Bhaskaracharya College of Applied Sciences

Practical File

Name: Anand Kumar Mishra

Roll No.: 2102006

Subject: Machine Learning

Course: B.Sc. (Hons.) Computer Science

Sem./Year: 6th/3rd

P1. Perform elementary mathematical operations like addition, multiplication, division and exponentiation.

In [ ]:

```
a =20
b = 6

print(a+b)
print(a*b)
print(a/b)
print(a**b)
```

```
26
120
3.3333333333333335
64000000
```

P2. Perform elementary logical operations like OR, AND, Checking for Equality, NOT, XOR

In [ ]:

```
a = True
b = False

print(a and b)
print(a or b)
print(a == b)
print(a ^ b)
```

```
False
True
False
True
```

P3. Create, initialize and display simple variables and simple strings and use simple formatting for variable.

In [ ]:

```
x = 10
y = 3.14
z = True

# Display the variables
print("x =", x)
print("y =", y)
print("z =", z)

# Create and initialize simple strings
name = "Alice"
greeting = "Hello, " + name + "!"

# Display the strings
print(greeting)
print("The length of the name is", len(name))
# Use simple formatting for variables

age = 20
print("My age is {} years old.".format(age))
```

```
height = 1.70
print(f"My height is {height:.2f} meters.")
```

```
x = 10
y = 3.14
z = True
Hello, Alice!
The length of the name is 5
My age is 20 years old.
My height is 1.70 meters.
```

P4. Create/Define single dimension / multi-dimension arrays, and arrays with specific values like array of all ones, all zeros, array with random values within a range, or a diagonal matrix.

In [ ]:

```
import numpy as np

sing_d = np.array([1,2,3,4,5])
two_d = np.array([[1,2,3],[4,5,6]])
ones_arr = np.ones(5)
zeros_arr = np.zeros(5)
random_arr = np.random.rand(3)
dia_mat = np.diag([1,2,3])

print(sing_d)
print(two_d)
print(ones_arr)
print(zeros_arr)
print(random_arr)
print(dia_mat)
```

```
[1 2 3 4 5]
[[1 2 3]
 [4 5 6]]
[1. 1. 1. 1. 1.]
[0. 0. 0. 0. 0.]
[0.88173833 0.22806814 0.43275916]
[[1 0 0]
 [0 2 0]
 [0 0 3]]
```

P5. Use command to compute the size of a matrix, size/length of a particular row/column, load data from a text file, store matrix data to a text file, finding out variables and their features in the current scope.

In [ ]:

```
print(two_d.shape)
print(len(two_d[1]))
print(len(two_d[:,0]))

import pandas as pd

data = np.loadtxt('data.txt')
print(data)
data = np.array([[1,2,3], [4,5,6], [7,8,9]])
np.savetxt('data1.txt', data)
data=np.loadtxt('data1.txt')
print(data)
```

```
(2, 3)
3
2
[[110.9  146.03]
 [ 44.83 211.82]
 [ 97.13 209.3 ]
 [105.64 164.21]]
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

P6. Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix

In [ ]:

```
print(ones_arr + sing_d)
print(sing_d + ones_arr)
print(np.dot(sing_d, ones_arr))
print(two_d[0])
```

```
[2. 3. 4. 5. 6.]
[2. 3. 4. 5. 6.]
15.0
[1 2 3]
```

P7. Perform other matrix operations like converting matrix data to absolute values, taking the negative of matrix values, adding/removing rows/columns from a matrix, finding the maximum or minimum values in a matrix or in a row/column, and finding the sum of some/all elements in a matrix.

In [ ]:

```
mat = np.array([[1,-2,3], [-4,-5,6], [7,8,-9]])

mat = np.abs(mat)
print(mat)

mat = -mat
print(mat)

# Define a matrix
mat = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Add a row to the matrix
new_row = np.array([10, 11, 12])
mat_new_row = np.vstack((mat, new_row))

