untitled3

September 1, 2024

1 PANDAS IN DATA SCIENCE

[ ]:
```
Getting Familiar with Pandas:
```

Pandas is a powerful and widely-used library in Python for data manipulation and analysis. It provides data structures like Series and DataFrame that make it easy to work with structured data, such as tables or time series

It provides two primary data structures: Series,Dataframe

With Pandas, you can easily: Read and write data from files (CSV, Excel, SQL, etc.). Filter, sort, and modify your data. Handle missing data by filling or removing it. Group and aggregate data for summary statistics. Merge and join datasets.

1.1 HOW TO IMPORT PANDAS

[1]:
```
import pandas as pd
```

1.2 Pandas Data Structures: DataFrames and Series

[ ]:
```
Series:
A one-dimensional labeled array.
Can hold any data type (integers, floats, strings, etc.).
Similar to a single column in a spreadsheet.

DataFrame:
A two-dimensional, labeled data structure.
Essentially a table with rows and columns, where each column is a Series.
Can be thought of as a collection of Series objects.
```

1.3 Creating Series and DataFrames

1.3.1 creating series

[5]:
```
#From a List

data = [10, 20, 30, 40, 50]
series = pd.Series(data)
print(series)
```

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

[7]:
```
# From a Dictionary:

data = {'a': 10, 'b': 20, 'c': 30}
series = pd.Series(data)
print(series)
```

```
a    10
b    20
c    30
dtype: int64
```

1.3.2 2. Creating a DataFrame:

[9]:
```
#From a Dictionary:

data = {
    'Name': ['VINU', 'VARMA', 'RACHI'],
    'Age': [19, 18, 45],
    'City': ['VSP', 'VJD', 'AKP']
}
df = pd.DataFrame(data)
print(df)
```

```
    Name  Age City
0   VINU   19  VSP
1  VARMA   18  VJD
2  RACHI   45  AKP
```

[11]:
```
#From a List of Lists:

data = [
    ['VINU', 19, 'VSP'],
    ['VARMA', 18, 'VJD'],
    ['RACHI', 45, 'AKP']
```

```
]
df = pd.DataFrame(data, columns=['Name', 'Age', 'City'])
print(df)
```

```
    Name  Age City
0   VINU   19  VSP
1  VARMA   18  VJD
2  RACHI   45  AKP
```

1.3.3 Creating a DataFrame from a CSV File

[13]:
```
# Load data from a CSV file into a DataFrame
df = pd.read_csv('data.csv')

# Display the DataFrame
print(df)
```

```
     Name  Age        City
0    Vinu   19         Vsp
1   varma   18  vijayawada
2   Rachi   47         Akp
3  dinesh   20       vizag
```

1.4 Selecting Data

Selecting Columns:

[15]:
```
df = pd.DataFrame({
    'Name': ['A', 'B', 'C'],
    'Age': [25, 30, 35],
    'City': ['MUMBAI', 'HYDERABAD', 'DELHI']
})

# Select the 'Name' column
names = df['Name']
print(names)
```

```
0    A
1    B
2    C
Name: Name, dtype: object
```

[17]:
```
# MULTIPLE COLUMNS

name_age = df[['Name', 'Age']]
print(name_age)
```

```
  Name  Age
```

```
0    A   25
1    B   30
2    C   35
```

1.5 Selecting Rows:

1.5.1 Using .iloc[] (Index-based Selection):

[19]:
```
first_row = df.iloc[0]
print(first_row)
```

```
Name         A
Age         25
City    MUMBAI
Name: 0, dtype: object
```

[21]:
```
mul_rows = df.iloc[0:2]
print(mul_rows)
```

```
  Name  Age       City
0    A   25     MUMBAI
1    B   30  HYDERABAD
```

1.5.2 2. Filtering Rows

[31]:
```
# Filter rows where 'Age' is greater than 30
filtered_df = df[df['Age'] > 30]
print(filtered_df)
```

```
      Age   City
Name
C      35  DELHI
```

[33]:
```
# Filter rows where 'Age' is greater than 30 and 'City' is 'Los Angeles'
filtered_df = df[(df['Age'] > 30) & (df['City'] == 'DELHI')]
print(filtered_df)
```

```
      Age   City
Name
C      35  DELHI
```

