57,18,90,77,32,21,20,40]

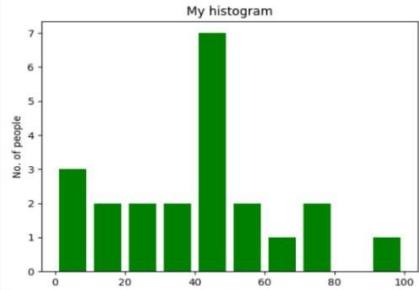
# setting the ranges and no. of intervals
range = (0, 100)
bins = 10

# plotting a histogram
plt.hist(ages, bins, range, color='green', histtype='bar', rwidth=0.8)

# x-axis label
plt.xlabel('age')
# frequency label
plt.ylabel('No. of people')
# plot title
plt.title('My histogram')

# function to show the plot
plt.show()

```



```

import matplotlib.pyplot as plt

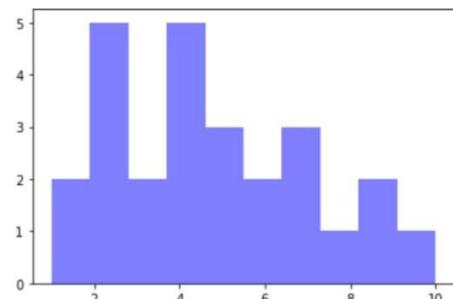
#generate fake data
x = [2,1,6,4,2,4,8,9,4,2,4,10,6,4,5,7,7,3,2,7,5,3,5,9,2,1]

#plot for a histogram
plt.hist(x, bins = 10, color='blue', alpha=0.5)
plt.show()

```

- Looking at the code snippet, I added two new arguments:

- **Bins** — is an argument specific to a histogram and allows the user to customize how many bins they want.
- **Alpha** — is an argument that displays the level of transparency of the data points.



```

import matplotlib.pyplot as plt

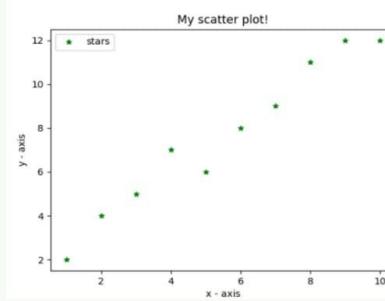
# x-axis values
x = [1,2,3,4,5,6,7,8,9,10]
# y-axis values
y = [2,4,5,7,6,8,9,11,12,12]

# plotting points as a scatter plot
plt.scatter(x, y, label= "stars", color="green", marker="*", s=30)

# x-axis label
plt.xlabel('x - axis')
# frequency label
plt.ylabel('y - axis')
# plot title
plt.title('My scatter plot!')
# showing legend
plt.legend()

# function to show the plot
plt.show()

```



Pandas - Cleaning Data of Wrong Format

Convert all cells in the 'Date' column into standard date.

```

import pandas as pd
df = pd.read_csv('data.csv')
df['Date'] = pd.to_datetime(df['Date'])
print(df.to_string())

```

`pd.to_datetime()` converts the "Date" column into a proper datetime format.

Removing Rows (NaT)

```
df.dropna(subset=['Date'], inplace = True)
```

21	60	'2020/12/21'	108
22	45	NaN	100
23	60	'2020/12/23'	130
24	45	'2020/12/24'	105
25	60	'2020/12/25'	102
26	60	20201226	100
27	60	'2020/12/27'	92

21	60	'2020-12-21'	108
22	45	NaT	100
23	60	'2020-12-25'	130
24	45	'2020-12-24'	105
25	60	'2020-12-25'	102
26	60	'2020-12-26'	100

```

x = [0, 1, 2, 3, 4, 5, 36]
y = [0, 2, 4, 6, 8, 10, 36]

# Create a list of colors based on the condition
# Green if x > 35 and y > 35, else Red
colors = ['green' if (xi > 35 and yi > 35) else 'red' for xi, yi in zip(x, y)]

# Plotting
plt.scatter(x, y, c=colors)
plt.title('Simple Scatter plot')
plt.xlabel('X - value')
plt.ylabel('Y - value')
plt.show()

```

