NumPy Exercises

NumPy is used for numerical computing with arrays. Below are exercises to practice array creation, manipulation, and operations.

Exercise 1: Array Creation and Modification

Task: Modify the code to create a 4x4 matrix of random integers between 10 and 50, then set all values greater than 40 to 0.

```
import numpy as np

# Original code
array = np.random.randint(1, 100, size=(3, 3))
print("Original array:\n", array)

# Modify the array (example: set values > 50 to 0)
array[array > 50] = 0
print("Modified array:\n", array)
```

Instructions:

- Change the array size to 4x4.
- Update the random integer range to 10–50.
- Modify the condition to set values greater than 40 to 0.
- Run the code and verify the output.

Expected Output:

A 4x4 array with random integers between 10 and 50, where all values > 40 are replaced with 0.

Exercise 2: Statistical Operations

Task: Modify the code to compute the median and standard deviation instead of the mean, and apply it to a 1D array of 20 random floats between 0 and 10.

```
import numpy as np

# Original code
array = np.random.randint(1, 100, size=10)
mean_value = np.mean(array)
print("Array:", array)
print("Mean:", mean_value)
```

Instructions:

- Change the array to a 1D array of 20 random floats (use np.random.uniform).
- Replace the mean calculation with the median and standard deviation.
- Print the array, median, and standard deviation.

Expected Output:
A 1D array of 20 floats, followed by its median and standard deviation.

Pandas Exercises

Pandas is used for data manipulation with DataFrames. Use the following exercises to practice data operations.

Exercise 3: DataFrame Filtering
Task: Modify the code to filter rows where age is greater than 25 and city is "New York".

```
import pandas as pd

# Original code
data = {
    'name': ['Alice', 'Bob', 'Charlie', 'David'],
    'age': [25, 30, 35, 20],
    'city': ['New York', 'Los Angeles', 'New York', 'Chicago']
}
df = pd.DataFrame(data)
filtered_df = df[df['age'] > 30]
print("Filtered DataFrame:\n", filtered_df)
```

Instructions:

- Update the filtering condition to select rows where age > 25 and city == "New York".
- Use the & operator for multiple conditions.
- Print the filtered DataFrame.

Expected Output:
A DataFrame showing only rows where age > 25 and city is "New York" (e.g., Charlie's row).

Exercise 4: Grouping and Aggregation

Task: Modify the code to group by city and compute the maximum age for each city.

```
import pandas as pd

# Original code
data = {
    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'age': [25, 30, 35, 20, 28],
    'city': ['New York', 'Los Angeles', 'New York', 'Chicago', 'Los Angel
}
df = pd.DataFrame(data)
grouped = df.groupby('city')['age'].mean()
print("Grouped means:\n", grouped)
```

Instructions:

- Change the aggregation function from mean() to max().
- Print the maximum age for each city.

Expected Output:

A Series showing the maximum age for each unique city (e.g., New York: 35, Los Angeles: 30, Chicago: 20).

Seaborn Exercises

Seaborn is used for statistical visualizations. Use the tips dataset from Seaborn for these exercises.

Exercise 5: Boxplot Customization

Task: Modify the code to create a boxplot of total_bill grouped by day, with points colored by sex.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Original code
tips = sns.load_dataset('tips')
sns.boxplot(data=tips, x='day', y='total_bill')
plt.show()
```

Instructions:

- Add the hue parameter to color the boxplot by sex.
- Add a title to the plot: "Total Bill by Day and Sex".
- Run the code to display the updated plot.

Expected Output:
A boxplot showing total_bill distributions by day, with different colors for sex (Male/Female), and a title.

Exercise 6: Correlation Heatmap
Task: Modify the code to create a heatmap of the correlation matrix for numerical columns in the tips dataset, with annotations.

```
import seaborn as sns

# Original code
tips = sns.load_dataset('tips')
sns.heatmap(tips.corr())
plt.show()
```

Instructions:

- Ensure the correlation matrix includes only numerical columns (use select_dtypes or specify columns).
- Add annotations to the heatmap to show correlation values (use annot=True).
- Set the color map to coolwarm for better visualization.

Expected Output:
A heatmap showing correlations between numerical columns (total_bill, tip, size) with values displayed on the cells.

Plotly Exercises

Plotly creates interactive visualizations. Use the tips dataset or synthetic data for these exercises.

Exercise 7: Interactive Scatter Plot
Task: Modify the code to create a scatter plot of total_bill vs. tip, with points colored by day and sized by size.

```
import plotly.express as px

# Original code
tips = px.data.tips()
fig = px.scatter(tips, x='total_bill', y='tip', color='sex')
fig.show()
```

Instructions:

- Change the color parameter to day.
- Add the size parameter to scale points by the size column.
- Add hover data to show time and smoker when hovering over points.

Expected Output:

An interactive scatter plot where points represent total_bill vs. tip, colored by day, sized by size, with hover info showing time and smoker.

Exercise 8: Bar Plot with Customization

Task: Modify the code to create a bar plot showing the average total_bill per day, with bars colored by sex.

```
import plotly.express as px

# Original code
tips = px.data.tips()
fig = px.bar(tips, x='day', y='total_bill')
fig.show()
```

Instructions:

- Modify the code to aggregate total_bill by day to show the mean (use tips.groupby('day')['total_bill'].mean() or Plotly's built-in aggregation).
- Add the color parameter to differentiate bars by sex.
- Update the layout to include a title: "Average Total Bill by Day and Sex".

Expected Output:

An interactive bar plot showing the average total_bill for each day, with bars split by sex and a title.