1.6 Modifying Data

[35]:
```
df['Salary'] = [50000, 60000, 70000]
print(df)
```

```
      Age       City  Salary
Name
A      25     MUMBAI   50000
```

```
B      30  HYDERABAD   60000
C      35      DELHI   70000
```

[61]:
```
df = pd.read_csv('wearable_tech_sleep_quality_1.csv')
```

[63]:
```
# Display the first few rows of the dataset
print("Original Data:")
print(df.head())
```

```
Original Data:
   Heart_Rate_Variability  Body_Temperature  Movement_During_Sleep  \
0               79.934283         37.199678               1.324822
1               67.234714         36.962317               1.855481
2               82.953771         36.529815               1.207580
3              100.460597         36.176532               1.692038
4               65.316933         36.849112               0.106385

   Sleep_Duration_Hours  Sleep_Quality_Score  Caffeine_Intake_mg  \
0              4.638289                  1.0          107.624032
1              6.209422                  1.0          104.658589
2              6.879592                 10.0            0.000000
3             10.331531                  1.0          116.990981
4              8.334830                  1.0          223.282908

   Stress_Level  Bedtime_Consistency  Light_Exposure_hours
0      2.771837             0.657037              7.933949
1      3.738138             0.144464              6.992699
2      3.115880             0.642949              7.655250
3      3.904008             0.453255              9.429463
4      4.571699             0.641492             10.555713
```

[65]:
```
# Step 2: Handle Missing Data
# 2.1: Check for missing data
print("\nMissing Data Check:")
print(df.isnull().sum())
```

```
Missing Data Check:
Heart_Rate_Variability    0
Body_Temperature          0
Movement_During_Sleep     0
Sleep_Duration_Hours      0
Sleep_Quality_Score       0
Caffeine_Intake_mg        0
Stress_Level              0
Bedtime_Consistency       0
Light_Exposure_hours      0
dtype: int64
```

[87]:
```python
# Handle missing values
df_filled = df.fillna(df.mean(numeric_only=True))
```

[89]:
```python
print("\nDataFrame after handling missing values:")
print(df_filled.head())
```

```
DataFrame after handling missing values:
   Heart_Rate_Variability  Body_Temperature  Movement_During_Sleep  \
0               79.934283         37.199678               1.324822
1               67.234714         36.962317               1.855481
2               82.953771         36.529815               1.207580
3              100.460597         36.176532               1.692038
4               65.316933         36.849112               0.106385

   Sleep_Duration_Hours  Sleep_Quality_Score  Caffeine_Intake_mg  \
0              4.638289                  1.0          107.624032
1              6.209422                  1.0          104.658589
2              6.879592                 10.0            0.000000
3             10.331531                  1.0          116.990981
4              8.334830                  1.0          223.282908

   Stress_Level  Bedtime_Consistency  Light_Exposure_hours
0      2.771837             0.657037              7.933949
1      3.738138             0.144464              6.992699
2      3.115880             0.642949              7.655250
3      3.904008             0.453255              9.429463
4      4.571699             0.641492             10.555713
```

[91]:
```python
# Check columns in the DataFrame
print("\nColumns in DataFrame:")
print(df_filled.columns)

# Check the first few rows to confirm 'Date' column
print("\nFirst few rows of DataFrame:")
print(df_filled.head())
```

```
Columns in DataFrame:
Index(['Heart_Rate_Variability', 'Body_Temperature', 'Movement_During_Sleep',
       'Sleep_Duration_Hours', 'Sleep_Quality_Score', 'Caffeine_Intake_mg',
       'Stress_Level', 'Bedtime_Consistency', 'Light_Exposure_hours'],
      dtype='object')

First few rows of DataFrame:
   Heart_Rate_Variability  Body_Temperature  Movement_During_Sleep  \
0               79.934283         37.199678               1.324822
1               67.234714         36.962317               1.855481
```

```
2           82.953771         36.529815              1.207580
3          100.460597         36.176532              1.692038
4           65.316933         36.849112              0.106385

   Sleep_Duration_Hours  Sleep_Quality_Score  Caffeine_Intake_mg  \
0              4.638289                  1.0          107.624032
1              6.209422                  1.0          104.658589
2              6.879592                 10.0            0.000000
3             10.331531                  1.0          116.990981
4              8.334830                  1.0          223.282908

   Stress_Level  Bedtime_Consistency  Light_Exposure_hours
0      2.771837             0.657037              7.933949
1      3.738138             0.144464              6.992699
2      3.115880             0.642949              7.655250
3      3.904008             0.453255              9.429463
4      4.571699             0.641492             10.555713
```

