ML-Assignment 2

Name: *Swayam Swarup Jethi*

Section: *2C1*

Registration Number: *2341010085*

Serial Number: *31*

Date of Submission: */10/2025*

1. Create a Pandas Series from a Python list [10, 20, 30, 40, 50] with customindexes ['a','b','c','d','e'].

(a) Print the first 3 elements.

(b) Access the element at index 'c'.

In [83]:

```
import pandas as pd
import numpy as np
arr=np.array([10,20,30,40,50])
s=pd.Series(arr, index=['a','b','c','d','e'])

print(f"first 3 elements:\n{s[:3]}")
print("element at index 'c':",s['c'])
```

```
first 3 elements:
a    10
b    20
c    30
dtype: int64
element at index 'c': 30
```

2. Create a Series from a NumPy array of random integers between 1–100(size=
10).

(a) Find the maximum, minimum, and mean values.

(b) Apply a function to square each element.

In [84]:

```
import random
arr=np.random.randint(1,100,size=10)
print(arr)
print(f"Max={s2.max()} \nMin={s2.min()} \nMean={s2.mean()}\n")
s2=pd.Series(arr**2,index=arr)
print(f"x          x²\n{s2**2}")
```

```
[96 78 61  2  2 98 88 19  7 90]
Max=7921
Min=441
Mean=3693.0

x          x²
96    84934656
78    37015056
61    13845841
2           16
2           16
98    92236816
88    59969536
19      130321
7         2401
```

```
90    65610000
dtype: int64
```

3. Given a Series with values [5, np.nan, 8, np.nan, 12]:

(a) Check for missing values.

(b) Fill missing values with forward fill (ffill).

(c) Drop missing values.

In [85]:

```
s3=pd.Series([5,np.nan,8,np.nan,12])
print(s3.isnull()) #isna can be used instead of isnull
print(f"No. of missing values: {s3.isnull().sum()}\n")
print(f"Filling missing values with forward fill: \n{s3.ffill()}\n")
print(f"Dropping Missing Values: \n{s3.dropna()}")
```

```
0    False
1     True
2    False
3     True
4    False
dtype: bool
No. of missing values: 2

Filling missing values with forward fill:
0     5.0
1     5.0
2     8.0
3     8.0
4    12.0
dtype: float64

Dropping Missing Values:
0     5.0
2     8.0
4    12.0
dtype: float64
```

4. Convert a Python dictionary data = {'Math': 85, 'Science': 90, 'English': 88} into a Series. Retrieve the value for 'Science'.

In [86]:

```
data = {'Math': 85, 'Science': 90, 'English': 88}
s4=pd.Series(data)
print(s4)
print(f"value for Science: {s4['Science']}")
```

```
Math       85
Science    90
English    88
dtype: int64
value for Science: 90
```

5. Create a DataFrame using a dictionary:

data = { 'Name': ['Amit', 'Riya', 'John', 'Sara'],

'Age': [25, 30, 22, 28],

'Salary': [50000, 60000, 55000, 65000]. }

(a) Select all rows where Age > 25.

(b) Select rows where Salary is between 55,000 and 65,000.

In [87]:

```
data = { 'Name': ['Amit', 'Riya', 'John', 'Sara'],'Age': [25, 30, 22, 28],
        'Salary': [50000, 60000, 55000, 65000] }
```

```
df=pd.DataFrame(data)
print(f"Rows with Age>25: \n{df[df['Age']>25]}\n")
print(f"Rows with 55k<Salary<65k: \n{df[(df['Salary'] > 55000) & (df['Salary'
```

```
Rows with Age>25:
   Name  Age  Salary
1  Riya   30   60000
3  Sara   28   65000

Rows with 55k<Salary<65k:
   Name  Age  Salary
1  Riya   30   60000
```

6. Create a DataFrame:

data = { 'Department': ['HR','IT','HR','IT','Finance'],

'Employee': ['A','B','C','D','E'],

'Salary': [40000, 50000, 42000, 55000, 60000]. }

(a) Find average salary per department.

(b) Count employees in each department.

(c) Sort employees by Salary in ascending order.

