

PLACEMENT REFRESHER PROGRAM

Session 8: Python 3
Data Handling & Visualization

By
Ritesh Kumar Pandey

Agenda

- Data Handling
 - Numpy
 - Pandas
- Data Visualization
 - Matplotlib
 - Seaborn
 - Plotly

- NumPy - Numerical Python
- highly flexible, optimized, open-source package meant for array processing
- provides tools for delivering high-end performance while dealing with N-dimensional powerful array objects
- beneficial for performing scientific computations, mathematical, and logical operations, sorting operations, I/O functions, basic statistical and linear algebra-based operations along with random simulation and broadcasting functionalities.

Input: `my_list = [[1, 2, 3], [4, 5, 6]]`

Output: `[1, 2, 3, 4, 5, 6]`

Input: `my_list = [[1, 2, 3], [4, 5, 6]]`

Output: `[1, 2, 3, 4, 5, 6]`

1. Using List
2. Using Numpy

Input: my_list = [[1, 2, 3], [4, 5, 6]]

Output: [1, 2, 3, 4, 5, 6]

```
my_list = [[1, 2, 3], [4, 5, 6]]
```

```
flat_list = []
```

```
for sublist in my_list:
```

```
    for num in sublist:
```

```
        flat_list.append(num)
```

```
print(flat_list)
```

```
arr = np.array(my_list,  
dtype=object)
```

```
flat_arr = arr.flatten()
```

```
print (flat_arr)
```

How are NumPy arrays better than Python's lists?

- Python lists support storing heterogeneous data types whereas NumPy arrays can store datatypes of one nature itself. NumPy provides extra functional capabilities that make operating on its arrays easier which makes NumPy array advantageous in comparison to Python lists as those functions cannot be operated on heterogeneous data.
- NumPy arrays are treated as objects which results in minimal memory usage. Since Python keeps track of objects by creating or deleting them based on the requirements, NumPy objects are also treated the same way. This results in lesser memory wastage.
- NumPy arrays support multi-dimensional arrays.
- NumPy provides various powerful and efficient functions for complex computations on the arrays.
- NumPy also provides various range of functions for BitWise Operations, String Operations, Linear Algebraic operations, Arithmetic operations etc. These are not provided on Python's default lists.

Write a program for creating an integer array with values belonging to the range 10 and 60

Write a program for creating an integer array with values belonging to the range 10 and 60

```
import numpy as np
arr = np.arange(10, 60)
print(arr)
```

```
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
 58 59]
```

Write a program to add a border of zeros around the existing array.

Write a program to add a border of zeros around the existing array.

```
import numpy as np

# Create NumPy arrays filled with ones
ones_arr = np.ones((4,4))

print("Transformed array:")
transformed_array = np.pad(ones_arr, pad_width=1, mode='constant',
constant_values=0)
print(transformed_array)
```

What is the output of the below code snippet?

```
import numpy as np
arr1 = np.array([7,8,9,10])
arr2 = np.array([1,2,3,4])
arr3 = arr1 + arr2
arr3 = arr3*arr1
print (arr3[2])
```

What is the output of the below code snippet?

```
import numpy as np
arr1 = np.array([7,8,9,10])
arr2 = np.array([1,2,3,4])
arr3 = arr1 + arr2
arr3 = arr3*arr1
print (arr3[2])
```

108

Which of the following is the correct way of creating an array of type float?

1. `a = np.array([4,3,2,1]).toFloat()`
2. `a = np.float([4,3,2,1])`
3. `a = np.array([4,3,2,1], dtype='f')`
4. `a = np.array([4,3,2,1], type='float')`

Which of the following is the correct way of creating an array of type float?

1. `a = np.array([4,3,2,1]).toFloat()`
2. `a = np.float([4,3,2,1])`
3. `a = np.array([4,3,2,1], dtype='f')`
4. `a = np.array([4,3,2,1], type='float')`

- Pandas - an open-source Python package
- most commonly used for data science, data analysis, and machine learning tasks
- built on top of another library named Numpy
- provides various data structures and operations for manipulating numerical data and time series and is very efficient in performing various functions like data visualization, data manipulation, data analysis, etc.

What data structures are provided by Pandas?

1. Arrays
2. Lists
3. Series and DataFrames
4. Dictionaries

What data structures are provided by Pandas?

1. Arrays
2. Lists
3. Series and DataFrames
4. Dictionaries

- **Series** - It is a one-dimensional array-like structure with homogeneous data which means data of different data types cannot be a part of the same series. It can hold any data type such as integers, floats, and strings and its values are mutable i.e. it can be changed but the size of the series is immutable i.e. it cannot be changed.
- **DataFrame** - It is a two-dimensional array-like structure with heterogeneous data. It can contain data of different data types and the data is aligned in a tabular manner. Both size and values of DataFrame are mutable.
- **Panel** - The Pandas have a third type of data structure known as Panel, which is a 3D data structure capable of storing heterogeneous data but it isn't that widely used.

What is the difference between the `loc[]` and `iloc[]` methods in Pandas?

1. The `loc[]` method selects rows based on their labels, while `iloc[]` selects rows based on their positions.
2. The `loc[]` method selects columns based on their labels, while `iloc[]` selects columns based on their positions.
3. The `loc[]` method can only be used on DataFrames, while `iloc[]` can be used on both DataFrames and Series.
4. The `loc[]` method is faster than `iloc[]` for large datasets.

What is the difference between the `loc[]` and `iloc[]` methods in Pandas?

1. The `loc[]` method selects rows based on their labels, while `iloc[]` selects rows based on their positions.
2. The `loc[]` method selects columns based on their labels, while `iloc[]` selects columns based on their positions.
3. The `loc[]` method can only be used on DataFrames, while `iloc[]` can be used on both DataFrames and Series.
4. The `loc[]` method is faster than `iloc[]` for large datasets.

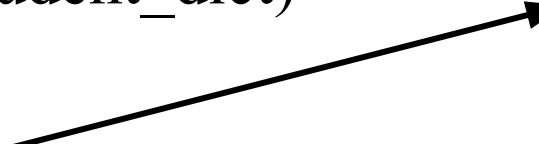
```
import pandas as pd
```

```
student_dict = {'Name': ['Kate', 'Harry', 'Sheila'], 'Age': [10, 14, 12], 'Marks': [85, 77, 91]}
```

```
# create DataFrame from dict
```

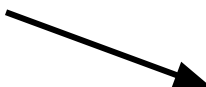
```
df = pd.DataFrame(student_dict)
```

```
print(df.iloc[[0, 2]])
```



	Name	Age	Marks
0	Kate	10	85
2	Sheila	12	91

```
print(df.loc[(df.Name=='Kate')])
```



	Name	Age	Marks
0	Kate	10	85

Basics & I/O:

- `pd.read_csv(filename)`: Read a comma-separated values file.
- `pd.read_excel(filename)`: Read an Excel file.
- `pd.read_sql(query, connection)`: Read from a SQL table/database.
- `df.to_csv(filename)`: Write to a CSV file.
- `df.to_excel(filename)`: Write to an Excel file.
- `df.head(n)`: Display the first `n` rows.
- `df.tail(n)`: Display the last `n` rows.
- `df.describe()`: Summary statistics.

Data Creation:

- `pd.DataFrame(data)`: Create a DataFrame.
- `pd.Series(data)`: Create a Series.

Selection:

- `df[col]`: Select column by column name.
- `df[[col1, col2]]`: Select multiple columns.
- `df.iloc[row, col]`: Select by row and column integer indices.
- `df.loc[row_label, col_label]`: Select by row and column labels.

Filtering:

- `df[df[col] > value]`: Rows where the column is greater than value.
- `df.query("col > value")`: Use query method to filter rows.
- `df[df[col].isin(values)]`: Rows where the column is in values.

Data Cleaning:

- `df.dropna()`: Drop missing values.
- `df.fillna(value)`: Fill missing values.
- `df.replace(old_val, new_val)`: Replace values.
- `df.drop_duplicates()`: Drop duplicate rows.

Data Transformation:

- `df.set_index(col)`: Set column as index.
- `df.reset_index()`: Reset index to default integer index.
- `df.pivot_table()`: Create a pivot table.
- `df.melt()`: Unpivot a DataFrame.

Combining Data:

- `pd.concat([df1, df2])`: Concatenate DataFrames.
- `df1.append(df2)`: Append rows of DataFrames.
- `pd.merge(df1, df2, on=col)`: Merge DataFrames using a column.

