Lesson 4 Assignment

Data Processing - Python Development and Pandas

Copy each task (including'Your code here:') to a different cell and complete the codes for each task. Add your own comments where necessary

Remember to push your changes to github and submit a PDF copy of your completed notebook with the outputs.

```
# Python and Pandas Assessment Notebook

import pandas as pd
import numpy as np
from functools import reduce

# Task 1: Create a DataFrame using Pandas
# Create a DataFrame with the following data:
# Name: Alice, Bob, Charlie, David
# Age: 25, 30, 35, 28
# City: New York, San Francisco, Los Angeles, Chicago

# Your code here:

data= {'Name':['Alice','Bob','Charlie','David'],'Age':[25,30,35,28],
       'City':['New York','San Francisco','Los Angeles', 'Chicago']}
print("-" * 50)
#convert to dataframe

df = pd.DataFrame(data)
display(df)
print("-" * 50)
# Handling Potential Errors:

try:
   print(df['Gender'])
except KeyError as e:
   print(f"Error:{e}, the column 'Gender' doesn't exist")
print("-" * 50)

# Task 2: Row and Column Manipulation
# Drop the 'City' column from the DataFrame created in Task 1

# Your code here:

try:
  df_modified = df.drop(columns=['City'])
  print(" The column 'City' has successfully been dropped")
except KeyError as e:
  print(f"Error:{e},the column 'City' doesn't exist")
display(df)
print("-" * 50)

# Verify the resulting columns
print("\n Columns after dropping 'City':", list(df_modified.columns))
print("-" * 50)

# Display the modified DataFrame
print("\n Updated DataFrame:")
print(df_modified)
print("-" * 50)

# Task 3: Handling Null Values
# Create a new DataFrame with some null values and fill them

# Your code here:

data_null= {'Name':['Alice','None','Charlie','David'],'Age':[25,30,None,28],
       'City':['New York','San Francisco','None', 'Chicago']}

data_null_df = pd.DataFrame(data_null)
print(" Modified DataFrame with Null Values:")
display(data_null_df)
```

```
print("-" * 50)

# --- Method 1) Fill nulls with a specific value per column type ---
df_filled = data_null_df
df_filled["Name"] = df_filled["Name"].fillna("Unknown")
df_filled["Age"] = df_filled["Age"].fillna(0)
df_filled["City"] = df_filled["City"].fillna("Not Provided")

print("\n Method 1: Filled Null Values (Custom Replacement):")
print(df_filled)
print("-" * 50)

# --- Method 2: Drop rows with any null values ---
df_dropped = data_null_df.dropna
print("\n Method 2: Dropped Rows Containing Any Nulls:")
print(df_dropped)


# --- Method 3: Fill using forward fill (propagate previous value) ---
df_ffill = data_null_df.fillna(method='ffill')
print("\ Method 3: Forward Fill (uses previous row's value):")
display(df_ffill)
print("-" * 50)

# The 4 method is backfill.

# Task 4: GroupBy and Describe
# Using the following DataFrame, group by 'Category' and describe the 'Value' column

df_group = pd.DataFrame({
    'Category': ['A', 'B', 'A', 'B', 'A', 'C'],
    'Value': [10, 20, 15, 25, 30, 35]
})
print("-" * 50)

# Your code here:

print("Original DataFrame:")
data = pd.DataFrame(df_group) # coverted data into dataframe
display(data)
print("-" * 50)

df_group = pd.DataFrame(data)
grouped_df = data.groupby('Category') # groupd the categ col into df

summary_stats = grouped_df['Value'].describe() # provide stats of value col

print("Grouped DataFrame:")
display(grouped_df)
print("-" * 50)
print("Grouped DataFrame:")
display(summary_stats)
display(grouped_df.describe())
print("-" * 50)

# Task 5: Concatenation and Merging
# Concatenate two DataFrames vertically and merge them horizontally

df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})
df3 = pd.DataFrame({'C': [9, 10], 'D': [11, 12]})

# Your code here:

# Create sample DataFrames
df1 = pd.DataFrame({
    "ID": [1, 2, 3],
    "Name": ["Alice", "Bob", "Charlie"],
    "Score": [85, 90, 88]
})

df2 = pd.DataFrame({
    "ID": [4, 5, 6],
    "Name": ["Diana", "Ethan", "Fiona"],
    "Score": [92, 78, 95]
})

