Python

Experiment No. 12

Aim: To implement different machine learning packages numpy, pandas and matplotlib

Problem Statements:

1. Create a result array by adding the following two NumPy arrays. Next, modify the result array by calculating the square of each element
2. Import Iris dataset and solve the following:
    a. From the given dataset print the first and last five rows
    b. to check for null values
    c. count of null values in each column
    d. Fill in the null values with zero, last valid observation and next valid observations along axes or with values.
    e. Replace null values with other value and drop null values
    f. Sort any one column in descending order
    g. Find the most expensive data
    h. Concatenate two data frames
3. Visualize the Iris data using matplotlib library for the following:
    a. Linear regression
    b. Heatmap
    c. Scatter plot
    d. Histogram

Theory:

Data Analysis
Data Analysis is a process of inspecting, cleaning, transforming, and modeling data with the goal of discovering useful information, suggesting conclusions, and supporting decision-making.
Steps for Data Analysis, Data Manipulation and Data Visualization:

1. Transform Raw Data in a Desired Format
2. Clean the Transformed Data (Step 1 and 2 also called as a Pre-processing of Data)
3. Prepare a Model
4. Analyse Trends and Make Decisions

NumPy

NumPy is a package for scientific computing.

1. Multi-dimensional array
2. Methods for processing arrays
3. Element by element operations
4. Mathematical operations like logical, Fourier transform, shape manipulation, linear algebra and random number generation

Ndarray - NumPy Array

The ndarray is a multi-dimensional array object consisting of two parts -- the actual data, some metadata which describes the stored data. They are indexed just like sequence are in Python, starting from 0

- Each element in ndarray is an object of data-type object called dtype
- An item extracted from ndarray, is represented by a Python object of an array scalar type

Single Dimensional Array

Creating a Numpy Array

```
# Creating a single-dimensional array
a =np.array([1,2,3]) # Calling the array function
print(a)

# Creating a multi-dimensional array
# Each set of elements within a square bracket indicates a row
# Array of two rows and two columns
b =np.array([[1,2], [3,4]])
print(b)

# Creating an ndarray by wrapping a list
#Creating a list
list1=[1,2,3,4,5]
arr = np.array(list1) # Wrapping the list
print(arr)

# Creating an array of numbers of a specified range
arr1 =np.arange(10, 100) # Array of numbers from 10 up to and
excluding 100
print(arr1)

# Creating a 5x5 array of zeroes
arr2 =np.zeros((5,5))
print(arr2)

# Creating Arrays from Existing Data
x = [1,2,3]
```

```
# Used for converting Python sequences into ndarrays
c =np.asarray(x) #np.asarray(a, dtype = None, order = None)
print(c)
```

```
[1 2 3]
[[1 2]
 [3 4]]
[1 2 3 4 5]
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99]
[[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 2 3]
```

Indexing of NumPy Arrays

```
# NumPy array indexing is identical to Python's indexing scheme
arr5 =np.arange(2, 20)
element = arr5[6]
print(element)

arr6 =np.arange(20)
arr_slice=slice(1, 10, 2) # Start, stop & step
element2 = arr6[6]
print(arr6[arr_slice])

# Extracting specific rows and columns using Slicing
d =np.array([[1,2,3], [3,4,5], [4,5,6]])
print(d[0:2, 0:2]) # Slice the first two rows and the first two
columns
```

```
8
[1 3 5 7 9]
[[1 2]
 [3 4]]
```

NumPy Array Attributes

```
print(d.shape) # Returns a tuple consisting of array dimensions
print(d.ndim) # Attribute returns the number of array dimensions
print(a.itemsize) # Returns the length of each element of array in
bytes

y =np.empty([3,2], dtype=int) # Creates an uninitialized array of
specified shape and dtype
print(y)

# Returns a new array of specified size, filled with zeros
z =np.zeros(5) # np.zeros(shape, dtype = float)
```

```
print(z)
```

```
(3, 3)
2
8
[[     96664535878756                  0]
 [    132511938551008     132511938551008]
 [7162247753406823781 2314885530449946484]]
[0. 0. 0. 0. 0.]
```

Numpy | Iterating Over Array

NumPy package contains an iterator object numpy.nditer. It is an efficient multidimensional iterator object using which it is possible to iterate over an array. Each element of an array is visited using Python's standard Iterator interface.

```
for x in geek.nditer(b):

print(x)
```

Controlling Iteration Order:

There are times when it is important to visit the elements of an array in a specific order, irrespective of the layout of the elements in memory. The nditer object provides an order parameter to control this aspect of iteration. The default, having the behavior described above, is order='K' to keep the existing order. This can be overridden with order='C' for C order and order='F' for Fortran order.

Modifying Array Values:

The nditer object has another optional parameter called **op_flags**. Its default value is read-only, but can be set to read-write or write-only mode. This will enable modifying array elements using this iterator.

Code:

```
# Python program for
# iterating over array
# using particular order

import numpy as geek

# creating an array using arrange
# method
a = geek.arange(12)

# shape array with 3 rows and
# 4 columns
a = a.reshape(3,4)

print('Original array is:')
```

```
print(a)
print()

print('Modified array in C-style order:')

# iterating an array in a given
# order
for x in geek.nditer(a, order ='C'):
    print(x)
```

Output:

```
Original array is:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

Modified array in C-style order:
0
1
2
3
4
5
6
7
8
9
10
11
```

Modifying Array Values:

The nditer object has another optional parameter called **op_flags**. Its default value is read-only, but can be set to read-write or write-only mode. This will enable modifying array elements using this iterator.

Code:

```
# Python program for
# modifying array values

import numpy as geek

# creating an array using arrange
# method
a = geek.arange(12)

# shape array with 3 rows and
# 4 columns
a = a.reshape(3,4)
print('Original array is:')
print(a)
print()
```

```
# modifying array values
for x in geek.nditer(a, op_flags=['readwrite']):
    x[...] =5*x
print('Modified array is:')
print(a)
```

Output:

```
Original array is:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

Modified array is:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]
```

Pandas

Pandas is an open-source Python library providing efficient, easy-to-use data structure and data analysis tools. The name Pandas is derived from "Panel Data" - an Econometrics from Multidimensional Data. Pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-type columns.
- Ordered and unordered time series data.
- Arbitary matrix data with row and column labels.
- Any other form observational/statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure.