Aggregation:

- `df.groupby(col)`: Group by column.
- `df.agg(functions)`: Aggregate using one or more functions.

Sorting & Ranking:

- `df.sort_values(by=col)`: Sort by column.
- `df.rank()`: Rank rows.

Apply Functions:

- `df.apply(func)`: Apply a function.
- `df[col].map(func)`: Apply a function to a column.

Time Series:

- `pd.to_datetime(col)`: Convert a column to datetime.
- `df.resample()`: Resample time-series data.
- `df.asfreq()`: Convert time series frequency.

Text Data:

- `df[col].str.split()`: Split string values.
- `df[col].str.contains(pattern)`: Check if string contains a pattern.
- `df[col].str.replace(old, new)`: Replace text.

Categorical Data:

- `df[col].astype("category")`: Convert column to categorical type.
- `df[col].cat.set_categories()`: Set categories.

Missing Data:

- `df.isna()`: Check for missing values.
- `df.notna()`: Check for non-missing values.

Plotting:

- `df.plot()`: Plot data.
- `df[col].hist()`: Plot a histogram.

Statistical Operations:

- `df[col].mean()`: Mean of column.
- `df[col].median()`: Median of column.
- `df.corr()`: Correlation matrix.

String Methods:

- `df[col].str.lower()`: Convert to lowercase.
- `df[col].str.upper()`: Convert to uppercase.
- `df[col].str.strip()`: Strip whitespaces.

Renaming & Reordering:

- `df.rename(columns=dict)`: Rename columns.
- `df.reorder_levels()`: Reorder levels on multi-level index.

Other Operations:

- `df.memory_usage()`: Memory usage of each column.
- `df.info()`: Concise summary of the DataFrame.
- `df.shape`: Return a tuple representing the dimensionality of the DataFrame.
- `df.dtypes`: Return the dtypes in the DataFrame.
- `df.columns`: Return the column labels of the DataFrame.
- `df.values`: Return a Numpy representation of the DataFrame.
- `df.T`: Transpose the DataFrame.
- `df.clip(lower, upper)`: Trim values at input threshold(s).
- `df.abs()`: Return a Series/DataFrame with absolute numeric value of each element.
- `df.all()`: Return whether all elements are True.
- `df.any()`: Return whether any element is True.
- `df.count()`: Count non-NA cells for each column or row.
- `df.empty`: Indicator whether DataFrame is empty.
- `df.bool()`: Return the bool of a single element Pandas object.
- `df.kurt()`: Return unbiased kurtosis.

Other Operations:

- `df.idxmax()`: Return index of first occurrence of maximum value.
- `df.idxmin()`: Return index of first occurrence of minimum value.
- `df.mode()`: Return the mode(s) of the dataset.
- `df.nunique()`: Count distinct observations.
- `df.quantile(q)`: Return value at the given quantile.
- `df.round()`: Round a DataFrame to a variable number of decimal places.
- `df.sem()`: Return unbiased standard error of the mean.
- `df.skew()`: Return unbiased skew.
- `df.to_dict()`: Convert the DataFrame to a dictionary.
- `df.to_string()`: Render a DataFrame to a console-friendly tabular output.

- Data visualization is the representation of data in a graphical or pictorial format that makes it easy to understand and interpret.
- It is important because it allows analysts to see patterns, trends, and insights that may be difficult to discern from raw data.

Python Libraries:

- Matplotlib : <https://matplotlib.org/>
- Seaborn : <https://seaborn.pydata.org/>
- Plotly Express (Interactive Graphs) : <https://plotly.com/python/plotly-express/>

Which of the given plot is used to give statistical summary ?

1. Bar
2. Scatter
3. Box
4. Line

Which of the given plot is used to give statistical summary ?

1. Bar
2. Scatter
3. Box
4. Line

Consider the code to create a bar chart -

```
import matplotlib.pyplot as plt  
weekdays = ['sun', 'mon', 'tue']  
sale = [12, 13, 11]  
plt.bar(weekdays, sale)
```

But when executing this code, no graph is displayed. Why ?

Consider the code to create a bar chart -

```
import matplotlib.pyplot as plt  
weekdays = ['sun', 'mon', 'tue']  
sale = [12, 13, 11]  
plt.bar(weekdays, sale)
```

But when executing this code, no graph is displayed. Why ?

```
plt.show()
```

THANK YOU