SESSION 6

1. Filtering data in DataFrame

For analyzing the data, we need a lot of filtering operations. Pandas provide a `query()` method to filter the DataFrame.

Syntax

```
DataFrame.query(expr, inplace=False, **kwargs)
```

- *expr*: Refers to an expression in string form to filter data.
- *inplace*: If the value is True, it makes the changes in the original DataFrame.
- *kwargs*: Refers to the other keyword arguments.

In [ ]:

```
# dataframe creation
import pandas as pd
```

```
Core_Dataframe = pd.DataFrame({'Emp_No': ['Emp1','Emp2','Emp3','Emp4'],
'Employee_Name':  ['Arun', 'selva', 'rakesh', 'arjith'],
'Employee_dept': ['CAD', 'CAD', 'DEV', 'CAD']})
print("   THE CORE DATAFRAME ")
print(Core_Dataframe)
print("")

# Dataframe query
Queried_Dataframe = Core_Dataframe.query('Employee_dept == "DEV"')
print("   THE QUERIED DATAFRAME ")
print(Queried_Dataframe)
print("")
```

```
   THE CORE DATAFRAME
  Emp_No Employee_Name Employee_dept
0   Emp1          Arun           CAD
1   Emp2         selva           CAD
2   Emp3        rakesh           DEV
3   Emp4        arjith           CAD

   THE QUERIED DATAFRAME
  Emp_No Employee_Name Employee_dept
2   Emp3        rakesh           DEV
```

---

2. Combining in DataFrame

- The method of combining the DataFrame using common fields is called "joining".
- The method that we use for combining the DataFrame is a `join()` method. The columns that contain common values are called "join key".

Inner joins

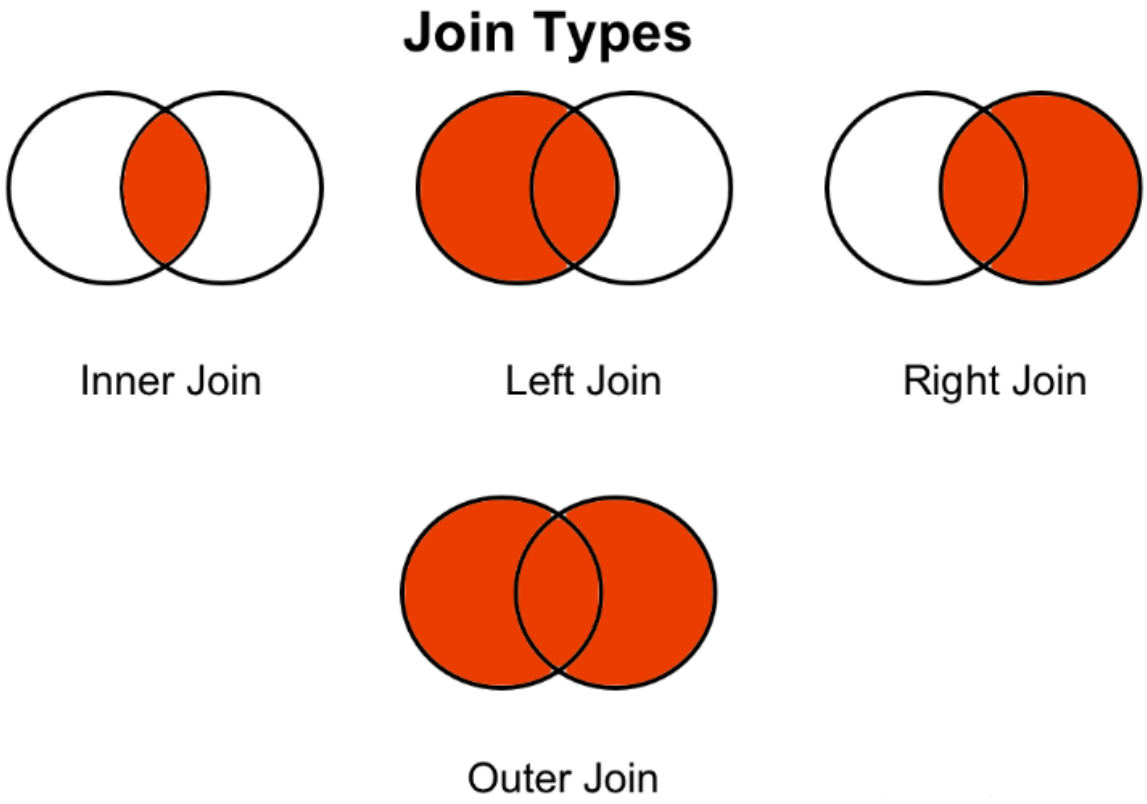
- Inner join can be defined as the most commonly used join.
- Basically, its main task is to combine the two DataFrames based on a join key and returns a new DataFrame.
- The returned DataFrame consists of only selected rows that have matching values in both of the original DataFrame.