Pandas provides three data structure - all of which are build on top of the NumPy array - all the data structures are value-mutable

- Series (1D) - labeled, homogenous array of immutable size
- DataFrames (2D) - labeled, heterogeneously typed, size-mutable tabular data structures
- Panels (3D) - Labeled, size-mutable array

```
import pandas as pd
```

Series

1. A Series is a single-dimensional array structures that stores homogenous data i.e., data of a single type.
2. All the elements of a Series are value-mutable and size-immutable
3. Data can be of multiple data types such as ndarray, lists, constants, series, dict etc.
4. Indexes must be unique, hashable and have the same length as data. Defaults to np.arrange(n) if no index is passed.
5. Data type of each column; if none is mentioned, it will be inferred; automatically
6. Deep copies data, set to false as default

```
# Creating an empty Series
series=pd.Series() # The Series() function creates a new Series
print(series)

# Creating a series from an ndarray
# Note that indexes are a assigned automatically if not specifies
arr=np.array([10,20,30,40,50])
series1 =pd.Series(arr)
print(series1)

# Creating a series from a Python dict
# Note that the keys of the dictionary are used to assign indexes
during conversion
data = {'a':10, 'b':20, 'c':30}
series2 =pd.Series(data)
print(series2)
```

```
# Retrieving a part of the series using slicing
print(series1[1:4])
```

```
Series([], dtype: object)
0    10
1    20
2    30
3    40
4    50
dtype: int64
a    10
b    20
c    30
dtype: int64
1    20
2    30
3    40
dtype: int64
```

DataFrames

1. A DataFrame is a 2D data structure in which data is aligned in a tabular fashion consisting of rows & columns
2. A DataFrame can be created using the following constructor - pandas.DataFrame(data, index, dtype, copy)
3. Data can be of multiple data types such as ndarray, list, constants, series, dict etc.
4. Index Row and column labels of the dataframe; defaults to np.arrange(n) if no index is passed
5. Data type of each column
6. Creates a deep copy of the data, set to false as default

```
# Converting a list into a DataFrame
list1 = [10, 20, 30, 40]
table =pd.DataFrame(list1)
print(table)

# Creating a DataFrame from a list of dictionaries
data = [{'a':1, 'b':2}, {'a':2, 'b':4, 'c':8}]
table1 =pd.DataFrame(data)
print(table1)
# NaN (not a number) is stored in areas where no data is provided

# Creating a DataFrame from a list of dictionaries and
accompaying row indices
table2 =pd.DataFrame(data, index = ['first', 'second'])
# Dict keys become column lables
print(table2)

# Converting a dictionary of series into a DataFrame
data1 = {'one':pd.Series([1,2,3], index = ['a', 'b', 'c']),
'two':pd.Series([1,2,3,4], index = ['a', 'b', 'c', 'd'])}
table3 =pd.DataFrame(data1)
```

```
print(table3)
# the resultant index is the union of all the series indexes passed
```

```
    0
0  10
1  20
2  30
3  40
   a  b    c
0  1  2  NaN
1  2  4  8.0
        a  b    c
first   1  2  NaN
second  2  4  8.0
    one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4
```

DataFrame - Addition & Deletion of Columns

```
# A new column can be added to a DataFrame when the data is passed as a Series
table3['three'] =pd.Series([10,20,30], index = ['a', 'b', 'c'])
print(table3)

# DataFrame columns can be deleted using the del() function
del table3['one']
print(table3)

# DataFrame columns can be deleted using the pop() function
table3.pop('two')
print(table3)
```

```
    one  two  three
a  1.0    1   10.0
b  2.0    2   20.0
c  3.0    3   30.0
d  NaN    4    NaN
    two  three
a    1   10.0
b    2   20.0
c    3   30.0
d    4    NaN
   three
a   10.0
b   20.0
c   30.0
d    NaN
```

DataFrame - Addition & Deletion of Rows

```
# DataFrame rows can be selected by passing the row lable to the loc() function
print(table3.loc['c'])
```

```
# Row selection can also be done using the row index
print(table3.iloc[2])
```

```
three    30.0
Name: c, dtype: float64
three    30.0
Name: c, dtype: float64
```

Importing & Exporting Data

```
# Data can be loaded into DataFrames from input data stored in the CSV format using the read_csv() function
table_csv=pd.read_csv('../input/Cars2015.csv')

# Data present in DataFrames can be written to a CSV file using the to_csv() function
# If the specified path doesn't exist, a file of the same name is automatically created
table_csv.to_csv('newcars2015.csv')

# Data can be loaded into DataFrames from input data stored in the Excelsheet format using read_excel()
sheet =pd.read_excel('cars2015.xlsx')
```

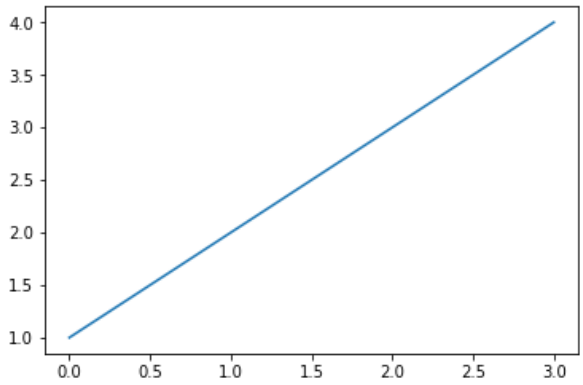
Matplotlib

1. Matplotlib is a Python library that is specially designed for the development of graphs, charts etc., in order to provide interactive data visualisation
2. Matplotlib is inspired from the MATLAB software and reproduces many of it's features

```
# Import Matplotlib submodule for plotting
import matplotlib.pyplot as plt
```

