Machine Learning

Video 11:

What are Tensors:

What is Tensor?

-It is a data structure

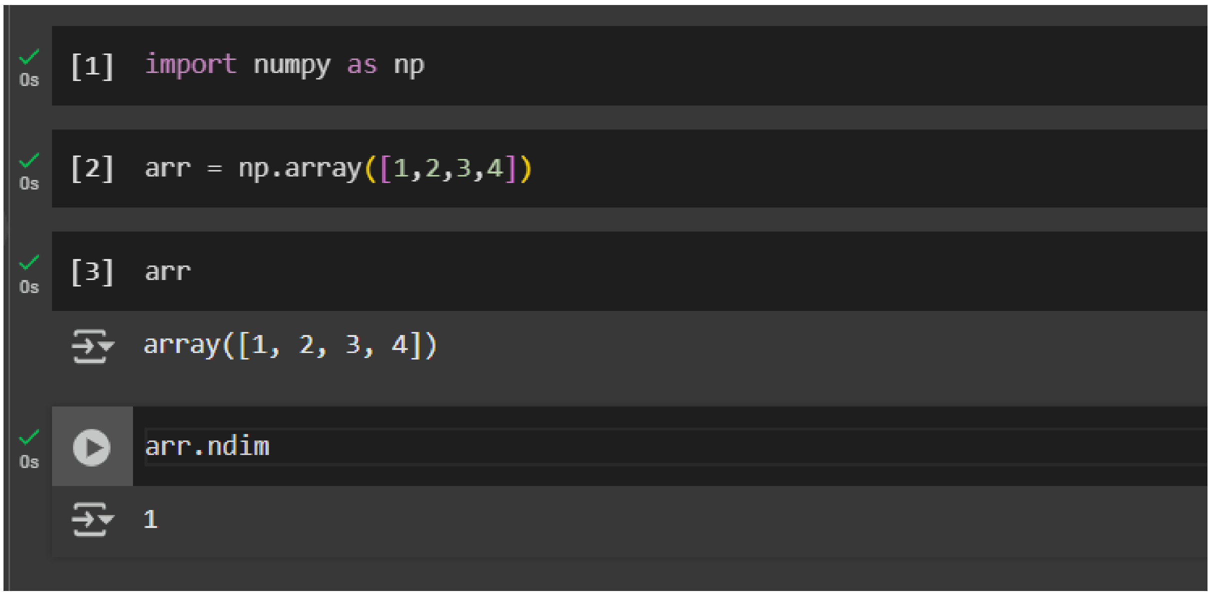
-It is a container for number

In machine learning, tensors are multi-dimensional arrays used to store data like images, text, or numerical values. They enable efficient computation by representing input, weights, and outputs in algorithms like neural networks. Tensors are manipulated using operations like addition, multiplication, and transformations, often using libraries like TensorFlow or PyTorch.

What is 1D Tensor?

A 1D tensor is essentially a one-dimensional array or vector. It consists of a sequence of elements arranged in a single row or column. In machine learning, it can represent data like a list of values, such as a series of numerical inputs or a feature vector.

The below code shows an example of 1D tensor:

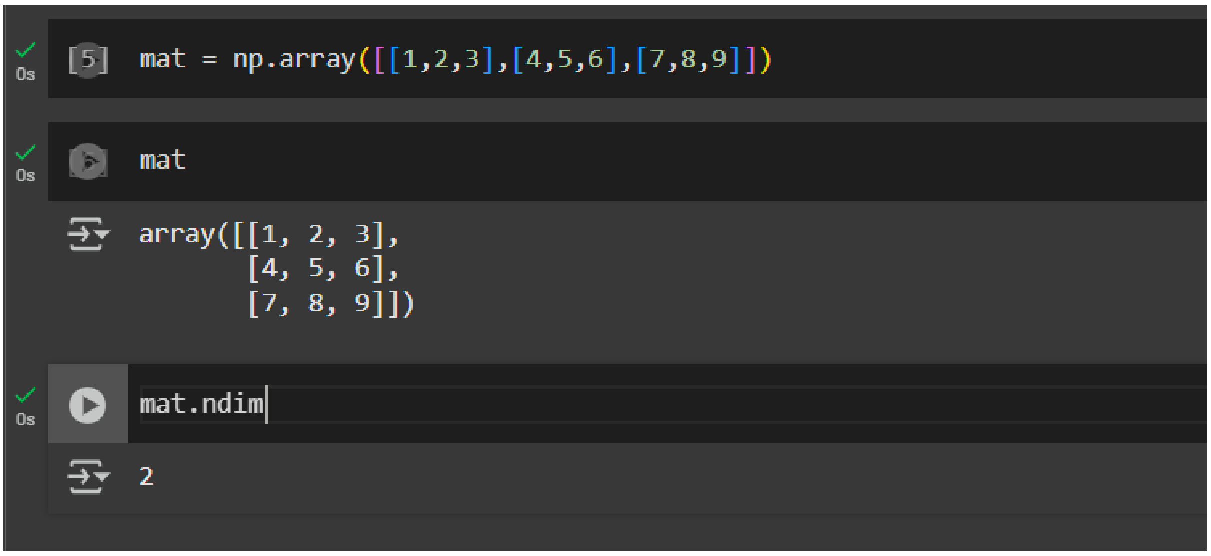


```
[1] import numpy as np

[2] arr = np.array([1,2,3,4])

[3] arr

array([1, 2, 3, 4])

arr.ndim

1
```