# Remove a column from the matrix
mat_remove_col = np.delete(mat, 1, axis=1)

# Display the results
print(mat_new_row)
print(mat_remove_col)

# Find the maximum value in the matrix
max_val = np.max(mat)

# Find the minimum value in the matrix
min_val = np.min(mat)

```

```
# Find the maximum value in each column
max_col = np.max(mat, axis=0)

# Find the minimum value in each row
min_row = np.min(mat, axis=1)

# Display the results
print(max_val)
print(min_val)
print(max_col)
print(min_row)

# Find the sum of all elements in the matrix
sum_all = np.sum(mat)

# Find the sum of elements in each row
sum_row = np.sum(mat, axis=1)

# Find the sum of elements in each column
sum_col = np.sum(mat, axis=0)

# Display the results
print(sum_all)
print(sum_row)
print(sum_col)
```

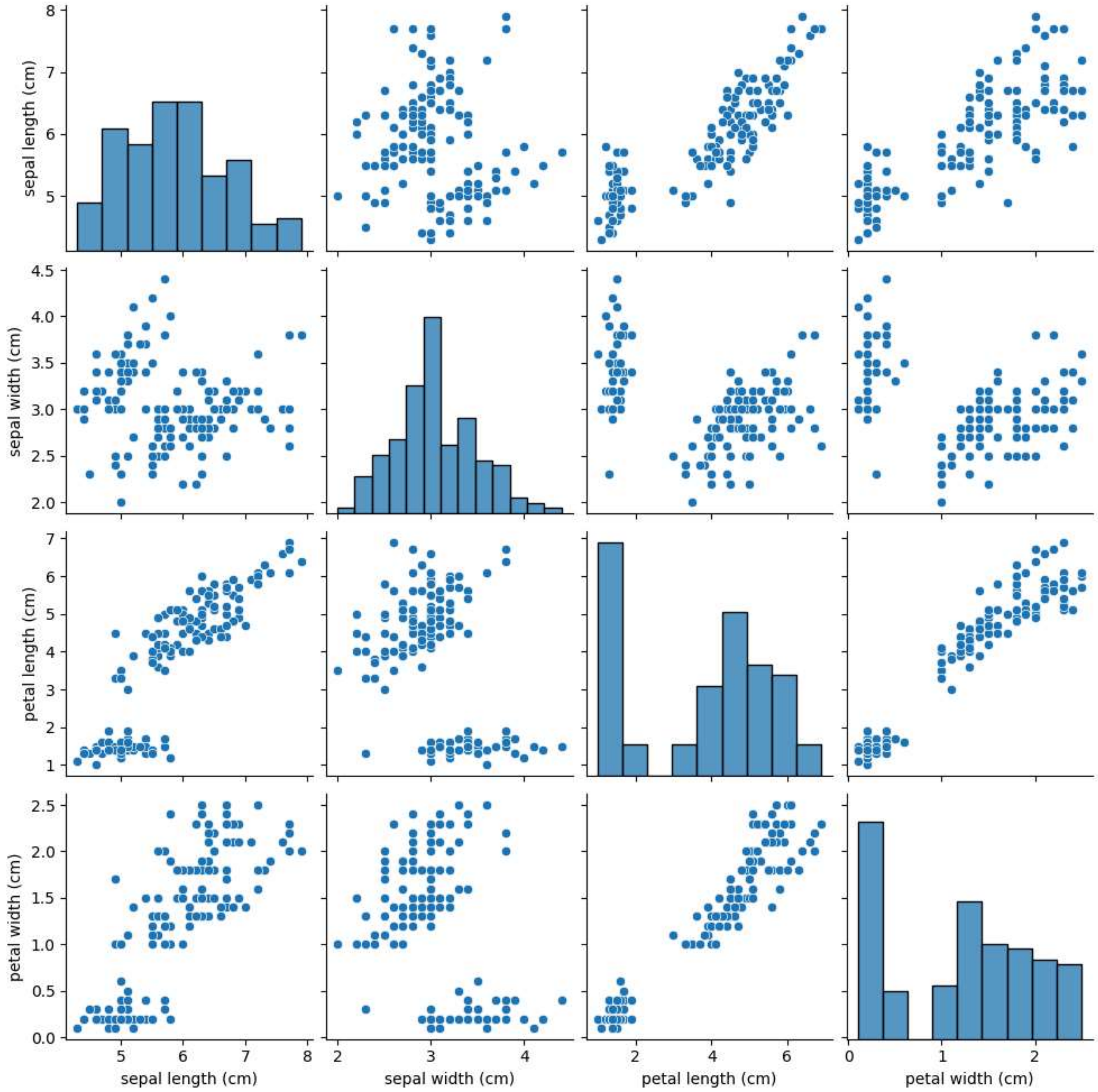
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[-1 -2 -3]
 [-4 -5 -6]
 [-7 -8 -9]]
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
[[1 3]
 [4 6]
 [7 9]]
9
1
[7 8 9]
[1 4 7]
45
[ 6 15 24]
[12 15 18]
```