(d) Sort by Department then by Salary (descending)

In [88]:

```
data = {'Department': ['HR','IT','HR','IT','Finance'],
'Employee': ['A','B','C','D','E'],
'Salary': [40000, 50000, 42000, 55000, 60000]}
df=pd.DataFrame(data)
print(f"Avg Salary Per Dept: \n{df.groupby('Department')['Salary'].mean()}\n"
print(f"No. of Employees Per Dept: \n{df.groupby('Department')['Employee'].co
print(f"\nEmployees sorted by Salary (ascending):\n{df.sort_values(by='Salary
print("\nSorted by Department and Salary (descending):")
print(df.sort_values(by=['Department', 'Salary'], ascending=[True, False]))
```

```
Avg Salary Per Dept:
Department
Finance    60000.0
HR         41000.0
IT         52500.0
Name: Salary, dtype: float64

No. of Employees Per Dept:
Department
Finance    1
HR         2
IT         2
Name: Employee, dtype: int64

Employees sorted by Salary (ascending):
  Department Employee  Salary
0         HR        A   40000
2         HR        C   42000
1         IT        B   50000
3         IT        D   55000
4    Finance        E   60000

Sorted by Department and Salary (descending):
  Department Employee  Salary
4    Finance        E   60000
2         HR        C   42000
0         HR        A   40000
3         IT        D   55000
1         IT        B   50000
```

1. Given a DataFrame with duplicate rows, perform the following tasks:

- **(a)** Identify and display only the duplicate rows.
- **(b)** Drop duplicates while:
  - Keeping the first occurrence.
  - Keeping the last occurrence.
  - Removing all duplicates.
- **(c)** Drop duplicates based on specific columns.
- **(d)** Count the number of duplicate rows using `duplicated().sum()` .
- **(e)** Extract only the unique values from a single column (e.g., `Name`).

In [89]:

```
data={
    'Name': ['Amit', 'Riya', 'Amit', 'John', 'Riya', 'Sara', 'Amit'],
    'Age': [25, 30, 25, 22, 30, 28, 25],
    'Salary': [50000, 60000, 50000, 55000, 60000, 65000, 50000]
}
df=pd.DataFrame(data)
print(df)
print("\n(a) Duplicate rows:")
print(df[df.duplicated(keep=False)])
print("\n(b) Drop duplicates, keeping the first occurrence:")
print(df.drop_duplicates(keep='first'))
print("\nDrop duplicates, keeping the last occurrence:")
print(df.drop_duplicates(keep='last'))
print("\nDrop duplicates, removing all duplicates (no duplicates remain):")
print(df.drop_duplicates(keep=False))
print("\n(c) Drop duplicates based on 'Name' and 'Age':")
print(df.drop_duplicates(subset=['Name', 'Age'], keep='first'))
print("\n(d) Number of duplicate rows:")
print(df.duplicated().sum())
print("\n(e) Unique values in 'Name' column:")
print(df['Name'].unique())
```

```
   Name  Age  Salary
0  Amit   25   50000
1  Riya   30   60000
2  Amit   25   50000
3  John   22   55000
4  Riya   30   60000
5  Sara   28   65000
6  Amit   25   50000

(a) Duplicate rows:
   Name  Age  Salary
0  Amit   25   50000
1  Riya   30   60000
2  Amit   25   50000
4  Riya   30   60000
6  Amit   25   50000

(b) Drop duplicates, keeping the first occurrence:
   Name  Age  Salary
0  Amit   25   50000
1  Riya   30   60000
3  John   22   55000
5  Sara   28   65000

Drop duplicates, keeping the last occurrence:
   Name  Age  Salary
```

```
3  John   22   55000
4  Riya   30   60000
5  Sara   28   65000
6  Amit   25   50000

Drop duplicates, removing all duplicates (no duplicates remain):
   Name  Age  Salary
3  John   22   55000
5  Sara   28   65000

(c) Drop duplicates based on 'Name' and 'Age':
   Name  Age  Salary
0  Amit   25   50000
1  Riya   30   60000
3  John   22   55000
5  Sara   28   65000

(d) Number of duplicate rows:
3

(e) Unique values in 'Name' column:
['Amit' 'Riya' 'John' 'Sara']
```

1. Given a DataFrame containing `NaN` values, perform the following tasks:

- **(a)** Detect missing values.
- **(b)** Count missing values per column.
- **(c)** Drop rows:
  - With any `NaN` values.
  - Where all values are `NaN` .
  - Where `NaN` appears in specific columns (e.g., `Age` or `Salary`).
- **(d)** Fill missing values:
  - With a fixed value (e.g., 0 or "Unknown").
  - With the mean of the column.
  - Using forward fill and backward fill.
  - Using linear interpolation.
- **(e)** Compare the results of forward fill versus interpolation on the same dataset.

