Beginner NumPy Practice Problems – Solutions

1. Create 1D and 2D arrays
Solution:
```
import numpy as np
arr1 = np.array([2, 4, 6, 8]) # 1D array
arr2 = np.array([[1, 2, 3],
[4, 5, 6]]) # 2×3 array
arr1.shape -> (4,)
arr2.shape -> (2, 3)
```

2. Check basic properties
Solution:
```
X = np.array([[1, 2, 3],
[4, 5, 6]])
print(X.shape) # (2, 3)
print(X.ndim) # 2 (2D array)
print(X.dtype) # usually int64 or int32 depending on system
```

3. Zeros and ones
Solution:
```
zeros_3x3 = np.zeros((3, 3))
ones_2x4 = np.ones((2, 4))
```

4. arange vs linspace
Solution:
```
a = np.arange(0, 10, 2) # [0 2 4 6 8]
b = np.linspace(0, 1, 5) # [0. 0.25 0.5 0.75 1. ]
```

5. Indexing single elements
Solution:
```
X = np.array([[800, 2, 15],
[950, 2, 10],
[1200, 3, 8],
[1500, 3, 5]])
first_element = X[0, 0] # 800
third_row_bedrooms = X[2, 1] # 3
last_element = X[-1, -1] # 5
```

6. Row and column selection
Solution:
```
first_row = X[0, :] # [800, 2, 15]
second_column = X[:, 1] # [2, 2, 3, 3]
```

7. Slicing rows and columns
Solution:
```
first_two_rows = X[:2, :] # rows with 800 and 950
rows_1_to_3_cols_0_to_1 = X[1:4, 0:2]
```

# [[950, 2],
# [1200, 3],
# [1500, 3]]

8. Feature/target split
Solution:
X = data[:, :3] # features: size, bedrooms, age
y = data[:, 3] # target: price
X.shape -> (4, 3), y.shape -> (4,)

9. Mean and standard deviation
Solution:
overall_mean = data.mean()
overall_std = data.std()
Numerical values (approx):
overall_mean ≈ 27781.12
overall_std ≈ 49878.97

10. Mean per column
Solution:
col_means = data.mean(axis=0)
Column means (approx):
size mean ≈ 1112.50
bedrooms mean ≈ 2.50
age mean ≈ 9.50
price mean ≈ 110000.00
The mean of the last column is the average house price.

11. Min and max per feature
Solution:
feature_mins = data[:, :3].min(axis=0)
feature_maxs = data[:, :3].max(axis=0)
Minimums (size, bedrooms, age): [800. 2. 5.]
Maximums (size, bedrooms, age): [1500. 3. 15.]

12. Reshape and flatten
Solution:
a = np.array([1, 2, 3, 4, 5, 6])
a_2x3 = a.reshape(2, 3) # [[1 2 3], [4 5 6]]
a_flat = a_2x3.flatten() # [1 2 3 4 5 6]

13. Vertical and horizontal stacking
Solution:
A = np.array([[1, 2],
[3, 4]])
B = np.array([[5, 6],
[7, 8]])
v_stacked = np.vstack((A, B))
[[1 2],

```
# [3 4],
# [5 6],
# [7 8]]
h_stacked = np.hstack((A, B))
# [[1 2 5 6],
# [3 4 7 8]]
```

14. Scalar operations
Solution:

```
X = np.array([10, 20, 30, 40])
X1 = X - 10 # [0, 10, 20, 30]
X2 = X / 10 # [1., 2., 3., 4.]
```

15. Elementwise operations
Solution:

```
a = np.array([1, 2, 3])
b = np.array([10, 20, 30])
a_plus_b = a + b # [11, 22, 33]
a_times_b = a * b # [10, 40, 90]
a_div_b = a / b # [0.1, 0.1, 0.1]
```

These are called elementwise operations because NumPy applies the operation on each pair of elements at the same index.

16. Vector dot product
Solution:

```
w = np.array([1, 2, 3])
x = np.array([4, 5, 6])
dot1 = np.dot(w, x)
dot2 = w @ x
# dot1 = dot2 = 1*4 + 2*5 + 3*6 = 4 + 10 + 18 = 32
```

17. Matrix–vector multiplication (simple model)
Solution:

```
X = np.array([[1, 2, 3],
[4, 5, 6]])
w = np.array([0.1, 0.2, 0.3])
b = 0.5
y_pred = X @ w + b
# First row: 1*0.1 + 2*0.2 + 3*0.3 + 0.5 = 1.9
# Second row: 4*0.1 + 5*0.2 + 6*0.3 + 0.5 = 3.7
```

So y_pred = [1.9, 3.7]. Each element is a predicted output for one row (one data point).

18. Transpose and sums
Solution:

```
A = np.array([[1, 2, 3],
[4, 5, 6]])
A_T = A.T
# A_T = [[1, 4],
# [2, 5],
# [3, 6]]
sum_axis0 = A.sum(axis=0) # [5, 7, 9]
sum_axis1 = A.sum(axis=1) # [6, 15]
```
axis=0: sum down each column.
axis=1: sum across each row.

19. Boolean indexing (filtering)
Solution:
```
prices = np.array([75000, 90000, 120000, 155000, 60000])
mask = prices > 100000
# mask -> [False, False, True, True, False]
expensive = prices[mask] # [120000, 155000]
count_expensive = expensive.size # 2
```

20. Shuffling dataset (X and y together)
Solution:
```
X = data[:, :3]
y = data[:, 3]
indices = np.random.permutation(len(X))
X_shuffled = X[indices]
y_shuffled = y[indices]
```
Now the rows in X_shuffled and the labels in y_shuffled are still matched but in random order.