Bala Vidya Mandir Sr.Sec. School

Class: XII- ARTIFICIAL INTELLIGENCE

List of practical programs - 2025-20

I. PYTHON PROGRAMS

1. **a) Create a Python program to generate a 2D array using NumPy and access its elements using forward indexing, backward indexing, and implement slicing.**

   Title: 2D NumPy Array – Indexing & Slicing

   Aim: To write a Python program that generates a 2D NumPy array and accesses its elements using forward indexing, backward indexing, and slicing.

   Algorithm:

   1. Import the numpy library.
   2. Create a 2D NumPy array using np.array() or np.arange().reshape().
   3. Access elements using forward indexing (positive indices).
   4. Access elements using backward indexing (negative indices).
   5. Access subsets of the array using slicing (array[start:end, start:end]).
   6. Print all outputs.

Program:

```
# Step 1: Create a 2D Array
import numpy as np
arr = np.array([[10, 20, 30],
               [40, 50, 60],
               [70, 80, 90]])
print("Original 2D Array:\n", arr)

# Forward Indexing

print("Element at [0][0]:", arr[0][0])

print("Element at [2][2]:", arr[2][2])

# Backward Indexing

print("Element at [-1][-1]:", arr[-1][-1])

print("Element at [-3][-3]:", arr[-3][-3])

# Slicing

print("First two rows and first two columns:\n", arr[:2, :2])

print("Last two rows and last column:\n", arr[-2:, -1])

print("All elements in second column:\n", arr[:, 1])
```

Result:

```
Original 2D Array:
 [[10 20 30]
 [40 50 60]
 [70 80 90]]
Element at [0][0]: 10
Element at [2][2]: 90
Element at [-1][-1]: 90
Element at [-3][-3]: 10
First two rows and first two columns:
 [[10 20]
 [40 50]]
Last two rows and last column:
 [60 90]
All elements in second column:
 [20 50 80]
```

1. **(b) Create a Python program to perform element-wise addition, subtraction, multiplication, and division on two 2D NumPy arrays of the same shape.**

Title: 2D NumPy Array – Indexing & Slicing

Aim: **To perform element-wise addition, subtraction, multiplication, and division on two 2D NumPy arrays of the same shape.**

Algorithm:

1. **Import the numpy library.**
2. **Create two 2D NumPy arrays of the same shape.**
3. **Perform element-wise addition using +.**
4. **Perform element-wise subtraction using -.**
5. **Perform element-wise multiplication using *.**
6. **Perform element-wise division using /.**
7. **Print results of all operations.**

1(b).Program:

```
import numpy as np

# Create two 2D arrays

arr1 = np.array([[10, 20, 30],

                 [40, 50, 60]])

arr2 = np.array([[5, 4, 3],

                 [2, 1, 6]])


# Arithmetic operations

print("Addition:\n", arr1 + arr2)

print("Subtraction:\n", arr1 - arr2)

print("Multiplication:\n", arr1 * arr2)

print("Division:\n", arr1 / arr2)
```

Sample Output:

```
Addition:

 [[15 24 33]

  [42 51 66]]

Subtraction:

 [[ 5 16 27]

  [38 49 54]]

Multiplication:

 [[50 80 90]

  [80 50 360]]

Division:

 [[2.   5.   10.  ]

 [20.  50.   10. ]]
```

2.Create a Python program to create a Pandas Series using a dictionary and access its elements using indexing and slicing.

Aim:

To create a Pandas Series using a dictionary and access its elements using indexing and slicing.

Algorithm:

1. **Import the Pandas library.**
2. **Create a dictionary with key-value pairs.**
3. **Convert it into a Pandas Series using pd.Series().**
4. **Access elements by index.**
5. **Access multiple elements using slicing.**
6. **Print all outputs.**

Program:

```
import pandas as pd

# Create dictionary

data = {'a': 10, 'b': 20, 'c': 30, 'd': 40}

# Create Series

series = pd.Series(data)

print("Original Series:\n", series)

# Accessing elements

print("Element at index 'b':", series['b'])

print("Elements from index 'b' to 'd':\n", series['b':'d'])
```

Sample Output:

```
Original Series:
a    10
b    20
c    30
d    40
dtype: int64
Element at index 'b': 20
Elements from index 'b' to 'd':
b    20
c    30
d    40
dtype: int64
```

3. Create a Python program using DataFrame for "Employee" Details (Emp_ID, Name, Department, Salary) from a Dictionary of Lists and Perform the following operations:

(a) Display the entire DataFrame

(b) Display the first 5 records

(c) Display the last 3 records

Aim:

To create an Employee Details DataFrame and display the entire DataFrame, the first 5 records, and the last 3 records.