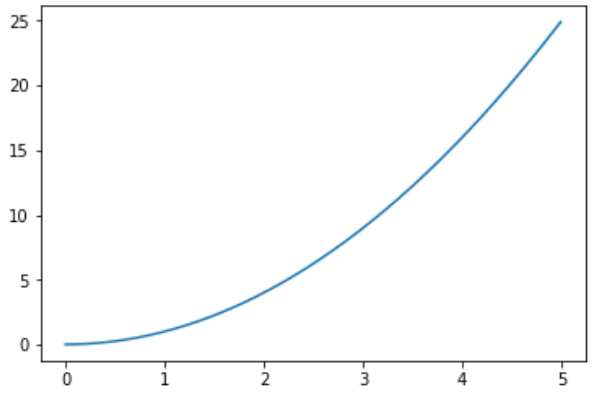
Plotting in Matplotlib

```
plt.plot([1,2,3,4]) # List of vertical co-ordinates of the points plotted
plt.show() # Displays plot
# Implicit X-axis values from 0 to (N-1) where N is the length of the list
```



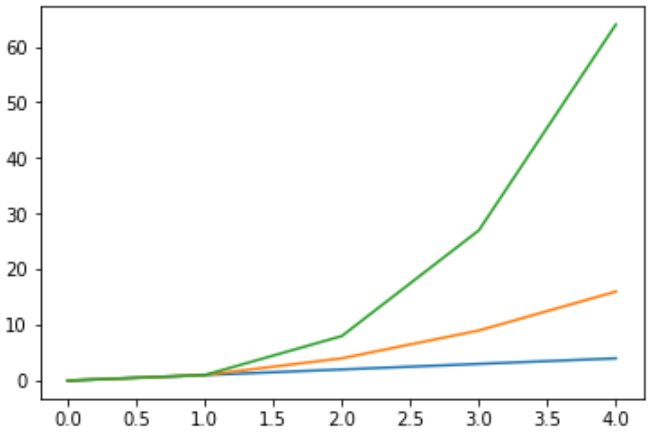
 # We can use NumPy to specify the values for both axes with greater precision

```
x =np.arange(0, 5, 0.01)
plt.plot(x, [x1**2for x1 in x]) # vertical co-ordinates of the points plotted: y = x^2
plt.show()
```

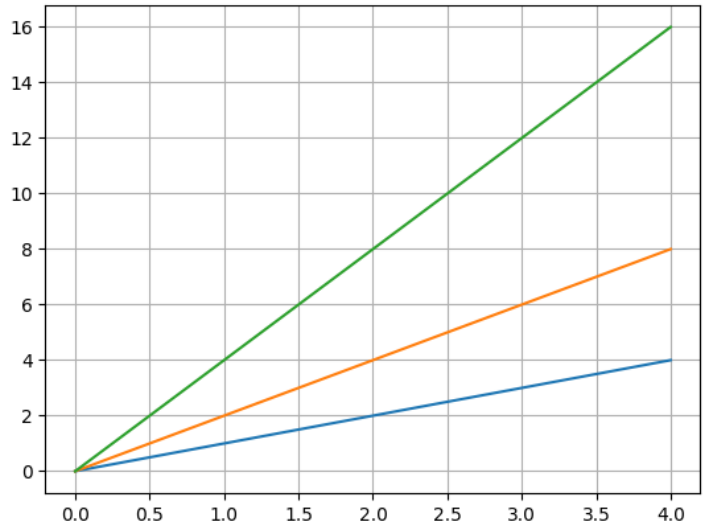


Multiline Plots

```
# Multiple functions can be drawn on the same plot
x =range(5)
plt.plot(x, [x1 for x1 in x])
plt.plot(x, [x1*x1 for x1 in x])
plt.plot(x, [x1*x1*x1 for x1 in x])
plt.show()
```

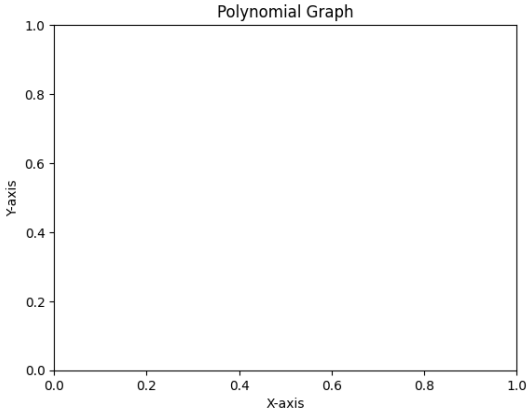



```
# The grid() function adds a grid to the plot
# grid() takes a single Boolean parameter
# grid appears in the background of the plot

x = range(5)

plt.plot(x, [x1 for x1 in x],
         x, [x1*2 for x1 in x],
         x, [x1*4 for x1 in x])
plt.grid(True)
plt.show()
```



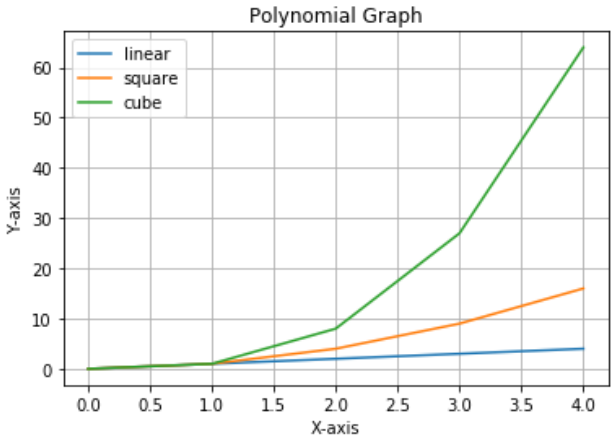
Adding labels and title

```
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title("Polynomial Graph") # Pass the title as a parameter to
title()
plt.show()
```



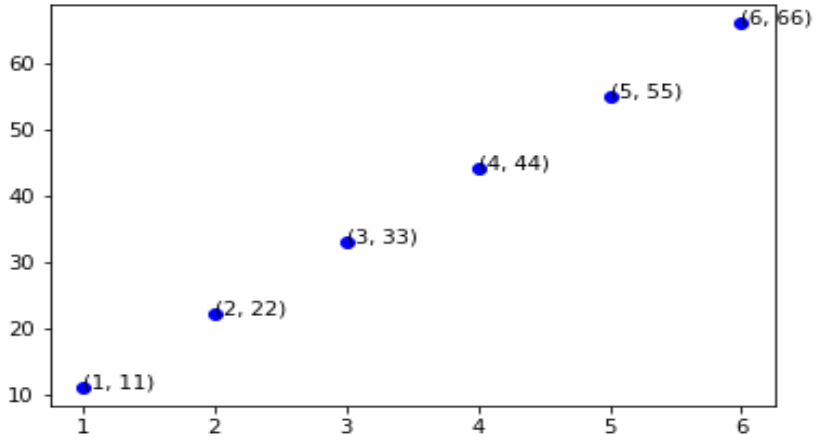
 # Legends explain the meaning of each line in the graph

```
x =np.arange(5)

plt.plot(x, x, label ='linear')
plt.plot(x, x*x, label ='square')
plt.plot(x, x*x*x, label ='cube')
plt.grid(True)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title("Polynomial Graph")
plt.legend()
plt.show()
```



Markers

```
x = [1, 2, 3, 4, 5, 6]
y = [11, 22, 33, 44, 55, 66]
plt.plot(x, y, 'bo')
for i in range(len(x)):
    x_cord= x[i]
    y_cord= y[i]
    plt.text(x_cord, y_cord, (x_cord, y_cord), fontsize=10)
plt.show()
```




```
# Saving Plots as image
plt.savefig('plot.png')
```

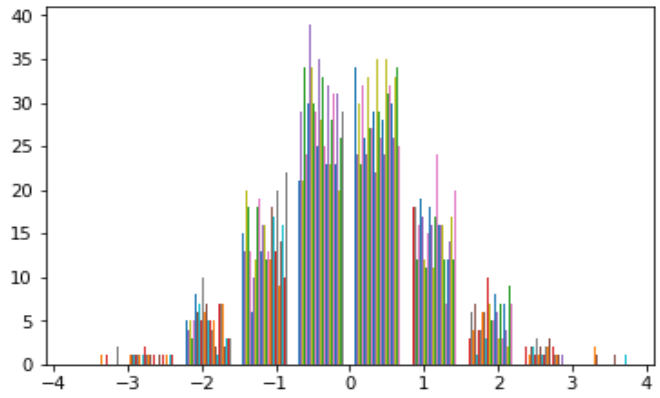
Plot Types

Matplotlib provides many types of plot formats for visualising information

- Scatter Plot
- Histogram
- Bar Graph
- Pie Chart

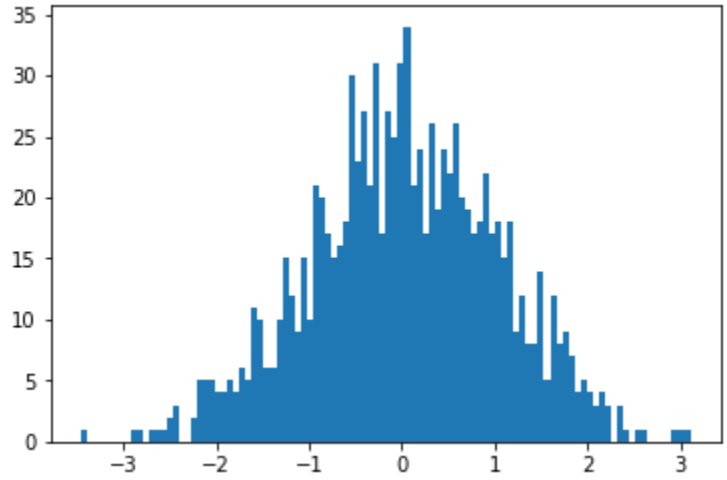
 # Histograms display the distribution of a variable over a range of frequencies or values

```
y = np.random.randn(100, 100)    # 100x100 array of a Gaussian
distribution
plt.hist(y)     # Function to plot the histogram takes the dataset as
the parameter
plt.show()
```



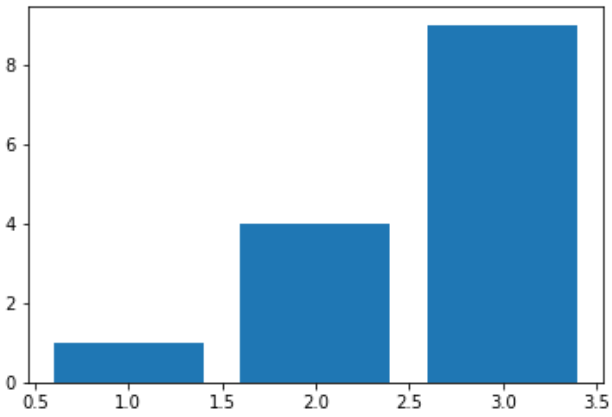
 # Histogram groups values into non-overlapping categories called bins

```
# Default bin value of the histogram plot is 10
y = np.random.randn(1000)
plt.hist(y, 10)
plt.show()
```

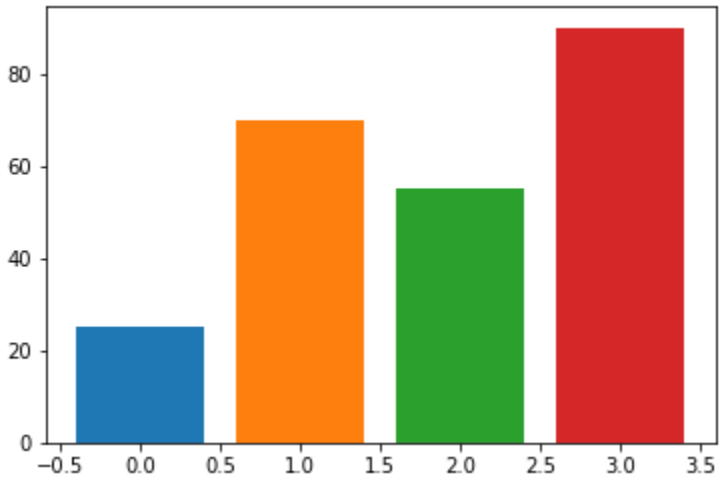


 # Bar charts are used to visually compare two or more values using rectangular bars

```
# Default width of each bar is 0.8 units
# [1,2,3] Mid-point of the lower face of every bar
# [1,4,9] Heights of the successive bars in the plot
plt.bar([1,2,3], [1,4,9])
plt.show()
```