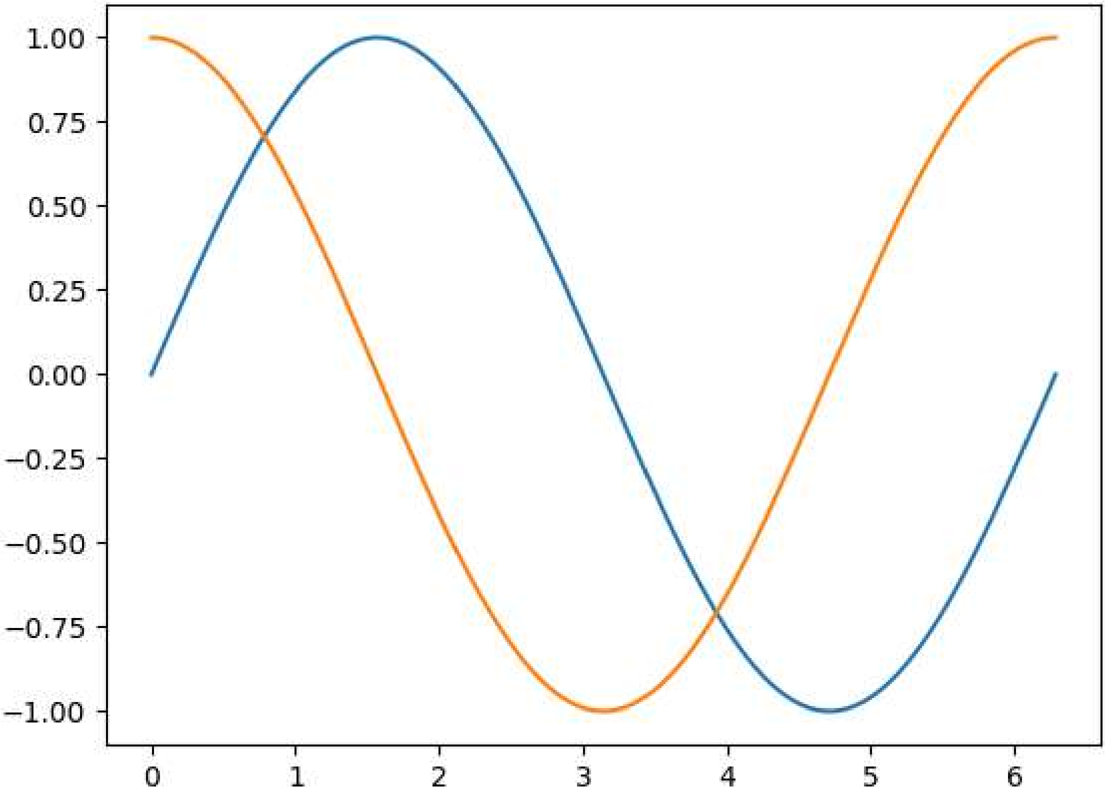
P8. Create various type of plots/charts like histograms, plot based on sine/cosine function based on data from a matrix. Further label different axes in a plot and data in a plot.