In [90]:

```python
import pandas as pd
import numpy as np

data = {
    'Name': ['Amit', 'Riya', np.nan, 'John', 'Sara', np.nan],
    'Age': [25, np.nan, 22, np.nan, 28, 30],
    'Salary': [50000, 60000, np.nan, 55000, np.nan, np.nan]
}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)

print("\n(a) Detect missing values (True indicates NaN):")
print(df.isna())

print("\n(b) Count missing values per column:")
print(df.isna().sum())

print("\n(c) Drop rows with any NaN values:")
print(df.dropna())
print("\nDrop rows where all values are NaN:")
print(df.dropna(how='all'))

print("\nDrop rows where NaN appears in 'Age' or 'Salary':")
print(df.dropna(subset=['Age', 'Salary']))

print("\n(d) Fill missing values with fixed value (0 for numeric, 'Unknown' f
df_fixed = df.fillna({'Name': 'Unknown', 'Age': 0, 'Salary': 0})
print(df_fixed)

print("\nFill missing values with mean of the column:")
df_mean = df.copy()
df_mean['Age'] = df_mean['Age'].fillna(df_mean['Age'].mean())
df_mean['Salary'] = df_mean['Salary'].fillna(df_mean['Salary'].mean())
print(df_mean)

print("\nFill missing values with forward fill:")
print(df.ffill())

print("\nFill missing values with backward fill:")
print(df.bfill())

print("\nFill missing values with linear interpolation:")
print(df.interpolate(method='linear'))

print("\n(e) Comparison of forward fill vs interpolation:")
df_ffill = df.ffill()
df_interp = df.interpolate(method='linear')
print("\nForward fill result:")
print(df_ffill)
print("\nInterpolation result:")
print(df_interp)
```

```
Original DataFrame:
   Name   Age   Salary
0  Amit  25.0  50000.0
1  Riya   NaN  60000.0
2   NaN  22.0      NaN
3  John   NaN  55000.0
4  Sara  28.0      NaN
5   NaN  30.0      NaN

(a) Detect missing values (True indicates NaN):
    Name    Age  Salary
0  False  False   False
1  False   True   False
2   True  False    True
3  False   True   False
4  False  False    True
5   True  False    True

(b) Count missing values per column:
Name      2
Age       2
Salary    3
dtype: int64

(c) Drop rows with any NaN values:
   Name   Age   Salary
0  Amit  25.0  50000.0

Drop rows where all values are NaN:
   Name   Age   Salary
```

```
0  Amit  25.0  50000.0
1  Riya   NaN  60000.0
2   NaN  22.0      NaN
3  John   NaN  55000.0
4  Sara  28.0      NaN
5   NaN  30.0      NaN

Drop rows where NaN appears in 'Age' or 'Salary':
   Name   Age   Salary
0  Amit  25.0  50000.0

(d) Fill missing values with fixed value (0 for numeric, 'Unknown' for Name):
      Name   Age   Salary
0     Amit  25.0  50000.0
1     Riya   0.0  60000.0
2  Unknown  22.0      0.0
3     John   0.0  55000.0
4     Sara  28.0      0.0
5  Unknown  30.0      0.0

Fill missing values with mean of the column:
   Name    Age   Salary
0  Amit  25.00  50000.0
1  Riya  26.25  60000.0
2   NaN  22.00  55000.0
3  John  26.25  55000.0
4  Sara  28.00  55000.0
5   NaN  30.00  55000.0

Fill missing values with forward fill:
   Name   Age   Salary
0  Amit  25.0  50000.0
1  Riya  25.0  60000.0
2  Riya  22.0  60000.0
3  John  22.0  55000.0
4  Sara  28.0  55000.0
5  Sara  30.0  55000.0

Fill missing values with backward fill:
   Name   Age   Salary
0  Amit  25.0  50000.0
1  Riya  22.0  60000.0
2  John  22.0  55000.0
3  John  28.0  55000.0
4  Sara  28.0      NaN
5   NaN  30.0      NaN

Fill missing values with linear interpolation:
   Name   Age   Salary
0  Amit  25.0  50000.0
1  Riya  23.5  60000.0
2   NaN  22.0  57500.0
3  John  25.0  55000.0
4  Sara  28.0  55000.0
5   NaN  30.0  55000.0

(e) Comparison of forward fill vs interpolation:

Forward fill result:
   Name   Age   Salary
0  Amit  25.0  50000.0
1  Riya  25.0  60000.0
2  Riya  22.0  60000.0
3  John  22.0  55000.0
```

```
4  Sara  28.0  55000.0
5  Sara  30.0  55000.0

Interpolation result:
   Name   Age   Salary
0  Amit  25.0  50000.0
1  Riya  23.5  60000.0
2   NaN  22.0  57500.0
3  John  25.0  55000.0
4  Sara  28.0  55000.0
5   NaN  30.0  55000.0
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: