ALA Lab Sessional 1 Jupyter Notebook

1. VS Code is a MUST

1. You should code in VS Code. Google Colab not allowed

2. Pre-requisites for lab in VS Code

1. Install PDF export support https://saturncloud.io/blog/how-to-export-jupyter-notebook-by-vscode-in-pdf-format/

3. At the end, export your notebook as html

1. Open VS Code command palette Shift + Ctrl + P
2. Type "Export Jupyter Notebook" in the search bar and select "Export Jupyter Notebook to HTML"
3. Upload your html AND ipynb here: https://tinyurl.com/yuvkh4ep (In the prompt, put your name correctly else your submission will be rejected)

4. Lab Sessional Summary

1. A code template is given to you in lab sesional in this jupyter notebook
2. The template will follow a linear sequence of TODOs appropriately labelled with question marks
3. You will have to fill the TODO question marks to compile those notebook cells and proceed to next cells
4. You can think of the linear sequence of TODO templates as a guided thought process.
5. Sessional is open book. Google, github, browse product documentation ChatGPT - Do anything you want, except copying from your classmates (Sending questions to your seniors and seeking answers is prohibited). If you are caught carrying out these illegal activities, you will be reported to MSIS Director for immediate action.
6. You CANNOT replace the TODO code template with some other code copied from stack overflow, ChatGPT etc. All your browsing and search should give you insights into finally how you can fit that into the framework I provide for the thought process of solving the problem. You will have to mandatorily fill the question marks and proceed with the lab problem.

In [ ]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
```

Problem Definition

Green Manipal Initiative encourages two wheeler ride sharing for work in Manipal Univeristy from same residential localities. Green Manipal Initiative wants to pair folks with maximum

overlap in their interest/hobbies and work schedules for ride sharing because it has discovered over time that people tend to strike conversations during their ride sharing and having many common interests makes ride sharing sustain for a long time.

Towards this, it created questionnaires with a series of questions with Yes/No (1/0) answer options. The following questions were asked

1. Are you interested in philately?
2. Are you interested in bird watching?
3. Do you love planting trees?
4. Would you participate in clean manipal drive?
5. Do you own pets?
6. Do you subscribe to organic pesticide free farming?
7. Do you work normal 9-5 weekdays?

The response for the questionnaire by 10 residents of Malpe locality driving daily to Manipal was put in a dataframe and given to you as follows

In [ ]:
```
df_green = pd.DataFrame({
                         'name': ['Mohit', 'Varshini', 'Sunil', 'Rahul', 'Atishay
                         'philately' : [1, 0, 0, 0, 1, 0, 1, 0, 1, 0],
                         'birdwatching' : [0, 1, 0, 0, 1, 1, 0, 0, 1, 1],
                         'planting' : [1, 1, 0, 1, 0, 0, 0, 0, 0, 1],
                         'cleanmanipal' : [1, 0, 1, 1, 1, 0, 0, 0, 0, 1],
                         'pets': [0, 1, 1, 0, 0, 0, 0, 1, 0, 0],
                         'organic': [1, 1, 0, 1, 0, 0, 1, 0, 1, 1],
                         'normal_workweek': [1, 1, 1, 0, 1, 0, 0, 1, 1, 1]
                         })
df_green
```

Out[ ]:

	name	philately	birdwatching	planting	cleanmanipal	pets	organic	normal_workweek
0	Mohit	1	0	1	1	0	1	1
1	Varshini	0	1	1	0	1	1	1
2	Sunil	0	0	0	1	1	0	1
3	Rahul	0	0	1	1	0	1	0
4	Atishay	1	1	0	1	0	0	1
5	Keerthana	0	1	0	0	0	0	0
6	Vishwas	1	0	0	0	0	1	0
7	Mithilesh	0	0	0	0	1	0	1
8	Shreyas	1	1	0	0	0	1	1
9	Rakshith	0	1	1	1	0	1	1

Green Manipal Initiative has contacted you to identify top 2 pairs of potential ride sharers with most aligned interests.

You as the budding data scientist devised a formula to find the similarity between any two potential ride sharers a and b as the intersection over union (IoU)

$$similarity = \frac{a \cap b}{a \cup b}$$

a and b are two potential ride sharers as records in data frame.

For e.g.

In [ ]:
```
a = df_green.iloc[0]
print("First Ride sharer response record in dataframe = ")
print(a)
```

```
First Ride sharer response record in dataframe =
name               Mohit
philately              1
birdwatching           0
planting               1
cleanmanipal           1
pets                   0
organic                1
normal_workweek        1
Name: 0, dtype: object
```

In [ ]:
```
b = df_green.iloc[5]
print("Sixth Ride sharer response record in dataframe = ")
print(b)
```

```
Sixth Ride sharer response record in dataframe =
name               Keerthana
philately                  0
birdwatching               1
planting                   0
cleanmanipal               0
pets                       0
organic                    0
normal_workweek            0
Name: 5, dtype: object
```

The similarity formula between any two records a and b as given earlier is

$$similarity = \frac{a \cap b}{a \cup b}$$

where we can further use the following formulas in linear algebra

1. Formula for intersection of two records a and b: $a \cap b = a^T b$
2. Formula for union of two records a and b: $a \cup b = ||a||^2 + ||b||^2 - a \cap b$

where $||a||^2$ and $||b||^2$ squared L2 norm of the records a and b

In [ ]:
```
# TODO: 1

# Question 1: 7 questions were asked to each person, what is the size of response v
# Enter your answer here: ____7_____
```

In [ ]:
```
# TODO: 2

# Following are a and b records as 1D numpy array
a = np.array([1, 0, 1, 0, 0, 1, 1])
b = np.array([1, 1, 0, 1, 0, 0, 1])

# The intersection of a and b as per the formula given earlier
```

```
# Complete the following code in place of question mark to implement intersection
a_intersect_b = np.dot(a, b)
print(a_intersect_b)
```

```
2
```

In [ ]:
```
# TODO: 3

# Following are a and b records as 1D numpy array
a = np.array([1, 0, 1, 0, 0, 1, 1])
b = np.array([1, 1, 0, 1, 0, 0, 1])

# The union of a and b as per the formula given earlier
# Complete the following code in place of question mark to implement the union
a_union_b = np.linalg.norm(a) ** 2 + np.linalg.norm(b) ** 2 - np.dot(a, b)
print(a_union_b)
```

```
6.0
```

In [ ]:
```
# TODO: 4
# Similarity is also known as Intersection over union (IoU)
# Put the code for calculating the intersection over union into a single function
# 1. Fix compilation errors due to syntax mistake
# 2. Fill question marks

def intersection_over_union(a, b):
    a_intersect_b = np.dot(a, b)
    a_union_b = np.linalg.norm(a) ** 2 + np.linalg.norm(b) ** 2 - np.dot(a, b)
    return a_intersect_b/a_union_b
```

In [ ]:
```
# TODO: 5

# Following are a and b records as 1D numpy array
a = np.array([1, 0, 1, 0, 0, 1, 1])
b = np.array([1, 1, 0, 1, 0, 0, 1])

# fill the question marks
# call the function intersection_over_union for a and b and display the results
iou = intersection_over_union(a, b)
print(iou)
```

```
0.3333333333333333
```

The distance between any pair of potential ride sharers and b is
$d_{iou}(a, b) = 1 - similarity$

In [ ]:
```
# TODO: 6

# Following are a and b records as 1D numpy array
a = np.array([1, 0, 1, 0, 0, 1, 1])
b = np.array([1, 1, 0, 1, 0, 0, 1])

# Fill the question marks
dist = 1 - intersection_over_union(a, b)

# Print the IoU distance
print(dist)
```

```
0.6666666666666667
```

In [ ]:
```
# TODO: 7

#Make a function for iou distance
```

```
def dist_iou(a, b):
    return 1 - intersection_over_union(a, b)
```

In [ ]:
```
# TODO: 8

# Following are a and b records as 1D numpy array
a = np.array([1, 0, 1, 0, 0, 1, 1])
b = np.array([1, 1, 0, 1, 0, 0, 1])