In [ ]:

```
#Loading the modules

import matplotlib.pyplot as plt
from sklearn.datasets import *
import seaborn as sns
```

```
# Loading the iris dataset

data = load_iris(as_frame=True)

sns.pairplot(data.data)
plt.show()


x = np.linspace(0, 2*np.pi, 100)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the sine and cosine functions
plt.plot(x, y_sin, label='sin(x)')
plt.plot(x, y_cos, label='cos(x)')
plt.show()
```



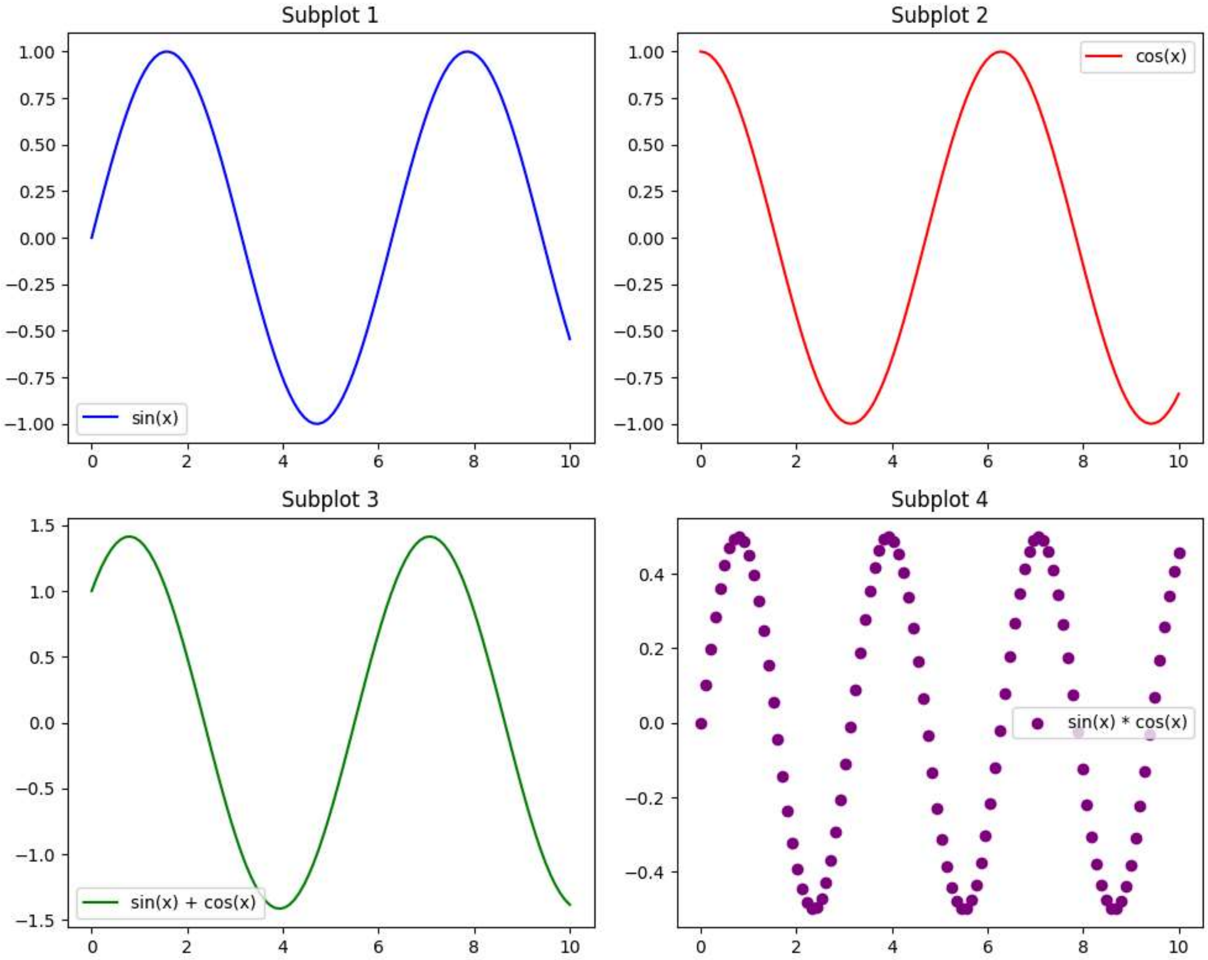


P9. Generate different subplots from a given plot and colour plot data

In [ ]:

```
# Generate sample data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Create a figure with subplots
fig, axs = plt.subplots(2, 2, figsize=(10, 8))

# Plot data on the first subplot
axs[0, 0].plot(x, y1, color='blue', label='sin(x)')
axs[0, 0].set_title('Subplot 1')
axs[0, 0].legend()

# Plot data on the second subplot
axs[0, 1].plot(x, y2, color='red', label='cos(x)')
axs[0, 1].set_title('Subplot 2')
axs[0, 1].legend()

# Plot data on the third subplot
axs[1, 0].plot(x, y1 + y2, color='green', label='sin(x) + cos(x)')
axs[1, 0].set_title('Subplot 3')
axs[1, 0].legend()

# Plot data on the fourth subplot
axs[1, 1].scatter(x, y1 * y2, color='purple', label='sin(x) * cos(x)')
axs[1, 1].set_title('Subplot 4')
axs[1, 1].legend()
```

```
# Adjust layout for better spacing
plt.tight_layout()

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



P10. Use conditional statements and different type of loops based on simple example/s

In [ ]:

```
# A simple loop to find odd and even numbers between 1 to 100

even, odd = [], []
for i in range(1, 101):
  if i%2==0:
    even.append(i)
  else:
    odd.append(i)

print("odd numbers are: ", odd)
print("even numbers are: ", even)
```

```
odd numbers are:  [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 3
5, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 7
7, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99]
even numbers are:  [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34,
36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76,
78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100]
```

P11. Perform vectorized implementation of simple matrix operation like finding the transpose of a matrix, adding, subtracting or multiplying two matrices.

In [ ]:

```
# Define two matrices
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
# Transpose of a matrix
A_T = A.T
print("Transpose of matrix A:\n", A_T)
# Adding two matrices
C = A + B
print("Sum of matrices A and B:\n", C)
# Subtracting two matrices
D = A - B
print("Difference of matrices A and B:\n", D)
# Multiplying two matrices
E = A.dot(B)
print("Product of matrices A and B:\n", E)
```

```
Transpose of matrix A:
 [[1 3]
 [2 4]]
Sum of matrices A and B:
 [[ 6  8]
 [10 12]]
Difference of matrices A and B:
 [[-4 -4]
 [-4 -4]]
Product of matrices A and B:
 [[19 22]
 [43 50]]
```

P12. Implement Linear Regression problem. For example, based on a dataset comprising of existing set of prices and area/size of the houses, predict the estimated price of a given house.

In [ ]:

```
data = pd.read_csv('housing.csv')
data.columns
```

Out[ ]:

```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')
```

In [ ]:

```
from sklearn.preprocessing import *
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import *
from sklearn.linear_model import LinearRegression
from sklearn.metrics import *
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')

X = data[['area', 'parking', 'stories', 'furnishingstatus', 'basement']]
y = data['price']
```

```
le = LabelEncoder()
X['furnishingstatus'] = le.fit_transform(X[['furnishingstatus']])
X['basement'] = le.fit_transform(X[['basement']])
print(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

pipe = Pipeline(steps=[
    ("scale", StandardScaler()),
    ("model", LinearRegression())
])

pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)

print("MSE: ", mean_squared_error(y_test, y_pred))

print("Score: ", r2_score(y_pred, y_test))
```

```
     area  parking  stories  furnishingstatus  basement
0    7420        2        3                 0         0
1    8960        3        4                 0         0
2    9960        2        2                 1         1
3    7500        3        2                 0         1
4    7420        2        2                 0         1
..    ...      ...      ...               ...       ...
540  3000        2        1                 2         1
541  2400        0        1                 1         0
542  3620        0        1                 2         0
543  2910        0        1                 0         0
544  3850        0        2                 2         0