Algorithm:

1. **Import the Pandas library.**
2. **Create a dictionary of employee details.**
3. **Convert it to a DataFrame.**
4. **Display the full DataFrame.**
5. **Display the first 5 records using .head(5).**
6. **Display the last 3 records using .tail(3).**

Program:

```
import pandas as pd

# Create employee dictionary

emp_data = {

   'Emp_ID': [101, 102, 103, 104, 105, 106],

   'Name': ['John', 'Alice', 'Bob', 'Sara', 'Mike', 'Emma'],

   'Department': ['HR', 'IT', 'Finance', 'IT', 'HR', 'Finance'],

   'Salary': [50000, 60000, 55000, 62000, 58000, 61000]

}

# Create DataFrame

df = pd.DataFrame(emp_data)

print("Full DataFrame:\n", df)

print("\nFirst 5 Records:\n", df.head(5))

print("\nLast 3 Records:\n", df.tail(3))
```

Sample Output:

```
Full DataFrame:
   Emp_ID   Name Department  Salary
0     101   John         HR   50000
1     102  Alice         IT   60000
2     103    Bob    Finance   55000
3     104   Sara         IT   62000
4     105   Mike         HR   58000
5     106   Emma    Finance   61000
First 5 Records:
   Emp_ID   Name Department  Salary
0     101   John         HR   50000
1     102  Alice         IT   60000
2     103    Bob    Finance   55000
3     104   Sara         IT   62000
4     105   Mike         HR   58000
Last 3 Records:
   Emp_ID  Name Department  Salary
3     104  Sara         IT   62000
4     105  Mike         HR   58000
5     106  Emma    Finance   61000
```

4. Create a Python program using DataFrame from a Series for storing Health Parameters (RBC, WBC, Platelets, Hemoglobin) Patient-Wise and Access Rows/Columns.

Aim:

To create a Health Parameters DataFrame from Pandas Series for each parameter and access rows and columns.

Algorithm:

1. Import the Pandas library.
2. Create individual Pandas Series for RBC, WBC, Platelets, and Hemoglobin with patient names as the index.
3. Combine these Series into a dictionary.
4. Create a DataFrame from this dictionary.
5. Access a specific row using .loc[].
6. Access a specific column using df['ColumnName'].
7. Print the results.

Program:

```
import pandas as pd

# Creating Series for each parameter

rbc = pd.Series([4.5, 4.8, 4.3], index=['Patient1', 'Patient2', 'Patient3'])

wbc = pd.Series([6000, 7500, 7200], index=['Patient1', 'Patient2', 'Patient3'])

platelets = pd.Series([250000, 270000, 260000], index=['Patient1', 'Patient2', 'Patient3'])

hemoglobin = pd.Series([13.5, 14.2, 13.0], index=['Patient1', 'Patient2', 'Patient3'])

# Creating DataFrame

health_df = pd.DataFrame({

    'RBC': rbc,

    'WBC': wbc,

    'Platelets': platelets,

    'Hemoglobin': hemoglobin

})

print("Health Parameters DataFrame:\n", health_df)

# Accessing specific row

print("\nDetails of Patient2:\n", health_df.loc['Patient2'])

# Accessing specific column

print("\nRBC Values:\n", health_df['RBC'])
```

Sample Output:

```
Health Parameters DataFrame:
          RBC   WBC  Platelets  Hemoglobin
Patient1  4.5  6000     250000        13.5
Patient2  4.8  7500     270000        14.2
Patient3  4.3  7200     260000        13.0
Details of Patient2:
RBC           4.8
WBC        7500.0
Platelets 270000.0
Hemoglobin   14.2
Name: Patient2, dtype: float64
RBC Values:
Patient1    4.5
Patient2    4.8
Patient3    4.3
Name: RBC, dtype: float64
```

5. Create a Python program using DataFrame for Plant Growth Profile (Light Intensity, CO_2 Level, Water Volume, Growth Rate) from a Dictionary of Lists and Perform Operations to Add a New Column, add a New Row, and Delete a Row.

