Pandas Data Manipulation Tasks

Pandas is a popular library for data manipulation and analysis. It provides data structures such as DataFrames that make it easy to work with structured data. Pandas offers functions for data cleaning, transformation, and exploration, making it useful for data pre-processing tasks in machine learning.

1. Load Dataset and Perform Operations

In [3]:
```
import pandas as pd
import numpy as np
```

```
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\.libs\libopenblas.EL2C6PLE4ZYW3ECEVIV3OXXGRN2NRFM2.gfortran-win_amd64.dll
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\.libs\libopenblas.XWYDX2IKJW2NMTWSFYNGFUWKQU3LYTCZ.gfortran-win_amd64.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

In [4]:
```
df=pd.read_csv('employee_data_100_records.csv')
df.head()
#df.tail()
```

Out[4]:

	Employee_ID	Name	Department	Salary	Join_Date	Performance_Score	Bonus	Locatio
0	101	Employee_1	HR	103828.0	1/1/2015	98.0	9454.0	Bosto
1	102	Employee_2	IT	104145.0	1/31/2015	78.0	NaN	Nev Yor
2	103	Employee_3	Operations	58164.0	3/2/2015	79.0	5765.0	Bosto
3	104	Employee_4	Finance	56490.0	4/1/2015	91.0	9691.0	Nev Yor
4	105	Employee_5	HR	85256.0	5/1/2015	95.0	3749.0	Chicag

In [5]:
```
df.shape
```

Out[5]: (100, 10)

In [6]:
```
#to find basic information on data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Employee_ID        100 non-null    int64  
 1   Name               100 non-null    object 
 2   Department         100 non-null    object 
 3   Salary             98 non-null     float64
 4   Join_Date          100 non-null    object 
 5   Performance_Score  97 non-null     float64
 6   Bonus              86 non-null     float64
 7   Location           100 non-null    object 
 8   Manager_ID         100 non-null    int64  
 9   Status             100 non-null    object 
dtypes: float64(3), int64(2), object(5)
memory usage: 7.9+ KB
```

In [7]:
```
#selecting column from dataframe
df['Department']
print(df.Department)
```

```
0            HR
1            IT
2    Operations
3       Finance
4            HR
        ...    
95   Operations
96           HR
97   Operations
98   Operations
99   Operations
Name: Department, Length: 100, dtype: object
```

In [8]:
```
#To find What are unique values in particular column
print(df['Status'].unique())

```

```
['Resigned' 'On Leave' 'Active' 'Retired']
```

In [9]:
```
#to find number of unique values for all columns
print(df['Department'].nunique())
```