[545 rows x 5 columns]
MSE:  2428100925301.8774
Score:  -0.3845000989559064
```

P13. Based on multiple features/variables perform Linear Regression. For example, based on a number of additional features like number of bedrooms, servant room, number of balconies, number of houses of years a house has been built – predict the price of a house.

In [ ]:

```
col = ['furnishingstatus', 'basement', 'mainroad', 'guestroom', 'basement', 'hotwat
num = ['bedrooms', 'bathrooms', 'area', 'stories', 'parking']

ct = ColumnTransformer(transformers=[
    ("encode", OneHotEncoder(), col),
    ("scale", StandardScaler(), num)
])

y = data['price']
X = data.drop(columns=['price'])

X = ct.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
pipe = Pipeline(steps=[
    ("scale", StandardScaler()),
    ("model", LinearRegression())
])

pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)

print("MSE: ", mean_squared_error(y_test, y_pred))

print("Score: ", r2_score(y_pred, y_test))
```

```
MSE:  1783348677178.379
Score:  0.21456209032472107
```

P14. Implement a classification/ logistic regression problem. For example based on different features of students data, classify, whether a student is suitable for a particular activity. Based on the available dataset, a student can also implement another classification problem like checking whether an email is spam or not.

In [ ]:

```
data, target = load_wine(as_frame=True, return_X_y=True)

data.head()
```

Out[ ]:

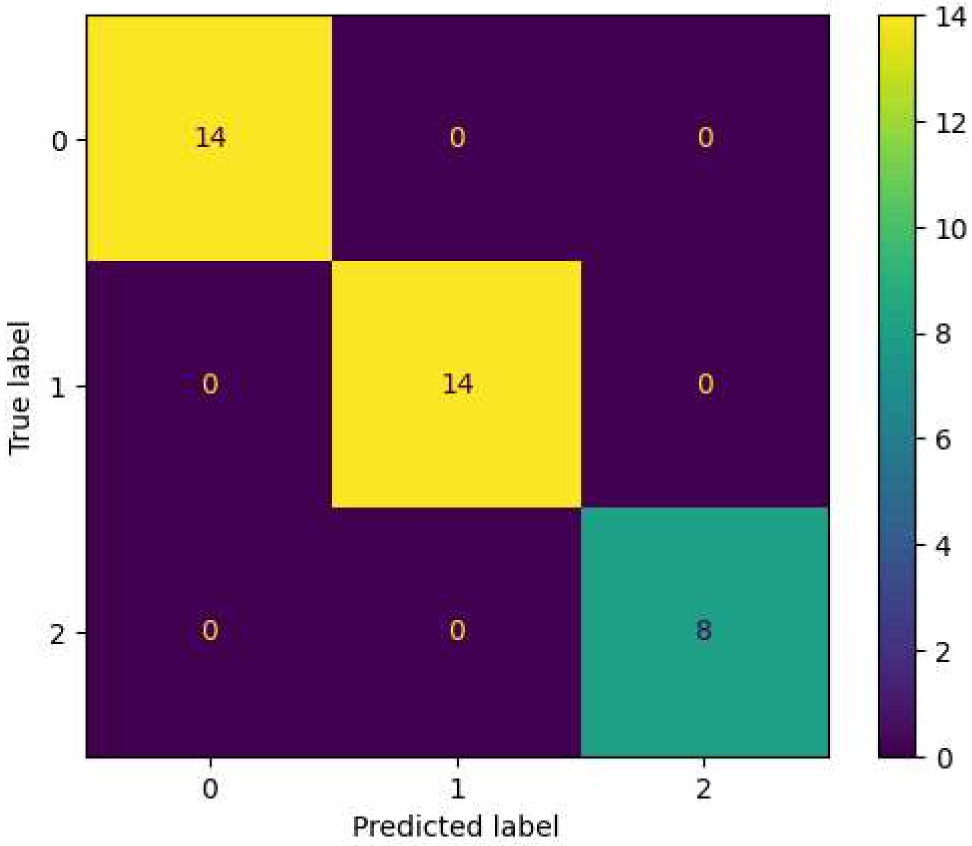
	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonfl
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	

In [ ]:

```
from sklearn.linear_model import LogisticRegression


X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, ra
pipe = Pipeline(steps=[
    ("scale", MinMaxScaler()),
    ("model", LogisticRegression(random_state=42))
])

pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)

cm = confusion_matrix(y_test, y_pred, labels=pipe[1].classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=pipe[1].classes_)
disp.plot()
```

```
print(classification_report(y_test, y_pred), "\n")
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      1.00      1.00        14
           2       1.00      1.00      1.00         8

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```

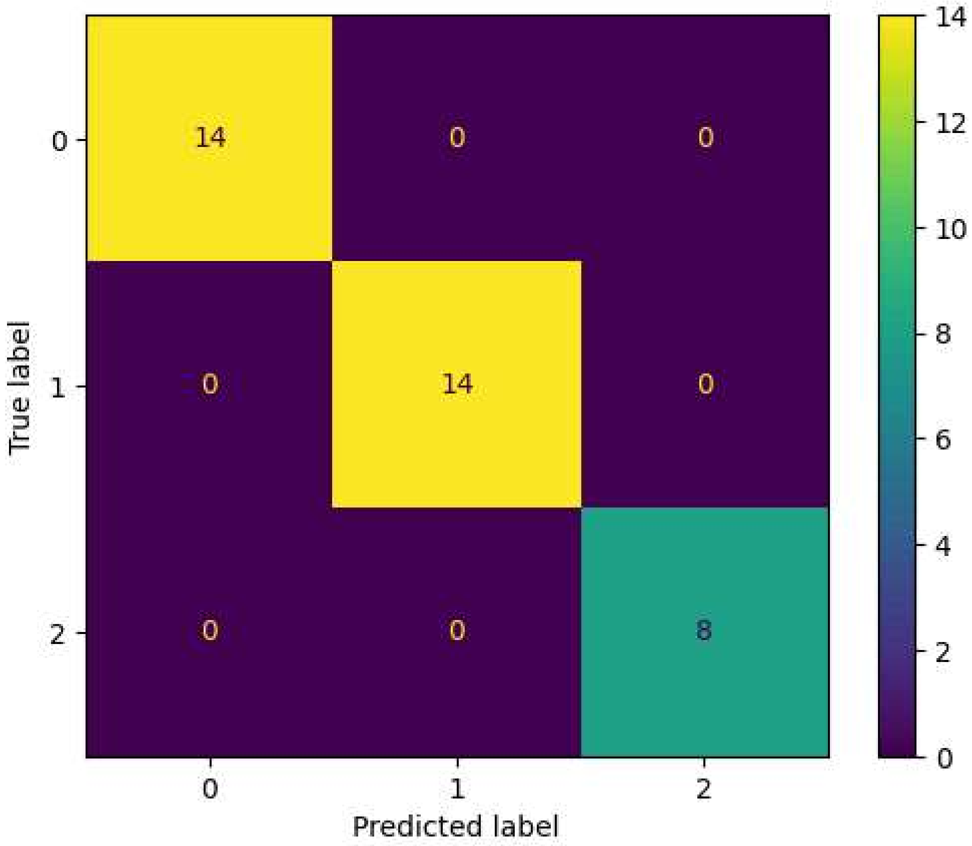


P15. Use some function for regularization of dataset based on problem P14

```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, ra
        pipe = Pipeline(steps=[
            ("scale", MinMaxScaler()),
            ("model", LogisticRegression(random_state=42, penalty='l1', solver='liblinear')
        ])

        pipe.fit(X_train, y_train)

        y_pred = pipe.predict(X_test)

        cm = confusion_matrix(y_test, y_pred, labels=pipe[1].classes_)
        disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=pipe[1].classes_)
        disp.plot()
```

```
print(classification_report(y_test, y_pred), "\n")
