Artificial Intelligence for Cybersecurity
Experiment 2
Working with Python for artificial intelligence

Python is one of the most popular programming languages used for artificial intelligence, and having a strong foundation in Python programming is crucial for success in this field. Here are some important Python skills for implementing machine learning algorithms:

- **Understanding the basics of Python:** Before starting with machine learning, it is essential to have a good understanding of Python syntax, data structures, control structures, functions, and modules.
- **Data visualisation:** Visualising data is an important aspect of exploratory data analysis and model interpretation. Skills in data visualisation libraries such as Matplotlib and Seaborn are essential for effective communication of results.
- **Familiarity with key Python libraries:** There are several libraries in Python that are used extensively in machine learning, such as NumPy, Pandas, Matplotlib, Scikit-learn, TensorFlow, and Keras. It is important to become familiar with these libraries and understand their capabilities.
- **Data manipulation and cleaning:** Handling and cleaning data is an important aspect of machine learning. It is essential to have the ability to extract, transform, and load data from various sources, and perform data cleaning and preprocessing.
- Algorithm implementation: It is important to have a good understanding of the different types of machine learning algorithms, including supervised and unsupervised learning, and to be able to implement them using Python libraries.
- Model evaluation and optimisation: Evaluating and optimising models is an important step in the machine learning workflow. Skills in cross-validation, hyperparameter tuning, and model selection are critical for achieving the best possible performance from machine learning models.

Basic Python

This program includes multiple data structures such as lists, tuples, dictionaries, arrays, and data frames. It also incorporates multiple control structures, such as for loops and while loops. There are also two functions defined within the program to perform simple operations.

In addition, this program uses multiple modules including NumPy, Pandas, and Matplotlib for data manipulation, analysis, and visualisation.

```
# Importing modules
import numpy as np
```

```
import matplotlib.pyplot as plt

# Defining functions
def add_numbers(x, y):
    """
    Adds two numbers and returns the result.
    """
    return x + y

def count_words(string):
    """
    Counts the number of words in a string.
    """
    words = string.split()
    return len(words)

# Defining data structures
my_list = [10, 21, 30, 41, 50]
my_tuple = ('a', 'b', 'c', 'd', 'e')
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
my_array = np.array([11, 22, 33, 44, 55])

# Using control structures
for num in my_list:
    if num % 2 == 0:
        print(f"{num} is even")
    else:
        print(f"{num} is odd")

while count_words(input("Enter a sentence: ")) < 5:
    print("Please enter a sentence with at least 5 words.")


# Using data visualisation
plt.plot(my_list, my_array)
plt.title("A Plot of My List and My Array")
plt.xlabel("My List")
plt.ylabel("My Array")
plt.show()
```

Customise the properties of a plot

Here are some ways to customise the appearance of a Matplotlib plot:

1. Change the colour of the lines or markers: You can specify the colour of the lines or markers in the plot using the colour argument. For example:

   ```
   plt.plot(x, y, color='red')
   ```

2. Change the line style: You can specify the style of the lines in the plot using the linestyle argument. For example:

```
plt.plot(x, y, linestyle='dashed')
```

3. Add markers to the lines: You can add markers to the lines in the plot using the marker argument. For example:

```
plt.plot(x, y, marker='o')
```

4. Change the axis labels and titles: You can specify the labels and titles for the x and y axes, as well as the title for the plot, using the xlabel, ylabel, and title functions, respectively. For example:

```
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line graph using Matplotlib in Google Colab')
```

5. Add grid lines: You can add grid lines to the plot using the grid function. For example:

```
plt.grid()
```

6. Customise the limits of the axes: You can specify the limits of the x and y axes using the xlim and ylim functions, respectively. For example:

```
plt.xlim(0, 6)
plt.ylim(0, 6)
```

These are just a few examples of the many ways you can customise the appearance of a Matplotlib plot. You can explore the Matplotlib documentation (https://matplotlib.org/stable/index.html) for more information on other customisation options and advanced plot properties.

NumPy

This example program demonstrates several important features of NumPy and SciPy. First, it creates a NumPy array of random values using the np.random.randn() function. Then, it computes the mean and standard deviation of the array using the np.mean() and np.std() functions, respectively. Additionally, it performs element-wise arithmetic operations on a NumPy array of integers using standard arithmetic operators.

```
# Importing the required libraries
import numpy as np
```

```
# Creating a NumPy array of random values
arr = np.random.randn(100)

# Computing the mean and standard deviation of the array
mean = np.mean(arr)
std_dev = np.std(arr)

# Printing the mean and standard deviation
print("Mean:", mean)
print("Standard deviation:", std_dev)

# Creating a NumPy array of integers
arr_int = np.array([1, 2, 3, 4, 5])

# Performing element-wise arithmetic operations on the array
add_arr = arr_int + 5
sub_arr = arr_int - 2
mult_arr = arr_int * 3
div_arr = arr_int / 2

# Printing the result of each operation
print("Addition:", add_arr)
print("Subtraction:", sub_arr)
print("Multiplication:", mult_arr)
print("Division:", div_arr)
```

Pandas

This program demonstrates several important features of Pandas. First, it creates a DataFrame from a dictionary containing some sample data. Then, it selects a subset of columns using the double bracket notation. Next, it filters rows based on a condition using boolean indexing. It also groups the DataFrame by a column and calculates statistics such as mean using the groupby() and mean() functions, respectively. Finally, it sorts the DataFrame by a column using the sort_values() function.

```
# Importing the required libraries
import pandas as pd

# Creating a dictionary with some sample data
data = {'Name': ['John', 'Jane', 'Bob', 'Alice'],
        'Age': [25, 30, 40, 35],
        'Gender': ['Male', 'Female', 'Male', 'Female'],
        'Salary': [50000, 60000, 80000, 70000]}

# Creating a pandas DataFrame from the dictionary
df = pd.DataFrame(data)

# Printing the DataFrame
print("Original DataFrame:")
```

```
print(df)

# Selecting a subset of columns
df_subset = df[['Name', 'Salary']]

# Printing the subset DataFrame
print("\nSubset DataFrame:")
print(df_subset)

# Filtering rows based on a condition
df_filtered = df[df['Age'] >= 30]

# Printing the filtered DataFrame
print("\nFiltered DataFrame:")
print(df_filtered)

# Grouping the DataFrame by a column and calculating statistics
df_grouped = df.groupby('Gender').mean()

# Printing the grouped DataFrame
print("\nGrouped DataFrame:")
print(df_grouped)

# Sorting the DataFrame by a column
df_sorted = df.sort_values('Age')

# Printing the sorted DataFrame
print("\nSorted DataFrame:")
print(df_sorted)
```

Data manipulation and cleaning

This program demonstrates several important features of data manipulation and cleaning using Pandas, NumPy, and SciPy. First, it creates a sample dataset using a Python dictionary and converts it to a Pandas DataFrame. The sample dataset includes two missing values in the Income column. First, the program removes the null values using the dropna() function. Then, it fills the missing values with the mean of the column using the fillna() function from Pandas. It also adds a new column to the dataframe using the pd.cut() function to categorise the age into age groups. Additionally, it computes the z-score of the income column using the zscore() function from the SciPy library. Finally, it groups the dataframe by the gender and age group columns using the groupby() function and computes the mean income using the mean() function.

```
# Importing the required libraries
import pandas as pd
from scipy.stats import zscore

# Creating a sample dataset
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'Dave', 'Emily', 'Frank'],
```

```
    'Age': [25, 30, 35, 40, 45, 50],
    'Gender': ['F', 'M', 'M', 'M', 'F', 'M'],
    'Income': [50000, 60000, 70000, 80000, np.nan, np.nan]
}
df = pd.DataFrame(data)

# Printing the initial dataframe
print("Initial DataFrame:")
print(df)

# Removing null values
df.dropna(inplace=True)

df = pd.DataFrame(data)
# Filling missing values with the mean of the column
df.fillna(df.mean(numeric_only=True), inplace=True)

# Adding a new column to the dataframe
df['Age_Group'] = pd.cut(df['Age'], bins=[18, 30, 40, 50], labels=['18-30',
'31-40', '41-50'])

# Computing the z-score of the income column
df['Income_zscore'] = zscore(df['Income'])

# Grouping the dataframe by the gender and age group columns and computing
the mean income
grouped_df = df.groupby(['Gender',
'Age_Group'])['Income'].mean().reset_index()

# Printing the final grouped dataframe
print("Final Grouped DataFrame:")
print(grouped_df)
```

Practice tasks

1. Generate some random data and visualise it using different plot, such as line graph, scatter plot and bar plot. Give necessary labels and titles to the plots.
2. Do the following with Pandas:
    a. Load a dataset of your choice into a pandas DataFrame.
    b. Print the first five rows of the DataFrame.
    c. Select a subset of columns and print the resulting DataFrame.
    d. Filter the rows based on a condition and print the resulting DataFrame.
    e. Group the DataFrame by a column of your choice and calculate the median of another column.
    f. Sort the DataFrame by a column of your choice in ascending order and print the resulting DataFrame.
3. Do the following:

a. Create a sample dataset with at least 10 rows and 4 columns, including one column with null values and one column with missing values.
b. Load the dataset into a Pandas dataframe and display the first 5 rows using the head() function.
c. Use the isnull() function to count the number of null values in the dataset.
d. Use the fillna() function to fill the missing values in the dataset with the median of the column.
e. Add a new column to the dataframe that categorises the data in one of the existing columns. For example, if one column is "age", you could add a new column called "age group" that categorises ages as "young", "middle-aged", or "old".
f. Use the groupby() function to group the dataframe by one or more columns and compute a statistical measure, such as the mean or standard deviation.
g. Display the final grouped dataframe.