Data Science essentials for AI

Thursday, 15 May 2025 17:46

NumPy

Numerical python is foundation library in python for numerical computation

- Performance
- Easy of use
- Integration

-

- Import numpy
-
- Import numpy as np
-
- Creating arrays
  -
  - From a List
    -
    - import numpy as ns
    - arr = ns.array([1,2,3,4])
    - print(arr)
  -
  - From buildin functions

```
arr = ns.array([1,2,3,4])
# print(arr)
    zeroes = np.zeros((3,3)) # type: ignore
    print(zeroes)
```

```
import numpy as np
ones = np.ones((2,4))
print(ones)
```

Linspace

```
import numpy as np
linspace_array = np.linspace(0, 1, 5)
print(linspace_array)
```

Manipulating arrays

Change shape

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
reshaped = arr.reshape((3,3))
print(reshaped)
Out:-
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Add dimensions

```
arr = np.array([1, 2, 3])
expanded = arr[:, np.newaxis]
print(expanded)

Output:-

[[1]
 [2]
 [3]]
```

- Basic operation of array

```
import numpy as np
a = np.array([1, 2, 3,])
b = np.array([5, 6, 7])
print(a + b)
print(a * b)
print(a / b)
```

Output:-

[6 8 10]
[5 12 21]
[0.2 0.33333333 0.42857143]

```
arr = np.array([4, 16, 25])
print(np.sqrt(arr))
```

Output:

[2. 4. 5.]

```
arr = np.array([4, 16, 25])
print(np.sqrt(arr))
print(np.sum(arr))
print(np.mean(arr))
```

output

45
15.0

Array Indexing, Slicing and reshaping

Indexing: - we can get element for a particular index

```
import numpy as np
arr = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90])
print(arr[2]
Out:-
30
```

Slicing:- where we can slice an array into a smaller array

```
import numpy as np
arr = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90])
print(arr[1:5])

Print(arr[1:])

Out

[20 30 40 50]
20 30 40 50 60 70 80 90
```

Reshaping:- we can reshape particular array.

```
reshaped = arr.reshape(2,7)
print(reshaped)
```

Advanced numpy operations

Broadcasting

Allows numpy to perform arithmetic operations on arrays of different shapes
Smaller arrays are automatically expanded to match the shape of larger arrays

Rules of brodcasting

1. Dimension is compatible if
   1) It matches the other array's dimension
   2) One of the dimensions is 1

```
import numpy as np
# arr = np.array([1, 2, 3])
# print(arr +10)
matrx = np.array([[1, 2, 3], [4, 5, 6]])
vector = np.array([1, 0, 1])
print(matrx + vector)
```

Aggregation functions

Aggregation functions compute summary statistics for array

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print("Sum: ", np.sum(arr))
print("Mean: ",np.mean(arr))
print("Min: ", np.min(arr))
print("Standard Deviation: ", np.std(arr))
print("Sum along rows: ", np.sum(arr, axis=1))
print("Sum along columns: ", np.sum(arr, axis=0))
```

out:-

Sum: 21
Mean: 3.5
Min: 1
Standard Deviation: 1.707825127659933
Sum along rows: [6 15]
Sum along columns: [5 7 9]

Boolean Indexing and filtering

Boolean arrays or conditions to filter elements from an array

```
import numpy as np
arr = np.array([1,2,3,4,5,6])
evens = arr[arr % 2 == 0]
print("Evens : ", evens)
arr[arr > 3] = 0
print("Modifiying arr : ", arr)
```

out:-
Evens : [2 4 6]
Modifiying arr : [1 2 3 0 0 0]

Random Number Generation and setting seeds

For generating random number

Np.random

```
import numpy as np
random_array = np.random.rand(3, 3)
print("Random Array: \n", random_array)
```

Out:-

```
import numpy as np

random_integers = np.random.randint(0, 10, size=(2,3))
print("Random Integers: \n", random_integers)
```

Setting seed

If we need perticular value we use seed

```
import numpy as np
np.random.seed(42)
random_integers = np.random.randint(0, 10, size=(2,3))
print("Random Integers: \n", random_integers)
```

Pandas:-

It's library. Using for data manipulation and analysis
It's easy to use data structure.

Pandas data structure:-

→ Series
- Series is a one dimensional labeled array capable of holding data of any type.

```
import pandas as pd
s = pd.Series([10, 20, 30], index=["a", "b", "c"])
print(s)
```

Out
a 10
b 20
c 30

→ DataFrame
- Data frame is a two dimensional labeled data structure, like a table

```
data = {"Name": ["Alice", "Bob"], "age": [25, 30]}
df =pd.DataFrame(data)
print(df)
```

Out:-

  Name age
0 Alice 25
1 Bob 30

Basic DataFrame operations

If we want to view the data

```
import pandas as pd
s = pd.Series([10, 20, 30], index=["a", "b", "c"])
print(df.head()) # viewing data
print(df.tail(3)) # last three rows of data
print(df.info()) #provide the compleate information about the data
print(df.describe()) # statistical summary of the data
```

Selecting

If we want particular column or raw
This is also give multiple column as well

```
import pandas as pd
s = pd.Series([10, 20, 30], index=["a", "b", "c"])
print(df[["Name", "Age"]])
```

Filter rows

```
import pandas as pd
s = pd.Series([10, 20, 30], index=["a", "b", "c"])
print(df[df["Age"] > 25])
```

Selecting by particular positions

