```
from google.colab import files
uploaded = files.upload()
```

Choose Files flipkart_pro…0250405.csv

- **flipkart_products_20250405.csv**(text/csv) - 903362 bytes, last modified: 4/6/2025 - 100% done

```
Saving flipkart_products_20250405.csv to flipkart_products_20250405.csv
```

```
import pandas as pd

# Replace with your actual filename if different
df = pd.read_csv("flipkart_products_20250405.csv")
print(df.head())
```

```
                                      Product Name  Price (₹)  Rating (★)  \
0  Krishnamurthy-Devan Laboriosam Ultra Smartphon...  142247.04         3.2
1             Nanda-Mahal Dignissimos Lite Laptops 1  186922.43         4.1
2                   Choudhury LLC Amet Plus Decor 15   11843.41         5.0
3           Borah LLC Accusantium Lite Smartphones 9   10864.31         4.8
4               Murty Inc Placeat Pro Smartwatches 8   32950.41         4.5

   Number of Buyers  Total Sold  Available Stock Main Category  Sub Category  \
0              7348        4812              364   Electronics   Smartphones
1              2342         881              145   Electronics       Laptops
2               739        2580              206          Home         Decor
3              1543        4562             1585   Electronics   Smartphones
4              7702        4925             1064   Electronics  Smartwatches

   Discount (%)            Seller  Return Policy  \
0            45         RetailNet          False
1            55  Flipkart Assured          False
2            58       SuperComNet           True
3             0      ElectroWorld          False
4            18         MobileHub          False

                                        Product URL
0  https://www.flipkart.com/Krishnamurthy-Devan-L...
1  https://www.flipkart.com/Nanda-Mahal-Dignissim...
2  https://www.flipkart.com/Choudhury-LLC-Amet-Pl...
3  https://www.flipkart.com/Borah-LLC-Accusantium...
4  https://www.flipkart.com/Murty-Inc-Placeat-Pro...
```

```
# Step 1: Install and Import Necessary Libraries
import numpy as np
import pandas as pd

# Step 2: Load the dataset
file_path = "/content/flipkart_products_20250405.csv"  # Upload to Colab and set path
df = pd.read_csv(file_path)

# Display first few rows
print("Data Sample:\n", df.head())

# Step 3: Convert a numerical column to NumPy array (e.g., ratings or prices)
# We'll use 'retail_price' and 'discounted_price' if available
if 'retail_price' in df.columns and 'discounted_price' in df.columns:
    prices = df[['retail_price', 'discounted_price']].to_numpy(dtype=np.float32)
else:
    prices = np.random.randint(100, 10000, size=(10, 2)).astype(np.float32)  # fallback

print("\nFixed-type Array:\n", prices)

# Step 4: Array Indexing and Slicing
print("\nFirst 5 retail prices:\n", prices[:5, 0])
print("\nEvery second discounted price:\n", prices[::2, 1])

# Step 5: Reshaping, Concatenation, Splitting
reshaped = prices.reshape(-1, 2)
concat = np.concatenate([prices, prices], axis=0)
split = np.split(prices, 2)

print("\nReshaped:\n", reshaped)
print("\nConcatenated:\n", concat.shape)
print("\nSplit:\n", [s.shape for s in split])

# Step 6: Universal Functions & Aggregations
print("\nDiscounts:\n", np.subtract(prices[:, 0], prices[:, 1]))
print("\nAverage Prices:\n", np.mean(prices, axis=0))
```

```python

# Step 7: Broadcasting Rules
discount_ratio = prices[:, 1] / prices[:, 0]
print("\nDiscount Ratios:\n", discount_ratio)

# Step 8: Comparisons and Boolean Masks
mask = prices[:, 1] < 1000
print("\nProducts with discounted price < 1000:\n", prices[mask])

# Step 9: Fancy Indexing
indices = [1, 3, 5]
print("\nFancy Indexed Rows:\n", prices[indices])

# Step 10: Sorting
sorted_prices = np.sort(prices[:, 1])  # Sort discounted prices
argsorted = np.argsort(prices[:, 1])
print("\nSorted Discounted Prices:\n", sorted_prices)
print("\nIndices of Sorted Prices:\n", argsorted)

# Step 11: Structured Arrays and Compound Types
required_columns = ['product_name', 'retail_price', 'discounted_price']  # Define required_columns here
if all(col in df.columns for col in required_columns):
    structured = np.array([
        (row['product_name'], row['retail_price'], row['discounted_price'])
        for _, row in df.head(5).iterrows()
    ], dtype=[('name', 'U50'), ('retail', 'f4'), ('discount', 'f4')])

    print("\nStructured Array:\n", structured)
    print("\nNames with discount > 500:\n", structured[structured['retail'] - structured['discount'] > 500]['name'])
else:
    print("\nError: One or more required columns are missing for creating the structured array.")
    print("Missing columns:", [col for col in required_columns if col not in df.columns])
```

```
 [ 303. 8968.]
 [ 435. 8662.]
 [ 640.  869.]
 [3205. 9890.]
 [2743. 3010.]
 [8962. 5411.]]

First 5 retail prices:
 [1476. 9569. 1548. 4226.  303.]

Every second discounted price:
 [1354. 8931. 8968.  869. 3010.]

Reshaped:
```

```
 [ 435. 8662.]]

Sorted Discounted Prices:
 [ 401.  869. 1354. 3010. 5411. 7350. 8662. 8931. 8968. 9890.]

Indices of Sorted Prices:
 [1 6 0 8 9 3 5 2 4 7]

Error: One or more required columns are missing for creating the structured array.
Missing columns: ['product_name', 'retail_price', 'discounted_price']
```

```
# ------- EXTRA NUMPY OPERATIONS -------

# Matrix Multiplication (just for demo)
mat = prices[:5, :]  # Take first 5 rows
mat_T = mat.T
dot_product = mat @ mat_T
print("\nMatrix Multiplication (5x2 @ 2x5):\n", dot_product)

# Clipping and Rounding
clipped = np.clip(prices[:, 1], 500, 5000)
rounded = np.round(prices[:, 1], -2)  # round to nearest 100
print("\nClipped Prices:\n", clipped)
print("\nRounded Prices:\n", rounded)

# Using np.where to filter and replace
price_check = np.where(prices[:, 1] < 1000, "Low", "High")
print("\nPrice Categories (using np.where):\n", price_check)

# Cumulative Sum and Product
cum_sum = np.cumsum(prices[:, 0])
cum_prod = np.cumprod(prices[:5, 1])
print("\nCumulative Sum of Retail Prices:\n", cum_sum)
print("\nCumulative Product of First 5 Discounted Prices:\n", cum_prod)

# Histogram Binning
hist, bins = np.histogram(prices[:, 1], bins=5)
print("\nHistogram Binning of Discounted Prices:\n", hist)
print("Bins:\n", bins)

# String vectorized operations (if column exists)
if 'product_name' in df.columns:
    names = df['product_name'].astype(str).values
    filtered_names = names[np.char.find(names, "Samsung") >= 0]
    print("\nProduct Names containing 'Samsung':\n", filtered_names[:5])

# NaN Handling (demo only if missing values exist)
if df.isnull().values.any():
    nan_mask = np.isnan(prices)
    prices_clean = np.nan_to_num(prices, nan=0.0)
    print("\nCleaned Prices (NaNs replaced):\n", prices_clean)

# Unique Values & Frequency (Categories, etc.)
if 'product_category_tree' in df.columns:
    categories = df['product_category_tree'].astype(str).values
    unique_cats, counts = np.unique(categories, return_counts=True)
    print("\nTop 5 Unique Categories:\n", list(zip(unique_cats, counts))[:5])

# Memory Usage of NumPy array
print("\nMemory usage of prices array (bytes):", prices.nbytes)

# View vs Copy Demonstration
view = prices[:5]
copy = prices[:5].copy()
view[0][0] = 999999  # This changes original
copy[0][0] = 111111  # This doesn't affect original
print("\nOriginal after modifying view:\n", prices[:5])
print("Original after modifying copy (unchanged):\n", prices[:5])

```

```
Matrix Multiplication (5x2 @ 2x5):
 [[ 4011892. 14666798. 14377422. 16189476. 12589900.]
 [14666798. 91726560. 18394144. 43385944.  6495575.]
 [14377422. 18394144. 82159064. 72184696. 80562256.]
 [16189476. 43385944. 72184696. 71881576. 67195280.]
 [12589900.  6495575. 80562256. 67195280. 80516832.]]

Clipped Prices:
```

```
 [1354.  500. 5000. 5000. 5000. 5000.  869. 5000. 3010. 5000.]

Rounded Prices:
 [1400.  400. 8900. 7400. 9000. 8700.  900. 9900. 3000. 5400.]

Price Categories (using np.where):
 ['High' 'Low' 'High' 'High' 'High' 'High' 'Low' 'High' 'High' 'High']

Cumulative Sum of Retail Prices:
 [ 1476. 11045. 12593. 16819. 17122. 17557. 18197. 21402. 24145. 33107.]

Cumulative Product of First 5 Discounted Prices:
 [1.3540000e+03 5.4295400e+05 4.8491223e+09 3.5641048e+13 3.1962892e+17]

Histogram Binning of Discounted Prices:
 [3 1 1 1 4]
Bins:
 [ 401.     2298.8    4196.6    6094.4004 7992.2    9890.    ]

Memory usage of prices array (bytes): 80

Original after modifying view:
 [[9.99999e+05 1.35400e+03]
 [9.56900e+03 4.01000e+02]
 [1.54800e+03 8.93100e+03]
 [4.22600e+03 7.35000e+03]
 [3.03000e+02 8.96800e+03]]
Original after modifying copy (unchanged):
 [[9.99999e+05 1.35400e+03]
 [9.56900e+03 4.01000e+02]
 [1.54800e+03 8.93100e+03]
 [4.22600e+03 7.35000e+03]
 [3.03000e+02 8.96800e+03]]
```