Aim: **To create a Plant Growth Profile DataFrame and perform operations to add a new column, add a new row, and delete a row.**

Algorithm:

1. **Import the Pandas library.**
2. **Create a dictionary with plant growth parameters (Light Intensity, CO_2 Level, Water Volume, Growth Rate).**
3. **Convert the dictionary into a DataFrame.**
4. **Add a new column with given values.**
5. **Add a new row using .loc[] or pd.concat().**
6. **Delete a row using .drop().**
7. **Display the DataFrame after each change.**

Program:

```
import pandas as pd

# Creating DataFrame

plant_data = {

   'Light Intensity': [200, 250, 300],

   'CO2 Level': [400, 420, 450],

   'Water Volume': [1.5, 1.8, 2.0],

   'Growth Rate': [5.2, 5.5, 6.0]

}

df_plant = pd.DataFrame(plant_data, index=['Day1', 'Day2', 'Day3'])

print("Original DataFrame:\n", df_plant)

# Adding new column

df_plant['Temperature'] = [25, 26, 27]

print("\nAfter Adding New Column:\n", df_plant)

# Adding new row

df_plant.loc['Day4'] = [320, 460, 2.2, 6.3, 28]

print("\nAfter Adding New Row:\n", df_plant)

# Deleting a row

df_plant = df_plant.drop('Day2')

print("\nAfter Deleting Row Day2:\n", df_plant)
```

Sample Output:

```
Original DataFrame:

     Light Intensity  CO2 Level  Water Volume  Growth Rate

Day1             200        400           1.5          5.2

Day2             250        420           1.8          5.5

Day3             300        450           2.0          6.0


After Adding New Column:

     Light Intensity  CO2 Level  Water Volume  Growth Rate  Temperature

Day1             200        400           1.5          5.2           25

Day2             250        420           1.8          5.5           26

Day3             300        450           2.0          6.0           27


After Adding New Row:

     Light Intensity  CO2 Level  Water Volume  Growth Rate  Temperature

Day1             200        400           1.5          5.2           25

Day2             250        420           1.8          5.5           26

Day3             300        450           2.0          6.0           27

Day4             320        460           2.2          6.3           28


After Deleting Row Day2:

     Light Intensity  CO2 Level  Water Volume  Growth Rate  Temperature

Day1             200        400           1.5          5.2           25

Day3             300        450           2.0          6.0           27

Day4             320        460           2.2          6.3           28
```

7. **Create a Python program to display the attributes of the DataFrame.**

Aim:

To display the attributes of a DataFrame such as shape, size, column names, and index.

Algorithm:

1. **Import the Pandas library.**
2. **Create a dictionary of data.**
3. **Convert it into a DataFrame.**
4. **Display the shape using .shape.**
5. **Display the size using .size.**
6. **Display the columns using .columns.**
7. **Display the index using .index.**

Program:

```
import pandas as pd

# Create sample DataFrame

data = {

   'Name': ['John', 'Alice', 'Bob'],

   'Age': [25, 30, 28],

   'City': ['Delhi', 'Mumbai', 'Chennai']

}

df = pd.DataFrame(data)

# Display attributes

print("Shape of DataFrame:", df.shape)

print("Size of DataFrame:", df.size)

print("Column Names:", df.columns)

print("Row Index:", df.index)
```

Sample Output:

```
Shape of DataFrame: (3, 3)

Size of DataFrame: 9

Column Names: Index(['Name', 'Age', 'City'], dtype='object')

Row Index: RangeIndex(start=0, stop=3, step=1)
```

7. Create a Python Program to Store and Read Employee Details Using a CSV File.

Aim:

To write a Python program to store employee details in a CSV file and read the contents of the file.