1.7 DATA TRANSFORMATION

[93]:
```python
# For example, let's add a new column that calculates sleep efficiency (assuming we have sleep_duration and awake_duration columns)
if 'sleep_duration' in df_filled.columns and 'awake_duration' in df_filled.columns:
    df_filled['sleep_efficiency'] = (df_filled['sleep_duration'] - df_filled['awake_duration']) / df_filled['sleep_duration']
    df_filled['sleep_efficiency'] = df_filled['sleep_efficiency'].apply(lambda x: x * 100)  # convert to percentage
```

[95]:
```python
print("\nData After Transformation:")
print(df_filled.head())
```

```
Data After Transformation:
   Heart_Rate_Variability  Body_Temperature  Movement_During_Sleep  \
0               79.934283         37.199678               1.324822
1               67.234714         36.962317               1.855481
2               82.953771         36.529815               1.207580
3              100.460597         36.176532               1.692038
4               65.316933         36.849112               0.106385

   Sleep_Duration_Hours  Sleep_Quality_Score  Caffeine_Intake_mg  \
0              4.638289                  1.0          107.624032
1              6.209422                  1.0          104.658589
2              6.879592                 10.0            0.000000
3             10.331531                  1.0          116.990981
4              8.334830                  1.0          223.282908
```

```
   Stress_Level  Bedtime_Consistency  Light_Exposure_hours
0      2.771837             0.657037              7.933949
1      3.738138             0.144464              6.992699
2      3.115880             0.642949              7.655250
3      3.904008             0.453255              9.429463
4      4.571699             0.641492             10.555713
```

Pandas functions to clean and preprocess data, such as handling missing values, removing duplicates, and data type conversions.

[ ]:
```
Handling Missing Values
Removing Duplicates
Data Type Conversions
```

[97]:
```python
# Display initial data summary
print("Initial Data Summary:")
print(df.info())
print("\nInitial Data Sample:")
print(df.head())
```

```
Initial Data Summary:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Heart_Rate_Variability 1000 non-null   float64
 1   Body_Temperature       1000 non-null   float64
 2   Movement_During_Sleep  1000 non-null   float64
 3   Sleep_Duration_Hours   1000 non-null   float64
 4   Sleep_Quality_Score    1000 non-null   float64
 5   Caffeine_Intake_mg     1000 non-null   float64
 6   Stress_Level           1000 non-null   float64
 7   Bedtime_Consistency    1000 non-null   float64
 8   Light_Exposure_hours   1000 non-null   float64
dtypes: float64(9)
memory usage: 70.4 KB
None

Initial Data Sample:
   Heart_Rate_Variability  Body_Temperature  Movement_During_Sleep  \
0               79.934283         37.199678               1.324822
1               67.234714         36.962317               1.855481
2               82.953771         36.529815               1.207580
3              100.460597         36.176532               1.692038
4               65.316933         36.849112               0.106385

   Sleep_Duration_Hours  Sleep_Quality_Score  Caffeine_Intake_mg  \
```

```
0           4.638289                 1.0         107.624032
1           6.209422                 1.0         104.658589
2           6.879592                10.0           0.000000
3          10.331531                 1.0         116.990981
4           8.334830                 1.0         223.282908

   Stress_Level  Bedtime_Consistency  Light_Exposure_hours
0      2.771837             0.657037              7.933949
1      3.738138             0.144464              6.992699
2      3.115880             0.642949              7.655250
3      3.904008             0.453255              9.429463
4      4.571699             0.641492             10.555713
```