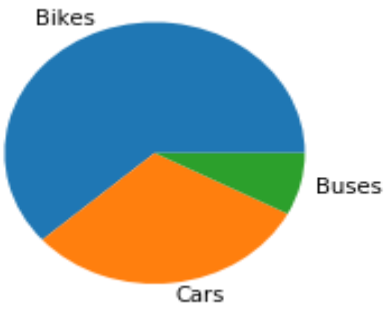



```
dictionary = {'A':25, 'B':70, 'C':55, 'D':90}
for i, key in enumerate(dictionary):
    plt.bar(i, dictionary[key])     # Each key-value pair is plotted
individually as dictionaries are not iterable
plt.show()
```



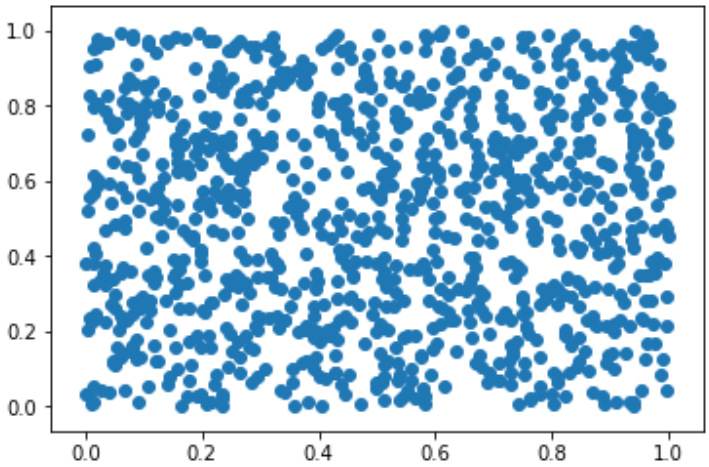
Pie Chart

```
plt.figure(figsize= (3,3))  # Size of the plot in inches
x = [40, 20, 5]     # Proportions of the sectors
labels = ['Bikes', 'Cars', 'Buses']
plt.pie(x, labels = labels)
plt.show()
```



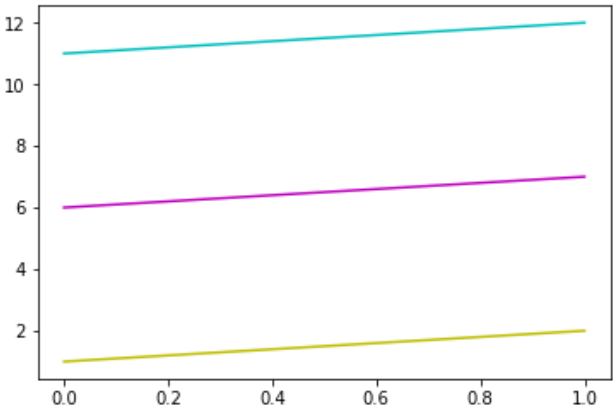
Scatter Plot

```
# Scatter plots display values for two sets of data, visualised as a
collection of points
# Two Gaussion distribution plotted
x = np.random.rand(1000)
y = np.random.rand(1000)
plt.scatter(x, y)
plt.show()
```



Styling

```
# Matplotlib allows to choose custom colours for plots
y = np.arange(1, 3)
plt.plot(y, 'y') # Specifying line colours
plt.plot(y+5, 'm')
plt.plot(y+10, 'c')
plt.show()
```

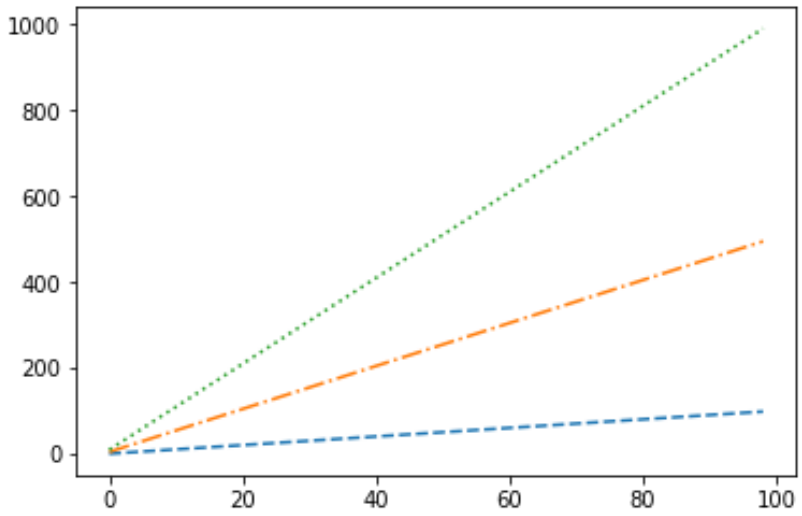


Color code:

- b = Blue
- c = Cyan
- g = Green
- k = Black
- m = Magenta
- r = Red
- w = White
- y = Yellow

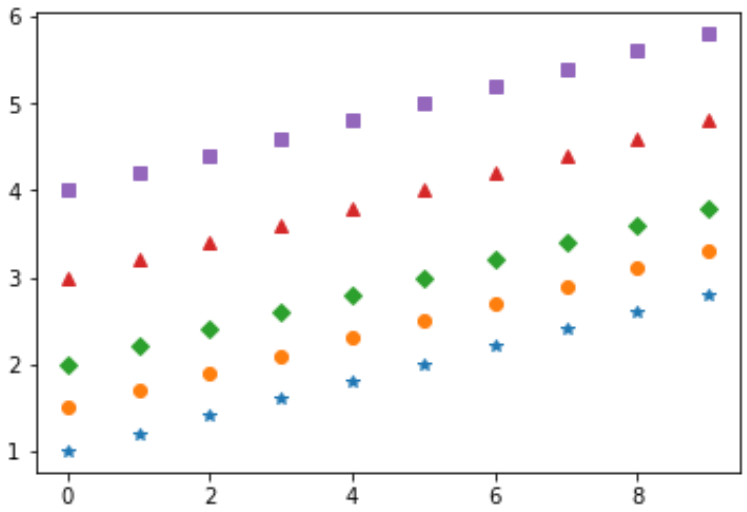
 # Matplotlib allows different line styles for plots

```
y = np.arange(1, 100)
plt.plot(y, '--', y*5, '-.', y*10, ':')
plt.show()
# - Solid line
# -- Dashed line
# -. Dash-Dot line
# : Dotted Line
```