```
import pandas as pd
s = pd.Series([10, 20, 30], index=["a", "b", "c"])
print(df.iloc[0])  # Accessing the first row
print(df.iloc[:, 0])  # Accessing the first column
```

Selecting by label

```
import pandas as pd
s = pd.Series([10, 20, 30], index=["a", "b", "c"])
print(df.loc[:, "Name"])
```

Data Transformations

Rename Columns

```
df = df.rename(column={"old_name": "new_name"})     #
```

Changing Data types

```
df["column_name"] = df["columns_name"].astype("float")
df["column_name"] = pd.to_datetime(df["column_name"])
```

Creating or modifying columns

```
df["new_column"] = df["existing_column"] * 2
```

Combining and Merging DataFrames

Concatenation (For combine)

```
combined = pd.concat([df1, df2], axis =0)
combined = pd.concat([df1, df2], axis=1)
```

Merging

```
merged = pd.merge(df1, df2, on="common_column")
merged = pd.merge(df1, df2='left', on="common_column")
merged = pd.merge(df1, df2, how="inner", on="common_column")
```

Joining

Joined = df1.join(df2, how='inner')

Aggregation Functions

Using groupby

```
 df.groupby("category_column")["numeric_column"].mean()
df.groupby("category_column").agg({"numeric_column":["mean", "max", "min"]})
```

Using pivot_table

```
pivot = df.pivot_table(
    values="numberic_column",
    index="category_column",
    aggfunc="mean"
)
```

Custom aggregation

```
def range_func(x):
    return x.max() - x.min()

df.groupby("category_column")["numeric_column"].agg(range_func)
```

Data Visualization with Matplotlib

a. Matplotlib
   i. It's a foundational python library for creating static, interactive, and animated plots.

Loading Data from Excel and other sources

Common data loading methods

Csv

```
import pandas as pd
s = pd.Series([10, 20, 30], index=["a", "b", "c"])
df = pd.read_csv("data.csv")
df = pd.read_excel("data.xlsx")
```

Excel

```
import pandas as pd
s = pd.Series([10, 20, 30], index=["a", "b", "c"])
df = pd.read_csv("data.csv")
df = pd.read_excel("data.xlsx")
```

Dictionary

```
data = {"Name": ["Alice", "Bob"], "age": [25, 30]}
df =pd.DataFrame(data)
print(df)
```

Saving Data

```
import pandas as pd
s = pd.Series([10, 20, 30], index=["a", "b", "c"])
df = pd.read_csv("data.csv")
df.to_csv("data.csv")
df = pd.read_excel("data.xlsx")
df.to_excel("data.xlsx")
```

Data Cleaning and preparation with pandas

Handling missing values

Pandas we can use it for handling missing data in excel

Drop Missing values

```
import pandas as pd
df = df.dropna()
Df = df.dropna(axis=1)  # it'll dop the column missing values
```

Fill Missing values

This is fill empty cell with zero

```
import pandas as pd
df["column_name"] = df[df"column_name"].filena(o) #fill column with NA

df.filena(method="ffill") #forward fill
df.fillna(method="bfill") #backward fill
df.fillna(value=0) #fillna with 0
df.fillna(value={"column_name": 0}) #fillna with 0 in specific column
```

Interpolation

```
df["column_name"] = df["column_name".interpolate] #this is will fill with add values which
similar
```

Grouping Data by Categories

Grouping data allows us to perform operation on subsets of data based on shared categories

Grouped = df.groupby("column_name")

Groupedby in pandas

1. Operations
   i. Iterate over groups

```
grouped = df.groupby("column_name")
for name,group in grouped:
    print(name)
    print(group)
```

2. Apply aggregation

```
grouped = df.groupby("column_name")
for name,group in grouped:
    print(name)
    print(group)
    grouped .mean()
    grouped.sum()
```

Calculating summary statistics for grouped data

Common statistics

Mean

```
df.groupby("category_column")["numeric_column"].mean()
```

Max

```
df.groupby("category_column")["numeric_column"].max()
```

Min

```
df.groupby("category_column")["numeric_column"].min()
```

Multi aggregation

```
df.groupby("category_column").agg({"numeric_column": ["mean", "max", "min"]})
```

#Baisplot

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [10, 20, 35, 45]
plt.plot(x, y)
plt.show()
```

Line Plot

```
#lineplot
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], label="Trend")
plt.title("Line Plot")
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
plt.legend()
plt.show()
```

Bar Plot

```
#barchart
categories = ['A', 'B', 'C', 'D']
values = [10, 20, 15, 25]
plt.bar(categories, values, color='skyblue')
plt.title("Bar Chart Example")
plt.show()
```

Histogram

```
#Histogram
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
plt.hist(data, bins=6, color="green", edgecolor="black")
plt.title("Histogram Example")
plt.show()
```

Scatterplot

```
#Scatter plot
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]
plt.scatter(x, y, color='yellow')
plt.title("Scatter Plot Example")
plt.show()
```

EDA:- Exploratory data analysis

EDA involves summarizing data sets to uncover patterns, relationships and insights.

1. Steps IN EDA
    i. Data cleaning :- it's just like data cleaning , handle missing values, remove duplicates and fix
    ii. Data Transformation :- Changing data types normalized data and create new feature if needed
    iii. Aggregation and filtering :- summarizing data and applying filters to focus on specific aspects