Welcome to the first weekly activity for the Machine Learning Course Fall 2023.

Task:Run the following code snippets

We are going to explore Numpy,a fundamental package in Python. NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

1) A fundamental library within scientific computing.

2) Central package of libraries, including scikit-learn, matplotlib, and pandas.

3) Designed for multi-dimensional arrays, optimizing performance for fast array operations.

Note: Many functions are implemented in C, with Python acting as a wrapper for the underlying C modules.

Brief overview of the use cases

1) array/ matrix

2) Mathematic operation

3) Random numbers

```
# to install the library
# !pip install numpy

import numpy as np

# Check the verion
np.__version__
```

```
'1.23.5'
```

```
# Create an array
arr_1d = np.array([1,2,3])
arr_1d
```

```
array([1, 2, 3])
```

The np.array() function is a fundamental function in NumPy that creates an array object from a sequence-like object or converts an existing array-like object into a NumPy array. It is a versatile function that allows you to create arrays of various dimensions and data types.

```
# Shape
print(f"Shape: {arr_1d.shape}")
# Dimention
print(f"Dimention: {arr_1d.ndim}")
# Data type
print(f"Data type: {arr_1d.dtype}")
# Total number of elements
print(f"Total number of elements: {arr_1d.size}")
# Size (in byte)
print(f"Size(in byte): {arr_1d.itemsize}")
```

```
Shape: (3,)
Dimention: 1
Data type: int64
```

```
Total number of elements: 3
Size(in byte): 8
```

```
# Access different elements
arr_1d[0]
```

```
1
```

```
# Modify the value
arr_1d[0] = 10
arr_1d
```

```
array([10,  2,  3])
```

```
# Create an array of 3 elements, all set to zero
np.zeros(3)
```

```
array([0., 0., 0.])
```

```
# Create an array of 3 elements, all set to one,
np.ones(3)
```

```
array([1., 1., 1.])
```

```
# Create 10 random numbers
np.random.rand(10)
```

```
array([0.86734224, 0.7952731 , 0.64598292, 0.70427166, 0.07118026,
       0.21336355, 0.16514643, 0.61070606, 0.16355436, 0.67712217])
```

```
# create a 2d array
arr_2d = np.array([[1,2,3],[4,5,6]])
arr_2d
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
# Shape
print(f"Shape: {arr_2d.shape}")
# Dimention
print(f"Dimention: {arr_2d.ndim}")
# Data type
print(f"Data type: {arr_2d.dtype}")
# Total number of elements
print(f"Total number of elements: {arr_2d.size}")
# Size (in byte)
print(f"Size(in byte): {arr_2d.itemsize}")
```

```
Shape: (2, 3)
Dimention: 2
Data type: int64
Total number of elements: 6
Size(in byte): 8
```

```
# Find specifi element
print(arr_2d[0])
print(arr_2d[0,2])
```

```
[1 2 3]
3
```

```
# Create a 3x2 array where all elements are set to one
arr_ones = np.ones([3,2])
arr_ones
```

```
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
```

```
# Matrix product
np.matmul(arr_ones, arr_2d)
```

```
array([[5., 7., 9.],
       [5., 7., 9.],
```

```
       [5., 7., 9.]])
```

Element-wise

1) add

2) multiply

3) divide

4) subtract

5) mod

```
np.add(arr_ones, arr_2d)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-37-f4a890a57efe> in <cell line: 1>()
----> 1 np.add(arr_ones, arr_2d)

ValueError: operands could not be broadcast together with shapes (3,2) (2,3)
```

SEARCH STACK OVERFLOW

Number of rows and colums should be same in order to add matrices.

```
# Create a 3x2 random array
np.random.rand(3,2)
```

```
array([[0.68734052, 0.23256218],
       [0.05365848, 0.72085687],
       [0.41444693, 0.93729647]])
```

Exercise 1: Write a Python program that creates a 3x3 NumPy array and performs the following operations:

1a.Find the sum of all the elements in the array.

1b.Find the minimum and maximum values in the array.

