1) Use Numpy to create single and multi-dimensional array and perform various operations using Python

```
import numpy as np

# Creating Arrays
single_dim = np.array([1, 2, 3, 4, 5])
multi_dim = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(single_dim);
print(multi_dim);

# Basic Operations
shape_single = single_dim_array.shape
shape_multi = multi_dim_array.shape
size_single = single_dim_array.size
size_multi = multi_dim_array.size
print(shape_single);
print(shape_multi);
print(size_single);
print(size_multi);

slice_array = single_dim_array[1:4]
indexed_element = multi_dim_array[1, 2]
print(slice_array);
print(indexed_element);
```

OutPut:

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

2) Use Pandas to access dataset, cleaning, manipulate data and analyze using Python

```
# Import pandas package
import pandas as pd

# Load data from CSV file into a DataFrame
data = pd.read_csv("nba.csv", index_col="Name")
print("Initial Data:\n", data.head())

data_cleaned = data.dropna()

data_cleaned = data_cleaned.drop_duplicates()

filtered_data = data_cleaned[data_cleaned['Age'] > 25]

# Print the cleaned and manipulated DataFrame
print("\nCleaned and Manipulated Data:\n", filtered_data)
```

output:

3) Use matplot library to plot graph for data visualization using Python.

pip install matplotlib – Library which support the matplotlib.

Commented [D1]:

```
import matplotlib.pyplot as plt
```

 # initializing the data

x = [10, 20, 30, 40]

y = [20, 25, 35, 55]

 # plotting the data

plt.plot(x, y)

 # Adding title to the plot

plt.title("Linear graph")

plt.xlabel("X-axis")

plt.ylabel("Y-axis")

plt.show()

output:

4) Determine probability, sampling and sampling distribution using Python

import numpy as np

import matplotlib.pyplot as plt

1. Probability

Probability of drawing an Ace from a standard deck of 52 cards

total_cards = 52

aces = 4

prob_ace = aces / total_cards

```python
print(f"Probability of drawing an Ace: {prob_ace}")

# 2. Sampling

# Population of 1000 individuals
population = np.arange(1, 1001)

# Random sample of 50 individuals
sample_size = 50
sample = np.random.choice(population, sample_size, replace=False)
print(f"Random sample of 50 individuals: {sample}")

# 3. Sampling Distribution

# Function to calculate the sample mean
def sample_mean(population, sample_size, num_samples):
    means = []
    for _ in range(num_samples):
        sample = np.random.choice(population, sample_size, replace=False)
        means.append(np.mean(sample))
    return means

# Parameters
num_samples = 1000
sample_means = sample_mean(population, sample_size, num_samples)

# Plot the sampling distribution of the sample mean
```

```
plt.hist(sample_means, bins=30, edgecolor='k')
plt.title('Sampling Distribution of the Sample Mean')
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')
plt.show()
```

OutPut:

```
Probability of drawing an Ace: 0.07692307692307693
Random sample of 50 individuals: [550 321 385 843 46 856 230 563 68 784 65
2 653 142 71 379 954 867 857
 840 17 782 788 56 730 370 895 48 570 967 196 105 147 769 441 847 679
 579 884 821 220 271 425 323 773 143 921 131 410 956 447]
```

Also have graph output.

5**) Determine frequency distributions, variability, average, and standard de viation usin g Python.**

- ➔ The basic formula for the average of n numbers x1, x2, ……xn is

  A = (x_1 + x_2 .... + x_n)/ n
- ➔ One can calculate the average by using **numpy.average()** function in python.
- ➔ The mathematical formula for variance is as follows,
- ➔ $Formula : \sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}$
- ➔ One can calculate the variance by using **numpy.var()** function in python.
- ➔ The mathematical formula for calculating standard deviation is as follows,
- ➔ $Standard Deviation = \sqrt{variance}$
- ➔ One can calculate the standard deviation by using **numpy.std()** function in python.

  # Python program to get average of a list

```
# Importing the NumPy module
import numpy as np
list = [2, 4, 4, 4, 5, 5, 7, 9]
print(np.average(list))
# Python program to get variance of a list
# Importing the NumPy module
import numpy as np
list = [2, 4, 4, 4, 5, 5, 7, 9]
print(np.var(list))
# Python program to get
# standard deviation of a list
# Importing the NumPy module
import numpy as np
list = [2, 4, 4, 4, 5, 5, 7, 9]
print(np.std(list))
```

6) Draw normal curves, correlation, correlation coefficient and scatter plots using Python.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Part 1: Drawing Normal Curves
mean = 0
std_dev = 1
```

```
x = np.linspace(mean - 4*std_dev, mean + 4*std_dev, 1000)
y = norm.pdf(x, mean, std_dev)