1.7.1 REMOVING DUPLICATES

[99]:
```python
# 2. Removing Duplicates

# Check for duplicate rows
duplicates = df_filled.duplicated().sum()
print(f"\nNumber of Duplicate Rows: {duplicates}")

# Remove duplicate rows
df_cleaned = df_filled.drop_duplicates()

print("\nData After Removing Duplicates:")
print(df_cleaned.head())
```

```
Number of Duplicate Rows: 0

Data After Removing Duplicates:
   Heart_Rate_Variability  Body_Temperature  Movement_During_Sleep  \
0               79.934283         37.199678               1.324822
1               67.234714         36.962317               1.855481
2               82.953771         36.529815               1.207580
3              100.460597         36.176532               1.692038
4               65.316933         36.849112               0.106385

   Sleep_Duration_Hours  Sleep_Quality_Score  Caffeine_Intake_mg  \
0              4.638289                  1.0          107.624032
1              6.209422                  1.0          104.658589
2              6.879592                 10.0            0.000000
3             10.331531                  1.0          116.990981
4              8.334830                  1.0          223.282908

   Stress_Level  Bedtime_Consistency  Light_Exposure_hours
0      2.771837             0.657037              7.933949
1      3.738138             0.144464              6.992699
```

```
2      3.115880             0.642949             7.655250
3      3.904008             0.453255             9.429463
4      4.571699             0.641492            10.555713
```

[ ]:
```
Pandas to perform data analysis, including generating summary statistics,
grouping data, and applying aggregate functions.
```

[ ]:
```
Generating Summary Statistics
Grouping Data
Applying Aggregate Functions
```

2 1. Generating Summary Statistics

[101]:
```python
# Summary statistics for numeric columns
print("Summary Statistics for Numeric Columns:")
print(df.describe())
```

```
Summary Statistics for Numeric Columns:
       Heart_Rate_Variability  Body_Temperature  Movement_During_Sleep  \
count             1000.000000       1000.000000            1000.000000
mean                70.386641         36.535418               2.005834
std                 19.584319          0.498727               0.983454
min                  5.174653         35.029806              -1.019512
25%                 57.048194         36.196879               1.352000
50%                 70.506012         36.531539               1.999749
75%                 82.958878         36.864441               2.660915
max                147.054630         38.096554               5.926238

       Sleep_Duration_Hours  Sleep_Quality_Score  Caffeine_Intake_mg  \
count           1000.000000          1000.000000         1000.000000
mean               7.471921             2.592946          148.260148
std                1.540699             2.979500           94.031760
min                3.105827             1.000000            0.000000
25%                6.393869             1.000000           80.630719
50%                7.500277             1.000000          145.717293
75%                8.500418             2.537789          211.244685
max               12.364639            10.000000          400.000000

       Stress_Level  Bedtime_Consistency  Light_Exposure_hours
count   1000.000000          1000.000000           1000.000000
mean       4.940956             0.504222              8.036684
std        2.032708             0.204137              2.023371
min        0.000000             0.000000              0.326689
25%        3.489725             0.361569              6.726291
50%        4.890507             0.500996              8.038248
75%        6.399490             0.644680              9.354408
max       10.000000             1.000000             14.754766
```