```
5
```

In [10]:
```
#to find number of unique values for each column in entire dataset
unq=df.nunique()
unq
```

Out[10]:
```
Employee_ID          100
Name                 100
Department             5
Salary                98
Join_Date            100
Performance_Score     40
Bonus                 85
Location               5
Manager_ID             5
Status                 4
dtype: int64
```

In [11]:
```
#To get summary of data
df.describe()
```

Out[11]:

	Employee_ID	Salary	Performance_Score	Bonus	Manager_ID
count	100.000000	98.000000	97.000000	86.000000	100.000000
mean	150.500000	86953.857143	79.412371	6808.406977	302.930000
std	29.011492	19761.040995	11.610657	1954.521415	1.437274
min	101.000000	50355.000000	60.000000	3223.000000	301.000000
25%	125.750000	71686.750000	69.000000	5248.000000	302.000000
50%	150.500000	86446.500000	78.000000	6632.500000	303.000000
75%	175.250000	104278.500000	91.000000	8503.750000	304.000000
max	200.000000	118577.000000	100.000000	9942.000000	305.000000

In [12]:
```
# 2. Filtering & Selection
f=df[(df['Department']=='HR')]
f.shape
```

Out[12]:
```
(23, 10)
```

In [13]:
```
f.shape
```

Out[13]:
```
(23, 10)
```

In [14]:
```
filterData = df[(df["Department"] == "HR") & (df["Location"] == "Chicago") &
filterData
```

Out[14]:

	Employee_ID	Name	Department	Salary	Join_Date	Performance_Score	Bonus	Locat
4	105	Employee_5	HR	85256.0	5/1/2015	95.0	3749.0	Chic
39	140	Employee_40	HR	64662.0	3/16/2018	62.0	6050.0	Chic
44	145	Employee_45	HR	106641.0	8/13/2018	70.0	9113.0	Chic
65	166	Employee_66	HR	77511.0	5/4/2020	85.0	9914.0	Chic
80	181	Employee_81	HR	96871.0	7/28/2021	67.0	4821.0	Chic
93	194	Employee_94	HR	92481.0	8/22/2022	62.0	8578.0	Chic

In [15]:
```
# 3. Sorting & Ranking
df_sort=df.sort_values(by="Salary",ascending=False)
df_sort.head()
```

Out[15]:

	Employee_ID	Name	Department	Salary	Join_Date	Performance_Score	Bonus	Loca
6	107	Employee_7	HR	118577.0	6/30/2015	81.0	8095.0	
13	114	Employee_14	Finance	118414.0	1/26/2016	90.0	4957.0	
48	149	Employee_49	IT	118020.0	12/11/2018	92.0	9797.0	Chic
88	189	Employee_89	IT	116983.0	3/25/2022	95.0	9445.0	Ang
24	125	Employee_25	Marketing	116262.0	12/21/2016	72.0	5649.0	

In [16]:
```
#sorting based on multiple values
Data_sort = df.sort_values(by=["Performance_Score","Salary"], ascending=[Fal
Data_sort.head()
```

Out[16]:

	Employee_ID	Name	Department	Salary	Join_Date	Performance_Score	Bonus	Locat
52	153	Employee_53	Operations	114620.0	4/10/2019	100.0	4451.0	
61	162	Employee_62	Marketing	56074.0	1/5/2020	99.0	7311.0	
0	101	Employee_1	HR	103828.0	1/1/2015	98.0	9454.0	Bos
89	190	Employee_90	Operations	73092.0	4/24/2022	98.0	3223.0	Bos
28	129	Employee_29	Marketing	95373.0	4/20/2017	97.0	NaN	Bos

In [17]:
```
#aggregation function
ag=df.Salary.mean()
ag
```

Out[17]:
```
86953.85714285714
```

In [18]:
```
# 4. Grouping & Aggregation
gpdf = df.groupby("Location")['Employee_ID'].count()
gpdf
```

Out[18]:
```
Location
Boston         19
Chicago        17
Los Angeles    17
New York       21
San Jose       26
Name: Employee_ID, dtype: int64
```

In [19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Employee_ID        100 non-null    int64  
 1   Name               100 non-null    object 
 2   Department         100 non-null    object 
 3   Salary             98 non-null     float64
 4   Join_Date          100 non-null    object 
 5   Performance_Score  97 non-null     float64
 6   Bonus              86 non-null     float64
 7   Location           100 non-null    object 
 8   Manager_ID         100 non-null    int64  
 9   Status             100 non-null    object 
dtypes: float64(3), int64(2), object(5)
memory usage: 7.9+ KB
```

In [20]:
```
# 6. Date-Time Handling
df["Join_Date"] = pd.to_datetime(df["Join_Date"])
#creating
df["Tenure_Years"] = (pd.to_datetime("today") - df["Join_Date"]).dt.days //
df.head()
#df.info()
```

Out[20]:

	Employee_ID	Name	Department	Salary	Join_Date	Performance_Score	Bonus	Locatio
0	101	Employee_1	HR	103828.0	2015-01-01	98.0	9454.0	Bosto
1	102	Employee_2	IT	104145.0	2015-01-31	78.0	NaN	Nev Yor
2	103	Employee_3	Operations	58164.0	2015-03-02	79.0	5765.0	Bosto
3	104	Employee_4	Finance	56490.0	2015-04-01	91.0	9691.0	Nev Yor
4	105	Employee_5	HR	85256.0	2015-05-01	95.0	3749.0	Chicag