Left Outer Join/Left outer joins:

- With a left outer join, all the records from the first dataframe will be displayed, irrespective of whether the keys in the first dataframe can be found in the second dataframe.
- Whereas, for the second dataframe, only the records with the keys in the second dataframe that can be found in the first dataframe will be displayed.

Left join/Left outer joins:

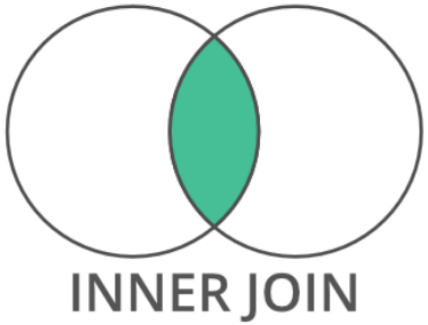
- If we want to add some information into the DataFrame without losing any of the data, we can simply do it through a different type of join called a "left outer join" or "left join"



2.1 Inner Join:

- Inner join is the most common type of join you'll be working with.

- It returns a dataframe with only those rows that have common characteristics.
- This is similar to the intersection of two sets.



In [ ]:

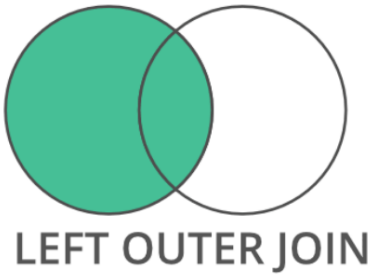
```
# Inner Join


# importing pandas
import pandas as pd

# Creating dataframe a
a = pd.DataFrame()

# Creating Dictionary
d = {'id': [1, 2, 10, 12,17],
        'val1': ['a', 'b', 'c', 'd','g']}

a = pd.DataFrame(d)

# Creating dataframe b
b = pd.DataFrame()

# Creating dictionary
d = {'id': [1, 2, 9, 8,17],
        'val1': ['p', 'q', 'r', 's','c']}
b = pd.DataFrame(d)

# inner join
df = pd.merge(a, b, on='id', how='inner')

# display dataframe
df
```

Out[ ]:

	id	val1_x	val1_y
0	1	a	p
1	2	b	q
2	17	g	c

2.2 Left Outer Join:

- For a left join, all the records from the first dataframe will be displayed.
- However, only the records with the keys in the second dataframe that can be found in the first dataframe will be displayed.



In [ ]:

```
# importing pandas
import pandas as pd

# Creating dataframe a
a = pd.DataFrame()

# Creating Dictionary
d = {'id': [1, 2, 10, 12],
        'val1': ['a', 'b', 'c', 'd']}

a = pd.DataFrame(d)

# Creating dataframe b
b = pd.DataFrame()

# Creating dictionary
d = {'id': [1, 2, 9, 8],
        'val1': ['p', 'q', 'r', 's']}
```

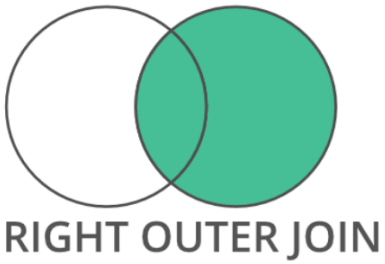
```
b = pd.DataFrame(d)

# left outer join
df = pd.merge(a, b, on='id', how='left')

# display dataframe
df
```

Out[ ]:

	id	val1_x	val1_y
0	1	a	p
1	2	b	q
2	10	c	NaN
3	12	d	NaN

2.3 Right Outer Join:

- For a right join, all the records from the second dataframe will be displayed.
- However, only the records with the keys in the first dataframe that can be found in the second dataframe will be displayed.



In [ ]:

```
# importing pandas
import pandas as pd

# Creating dataframe a
a = pd.DataFrame()

# Creating Dictionary
d = {'id': [1, 2, 10, 12],
        'val1': ['a', 'b', 'c', 'd']}
```

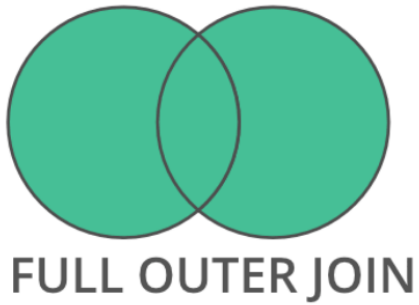
```
a = pd.DataFrame(d)

# Creating dataframe b
b = pd.DataFrame()

# Creating dictionary
d = {'id': [1, 2, 9, 8],
        'val1': ['p', 'q', 'r', 's']}
b = pd.DataFrame(d)

# right outer join
df = pd.merge(a, b, on='id', how='right')

# display dataframe
df
```

Out[ ]:

	id	val1_x	val1_y
0	1	a	p
1	2	b	q
2	9	NaN	r
3	8	NaN	s

2.4 Full Outer Join:

- A full outer join returns all the rows from the left dataframe, all the rows from the right dataframe, and matches up rows where possible, with NaNs elsewhere.
- But if the dataframe is complete, then we get the same output.



In [ ]:

```
# importing pandas
import pandas as pd

# Creating dataframe a
a = pd.DataFrame()

# Creating Dictionary
d = {'id': [1, 2, 10, 12],
        'val1': ['a', 'b', 'c', 'd']}

a = pd.DataFrame(d)

# Creating dataframe b
b = pd.DataFrame()

# Creating dictionary
d = {'id': [1, 2, 9, 8],
        'val1': ['p', 'q', 'r', 's']}
b = pd.DataFrame(d)

# full outer join
df = pd.merge(a, b, on='id', how='outer')

# display dataframe
df
```

Out[ ]:

	id	val1_x	val1_y
0	1	a	p
1	2	b	q
2	10	c	NaN
3	12	d	NaN
4	9	NaN	r
5	8	NaN	s

3. Merging in DataFrame

- Pandas `merge()` is defined as the process of bringing the two datasets together into one and aligning the rows based on the common attributes or columns.
- It is an entry point for all standard database join operations between DataFrame objects

In [ ]:
```
# Merge two DataFrames on multiple keys

import pandas as pd

left = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5']})

right = pd.DataFrame({
    'id':[1,2,3,4,5],
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5']})

print (pd.merge(left,right,on='subject_id'))
```

```
   id_x  Name_x subject_id  id_y Name_y
0     2     Amy       sub2     1  Billy
1     3   Allen       sub4     2  Brian
2     4   Alice       sub6     4  Bryce
3     5  Ayoung       sub5     5  Betty
```

---

4. Counting in DataFrame

- The Pandas `count()` is defined as a method that is used to count the number of non-NA cells for each column or row.
- The `axis` parameter in the count() method specifies the axis along which to perform the counting.
- By default, `axis=0` , which means that the counting is performed vertically, i.e., the number of non-null values in each column is counted.
- If you set `axis=1` , the counting is performed horizontally, i.e., the number of non-null values in each row is counted.

In [ ]:
```
# importing pandas as pd
import pandas as pd

# Creating a dataframe using dictionary
df = pd.DataFrame({"A":[-5, 8, 12, None, 5, 3],
                                "B":[-1, None, 6, 4, None, 3],
                                "C":["sam", "haris", "alex", "samantha", "peter", "nathan"]})

# Printing the dataframe
df
# axis = 0 indicates row
df.count(axis = 0)
```

Out[ ]:
```
A    5
B    4
C    6
dtype: int64
```

In [ ]:
```
# importing pandas as pd
import pandas as pd

# Creating a dataframe using dictionary
df = pd.DataFrame({"A":[-5, 8, 12, None, 5, 3],
                                "B":[-1, None, 6, 4, None, 3],
                                "C":["sam", "haris", "alex", "samantha", "peter", "nathan"]})

# Printing the dataframe
df
# axis = 1 indicates column
df.count(axis = 1)
```

Out[ ]:
```
0    3
1    2
2    3
3    2
4    2
5    3
dtype: int64
```

---

5. Functions in DataFrame

5.1 Aggregations

- It is defined as a function that returns a single aggregated value for each of the groups.
- We can perform several aggregation operations on the grouped data when the groupby object is created.
- DataFrame `groupby()` function allows us to rearrange the data by utilizing them on real-world data sets.
- Its primary task is to split the data into various groups. These groups are categorized based on some criteria.

In [ ]:
```
# Find the average co2 consumption for each car brand:
import pandas as pd

data = {
  'co2': [95, 90, 99, 104, 105, 94, 99, 104],
  'model': ['Citigo', 'Fabia', 'Fiesta', 'Rapid', 'Focus', 'Mondeo', 'Octavia', 'B-Max'],
  'car': ['Skoda', 'Skoda', 'Ford', 'Skoda', 'Ford', 'Ford', 'Skoda', 'Ford']
}

df = pd.DataFrame(data)

print(df.groupby(["car"]).mean())
```

```
         co2
car
Ford   100.5
Skoda   97.0
```

In [ ]:
```
# Find the average co2 consumption for each car brand without mean():
import pandas as pd

data = {
  'co2': [95, 90, 99, 104, 105, 94, 99, 104],
  'model': ['Citigo', 'Fabia', 'Fiesta', 'Rapid', 'Focus', 'Mondeo', 'Octavia', 'B-Max'],
  'car': ['Skoda', 'Skoda', 'Ford', 'Skoda', 'Ford', 'Ford', 'Skoda', 'Ford']
}

df = pd.DataFrame(data)

print(df.groupby(["car"]))
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000024C549F8A30>
```

5.2 View Groups

In [ ]:
```
import pandas as pd

data = {
  'co2': [95, 90, 99, 104, 105, 94, 99, 104],
   #          0        1       2        3        4       5        6          7
  'model': ['Citigo', 'Fabia', 'Fiesta', 'Rapid', 'Focus', 'Mondeo', 'Octavia', 'B-Max'],
  'car': ['Skoda', 'Skoda', 'Ford', 'Skoda', 'Ford', 'Ford', 'Skoda', 'Ford']
}

df = pd.DataFrame(data)

df.groupby('model').groups
```

Out[ ]:
```
{'B-Max': [7], 'Citigo': [0], 'Fabia': [1], 'Fiesta': [2], 'Focus': [4], 'Mondeo': [5], 'Octavia': [6], 'Rapid': [3]}
```

5.3 Select a Group

- Using the `get_group()` method, we can select a single group.

In [ ]:
```
# select single group from the below data
import pandas as pd

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

grouped = df.groupby('Year')
grouped.get_group(2015)
```

Out[ ]:

	Team	Rank	Year	Points
1	Riders	2	2015	789
3	Devils	3	2015	673
5	kings	4	2015	812

	Team	Rank	Year	Points
10	Royals	1	2015	804

In [ ]:
```
# select single Year from the below data

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

grouped = df.groupby('Year')
print(grouped.get_group(2014))
```

```
    Team  Rank  Year  Points
0  Riders     1  2014     876
2  Devils     2  2014     863
4   Kings     3  2014     741
9  Royals     4  2014     701
```

5.4 Filtration

- Filtration filters the data on a defined criteria and returns the subset of data. The `filter()` function is used to filter the data.

In [ ]:
```
# Filterin data in DataFrame
import pandas as pd
import numpy as np

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

# using the filter condition, we are asking to return the teams which have participated three or more times in IPL
df.groupby('Team').filter(lambda x: len(x)>=3)
```

Out[ ]:

	Team	Rank	Year	Points
0	Riders	1	2014	876
1	Riders	2	2015	789
4	Kings	3	2014	741
6	Kings	1	2016	756
7	Kings	1	2017	788
8	Riders	2	2016	694
11	Riders	2	2017	690

Filtration using CSV

- Filtration filters the CSV data on a defined criteria and returns the subset of data. The `filter()` function is used to filter the data.