 # Matplotlib provides customization options for markers

```
y = np.arange(1, 3, 0.2)
plt.plot(y, '*',
         y+0.5, 'o',
         y+1, 'D',
         y+2, '^',
         y+3, 's') # Specifying line styling
plt.show()
```



Heatmap

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. Heatmap is also defined by the name of the shading matrix. Heatmaps in Seaborn can be plotted by using the seaborn.heatmap() function.

```
seaborn.heatmap()
```

Syntax:

```
seaborn.heatmap(data, *, vmin=None, vmax=None, cmap=None, center=None,
annot_kws=None, linewidths=0, linecolor='white', cbar=True, **kwargs)
```

Important Parameters:

- ***data:*** 2D dataset that can be coerced into an ndarray.
- ***vmin,vmax:*** Values to anchor the colormap, otherwise they are inferred from the data and other keyword arguments.
- ***cmap:*** The mapping from data values to color space.
- ***center:*** The value at which to center the colormap when plotting divergent data.
- ***annot:*** If True, write the data value in each cell.
- ***fmt:*** String formatting code to use when adding annotations.
- ***linewidths:*** Width of the lines that will divide each cell.
- ***linecolor:*** Color of the lines that will divide each cell.
- ***cbar:*** Whether to draw a colorbar.

All the parameters except data are optional.

Scikit-learn

scikit-learn is an open-source Python library that implements a range of machine learning, pre-processing, cross-validation, and visualization algorithms using a unified interface.

Important features of scikit-learn:

- Simple and efficient tools for data mining and data analysis. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
- Accessible to everybody and reusable in various contexts.
- Built on the top of NumPy, SciPy, and matplotlib.
- Open source, commercially usable – BSD license

.

Step 1: Load a dataset

A dataset is nothing but a collection of data. A dataset generally has two main components:

- **Features**: (also known as predictors, inputs, or attributes) they are simply the variables of our data. They can be more than one and hence represented by a **feature matrix** ('X' is a common notation to represent feature matrix). A list of all the feature names is termed **feature names**.
- **Response**: (also known as the target, label, or output) This is the output variable depending on the feature variables. We generally have a single response column and it is represented by a **response vector** ('y' is a common notation to represent response vector). All the possible values taken by a response vector are termed **target names**.

Loading exemplar dataset:
scikit-learn comes loaded with a few example datasets like the iris and digits datasets for classification and the boston house prices dataset for regression.

Step 2: Splitting the dataset
One important aspect of all machine learning models is to determine their accuracy. Now, in order to determine their accuracy, one can train the model using the given dataset and then predict the response values for the same dataset using that model and hence, find the accuracy of the model.
But this method has several flaws in it, like:

- The goal is to estimate the likely performance of a model on **out-of-sample** data.
- Maximizing training accuracy rewards overly complex models that won't necessarily generalize our model.
- Unnecessarily complex models may over-fit the training data.

A better option is to split our data into two parts: the first one for training our machine learning model, and the second one for testing our model.

Step 3: Training the model

Now, it’s time to train some prediction models using our dataset. Scikit-learn provides a wide range of machine learning algorithms that have a unified/consistent interface for fitting, predicting accuracy, etc.

Q.1 Create a result array by adding the following two NumPy arrays. Next, modify the result array by calculating the square of each element

Code:

```
import numpy as np
```

```
a=np.array ([[1,2,3], [4,5,6]])
b=np.array([[6,7,8], [9,10,11]])
print("First array: ")
print(a,"\n")
print("Second array: ")
print(b)
result = a+b
print("Summation array: ")
print(result, "\n")
print("Sqaure array: ")
for x in np.nditer(result, op_flags = ['readwrite']):
    x[...] = x*x
print(result)
```

Output:

```
First array:
[[1 2 3]
 [4 5 6]]

Second array:
[[ 6  7  8]
 [ 9 10 11]]
Summation array:
[[ 7  9 11]
 [13 15 17]]

Sqaure array:
[[ 49  81 121]
 [169 225 289]]
```

Q.2 Import Iris dataset and solve the following:

a. From the given dataset print the first and last five rows
b. to check for null values
c. count of null values in each column
d. Fill in the null values with zero, last valid observation and next valid observations along axes or with values.
e. Replace null values with other value and drop null values
f. Sort any one column in descending order
g. Find the most expensive data
h. Concatenate two data frames

Iris Dataset link: https://www.kaggle.com/datasets/uciml/iris

Code:

```
import pandas as pd
```

```
df=pd.read_csv("IRIS.csv")
```

```
print("First five rows: ")
df.head()
```

```
First five rows:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
print("\nLast five rows: ")
df.tail()
```

```
Last five rows:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

```
print("\nChecking for null values:")
df.isnull().any()
```

```
Checking for null values:
```

	0
Id	False
SepalLengthCm	False
SepalWidthCm	False
PetalLengthCm	False
PetalWidthCm	False
Species	False

dtype: bool

```
print("\nNull value count per column:")
df.isnull().sum()
```

```
Null value count per column:
```

	0
Id	0
SepalLengthCm	0
SepalWidthCm	0
PetalLengthCm	0
PetalWidthCm	0
Species	0

dtype: int64

```
df.sort_values(by=['SepalLengthCm'], ascending=False)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
131	132	7.9	3.8	6.4	2.0	Iris-virginica
135	136	7.7	3.0	6.1	2.3	Iris-virginica
122	123	7.7	2.8	6.7	2.0	Iris-virginica
117	118	7.7	3.8	6.7	2.2	Iris-virginica
118	119	7.7	2.6	6.9	2.3	Iris-virginica
...	...	...	...	...	...	...
41	42	4.5	2.3	1.3	0.3	Iris-setosa
42	43	4.4	3.2	1.3	0.2	Iris-setosa
38	39	4.4	3.0	1.3	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
13	14	4.3	3.0	1.1	0.1	Iris-setosa

150 rows × 6 columns

```
df[df['SepalLengthCm']==df['SepalLengthCm'].max()]
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
131	132	7.9	3.8	6.4	2.0	Iris-virginica

Complete Code:

```
import pandas as pd

# Load dataset as a DataFrame
df=pd.read_csv("IRIS.csv")

# Print first 5 rows
print("First five rows: ")
df.head()
# Print last 5 rows
print("\nLast five rows: ")
df.tail()

# Check null values in dataset
print("\nChecking for null values:")
df.isnull().any()

# Count of Null in each column
print("\nNull value count per column:")
df.isnull().sum()

# Sort one column in descending order
df.sort_values(by=['SepalLengthCm'], ascending=False)

# Finding most expensive data
df[df['SepalLengthCm']==df['SepalLengthCm'].max()]
```

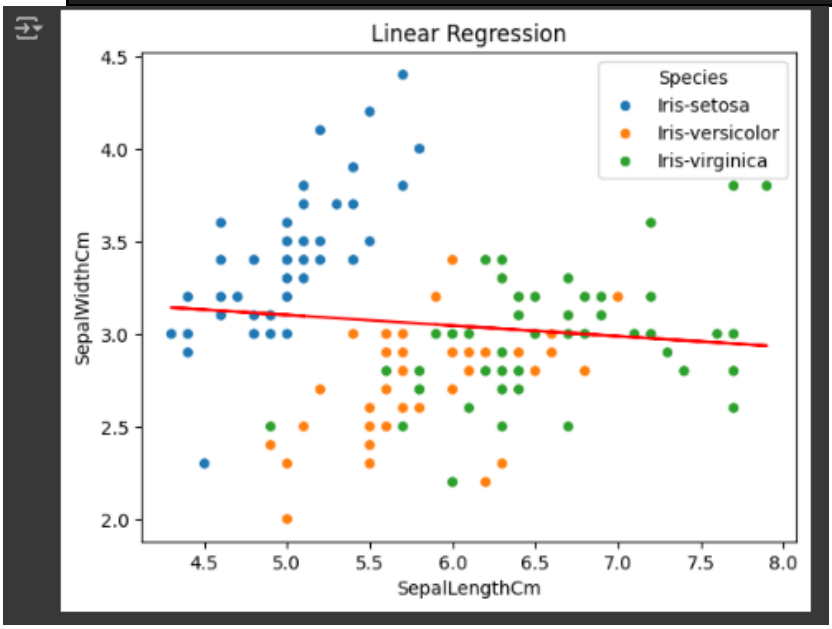
Q.3 Visualize the Iris data using matplotlib library for the following:

a. Linear regression
b. Heatmap
c. Scatter plot
d. Histogram

Code:

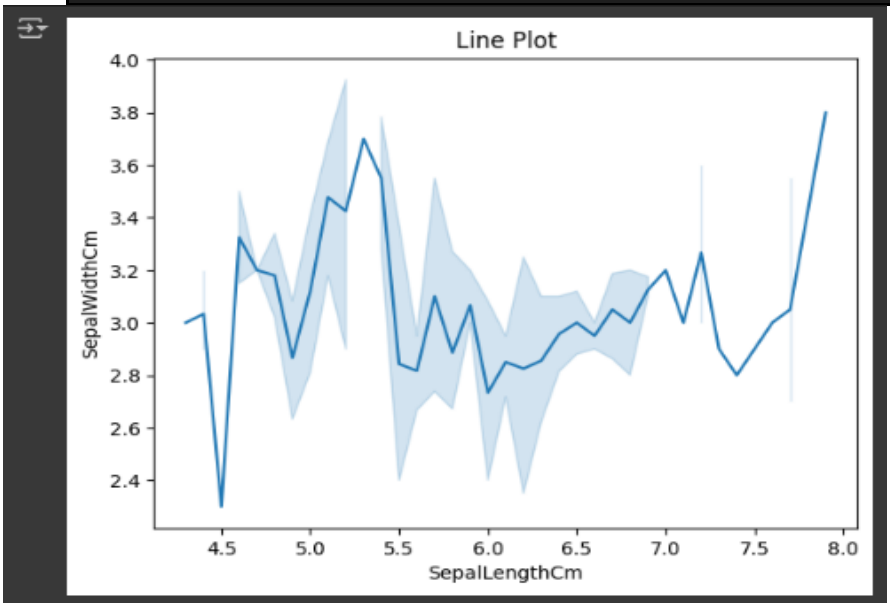
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

```
# Assuming 'SepalLengthCm' is the independent variable and
'SepalWidthCm' is the dependent variable
X = df['SepalLengthCm'].values.reshape(-1, 1)
y = df['SepalWidthCm']

model = LinearRegression()
model.fit(X, y)

# Create the scatter plot with hue for species
sns.scatterplot(x='SepalLengthCm', y='SepalWidthCm',
hue='Species', data=df)