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Employee_ID        100 non-null    int64
 1   Name               100 non-null    object
 2   Department         100 non-null    object
 3   Salary             98 non-null     float64
 4   Join_Date          100 non-null    datetime64[ns]
 5   Performance_Score  97 non-null     float64
 6   Bonus              86 non-null     float64
 7   Location           100 non-null    object
 8   Manager_ID         100 non-null    int64
 9   Status             100 non-null    object
 10  Tenure_Years       100 non-null    int64
dtypes: datetime64[ns](1), float64(3), int64(3), object(4)
memory usage: 8.7+ KB
```

In [22]:
```
# 1. Handling Missing Values
#1.1 filling salary column with average salary of employees
df["Salary"].fillna(df["Salary"].mean(), inplace=True)
#1.2 fill all null values with 0 in performance score
df["Performance_Score"].fillna(0, inplace=True)
#1.3 fill null values of Bonus column with median
df["Bonus"].fillna(df["Bonus"].median(),inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Employee_ID        100 non-null    int64
 1   Name               100 non-null    object
 2   Department         100 non-null    object
 3   Salary             100 non-null    float64
 4   Join_Date          100 non-null    datetime64[ns]
 5   Performance_Score  100 non-null    float64
 6   Bonus              100 non-null    float64
 7   Location           100 non-null    object
 8   Manager_ID         100 non-null    int64
 9   Status             100 non-null    object
 10  Tenure_Years       100 non-null    int64
dtypes: datetime64[ns](1), float64(3), int64(3), object(4)
memory usage: 8.7+ KB
```

In [23]:
```
# 7. Categorical Data Manipulation
df["Status"] = df["Status"].astype("category")
df["Department"] = df["Department"].astype("category")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Employee_ID        100 non-null    int64
 1   Name               100 non-null    object
 2   Department         100 non-null    category
 3   Salary             100 non-null    float64
 4   Join_Date          100 non-null    datetime64[ns]
 5   Performance_Score  100 non-null    float64
 6   Bonus              100 non-null    float64
 7   Location           100 non-null    object
 8   Manager_ID         100 non-null    int64
 9   Status             100 non-null    category
 10  Tenure_Years       100 non-null    int64
dtypes: category(2), datetime64[ns](1), float64(3), int64(3), object(2)
memory usage: 7.8+ KB
```

In [24]:
```
1 df.head()  # Display first few rows
```

Out[24]:

	Employee_ID	Name	Department	Salary	Join_Date	Performance_Score	Bonus	Locatio
0	101	Employee_1	HR	103828.0	2015-01-01	98.0	9454.0	Bosto
1	102	Employee_2	IT	104145.0	2015-01-31	78.0	6632.5	Nev Yor
2	103	Employee_3	Operations	58164.0	2015-03-02	79.0	5765.0	Bosto
3	104	Employee_4	Finance	56490.0	2015-04-01	91.0	9691.0	Nev Yor
4	105	Employee_5	HR	85256.0	2015-05-01	95.0	3749.0	Chicag

In [25]:
```
1 print(10/2)
```

```
5.0
```

In [26]:
```
1 df['Manager_ID'].unique()
```

Out[26]:
```
array([303, 301, 304, 302, 305], dtype=int64)
```

In [27]:
```
# 5. Merging & Joining
managers = pd.DataFrame({"Manager_ID": [301, 302, 303], "Manager_Name": ["Jo
merged_df = df.merge(managers, on="Manager_ID", how="left")
merged_df
```

Out[27]:

	Employee_ID	Name	Department	Salary	Join_Date	Performance_Score	Bonus	Loca
0	101	Employee_1	HR	103828.0	2015-01-01	98.0	9454.0	Bo
1	102	Employee_2	IT	104145.0	2015-01-31	78.0	6632.5	
2	103	Employee_3	Operations	58164.0	2015-03-02	79.0	5765.0	Bo
3	104	Employee_4	Finance	56490.0	2015-04-01	91.0	9691.0	
4	105	Employee_5	HR	85256.0	2015-05-01	95.0	3749.0	Chi
...	...	...	...	...	...	...	...	
95	196	Employee_96	Operations	84128.0	2022-10-21	80.0	8583.0	Ang
96	197	Employee_97	HR	104323.0	2022-11-20	64.0	6632.5	
97	198	Employee_98	Operations	84371.0	2022-12-20	94.0	6922.0	
98	199	Employee_99	Operations	62793.0	2023-01-19	69.0	4290.0	
99	200	Employee_100	Operations	108923.0	2023-02-18	71.0	6307.0	

100 rows × 12 columns