In [ ]:

```
# Import the pandas library
import pandas as pd

# Load the CSV file into a DataFrame
df = pd.read_csv('your_csv_file.csv')

# Use the filter() function to select the columns you want to filter. For example, if you want to filter based on the "age" and "g
filtered_df = df.filter(['age', 'gender'])

# Apply the desired filter to the selected columns using the query() function. For example, if you want to filter the rows where t
filtered_df = filtered_df.query('age >= 30')

# Save the filtered DataFrame to a new CSV file
# index=False - prevents pandas from writing the row index as a separate column in the CSV file
filtered_df.to_csv('filtered_csv_file.csv', index=False)
```

5.5 Fill NA on CSV using `fillna()`

- To fill missing values in a CSV file using a DataFrame in pandas, you can use the `fillna()` function.

In [ ]:
```
import pandas as pd

df = pd.read_csv('your_csv_file.csv')

# Use the fillna() function to fill missing values with a specific value or method. For example, to fill missing values with the m
# Here, df.mean() calculates the mean of each column, and inplace=True updates the DataFrame with the filled values
df.fillna(df.mean(), inplace=True)

df.to_csv('filled_csv_file.csv', index=False)
```

5.6 Group and Filter on CSV using `groupby()` & `filter()`

- To group and filter a CSV file using a DataFrame in pandas, you can use the `groupby()` function to group the rows based on a specific column, and then apply a filter using the `filter()` function.

In [ ]:
```
import pandas as pd

df = pd.read_csv('your_csv_file.csv')

# Use the groupby() function to group the rows based on a specific column. For example, if you want to group by the "gender" colum
grouped_df = df.groupby('gender')

# Use the filter() function to apply a filter to the grouped DataFrame. For example, if you want to filter the groups where the me
# Here, the lambda function applies the filter to each group by calculating the mean of the "age" column for each group
filtered_df = grouped_df.filter(lambda x: x['age'].mean() >= 30)


filtered_df.to_csv('filtered_csv_file.csv', index=False)
```

Homework Questions

1) Write a program for DataFrame Filtering using `Query()`

2) Python code to demonstrate how to use `merge()` function to perform an outer join on two DataFrames using Pandas library

3) Add data on the dataset using Left join

4) Write a program with input Dictionaries using Right Join

5) Write a program with input Dictionaries using Left Join

6) Write a program with the below DataFrame and execute Group by 'continent' and print one of the continents using pandas groupby function

```
  DataFrame({'continent' : ['Asia','NorthAmerica','NorthAmerica','Europe','Europe', 'Europe','Asia', 'Europe',
'Asia'],
              'country' : ['China', 'USA', 'Canada', 'Poland', 'Romania', 'Italy', 'India', 'Germany',
'Russia'],
              'Member_G20' : ['Y', 'N', 'Y', 'N', 'Y', 'N', 'Y', 'N', 'Y']})
```

7) Write a program using pandas groupby filter by column values and conditional aggregation on the below DataFrame.

```
  DataFrame({'continent' : ['Asia','NorthAmerica','NorthAmerica','Europe','Europe', 'Europe','Asia', 'Europe',
'Asia'],
              'country' : ['China', 'USA', 'Canada', 'Poland', 'Romania', 'Italy', 'India', 'Germany',
'Russia'],
              'Member_G20' : ['Y', 'N', 'Y', 'N', 'Y', 'N', 'Y', 'N', 'Y']})
```

- Execute Groupby continent who are G20 Member

8) Create the below DataFrame with four columns (`name, age, gender, and score`). Use the `filter()` function to select only two columns (`name and age`) and the `loc[]` function to filter the rows based on the condition that the age column is greater than or equal to 30.

```
'name': ['Alice', 'Bob', 'Charlie', 'Dave'],
'age': [25, 32, 30, 45],
'gender': ['F', 'M', 'M', 'M'],
'score': [80, 90, 70, 85]
```

9) Create two dataframes `df1 and df2`, each with a `key and value` column. Merge the dataframes based on the key column using the `merge()` function, and store in merged_df.

```
df1         'key': ['A', 'B', 'C', 'D'],
            'value': [1, 2, 3, 4]

df2         'key': ['B', 'D', 'E', 'F'],
            'value': [5, 6, 7, 8]
```

For solutions of Homework questions, please refer to the `HomeworkSolution.ipynb` file