# Fill the question marks
dist = dist_iou(a, b)

# Print the IoU distance
print(dist)
```

```
0.6666666666666667
```

Till now you have worked out a custom distance formula for ride sharing application.

Next task is to use the custom distance formula to find pairwise distances between the 10 Malpe-Manipal riders

In [ ]:
```
# Convert the riders dataframe into a numpy matrix
X = df_green.loc[:, df_green.columns != 'name'].to_numpy()

# print the type
print(f"type(X)={type(X)}")

# print the shape
print(f"X.shape={X.shape}")
```

```
type(X)=<class 'numpy.ndarray'>
X.shape=(10, 7)
```

1. There are 10 records in the numpy array X
2. You want to find pairwise distance between each of the 10 records
3. However you dont want to find distance of a record with self
4. You also dont want find the distance between records (b, a) when you would already found distance between (a, b)
5. Towards this you used a numpy function that gave the upper triangular indices

In [ ]:
```
# TODO: 9
# Get the right shifted version of upper triangular indices. Right shifted by one

idx1, idx2 = np.triu_indices(X.shape[0], k=1)
idx1, idx2
```

Out[ ]:
```
(array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2,
        2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7,
        8]),
 array([1, 2, 3, 4, 5, 6, 7, 8, 9, 2, 3, 4, 5, 6, 7, 8, 9, 3, 4, 5, 6, 7,
        8, 9, 4, 5, 6, 7, 8, 9, 5, 6, 7, 8, 9, 6, 7, 8, 9, 7, 8, 9, 8, 9,
        9]))
```

In [ ]:
```
# TODO: 10
# Use the index 1 to get first pair of riders

riders1 = X[idx1]

# What is the type and shape of riders1
```

```
print(type(riders1))
print(riders1.shape)
```

```
<class 'numpy.ndarray'>
(45, 7)
```

In [ ]:
```
# TODO: 11
# print first 5 entries of riders1
print(riders1[0:5])
```

```
[[1 0 1 1 0 1 1]
 [1 0 1 1 0 1 1]
 [1 0 1 1 0 1 1]
 [1 0 1 1 0 1 1]
 [1 0 1 1 0 1 1]]
```

In [ ]:
```
# TODO: 12
# use the index 2 to get second pair of riders

riders2 = X[idx2]

# print first 7 entries of riders1
print(riders2[0:7])
```

```
[[0 1 1 0 1 1 1]
 [0 0 0 1 1 0 1]
 [0 0 1 1 0 1 0]
 [1 1 0 1 0 0 1]
 [0 1 0 0 0 0 0]
 [1 0 0 0 0 1 0]
 [0 0 0 0 1 0 1]]
```

1. Instead of broadcasted vectorization, you will do programmatic looping over riders1 and riders2
2. Calculate the distance between the correpsonding elements in riders1 and riders2
3. Add the calculated distance to a numpy array

In [ ]:
```
# TODO: 13
# Create a empty numpy array to hold the calculated pairwise distances between ride
dist_arr = np.empty(X.shape[0])

for idx in np.arange(0,X.shape[0]):
    a = riders1[idx]
    b = riders2[idx]
    dist_arr[idx] = dist_iou(a, b)

# print the distances array
print(dist_arr)
```

```
[0.57142857 0.66666667 0.4        0.5        1.         0.6
 0.83333333 0.5        0.33333333 0.66666667]
```

1. Now you may want to obtain a sorted version of the distance array
2. But more than that you want a argsort and obtain the index of the original dist array

In [ ]:
```
sorted_dist_indices = np.argsort(dist_arr)
sorted_dist_indices
```

Out[ ]:
```
array([8, 2, 3, 7, 0, 5, 1, 9, 6, 4], dtype=int64)
```

In [ ]:
```
# TODO: 14
# Make a list of tuples of the indices from the the two upper triangular indices us