3 2. Grouping Data

[121]:
```python
if 'Body_Temperature' in df.columns:
    grouped_data = df.groupby('Body_Temperature').mean()
    print("\nGrouped Data (Mean of Numeric Columns) by 'Body_Temperature':")
    print(grouped_data)
```

```
Grouped Data (Mean of Numeric Columns) by 'Body_Temperature':
                  Heart_Rate_Variability  Movement_During_Sleep  \
Body_Temperature
35.029806                      56.378967               1.931366
35.039325                      60.446851               1.542698
35.051872                      61.587094               1.097948
35.063869                      58.462163               2.054934
35.075729                      66.286820               1.862628
…                                      …                      …
37.794782                      83.924127               2.902277
37.800842                      80.080930               1.284240
37.822172                      78.800289               2.876047
38.068874                      79.135064               1.763445
38.096554                      72.584424               2.668340

                  Sleep_Duration_Hours  Sleep_Quality_Score  \
Body_Temperature
35.029806                     5.846595             1.000000
35.039325                     8.406181             1.000000
35.051872                     8.783333             1.000000
35.063869                     7.259460             1.000000
35.075729                     7.000709             1.000000
…                                    …                    …
37.794782                     6.853312             1.000000
37.800842                     6.659089             6.832218
37.822172                     7.590129             1.000000
38.068874                     7.140747             1.000000
38.096554                     8.601956            10.000000

                  Caffeine_Intake_mg  Stress_Level  Bedtime_Consistency  \
Body_Temperature
35.029806                 150.075551      4.896916             0.628938
35.039325                 124.604829      2.644477             0.621590
35.051872                 200.168505      5.461222             0.474519
35.063869                 245.246197      2.953494             0.437812
35.075729                 254.691179      6.444894             0.619508
…                                  …             …                    …
37.794782                 243.864002      5.401397             0.288193
37.800842                  39.188754      3.761754             0.704325
```

```
37.822172                 317.344189      6.757928                0.462061
38.068874                 210.258027      8.612178                0.931824
38.096554                   0.000000      4.996572                0.283999

                  Light_Exposure_hours
Body_Temperature
35.029806                    10.958964
35.039325                     4.727194
35.051872                     6.642039
35.063869                     9.292259
35.075729                     8.301205
…                                  …
37.794782                     5.749604
37.800842                     6.750331
37.822172                     9.253754
38.068874                     7.285170
38.096554                     6.043954

[1000 rows x 8 columns]
```

3.1 AGGREGATING DATA

[123]:
```python
if 'Movement_During_Sleep' in df.columns:
    aggregated_data = df.groupby('Body_Temperature').agg({
        'Movement_During_Sleep': ['mean', 'sum', 'max', 'min']
    })
    print("\nAggregated Data (Mean, Sum, Max, Min) of 'Movement_During_Sleep'␣
↪by 'Body_Temperature':")
    print(aggregated_data)
```

```
Aggregated Data (Mean, Sum, Max, Min) of 'Movement_During_Sleep' by
'Body_Temperature':
                 Movement_During_Sleep
                                  mean       sum       max       min
Body_Temperature
35.029806                     1.931366  1.931366  1.931366  1.931366
35.039325                     1.542698  1.542698  1.542698  1.542698
35.051872                     1.097948  1.097948  1.097948  1.097948
35.063869                     2.054934  2.054934  2.054934  2.054934
35.075729                     1.862628  1.862628  1.862628  1.862628
…                                  …        …        …        …
37.794782                     2.902277  2.902277  2.902277  2.902277
37.800842                     1.284240  1.284240  1.284240  1.284240
37.822172                     2.876047  2.876047  2.876047  2.876047
38.068874                     1.763445  1.763445  1.763445  1.763445
38.096554                     2.668340  2.668340  2.668340  2.668340
```

```
[1000 rows x 4 columns]
```

3.2 advanced data manipulation techniques like merging, joining, and

concatenating DataFrames. for above data set

[125]:
```python
# Create sample DataFrames for demonstration
data1 = {
    'id': [1, 2, 3, 4],
    'sleep_duration': [7.5, 6.2, 8.0, 5.5],
    'steps': [10000, 8000, 12000, 6000]
}
data2 = {
    'id': [3, 4, 5, 6],
    'calories_burned': [300, 250, 350, 200],
    'heart_rate': [70, 65, 75, 60]
}


df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)
```

3.3 1. Merging DataFrames

[127]:
```python
merged_df = pd.merge(df1, df2, on='id', how='inner')  # Inner join
print("Merged DataFrame (inner join on 'id'):")
print(merged_df)
```

```
Merged DataFrame (inner join on 'id'):
   id  sleep_duration  steps  calories_burned  heart_rate
0   3             8.0  12000              300          70
1   4             5.5   6000              250          65
```

3.4 2. Joining DataFrames

[129]:
```python
# Set 'id' as the index for joining
df1.set_index('id', inplace=True)
df2.set_index('id', inplace=True)

# Join df1 with df2 on the index
joined_df = df1.join(df2, how='left')  # Left join
print("\nJoined DataFrame (left join on index):")
print(joined_df)
```

```

Joined DataFrame (left join on index):
    sleep_duration  steps  calories_burned  heart_rate
id
```

```
1               7.5  10000              NaN         NaN
2               6.2   8000              NaN         NaN
3               8.0  12000            300.0        70.0
4               5.5   6000            250.0        65.0
```