# Plot the normal curve
plt.plot(x, y, label='Normal Distribution')
plt.title('Normal Curve')
plt.xlabel('Value')
plt.ylabel('Probability Density')
plt.legend()
plt.show()

# Part 2: Calculating Correlation and Correlation Coefficient
np.random.seed(0)
x = np.random.randn(100)
y = 2 * x + np.random.randn(100) * 0.5

correlation_coefficient = np.corrcoef(x, y)[0, 1]
print(f"Correlation Coefficient: {correlation_coefficient}")

# Part 3: Creating Scatter Plots
plt.scatter(x, y, alpha=0.5)
plt.title('Scatter Plot of X and Y')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

7) Implement and analyze Linear regression in Python (Single variable & Multivariable)

```
# single variable.
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Step 1: Generate synthetic data
np.random.seed(0)
house_size = 2 * np.random.rand(100, 1) + 3  # house sizes between 3 and 5 (1000 sq ft)
house_price = 4 + 3 * house_size + np.random.randn(100, 1)  # linear relation with some noise

# Visualize the data
plt.scatter(house_size, house_price)
plt.xlabel("House Size (1000 sq ft)")
plt.ylabel("House Price ($1000)")
plt.title("House Size vs House Price")
plt.show()

# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(house_size, house_price, test_size=0.2, random_state=0)

# Step 3: Train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
# Step 4: Make predictions
y_pred = model.predict(X_test)

# Step 5: Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Single Variable Linear Regression")
print(f"Intercept: {model.intercept_[0]}")
print(f"Coefficient: {model.coef_[0][0]}")
print(f"Mean Squared Error: {mse}")

# Visualize the regression line
plt.scatter(X_test, y_test, color='black', label='Actual data')
plt.plot(X_test, y_pred, color='blue', linewidth=3, label='Regression line')
plt.xlabel("House Size (1000 sq ft)")
plt.ylabel("House Price ($1000)")
plt.title("House Size vs House Price (Test Data)")
plt.show()
```

output: Graph

```
#multi variable linear regression.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import mean_squared_error

# Step 1: Generate synthetic data
np.random.seed(0)
house_size = 2 * np.random.rand(100, 1) + 3  # house sizes between 3 and 5 (1000 sq ft)
num_bedrooms = np.random.randint(1, 6, size=(100, 1))  # number of bedrooms between 1 and 5
house_age = np.random.randint(0, 30, size=(100, 1))  # house age between 0 and 30 years
house_price = 4 + 3 * house_size + 1.5 * num_bedrooms - 0.5 * house_age + np.random.randn(100, 1)  # linear relation with some noise

# Creating a DataFrame
data = np.concatenate((house_size, num_bedrooms, house_age, house_price), axis=1)
columns = ['Size', 'Bedrooms', 'Age', 'Price']
df = pd.DataFrame(data, columns=columns)

# Step 2: Split the data into training and testing sets
X = df[['Size', 'Bedrooms', 'Age']]
y = df['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Step 3: Train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
# Step 4: Make predictions
y_pred = model.predict(X_test)

# Step 5: Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Multivariable Linear Regression")
print(f"Intercept: {model.intercept_}")
print(f"Coefficients: {model.coef_}")
print(f"Mean Squared Error: {mse}")

# Visualize the actual vs predicted prices
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Prices")
plt.show()
```

8) Implement and analyze Logistic regression in Python.

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,
confusion_matrix

# Load the dataset
iris = load_iris()
```

```
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target

# Filter to include only classes 0 and 1 for binary classification
df = df[df['target'] != 2]

# Separate features and target
X = df.drop('target', axis=1)
y = df['target']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)

print(f'Accuracy: {accuracy}')
```

```
print(f'Precision: {precision}')

print(f'Recall: {recall}')

print(f'Confusion Matrix:\n{confusion}')
```

output:

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
Confusion Matrix:
[[12  0]
 [ 0  8]]
```

9) Implement and analyze Decision tree algorithm in Python

```
 import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, conf
usion_matrix

# Load the dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target

# Separate features and target
X = df.drop('target', axis=1)
y = df['target']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
ate=42)

# Train the Decision Tree model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
```

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
confusion = confusion_matrix(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'Confusion Matrix:\n{confusion}')
```

output:

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

10) <u>Implement and analyze Random Forest algorithm in Python</u>

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix


# Load the dataset

iris = load_iris()

df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

df['target'] = iris.target


# Separate features and target

X = df.drop('target', axis=1)
```

```
y = df['target']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train the Random Forest model
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
confusion = confusion_matrix(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'Confusion Matrix:\n{confusion}')
```

output:

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
Confusion Matrix:
```

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```