Lab01

May 1, 2025

Import packages

[1]:
```
import numpy as np
```

Exercise 1: NumPy Advanced Operations 1.Create a random array of integers between 0 and 100

[2]:
```
arr =np.random.randint(0, 101, size=200)
print("Original array: ", arr)
```

```
Original array:  [ 57  84  64  32  69  78   6   6  50  22   2   0  17  69  26
10  23  17
  35  83  35  18  75  15  70  52  33  95  92  76  86  35   7  15  10  89
  11  57  35  16  48  94  82  50  60  47  58  59  29  23  79  26  53  23
   5  49  37  67 100  79  21  69  34  99  49  56  52  81  53  35  46  66
   3  89  66  28  13  38  31  48  71  69  64  84  47  24  20  32  89  59
  99  95  35  76  15  19  86  33  89   9  82  54  14  69  82   4  17  71
  68  64  53  16   3  53  60   0  70  64  60  14  67  47  79  45   8  78
  14  60  80  21  50  61  62  99  18  64  29  36  27  44  25  70  78  70
  18  79  49  28   5  93  53   5  84  96  61  42  43  13  56  11  84  61
   5  51  81  89  54  53   8  87   9  91  12  67  45  44  86  79  64  34
  93  53  65  48  36  91  16  56  90  27  43  46  27  86  92  66  26  48
  91  13]
```

2.Create a mask for values greater than 50

[3]:
```
mask = arr >= 50
filtered_arr = arr[mask]
print("Filtered array (values >= 50): ", filtered_arr)
```

```
Filtered array (values >= 50):  [ 57  84  64  69  78  50  69  83  75  70  52  95
92  76  86  89  57  94
  82  50  60  58  59  79  53  67 100  79  69  99  56  52  81  53  66  89
  66  71  69  64  84  89  59  99  95  76  86  89  82  54  69  82  71  68
  64  53  53  60  70  64  60  67  79  78  60  80  50  61  62  99  64  70
  78  70  79  93  53  84  96  61  56  84  61  51  81  89  54  53  87  91
  67  86  79  64  93  53  65  91  56  90  86  92  66  91]
```

3.Demostrate boradcasting

[4]:
```
small = np.arange(5) # 0, 1, 2, 3, 4
print("Small array: ", small)
large = np.arange(20).reshape(4,5)
print("Large array: \n", large)
result = large + small # Broadcasting occurs here
print("Result of broadcasting: \n", result)
```

```
Small array:  [0 1 2 3 4]
Large array:
 [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
Result of broadcasting:
 [[ 0  2  4  6  8]
 [ 5  7  9 11 13]
 [10 12 14 16 18]
 [15 17 19 21 23]]
```

4.Compute the dot product of two arrays of length 10

[5]:
```
a = np.random.rand(10)
b = np.linspace(0, 9, 10)
dp = np.dot(a, b)
print("Dot product of a and b: ", dp)
```

```
Dot product of a and b:  19.74502366276779
```

Exercise 2: Matplotlib Subplots

1. Prepare data for sine and cosine functions:

[6]:
```
x = np.linspace(0, 2*np.pi, 200)
y1 =np.sin(x)
y2 =np.cos(x)
```

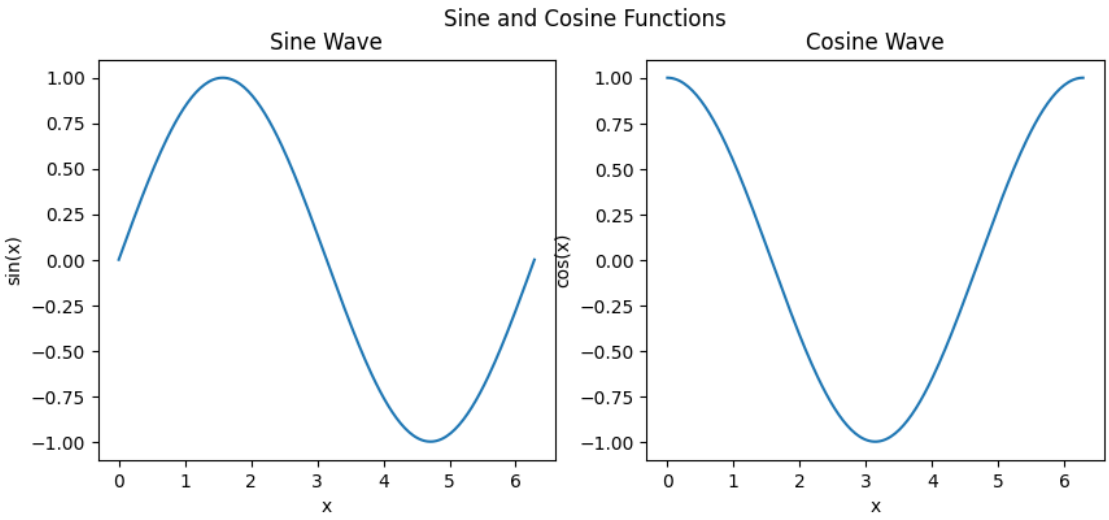
import plot

[7]:
```
import matplotlib.pyplot as plt
```

2. Create subplots:

[8]:
```
fig, axes = plt.subplots(1, 2, sharex=True, figsize=(10, 4))
axes[0].plot(x, y1)
axes[0].set(title="Sine Wave", xlabel="x", ylabel="sin(x)")
axes[1].plot(x, y2)
axes[1].set(title="Cosine Wave", xlabel="x", ylabel="cos(x)")
fig.suptitle("Sine and Cosine Functions")
plt.savefig('trig_functions.png')
```

```
plt.show()
```



Exercise 3: Pandas Cleaning & Preprocessing

1. Load Titanic dataset:

[9]:
```
import pandas as pd
url = 'https://raw.githubusercontent.com/datasciencedojo/datasets/master/
  ↪titanic.csv'
df = pd.read_csv(url)
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
```

```
memory usage: 83.7+ KB
None
```

2. Impute missing values:

[10]:
```
df['Age'].fillna(df['Age'].median(), inplace=True)  # Replace missing ages with␣
 ↪the median age
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)  # Replace␣
 ↪missing embarkation points with the most frequent value
```

```
C:\Users\ravin\AppData\Local\Temp\ipykernel_11068\3345290980.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df['Age'].fillna(df['Age'].median(), inplace=True)  # Replace missing ages
with the median age
C:\Users\ravin\AppData\Local\Temp\ipykernel_11068\3345290980.py:2:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)  # Replace
missing embarkation points with the most frequent value
```

4. Convert and detect outliers in Fare

[37]:
```
df['Fare_int'] = df['Fare'].round().astype(int) # Convert Fare to integer
Q1 = df['Fare_int'].quantile(0.25) # 25th percentile
Q3 = df['Fare_int'].quantile(0.75) # 75th percentile
IQR = Q3 - Q1 # Interquartile range
print("Q1: ", Q1,"Q2:", Q3,"IQR:", IQR) # Print Q1, Q3, and IQR for reference
outliers = df[(df['Fare_int'] < Q1 - 1.5 * IQR) | (df['Fare_int'] > Q3 + 1.5 *␣
 ↪IQR)]
print(outliers)
```

```
Q1:  8.0 Q2: 31.0 IQR: 23.0
     PassengerId  Survived  Pclass  \
1              2         1       1
27            28         0       1
31            32         1       1
34            35         0       1
52            53         1       1
..           …       …     …
846          847         0       3
849          850         1       1
856          857         1       1
863          864         0       3
879          880         1       1

                                                  Name     Sex   Age  SibSp  \
1    Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
27                        Fortune, Mr. Charles Alexander    male  19.0      3
31      Spencer, Mrs. William Augustus (Marie Eugenie)  female  28.0      1
34                                Meyer, Mr. Edgar Joseph    male  28.0      1
52             Harper, Mrs. Henry Sleeper (Myna Haxtun)  female  49.0      1
..                                                   …     …   …    …
846                             Sage, Mr. Douglas Bullen    male  28.0      8
849       Goldenberg, Mrs. Samuel L (Edwiga Grabowska)  female  28.0      1
856         Wick, Mrs. George Dennick (Mary Hitchcock)  female  45.0      1
863                  Sage, Miss. Dorothy Edith "Dolly"  female  28.0      8
879      Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)  female  56.0      0

     Parch    Ticket      Fare        Cabin Embarked  Fare_int
1        0  PC 17599   71.2833          C85        C        71
27       2     19950  263.0000  C23 C25 C27        S       263
31       0  PC 17569  146.5208          B78        C       147
34       0  PC 17604   82.1708          NaN        C        82
52       0  PC 17572   76.7292          D33        C        77
..     …       …       …           …      …       …
846      2  CA. 2343   69.5500          NaN        S        70
849      0     17453   89.1042          C92        C        89
856      1     36928  164.8667          NaN        S       165
863      2  CA. 2343   69.5500          NaN        S        70
879      1     11767   83.1583          C50        C        83

[116 rows x 13 columns]
```

Exercise 4: Pandas Essentials

1. Create and inspect Series:

[12]:
```
s1 = pd.Series ([1,2,4,5])
print(s1.shape, s1.index)
```

```
s2 = pd.Series ([1,2,4,5], index=['a', 'b', 'c', 'd'])
```

```
(4,) RangeIndex(start=0, stop=4, step=1)
```

2. Build DataFrame and summarize:

[13]:
```
df1 = pd.DataFrame({'name': ['Alice', 'Bob', 'Charlie'], 'score': [85, 92, 78]})
df2 = pd.DataFrame(np.random.randn(100, 3), columns=list('ABC'))

df1.head(), df1. tail () , df1. info () , df1. describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   name    3 non-null      object
 1   score   3 non-null      int64
dtypes: int64(1), object(1)
memory usage: 176.0+ bytes
```

[13]:
```
(      name  score
 0    Alice     85
 1      Bob     92
 2  Charlie     78,
       name  score
 0    Alice     85
 1      Bob     92
 2  Charlie     78,
 None,
        score
 count    3.0
 mean    85.0
 std      7.0
 min     78.0
 25%     81.5
 50%     85.0
 75%     88.5
 max     92.0)
```

3. Indexing with loc/iloc, sorting, and dropping:

[14]:
```
df1 = pd.DataFrame({'name': ['Alice', 'Bob', 'Charlie'], 'score': [85, 92, 78]})
df2 = pd.DataFrame(np.random.randn(100, 3), columns=list('ABC'))

df1.loc[0, 'score'], df2.iloc[2]
df2_sorted = df2.sort_values('A', ascending=False)
df2_sorted.drop(['B'], axis=1).head()
```

[14]:
```
           A         C
19  2.280309  0.842856
23  1.915891  1.206664
1   1.826032  2.048091
8   1.813568 -0.236729
36  1.795745 -1.103959
```

4. Handle missing data:

[15]:
```
df_nan = pd.DataFrame({'X': [1, None, 3], 'Y': [None, 2, 3]})
df_nan.dropna(), df_nan.fillna(0)
```

[15]:
```
(     X    Y
 2  3.0  3.0,
      X    Y
 0  1.0  0.0
 1  0.0  2.0
 2  3.0  3.0)
```

5. Excel I/O:

[36]:
```
df_weather = pd.read_excel('weather.xlsx')
df_weather.tail()
df_weather.to_excel('weather_updated.xlsx')
```

Exercise 5: Loading Open Dataset from UCI Repository

1. Load Wine dataset:

[ ]:
```
wine_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.
 ↪data'
cols = ['Class', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash',
    'Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols',
    'Proanthocyanins', 'Color intensity', 'Hue',
    'OD280/OD315 of diluted wines', 'Proline']
df_wine = pd.read_csv(wine_url, names=cols)
print(df_wine.head(10))
```

```
   Class  Alcohol  Malic acid   Ash  Alcalinity of ash  Magnesium  \
0      1    14.23        1.71  2.43               15.6        127
1      1    13.20        1.78  2.14               11.2        100
2      1    13.16        2.36  2.67               18.6        101
3      1    14.37        1.95  2.50               16.8        113
4      1    13.24        2.59  2.87               21.0        118
5      1    14.20        1.76  2.45               15.2        112
6      1    14.39        1.87  2.45               14.6         96
7      1    14.06        2.15  2.61               17.6        121
8      1    14.83        1.64  2.17               14.0         97
```

```
9      1    13.86        1.35  2.27               16.0         98

   Total phenols  Flavanoids  Nonflavanoid phenols  Proanthocyanins  \
0           2.80        3.06                  0.28             2.29
1           2.65        2.76                  0.26             1.28
2           2.80        3.24                  0.30             2.81
3           3.85        3.49                  0.24             2.18
4           2.80        2.69                  0.39             1.82
5           3.27        3.39                  0.34             1.97
6           2.50        2.52                  0.30             1.98
7           2.60        2.51                  0.31             1.25
8           2.80        2.98                  0.29             1.98
9           2.98        3.15                  0.22             1.85

   Color intensity   Hue  OD280/OD315 of diluted wines  Proline
0             5.64  1.04                          3.92     1065
1             4.38  1.05                          3.40     1050
2             5.68  1.03                          3.17     1185
3             7.80  0.86                          3.45     1480
4             4.32  1.04                          2.93      735
5             6.75  1.05                          2.85     1450
6             5.25  1.02                          3.58     1290
7             5.05  1.06                          3.58     1295
8             5.20  1.08                          2.85     1045
9             7.22  1.01                          3.55     1045
```

2. Group by class:

[ ]:
```
wine_means = df_wine.groupby("Class").mean()
print(wine_means)
```

```
         Alcohol  Malic acid       Ash  Alcalinity of ash   Magnesium  \
Class
1      13.744746    2.010678  2.455593          17.037288  106.338983
2      12.278732    1.932676  2.244789          20.238028   94.549296
3      13.153750    3.333750  2.437083          21.416667   99.312500

       Total phenols  Flavanoids  Nonflavanoid phenols  Proanthocyanins  \
Class
1           2.840169    2.982373              0.290000         1.899322
2           2.258873    2.080845              0.363662         1.630282
3           1.678750    0.781458              0.447500         1.153542

       Color intensity       Hue  OD280/OD315 of diluted wines      Proline
Class
1             5.528305  1.062034                      3.157797  1115.711864
2             3.086620  1.056282                      2.785352   519.507042
3             7.396250  0.682708                      1.683542   629.895833
```

Exercise 6: scikit-learn Iris Dataset (Extended)

1. Load and preview Iris:

```
[ ]: from sklearn import datasets

iris = datasets.load_iris()
df_iris = pd.DataFrame(iris.data, columns=iris.feature_names)
df_iris['target'] = iris.target
print(df_iris.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
```

2. Train/test split:

```
[ ]: from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    df_iris[iris.feature_names], df_iris['target'],
    test_size=0.3, random_state=42)
```

3. Model training and evaluation:

```
[ ]: from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13
```

```
              precision    recall  f1-score   support

    accuracy                         1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```