Algorithm:

1. **Import the csv module.**
2. **Create a list of lists containing employee details (Emp_ID, Name, Department, Salary).**
3. **Open a CSV file in write mode.**
4. **Create a CSV writer object using csv.writer().**
5. **Write the header row (column names) to the CSV file.**
6. **Write all employee records to the CSV file.**
7. **Close the file.**
8. **Open the CSV file in read mode.**
9. **Create a CSV reader object using csv.reader().**
10. **Read and display each record from the file.**

Program:

```
import csv

# Step 1: Store employee details in a CSV file

employee_data = [

    ["Emp_ID", "Name", "Department", "Salary"],

    [101, "John", "HR", 50000],

    [102, "Alice", "IT", 60000],

    [103, "Bob", "Finance", 55000]

]

# Writing to CSV

with open("employees.csv", "w", newline="") as file:

    writer = csv.writer(file)

    writer.writerows(employee_data)

print("Employee details stored in 'employees.csv'.")

# Step 2: Read employee details from the CSV file

print("\nReading data from 'employees.csv':")

with open("employees.csv", "r") as file:

    reader = csv.reader(file)

    for row in reader:

        print(row)
```

Sample Output:

```
Employee details stored in 'employees.csv'.


Reading data from 'employees.csv':
['Emp_ID', 'Name', 'Department', 'Salary']
['101', 'John', 'HR', '50000']
['102', 'Alice', 'IT', '60000']
['103', 'Bob', 'Finance', '55000']
```

Bala Vidya Mandir Sr. Sec. School

Class XII- Artificial Intelligence

List of practical programs 2025-26

Instructions for writing the AI Record Notebook

1. Refer to the posted document and follow the instructions given in class.
2. On the right side of the record notebook, neatly write:
   - Aim
   - Dataset
   - Procedure
   - Result
3. On the left side of the record notebook:
   - Workflow screenshots for each experiment are to be pasted. These screenshots will be provided after the school reopens.

Note: Programs 1 to 6 already written in the record

7. Differentiating Fruits and Vegetables using Orange

Aim:

To classify food items as either fruits or vegetables based on their nutritional characteristics using Orange Data Mining and to evaluate the performance of different classification models.

Dataset:

i. A dataset containing nutritional information of fruits and vegetables.

ii. Features include:

   - Energy (kcal/kJ)
   - Water content (g)
   - Protein (g)
   - Total Fat (g)
   - Carbohydrates (g)
   - Fiber (g)
   - Sugars (g)
   - Calcium (mg)

        - Iron (mg)
        - Magnesium (mg)
        - Phosphorus (mg)
        - Potassium (mg)
        - Sodium (g)

iii. The **target variable**: *Category* (Fruit / Vegetable).

iv. The dataset can be sourced from Kaggle or other nutritional databases.

Procedure:

Step 1: Load Dataset

- Launch **Orange Data Mining**.
- Drag the **File** widget to the canvas and load the dataset (CSV/Excel).
- Ensure the **target column (Fruit/Vegetable)** is marked as "Target."

Step 2: Split Dataset

- Drag the **Data Sampler** or **Data Split** widget.
- Connect it to the **File** widget.
- Split data into **training set (e.g., 70%)** and **testing set (e.g., 30%)**.

Step 3: Train Classification Models

- Add different classification models such as **Logistic Regression**, **Decision Tree**, **Random Forest** and **k-Nearest Neighbours (kNN)**
- Connect each model to the training dataset.

Step 4: Test and Evaluate Models

- Add the **Test & Score** widget.
- Connect both the training data and classifiers to it.
- The widget will calculate metrics like **Accuracy, Precision, Recall, and F1-score**.

Step 5: Interpret Results

- Add the **Confusion Matrix** widget to see correct vs incorrect classifications.
- Compare model performances based on evaluation metrics.
- Identify which algorithm performs best for this dataset.

Result:

- The dataset of fruits and vegetables was successfully classified using Orange.
- **Evaluation metrics obtained:**
    i. Decision Tree: Accuracy = __%, Precision = __%, Recall = __%, F1 = __%
    ii. Random Forest: Accuracy = __%, Precision = __%, Recall = __%, F1 = __%
    iii. Logistic Regression: Accuracy = __%, Precision = __%, Recall = __%, F1 = __%
    iv. kNN: Accuracy = __%, Precision = __%, Recall = __%, F1 = __%
- The **Random Forest classifier** performed best with the highest accuracy and balanced precision/recall.

8. Clustering Images of Birds and Animals using Orange

Aim:

To cluster images of birds and animals into distinct groups based on their visual characteristics using Orange Data Mining.

Dataset:

- A collection of images containing various species of **birds** (e.g., sparrows, pigeons, parrots) and **animals** (e.g., lions, tigers, dogs, cats).
- Each dataset folder should contain a sufficient number of images representing different species.