3.5 3. Concatenating DataFrames

[131]:
```
# Concatenate df1 and df2 horizontally (adding columns)
concat_horiz_df = pd.concat([df1, df2], axis=1)
print("\nConcatenated DataFrame (horizontally):")
print(concat_horiz_df)
```

```
Concatenated DataFrame (horizontally):
    sleep_duration    steps  calories_burned  heart_rate
id
1              7.5  10000.0              NaN         NaN
2              6.2   8000.0              NaN         NaN
3              8.0  12000.0            300.0        70.0
4              5.5   6000.0            250.0        65.0
5              NaN      NaN            350.0        75.0
6              NaN      NaN            200.0        60.0
```

4 Application in Data Science:

Pandas is a powerful library for data manipulation and analysis in Python, offering several advantages over traditional Python data structures for handling and analyzing data. Here's how the use of Pandas in the provided program can help a data science professional, and why it's beneficial compared to using native Python data structures:

4.1 Advantages of Using Pandas:

Efficient Data Handling: DataFrames and Series: Pandas introduces the DataFrame and Series objects, which are optimized for handling large datasets efficiently. These structures allow for easy manipulation of data with labels and are designed to handle data operations that would be cumbersome with lists or dictionaries.

Built-in Functions for Data Cleaning: Handling Missing Data: Pandas provides convenient methods like fillna(), dropna(), and isnull() to handle missing values. Removing Duplicates: With methods such as drop_duplicates(), Pandas simplifies the process of identifying and removing duplicate records.

Advanced Data Manipulation: Merging and Joining: Pandas supports sophisticated data merging and joining operations through functions like merge() and join() Concatenation: Using concat(), Pandas can efficiently concatenate DataFrames either vertically or horizontally. This capability is essential for combining multiple datasets and performing batch processing.

Data Aggregation and Analysis: GroupBy and Aggregation: Pandas allows for easy grouping of data using groupby() and performing aggregate functions such as mean(), sum(), and count().

This functionality enables data scientists to perform complex data analysis and summarization with concise code, rather than implementing these operations from scratch.

Flexible Data Transformation: Data Type Conversion: With methods like astype() and pd.to_datetime(), Pandas facilitates the conversion of data types, making it easier to ensure that data is in the correct format for analysis. Summary Statistics: The describe() method provides a quick way to generate summary statistics, helping data scientists to understand the distribution and characteristics of their data at a glance.

4.1.1 real-world examples where Pandas is essential, such as in data cleaning,

exploratory data analysis (EDA)

Pandas is a cornerstone of data science and is essential in many real-world scenarios. Here are some examples where Pandas proves invaluable, particularly in data cleaning and exploratory data analysis (EDA):

1. Data Cleaning Example: Financial Transactions Data Scenario: A financial institution receives daily transaction data from multiple sources, including internal systems and external partners. The data includes transaction amounts, timestamps, and account information. Challenges: Missing Values: Some transactions may have missing amounts or incomplete timestamps. Inconsistent Data Formats: Date and time formats might vary across different sources. Duplicate Records: Duplicate transactions could be recorded due to system errors. How Pandas Helps: Handling Missing Values: Pandas' fillna() and dropna() functions can be used to fill in or remove missing data. For example, df.fillna(df['amount'].mean()) fills missing amounts with the average transaction amount.

2. Exploratory Data Analysis (EDA) Example: E-Commerce Sales Analysis Scenario: An e-commerce company wants to understand its sales performance over the past year. The dataset includes sales transactions, product categories, customer information, and timestamps. Challenges: Summary Statistics: Understanding the central tendencies and dispersion of sales data. Group Analysis: Analyzing sales performance by product category or region. Trend Analysis: Identifying trends and patterns over time. How Pandas Helps: Generating Summary Statistics: Use df.describe() to obtain summary statistics like mean, median, and standard deviation for numeric columns, which helps in understanding overall sales performance. Grouping and Aggregation: df.groupby('category').agg({'sales_amount': ['mean', 'sum']}) aggregates sales data by product category, showing the average and total sales per category. Trend Analysis: Use df.groupby(df['date'].dt.to_period('M')).sum() to analyze monthly sales trends, helping to identify seasonal patterns and growth over time.