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      1.00      1.00        14
           2       1.00      1.00      1.00         8

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```

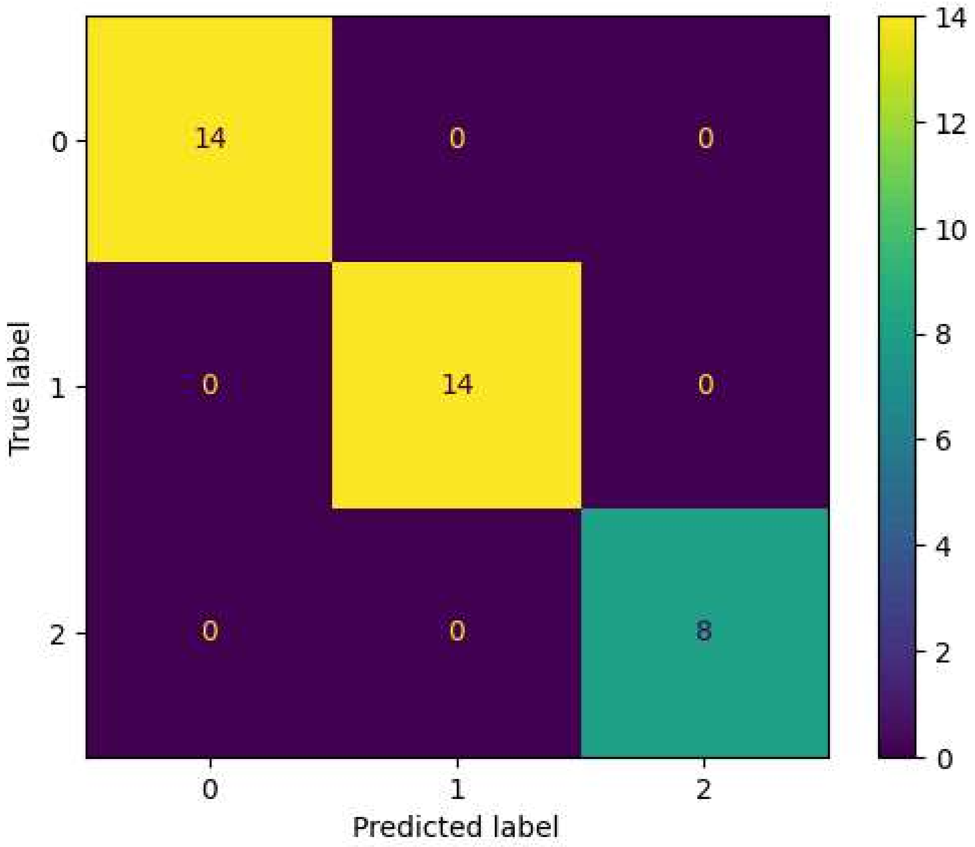


P16. Use some function for neural networks, like Stochastic Gradient Descent or backpropagation - algorithm to predict the value of a variable based on the dataset of problem 14

In [ ]:

```
from sklearn.neural_network import MLPClassifier
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')

X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, ra
pipe = Pipeline(steps=[
    ("scale", MinMaxScaler()),
    ("model", MLPClassifier(random_state=42))
])

param_grid = {
    'model__activation' : ['logistic', 'relu'],
    'model__solver' : ['sgd'],
    'model__learning_rate_init' : [0.001, 0.01, 0.0001, 0.1]
}

gs = GridSearchCV(pipe, param_grid=param_grid, cv=5)
gs.fit(X_train, y_train)

y_pred = gs.predict(X_test)

cm = confusion_matrix(y_test, y_pred, labels=gs.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=gs.classes_)
disp.plot()

print(classification_report(y_test, y_pred), "\n")
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      1.00      1.00        14
           2       1.00      1.00      1.00         8

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```


True label
Predicted label