triu_idx_list = list(zip(idx1, idx2))
print(triu_idx_list)
```

```
[(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (1, 2),
(1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9), (2, 3), (2, 4), (2, 5),
(2, 6), (2, 7), (2, 8), (2, 9), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9),
(4, 5), (4, 6), (4, 7), (4, 8), (4, 9), (5, 6), (5, 7), (5, 8), (5, 9), (6, 7),
(6, 8), (6, 9), (7, 8), (7, 9), (8, 9)]
```

1. You found earlier that sorted_dist_indices contains the indices of distance between riders sorted.
2. For e.g. 21 is the first entry. This means the 21st entry in dist_arr is the minimum distance
3. The 21st distance is the distance between 21st riders in riders1 and riders2
4. Thats why you zipped the two upper triangular indices, so that you can pull their 21st entry as follows

In [ ]:
```
# 21 is the index of those two records with minimum distance
# Which are they?
# They are obtained peek inside triu_idx_list
best_rideshare_index = triu_idx_list[21]  # 21st record in triu_idx_list gives a tu
print(best_rideshare_index)
```

```
(2, 7)
```

In [ ]:
```
# Use the 2 and 7 as the indices to look up in the data frame name column to get th
print(df_green["name"].iloc[2])
print(df_green["name"].iloc[7])
```

```
Sunil
Mithilesh
```

In [ ]:
```
# This cell combines the work of the two previous cells together to get the two bes
print(df_green["name"].iloc[best_rideshare_index[0]])
print(df_green["name"].iloc[best_rideshare_index[1]])
```

```
Sunil
Mithilesh
```

In [ ]:
```
# 8 is the index of records with second minimum distance. Refer sorted_dist_indices
# Pull up the corresponding rider index from
secondbest_rideshare_index = triu_idx_list[8]
print(df_green["name"].iloc[secondbest_rideshare_index[0]])
print(df_green["name"].iloc[secondbest_rideshare_index[1]])
```

```
Mohit
Rakshith
```

Combine all of the above logic into a single function

In [ ]:
```
# TODO: 15
# This is a the combined logic in all of the above cells you executed combined into
# Fill the question marks and complete the function
def ride_share_pairs_best_2(df, col_to_pair="name"):
    # extract X numpy array from the dataframe by removing the name column
    X = df_green.loc[:, df_green.columns != col_to_pair].to_numpy()
```

```
    # get the right shifted upper triangular indices
    idx1, idx2 = np.triu_indices(X.shape[0], k=1)

    # Get the two set of riders for whom we want to find pairwise distances
    riders1 = X[idx1]
    riders2 = X[idx2]

    # Create a empty dist arr to hold the distance between riders1 and riders2
    dist_arr = np.empty(X.shape[0])

    # Loop over riders1 and riders2 and calculate their paiwise distance using inte
    for idx in np.arange(0,X.shape[0]):
        a = riders1[idx]
        b = riders2[idx]
        dist_arr[idx] = dist_iou(a, b)

    # Sort the distances and get their indices in ascending order
    sorted_dist_indices = np.argsort(dist_arr)

    # zip the upper triangular indices to create the ridershare pair rider indices
    triu_idx_list = list(zip(idx1, idx2))

    # Select the best pair of rideshare riders with minimum iou distance between th
    min_triu_index_tpl = triu_idx_list[21]
    best_pair = (df[col_to_pair].iloc[min_triu_index_tpl[0]], df[col_to_pair].iloc[

    # Select the second best pair of rideshare riders with second minimum iou dista
    second_min_triu_index_tpl = triu_idx_list[8]
    second_best_pair = (df[col_to_pair].iloc[second_min_triu_index_tpl[0]], df[col_

    # Combine the two best pairs into a list
    two_best_pairs = [best_pair, second_best_pair]
    return two_best_pairs
```

In [ ]:
```
# TEST
# If this cell prints prints two pairs of rideshare riders as follows, then your fu
# [ (Sunil, Mithilesh), (Mohit, Rakshith)]
ride_share_pairs_best_2(df_green)
```

Out[ ]:
```
[('Sunil', 'Mithilesh'), ('Mohit', 'Rakshith')]
```

In [ ]:
```

```