1c.Calculate the mean and standard deviation of the array.

```
# Create a 3x3 NumPy array
array = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])
array
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
# Find the sum of all the elements in the array
sum_of_elements = np.sum(array)
print("Sum of elements:", sum_of_elements)
```

```
Sum of elements: 45
```

The np.sum() function in NumPy is used to calculate the sum of array elements along a specified axis or axes. It is a versatile function that allows you to perform element-wise summation across various dimensions of a NumPy array.

```
# Find the minimum and maximum values in the array
min_value = np.min(array)
max_value = np.max(array)
print("Minimum value:", min_value)
print("Maximum value:", max_value)
```

```
Minimum value: 1
Maximum value: 9
```

In NumPy, the functions np.min() and np.max() are used to find the minimum and maximum values in a given array, respectively.

```
# Calculate the mean and standard deviation of the array
mean_value = np.mean(array)
std_deviation = np.std(array)
print("Mean:", mean_value)
print("Standard Deviation:", std_deviation)
```

```
Mean: 5.0
Standard Deviation: 2.581988897471611
```

In NumPy, the functions .mean() and .std() are used to calculate the mean and standard deviation of an array, respectively. These functions provide useful statistical measures to analyze the central tendency and dispersion of the data in the array.

mean(): This function computes the arithmetic mean, which is the average of all the elements in the array. The mean is calculated as the sum of the values divided by the total number of elements.

std(): This function calculates the standard deviation, which measures the dispersion or spread of the values in the array. The standard deviation is a measure of how much the individual values deviate from the mean. A higher standard deviation indicates greater variability in the data.

Exercise 2:Using Iphone purchase dataset

The iphone purchase dataset has columns Gender,Age and salary of different individuals and their potential to purchase an Iphone.

Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays.

```
import pandas as pd
```

```
df = pd.read_csv("https://raw.githubusercontent.com/omairaasim/machine_learning/master/project_11_k_nearest_neighbor/ip
```

Creating a dataframe to store the dataset.DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects.

```
df.head()
```

	Gender	Age	Salary	Purchase Iphone
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0

```
df.head(10)
```

	Gender	Age	Salary	Purchase Iphone
0	Male	19	19000	0

The head function in Python displays the first five rows of the dataframe by default. It takes in a single parameter: the number of rows. We can use this parameter to display the number of rows of our choice.

3	Female	27	57000	0

```
df.tail()
```

	Gender	Age	Salary	Purchase Iphone
395	Female	46	41000	1
396	Male	51	23000	1
397	Female	50	20000	1
398	Male	36	33000	0
399	Female	49	36000	1

```
df.tail(7)
```

	Gender	Age	Salary	Purchase Iphone
393	Male	60	42000	1
394	Female	39	59000	0
395	Female	46	41000	1
396	Male	51	23000	1
397	Female	50	20000	1
398	Male	36	33000	0
399	Female	49	36000	1

```
# Select columns
df[["Age"]].head()
```

	Age
0	19
1	35
2	26
3	27
4	19

```
# Select list of columns
df[["Age", "Salary"]].head()
```

	Age	Salary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000

iloc: iloc stands for “integer location” and is a function in the Pandas library. It is used to select rows and columns from a Pandas DataFrame or a Series using integer-based indexing.

loc: selects rows using row labels

```
df.iloc[:,:].head()
```

	Gender	Age	Salary	Purchase Iphone
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0

```
df.iloc[:, 3].head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: Purchase Iphone, dtype: int64
```

```
df.iloc[1:3].head()
```

	Gender	Age	Salary	Purchase Iphone
1	Male	35	20000	0
2	Female	26	43000	0

```
df.iloc[:,[0,2]].head()
```

	Gender	Salary
0	Male	19000
1	Male	20000
2	Female	43000
3	Female	57000
4	Male	76000

```
df.iloc[4:,[2,3]].head(10)
```

	Salary	Purchase Iphone
4	76000	0
5	58000	0
6	84000	0
7	150000	1
8	33000	0
9	65000	0
10	80000	0
11	52000	0
12	86000	0
13	18000	0

```
df.iloc[:,:-1].head()
```

	Gender	Age	Salary
0	Male	19	19000
1	Male	35	20000
2	Female	26	43000
3	Female	27	57000
4	Male	19	76000

```
df.iloc[:5,:-1]
```

	Gender	Age	Salary
0	Male	19	19000
1	Male	35	20000
2	Female	26	43000
3	Female	27	57000
4	Male	19	76000

```
df.iloc[[1,5],:-1]
```

	Gender	Age	Salary
1	Male	35	20000
5	Male	27	58000

Explanation for below: It prints all the data who have "Age" as 30.

```
# Dictionaries are case-sensitive
df.loc[df["Age"] == 30]
```

	Gender	Age	Salary	Purchase Iphone
37	Male	30	49000	0
43	Male	30	15000	0
48	Male	30	135000	1
80	Male	30	80000	0
84	Female	30	62000	0
91	Female	30	116000	0
128	Male	30	17000	0
132	Male	30	87000	0
137	Male	30	107000	1
143	Male	30	89000	0
196	Female	30	79000	0

```
# Check Keys
df.keys()
```

```
Index(['Gender', 'Age', 'Salary', 'Purchase Iphone'], dtype='object')
```

```
# Features
X = df.iloc[:,:-1].values
X
```

```
array([['Male', 19, 19000],
       ['Male', 35, 20000],
       ['Female', 26, 43000],
       ...,
       ['Female', 50, 20000],
       ['Male', 36, 33000],
       ['Female', 49, 36000]], dtype=object)
```

```
x = df.iloc[3:10].values
x
```

```
array([['Female', 27, 57000, 0],
       ['Male', 19, 76000, 0],
       ['Male', 27, 58000, 0],
       ['Female', 27, 84000, 0],
       ['Female', 32, 150000, 1],
       ['Male', 25, 33000, 0],
       ['Female', 35, 65000, 0]], dtype=object)
```

```
# Label
y = df.iloc[:, 3].values
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1])
```

```
# Label
y = df.iloc[1:5, 0].values
y
```

```
array(['Male', 'Female', 'Female', 'Male'], dtype=object)
```

```
#Initializing the column names for the dataset
df.columns=[ "Gender","Age","Salary","PurchaseIphone"]
df.head(2)
```

	Gender	Age	Salary	PurchaseIphone
0	Male	19	19000	0
1	Male	35	20000	0

```
df
```

	Gender	Age	Salary	PurchaseIphone
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0
...	...	...	...	...
395	Female	46	41000	1
396	Male	51	23000	1
397	Female	50	20000	1
398	Male	36	33000	0
399	Female	49	36000	1

400 rows × 4 columns

```
df.columns
```

```
Index(['Gender', 'Age', 'Salary', 'PurchaseIphone'], dtype='object')
```

```
# If you have many columns, then you are in trouble
df.columns=[ "Gender","Age","PurchaseIphone"]

#We need to mention all the columns
#If even one misses then it throws an error
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-76-9632775bcc80> in <cell line: 2>()
      1 # If you have many columns, then you are in trouble
----> 2 df.columns=[ "Gender","Age","PurchaseIphone"]
      3
      4 #We need to mention all the columns
      5 #If even one misses then it throws an error

 4 frames
```

```
# If you have many columns, then you are in trouble
df.columns=[ "Gender","Age","Salary","PurchaseIphone"]
df.columns
```

```
Index(['Gender', 'Age', 'Salary', 'PurchaseIphone'], dtype='object')
```

```
df.rename(columns={'PurchaseIphone':"Purchase_Iphone"})
df.head(2)
```

	Gender	Age	Salary	PurchaseIphone
0	Male	19	19000	0
1	Male	35	20000	0

```
df=df.rename(columns={'PurchaseIphone':"Purchase_Iphone"})
df.head(2)
```

	Gender	Age	Salary	Purchase_Iphone
0	Male	19	19000	0
1	Male	35	20000	0

```
df.rename(columns={'PurchaseIphone':"Purchase_Iphone"}, inplace= True)
df.head(2)
```

	Gender	Age	Salary	Purchase_Iphone
0	Male	19	19000	0
1	Male	35	20000	0

```
unique_genders = np.unique(df["Gender"])
print("Unique Genders:", unique_genders)
```

```
Unique Genders: ['Female' 'Male']
```

The numpy.unique() function is used to find the unique elements of an array.Returns the sorted unique elements of an array.

```
df["Gender"].value_counts()
```

```
Female    204
Male      196
Name: Gender, dtype: int64
```

```
df.groupby("Gender").count()
```

Gender	Age	Salary	Purchase_Iphone
Female	204	204	204
Male	196	196	196

```
df.groupby("Age").sum().tail(6)
```

```
<ipython-input-84-a8803ab79c4e>:1: FutureWarning: The default value of numeric_only i
  df.groupby("Age").sum().tail(6)
```

Age	Salary	Purchase_Iphone
55	294000	3
56	297000	3
57	315000	5

```
mean_salary = np.mean(df["Salary"])
print("Mean Salary:", mean_salary)
```

```
Mean Salary: 69742.5
```

```
mean_purchaseIphone = np.mean(df["Purchase_Iphone"])
print("Mean Purchase Iphone:", mean_purchaseIphone)
```

```
Mean Purchase Iphone: 0.3575
```

```
min_salary = np.min(df["Salary"])
print("Minimum Salary:", min_salary)
```

```
Minimum Salary: 15000
```

```
min_purchaseIphone = np.mean(df["Purchase_Iphone"])
print("Min Purchase Iphone:", min_purchaseIphone)
```

```
Min Purchase Iphone: 0.3575
```

Practice 1 - Numpy

1) Create a 4X3 NumPy array called array

2) Create a 3x4 random array called rand_array

3) Check shape (side by side) for both rand_array, and array

4) Apply matrix product

```
array = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9],
                  [10,11,12]])
rand_array = np.random.rand(3,4) # Hint: If the original array has dimensions 4x3, make sure the rand_array has dimensic
print(rand_array.shape, array.shape)
print(np.matmul(array, rand_array))
```

```
(3, 4) (4, 3)
[[ 2.38007789  2.53654208  3.04662947  4.39000053]
 [ 7.03333614  5.629043    8.2604093  10.69557079]
 [11.6865944   8.72154393 13.47418912 17.00114104]
 [16.33985266 11.81404485 18.68796894 23.30671129]]
```

```
df
```

	Gender	Age	Income	PurchaseIphone
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0

Practice - Pandas

1) Rename "salary" to "income".

2) Calculate the counts of females and males.

3) Create separate datasets for females and males.

4) Determine the minimum and maximum ages.

5) Examine salary and iPhone purchase status.

```
df.rename(columns={"Salary":"Income"}, inplace=True)
print(df["Gender"].value_counts())
print ("-"*5)
female = df.loc[df["Gender"] == "Female"]
male = df.loc[df["Gender"] == "Male"]

min_age_female = np.min(female["Age"])
max_age_female = np.max(female["Age"])
min_age_male = np.min(male["Age"])
max_age_male = np.max(male["Age"])

print ("Young Female")
print (female.loc[female["Age"]==min_age_female])
print ("-"*5)
print ("Old Female")
print (female.loc[female["Age"]==max_age_female])
print ("-"*5)
print ("Young Male")
print (male.loc[male["Age"]==min_age_male])
print ("-"*5)
print ("Old Male")
print (male.loc[male["Age"]==max_age_male])
```

```
Female    204
Male      196
Name: Gender, dtype: int64
-----
Young Female
     Gender  Age  Income  PurchaseIphone
51   Female   18   44000               0
141  Female   18   68000               0
165  Female   18   86000               0
-----
Old Female
     Gender  Age  Income  PurchaseIphone
215  Female   60  108000               1
370  Female   60   46000               1
-----
Young Male
   Gender  Age  Income  PurchaseIphone
14   Male   18   82000               0
76   Male   18   52000               0
-----
Old Male
    Gender  Age  Income  PurchaseIphone
223   Male   60  102000               1
272   Male   60   42000               1
355   Male   60   34000               1
371   Male   60   83000               1
393   Male   60   42000               1
```

Answers with clarity

```
# 1) Rename "salary" to "income".
df.rename(columns={"Salary":"Income"}, inplace=True)

# 2) Calculate the counts of females and males.
print(df["Gender"].value_counts())

print ("-"*5)

# 3) Create separate datasets for females and males.
female = df.loc[df["Gender"] == "Female"]
male = df.loc[df["Gender"] == "Male"]

# 4) Determine the minimum and maximum ages.
min_age_female = np.min(female["Age"])
max_age_female = np.max(female["Age"])
min_age_male = np.min(male["Age"])
max_age_male = np.max(male["Age"])

# 5) Examine salary and iPhone purchase status.
print ("Young Female")
print (female.loc[female["Age"]==min_age_female])
print ("-"*5)
print ("Old Female")
print (female.loc[female["Age"]==max_age_female])
print ("-"*5)
print ("Young Male")
print (male.loc[male["Age"]==min_age_male])
print ("-"*5)
print ("Old Male")
print (male.loc[male["Age"]==max_age_male])
```

```
Female    204
Male      196
Name: Gender, dtype: int64
-----
Young Female
     Gender  Age  Income  PurchaseIphone
51   Female   18   44000               0
141  Female   18   68000               0
165  Female   18   86000               0
-----
Old Female
     Gender  Age  Income  PurchaseIphone
215  Female   60  108000               1
370  Female   60   46000               1
-----
Young Male
   Gender  Age  Income  PurchaseIphone
14   Male   18   82000               0
76   Male   18   52000               0
-----
Old Male
    Gender  Age  Income  PurchaseIphone
223   Male   60  102000               1
272   Male   60   42000               1
355   Male   60   34000               1
371   Male   60   83000               1
393   Male   60   42000               1
```

DRY (DO NOT REPEAT YOURSELF)

Double-click (or enter) to edit

Double-click (or enter) to edit

```
def age_gender_analysis(inp:str) -> None:
    """
    Analyze age and gender to identify potential buyers.

    Parameter
    ---------
```

```
    inp:
      The gender for which age analysis will be performed.

    """
    data = df.loc[df["Gender"] == inp]
    min_age = np.min(data["Age"])
    max_age = np.max(data["Age"])
    print (f"Young {inp}")
    print (data.loc[data["Age"]== min_age])
    print ("-"*5)
    print (f"Old {inp}")
    print (data.loc[data["Age"]== max_age])

df.rename(columns={"Salary":"Income"}, inplace=True)
stat = df["Gender"].value_counts()
print (stat)
print ("-"*5)
for inp in ["Female", "Male"]:
    age_gender_analysis(inp)
```

```
Female    204
Male      196
Name: Gender, dtype: int64
-----
Young Female
     Gender  Age  Income  PurchaseIphone
51   Female   18   44000               0
141  Female   18   68000               0
165  Female   18   86000               0
-----
Old Female
     Gender  Age  Income  PurchaseIphone
215  Female   60  108000               1
370  Female   60   46000               1
Young Male
   Gender  Age  Income  PurchaseIphone
14   Male   18   82000               0
76   Male   18   52000               0
-----
Old Male
    Gender  Age  Income  PurchaseIphone
223   Male   60  102000               1
272   Male   60   42000               1
355   Male   60   34000               1
371   Male   60   83000               1
393   Male   60   42000               1
```

```
Please share your takeaway from this session in 1 or 2 sentences.
```

```
  File "<ipython-input-73-dfdd0173c873>", line 1
    Please share your takeaway from this session in 1 or 2 sentences.
               ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

My takeaway from this session

1) Machines will surely not replace humans, but they will reduce human work by using the ability to learn without being explicitly programmed. Machine learning is not only used in AI applications; it can also have other uses, like in self-driving cars, etc.

2) Numpy is used to perform operations on arrays, whereas the Panda library is used to manipulate and work with data sets.

✓ 0s completed at 11:42 AM