What is 2D Tensor?

A 2D tensor is a two-dimensional array, often referred to as a matrix. It consists of rows and columns, forming a grid of values. In machine learning, 2D tensors can represent data such as images (with height and width dimensions) or a dataset with multiple features and examples.

How to generate a 2D tensor?



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

```
mat
```

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

```
mat.ndim
```

```
2
```

In the context of tensors:

- **Rank** refers to the number of dimensions (or axes) a tensor has. For example, a scalar has rank 0, a vector has rank 1, a matrix has rank 2, and so on.
- **Axes** are the specific dimensions along which data is organized. For example, a 2D tensor (matrix) has two axes: one for rows and one for columns.
- **Shape** represents the size of a tensor along each axis. For instance, a 2D tensor with shape (3, 4) means 3 rows and 4 columns. The shape is often written as a tuple.

Video 12:

Installing Anaconda For Data Science | Jupyter Notebook for Machine Learning | Google Colab for ML:

NA

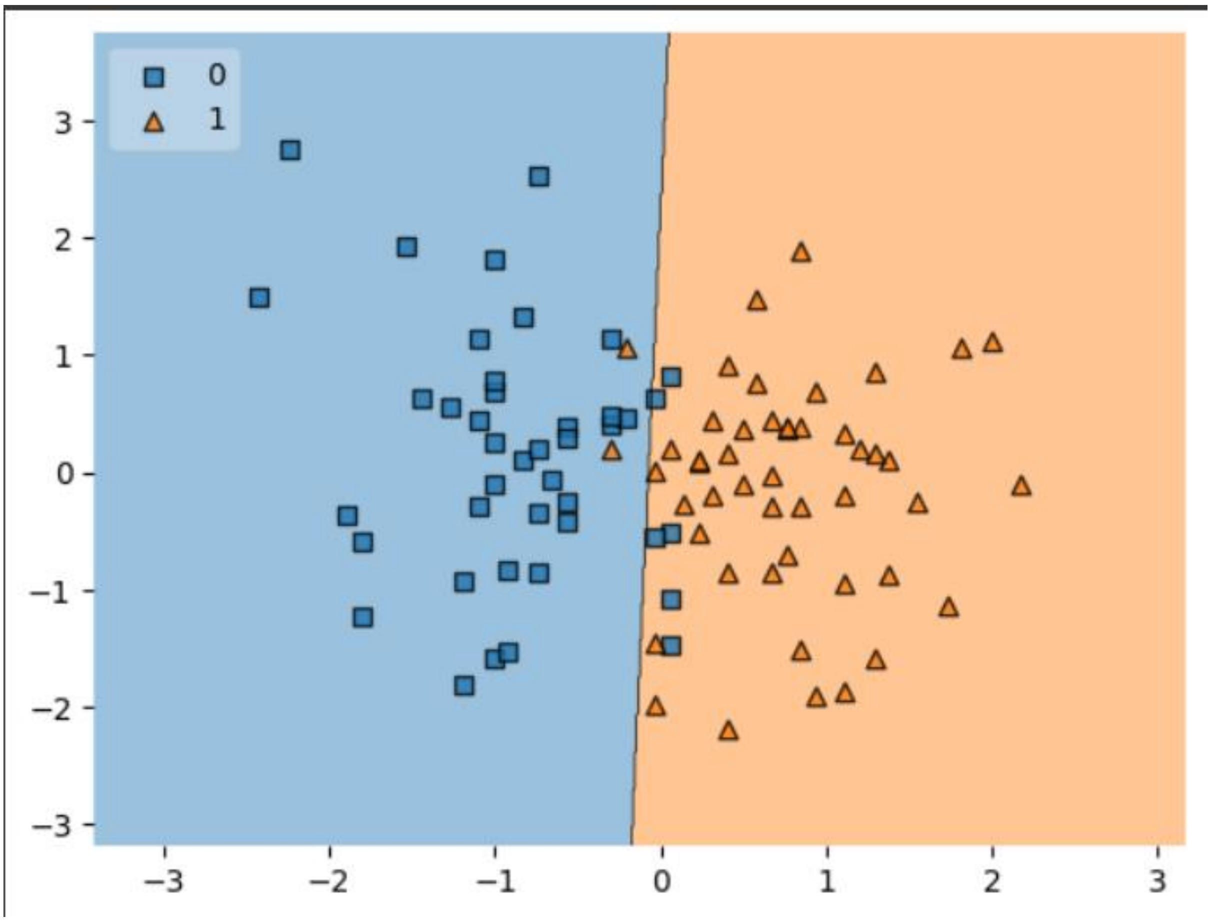
Video 13:

End to End Toy Project:

Code link: https://colab.research.google.com/drive/1RF7uZh9Ata3Msaq7RlWl6mCFX6S-udvR?usp=sharing

Data link: https://github.com/campusx-official/placement-project-logistic-regression

Output:



Video 14:

How to Frame a Machine Learning Problem:

Steps to follow:

1. **Business Problem to ML Problem**
   Translate business objectives into actionable machine learning tasks. Identify specific challenges, then define data-driven solutions and metrics.
2. **Types of Problem**
   Different types of ML problems include classification, regression, clustering, and recommendation. Each has distinct methodologies and goals.
3. **Current Solution**
   Analyze the existing solution's limitations. Determine how machine learning can improve efficiency, accuracy, or provide additional insights.
4. **Getting Data**
   Data collection involves sourcing, cleaning, and structuring data. Accurate, relevant datasets are essential for model training and validation.

5. **Metrics to Measure**
   Metrics are used to assess model performance. Choose metrics that align with business goals, such as accuracy, precision, recall.
6. **Online Vs Batch?**
   Online learning processes data in real-time, updating models continuously. Batch learning updates periodically, requiring stored data and periodic retraining.
7. **Check Assumptions**
   Validate assumptions about data distribution, model behavior, and business context. Test assumptions to avoid biases and improve model reliability.

Video 15:

Working with CSV files:

This video is totally concerned with data gathering technique. Basically, we have following ways to access the data:

1. CSV
2. JSON/SQL
3. Fetch API
4. Web Scraping

Here, we are concerned with CSV files:

When working with CSV files in Python, **Pandas** is a powerful library that simplifies the process of reading, writing, and manipulating these files. Here's a brief overview of the important parameters and techniques for handling CSVs:

1. **Importing Pandas:** You can easily import the Pandas library using import pandas as pd.

2. **Opening Local and URL CSV Files:**

   - To open a local file, use pd.read_csv('filename.csv').
   - For CSV files from a URL, use the requests library combined with StringIO to read the data into a Pandas DataFrame.

3. **Customizing CSV Read:**

   - **Separator (sep)**: Defines how values in the file are separated (e.g., commas, tabs).
   - **Column Names (names)**: Allows you to define column headers if missing.
   - **Index Column (index_col)**: Specifies a column to be used as the index for the DataFrame.
   - **Header (header)**: Defines the row that contains column names.
   - **Use Columns (usecols)**: Reads only a subset of columns for performance improvement.

4. **Additional Parameters:**

   - **Squeeze**: Converts a single-column DataFrame to a Series.

- **Skip Rows (skiprows)**: Skips specified rows before reading.
- **Number of Rows (nrows)**: Limits how many rows to read.
- **Encoding**: Defines the character encoding (useful for files with special characters).
- **Bad Lines (on_bad_lines)**: Specifies how to handle badly formatted rows (e.g., skip them).
- **Data Types (dtype)**: You can specify the data types for certain columns.
- **Dates (parse_dates)**: Automatically converts date columns to datetime objects.
- **Converters (converters)**: Allows applying custom functions to convert data during import.
- **NA Values (na_values)**: Specifies custom placeholders for missing data.

5. **Handling Large Files:** For large datasets, you can read data in **chunks** using chunksize. This lets you process the file piece by piece to save memory.

These features make it easy to import, clean, and analyze data from CSV files in a flexible and efficient manner.

Code link: https://colab.research.google.com/drive/1eEJITMtYXZsBM-W33KhFcwVIpmBpe42W?usp=sharing

Data link: https://github.com/campusx-official/100-days-of-machine-learning/tree/main/day15%20-%20working%20with%20csv%20files