# Plot the regression line
plt.plot(X, model.predict(X), color='red')

plt.title('Linear Regression')
plt.xlabel('SepalLengthCm')
plt.ylabel('SepalWidthCm')
plt.show()
```

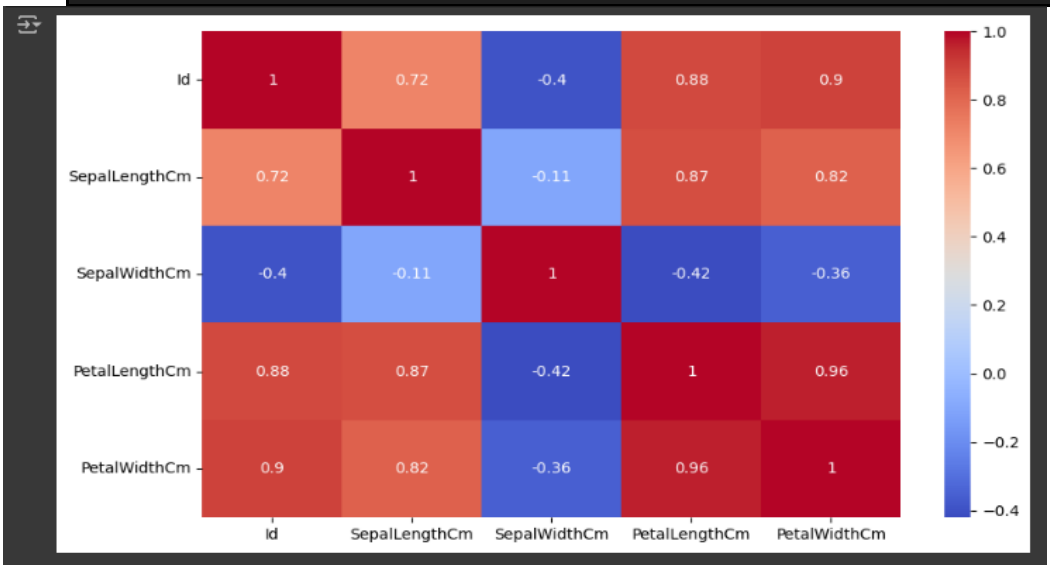


- Chaitanya Shah

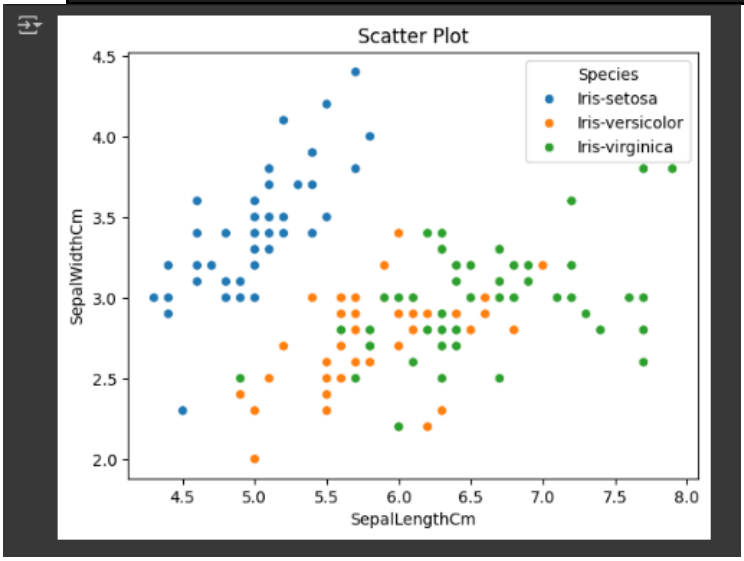
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.lineplot(x='SepalLengthCm', y='SepalWidthCm', data=df)
plt.title('Line Plot')
plt.xlabel('SepalLengthCm')
plt.ylabel('SepalWidthCm')
plt.show()
```



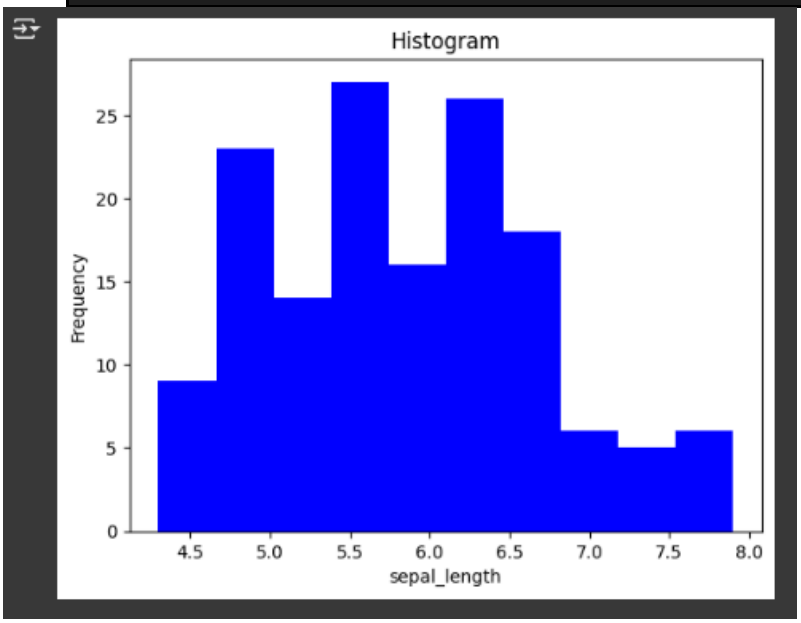

```
numeric_df = df.drop('Species', axis=1)
plt.figure(figsize=(10,6))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.show()
```



```
sns.scatterplot(x='SepalLengthCm', y='SepalWidthCm',
data=df, color='green', hue='Species')
plt.title('Scatter Plot')
plt.xlabel('SepalLengthCm')
plt.ylabel('SepalWidthCm')
plt.show()
```




```
plt.hist(df['SepalLengthCm'], bins=10, color='blue')
plt.title('Histogram')
plt.xlabel('sepal_length')
plt.ylabel('Frequency')
plt.show()
```



Complete Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

                # ---- Linear  Regression ----
# Assuming 'SepalLengthCm' is the independent variable and
'SepalWidthCm' is the dependent variable
X = df['SepalLengthCm'].values.reshape(-1, 1)
y = df['SepalWidthCm']

model = LinearRegression()
model.fit(X, y)

# Create the scatter plot with hue for species
sns.scatterplot(x='SepalLengthCm', y='SepalWidthCm',
hue='Species', data=df)

# Plot the regression line
plt.plot(X, model.predict(X), color='red')

plt.title('Linear Regression')
plt.xlabel('SepalLengthCm')
plt.ylabel('SepalWidthCm')
plt.show()


import seaborn as sns
import matplotlib.pyplot as plt

                # ---- Line Plot ----
sns.lineplot(x='SepalLengthCm', y='SepalWidthCm', data=df)
plt.title('Line Plot')
plt.xlabel('SepalLengthCm')
plt.ylabel('SepalWidthCm')
plt.show()


                # ---- Heatmap ----
numeric_df = df.drop('Species', axis=1)
plt.figure(figsize=(10,6))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.show()
```

```
                # ---- Scatter Plot ----
sns.scatterplot(x='SepalLengthCm', y='SepalWidthCm', data=df,
color='green', hue='Species')
plt.title('Scatter Plot')
plt.xlabel('SepalLengthCm')
plt.ylabel('SepalWidthCm')
plt.show()


                # ---- Histogram ----
plt.hist(df['SepalLengthCm'], bins=10, color='blue')
plt.title('Histogram')
plt.xlabel('sepal_length')
plt.ylabel('Frequency')
plt.show()
```

Conclusion: Thus studied different machine learning packages numpy, pandas and matplotlib