Procedure:

Step 1: Install Add-On

- Open Orange → go to **Options** → **Add-ons** → install **Image Analytics** → restart Orange.

Step 2: Import Dataset

- Drag and drop the **Import Images** widget.
- Load the dataset folder containing images of birds and animals.

Step 3: Image Visualization

- Connect **Image Viewer** to check if the images are imported correctly.

Step 4: Image Embedding

- Add the **Image Embedding** widget.
- Connect it to **Import Images** to convert images into numerical representations (features).

Step 5: Compute Distances

- Add the **Distance** widget.
- Connect it to **Image Embedding** and select **Cosine** as the distance measure.

Step 6: Clustering

- Add the **Hierarchical Clustering** widget.
- Connect it to **Distance**.
- Open the dendrogram to visualize how images are grouped into clusters.

Step 7: Cluster Analysis

- Select a cluster in the dendrogram.
- Connect **Image Viewer** to the clustering output and observe which images belong to each cluster.

Result:

The images of birds and animals are successfully clustered into distinct groups based on their visual features.

- One cluster contained mostly bird images.
- Another cluster contained mostly animal images.
- Sub-groups were observed within clusters (e.g., sparrows and pigeons grouped together, lions and tigers grouped together).

9. Text Analysis and Word Cloud using Orange

Aim:

To create a text corpus, preprocess it using normalization techniques, and analyze word frequencies using Word Clouds in Orange. Additionally, to compare word clouds before and after applying stemming/lemmatization.

Dataset:

- **Text Used:** *The Tortoise and the Hare* fable.

Text Extract:
Once upon a time, there was a hare who was very proud of how fast he could run. One day, he decided to challenge the tortoise to a race... In the end, the tortoise won the race, proving that slow and steady wins the race.

Procedure:

Step 1: Create Corpus

- Open Orange Data Mining.
- Install the **Text Add-On** from *Options → Add-ons*.
- Drag the **Create Corpus** widget onto the canvas.
- Paste the text of *"The Tortoise and the Hare"* inside the widget.
- Connect it to **Corpus Viewer** to see the raw text.

Step 2: Generate Raw Word Cloud

- Drag the **Word Cloud** widget and connect it to **Create Corpus**.
- Observe the raw word cloud, where frequent words like *"the"*, *"was"*, *"and"* appear larger (stopwords).

Step 3: Preprocess Text

- Add the **Preprocess Text** widget and connect it to **Create Corpus**.
- Apply the following preprocessing techniques:
    i. Convert text to **lowercase**.
    ii. **Tokenize** into words.
    iii. Remove **stopwords** (e.g., the, is, and).
    iv. Remove **punctuation**.

Step 4: Generate Preprocessed Word Cloud

- Connect **Preprocess Text** to **Word Cloud**.
- Observe the cleaned word cloud, where words like *"tortoise"*, *"hare"*, *"race"*, *"slow"*, *"steady"* appear more prominently.

Step 5: Apply Stemming / Lemmatization

- Add **Stem** or **Lemmatize** widget after **Preprocess Text**.
- Connect it to another **Word Cloud** widget.

- Observe that similar words merge (e.g., *“running”*, *“ran”* → *“run”*).

Result:

1. **Raw Word Cloud (Before Preprocessing):**
   - Frequent stopwords (*“the”*, *“and”*, *“was”*) dominate.
   - Words like *“tortoise”* and *“hare”* appear smaller.
2. **Preprocessed Word Cloud (After Stopword Removal):**
   - Words like *“tortoise”*, *“hare”*, *“race”*, *“slow”*, *“steady”* become prominent.
   - The word cloud is cleaner and reflects key ideas of the story.
3. **Word Cloud after Stemming/Lemmatization:**
   - Words with similar roots are merged (*“run”*, *“ran”*, *“running”* → *“run”*).
   - The cloud becomes concise and emphasizes main concepts like *“tortoise”*, *“hare”*, *“race”*, *“run”*, *“steady”*

Using Orange, we successfully preprocessed text, generated word clouds at different stages, and observed how normalization (stopword removal, stemming, lemmatization) improves clarity..

EXPERIMENT – 10: DATA STORYTELLING USING ORANGE DATA MINING TOOL

Aim:

To analyze and visualize the impact of the Mid-Day Meal Scheme on school dropout rates using Orange Data Mining Tool and create a data story based on the findings.

Tools Used:

Orange Data Mining Tool, CSV Dataset (Academic Year vs Dropout Rate)

Procedure:

1. Open the Orange Data Mining Tool.

2. Add a "File" widget and load the dataset (dropout_rate.csv)

3. Connect the File widget to a "Data Table" widget to view the dataset.

4. Connect the File widget to a "Scatter Plot" widget.

5. In the Scatter Plot, set X-axis to Academic Year and Y-axis to Dropout Rate (%).

6. Observe the trend — a downward slope indicates a decrease in dropout rate over time.

7. Add a "Correlations" widget connected to the File widget.

8. View the correlation value between Academic Year and Dropout Rate (%).

9. Record the correlation result and interpret it in context of the Mid-Day Meal Scheme.

Observations:

The Scatter Plot shows a clear decreasing trend of dropout rate from 2010 to 2014. The Correlations widget indicates a strong negative correlation (~ -0.93) between Academic Year and Dropout Rate.

Interpretation:

The analysis reveals that as years progress, dropout rates significantly decrease, reflecting the positive impact of the Mid-Day Meal Scheme. The correlation value suggests that the scheme played a vital role in improving student retention in schools.

Conclusion:

Using Orange Data Mining's File, Data Table, Scatter Plot, and Correlations widgets, we concluded that dropout rates declined from 4.8% to 1.5% between 2010 and 2014. The Mid-Day Meal Scheme contributed significantly to this improvement, promoting education through better attendance and reduced dropouts.

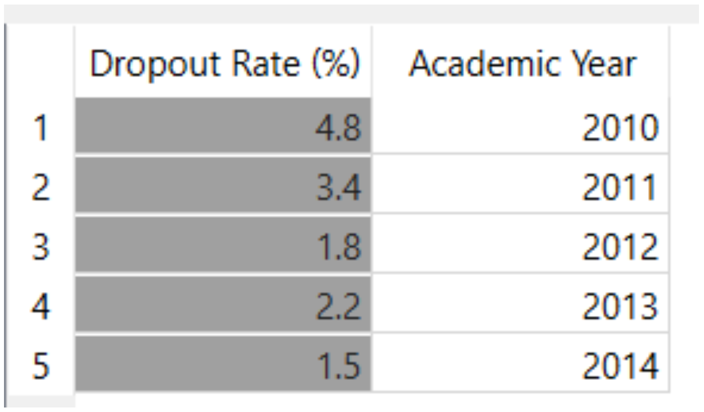
Note:Do not write the content which is in the text color RED

To be done in the left side of your record note book

Take the print of the following picture and paste in your record in the order as mentioned.

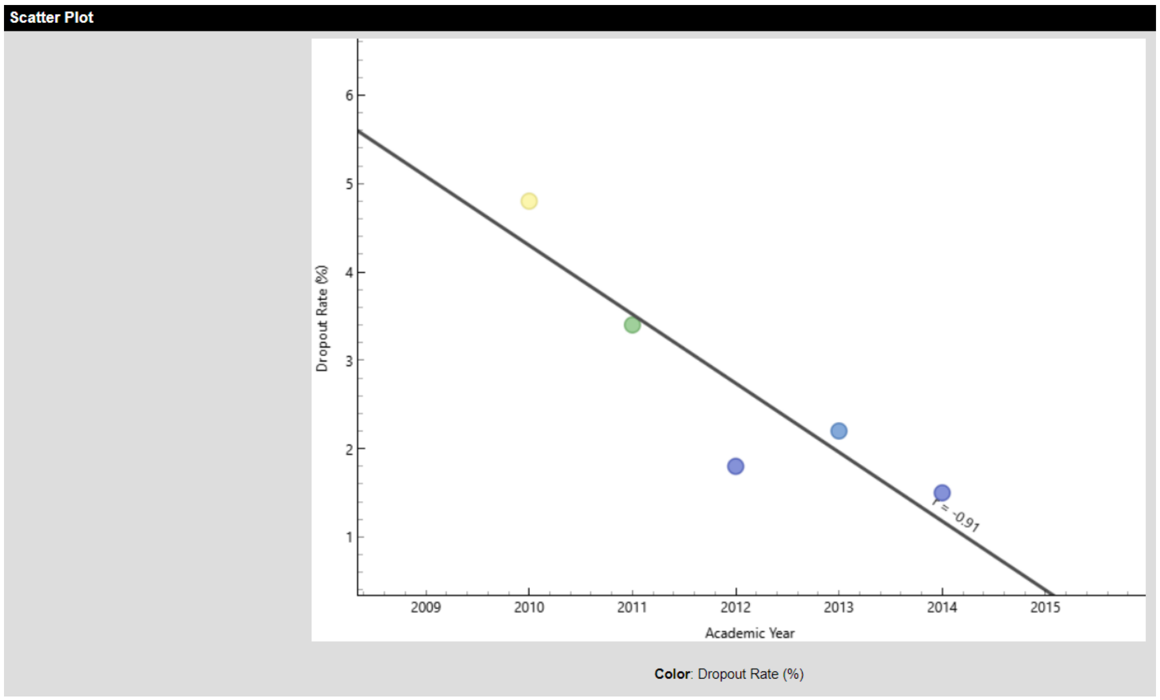
Workflow Diagram:(To be pasted in left side-Blank page)

1.Dataset-Data Table view



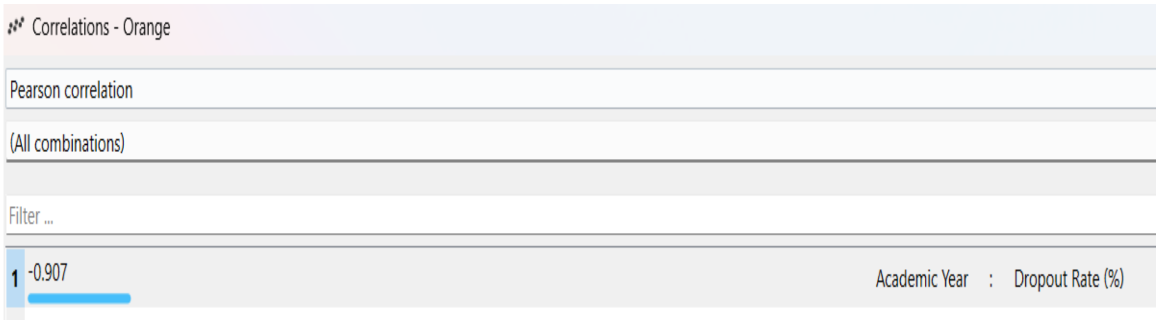
	Dropout Rate (%)	Academic Year
1	4.8	2010
2	3.4	2011
3	1.8	2012
4	2.2	2013
5	1.5	2014

2.Scatter Plot showing Dropout trend



Note 1 & 2 should be in the same page

3.Correlations Widget showing correlation value



Correlations - Orange

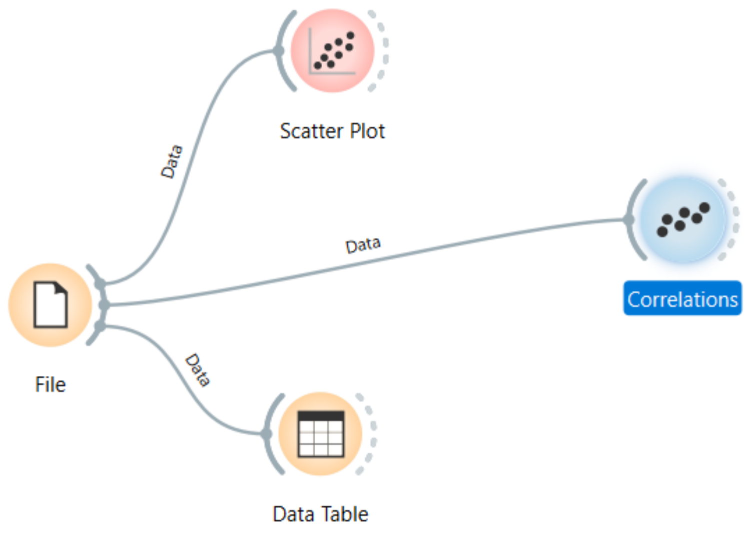
Pearson correlation

(All combinations)

Filter ...

```
1  -0.907                                   Academic Year  :  Dropout Rate (%)
```

4.Orange Workflow (File → Data Table → Scatter Plot → Correlations)



Note:3 and 4 should be in the same page.