```

```
df3 = pd.DataFrame({
    "ID": [1, 2, 3, 4, 5, 6],
    "City": ["Toronto", "Ottawa", "Vancouver", "Calgary", "Montreal", "Halifax"]
})
print("-" * 50)
# --- Step 1: Concatenate df1 and df2 vertically ---
df_concat = pd.concat([df1, df2], axis=0, ignore_index=True)
print("Step 1: Concatenated DataFrame (df1 + df2):")
print(df_concat)
print("-" * 50)
# --- Step 2: Merge the concatenated result with df3 horizontally ---
df_merged = pd.merge(df_concat, df3, on="ID", how="inner")
display(df_merged)

print("\n Step 2: Merged DataFrame (with df3):")
print(df_merged)
print("-" * 50)
# --- Step 3: Verify the shape and columns ---
print("\n Shape of the final DataFrame:", df_merged.shape)
print(" Columns:", list(df_merged.columns))

# Task 6: Tuples and Sets
# Create a tuple of fruits and a set of numbers.
# Then, try to add an element to each and observe the difference.

# Your code here:

# Create a tuple of fruits
fruits = ("Apple", "Banana", "Cherry")

# Create a set of numbers
numbers = {10, 20, 30, 40, 50}

display("Original Tuple:", fruits)
display("Original Set:", numbers)

print("-" * 50)

# Attempt to add a new element to both
print("\nAttempting to add a new element...")

try:
    fruits.append("Mango") # Adding to tuple (should fail)
except AttributeError as e:
    print(f"Error when adding to tuple: {e}")

numbers.add(60) ## Adding to set (should succeed)
print("Successfully added 60 to set:", numbers)
print("-" * 50)
# Task 7: Dictionaries
# Create a dictionary of student names and their scores.
# Then, update a student's score and add a new student.

# Your code here:

students = {
    "Alice": 85,
    "Bob": 90,
    "Charlie": 78
}

print("Original Dictionary:")
display(students)
print("-" * 50)

# Update the score of an existing student (e.g., Bob)
students["Bob"] = 95
print("Updated Bob's score:")
print(students)
print("-" * 50)

# Add a new student and their score
students["Diana"] = 88
print("Added a new student (Diana):")
print(students)
print("-" * 50)
```

```
# Task 8: Functions and Lambda
# Create a function to calculate the square of a number.
# Then, use a lambda function to do the same.

# Your code here:

# Define a regular function
def square(num):
    """Return the square of a number."""
    return num ** 2

# Define a lambda function for the same task
square_lambda = lambda x: x ** 2

# Test both functions with two inputs
inputs = [4, 7]

print("Regular Function Results:")
for i in inputs:
    print(f"The square of {i} is {square(i)}")

print("-" * 50)

print("Lambda Function Results:")
for i in inputs:
    print(f"The square of {i} is {square_lambda(i)}")

print("-" * 50)

# Task 9: Iterators and Generators
# Create an iterator for the first 5 even numbers.
# Then, create a generator for the same.

# Your code here:

# --- Part 1: Iterator Class ---
class EvenIterator:
    def __init__(self, limit=5): # Iterator that generates the first 5 even numbers."""
        self.limit = limit
        self.count = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.count < self.limit:
            result = self.count * 2
            self.count += 1
            return result
        else:
            raise StopIteration  # this indicates the end of the sequence

# --- Part 2: Generator Function ---
def even_generator(limit=5):  # Generator that yields the first 5 even numbers."""

    for i in range(limit):
        yield i * 2

# --- Part 3: Demonstration ---

print("Using Iterator Class:")
iterator = EvenIterator()
for num in iterator:
    print(num, end=" ")

print("\n" + "-" * 50)

print("Using Generator Function:")
for num in even_generator():
    print(num, end=" ")

print("\n" + "-" * 50)

# Task 10: Map, Reduce, and Filter
# Use map to square all numbers in a list.
# Use reduce to find the product of all numbers in a list.
# Use filter to get only even numbers from a list.

```

```
numbers = [1, 2, 3, 4, 5]

# Your code here:

# Define a list of numbers
numbers = [1, 2, 3, 4, 5, 6]

print("Original List:", numbers)
print("-" * 50)

# 1) Use map() to square all numbers ---
squared = list(map(lambda x: x ** 2, numbers))
print("Squared Numbers (map):", squared)
print("-" * 50)

# 2) Use reduce() to find the product of all numbers ---
product = reduce(lambda x, y: x * y, numbers)
print("Product of All Numbers (reduce):", product)

print("-" * 50)

#3) Use filter() to get only even numbers ---
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print("Even Numbers (filter):", even_numbers)
print("-" * 50)

# Task 11: Object-Oriented Programming - Creating a Class
# Create a class called 'Rectangle' with attributes 'length' and 'width'.
# Include methods to calculate area and perimeter.

# Your code here:

class Rectangle:

    def __init__(self, length, width): #Initialize the rectangle with length and width.
        self.length = length
        self.width = width

    def area(self): #"""Return the area of the rectangle."""
        return self.length * self.width

    def perimeter(self): #Return the perimeter of the rectangle."""
        return 2 * (self.length + self.width)

    def display(self):  #Display rectangle details
       print(f"Rectangle Length: {self.length}, Width: {self.width}, "
       f"Area: {self.area()}, Perimeter: {self.perimeter()}")

# Create Rectangle Objects

rect1 = Rectangle(10, 5)
rect2 = Rectangle(7, 3)

#Demonstration of their use
print("Demonstrating Rectangle Objects:")
print("-" * 50)

rect1.display()
print("-" * 50)

rect2.display()
print("-" * 50)

# Accessing attributes and methods directly
print(f"Area of rect1: {rect1.area()} | Perimeter: {rect1.perimeter()}")
print(f"Area of rect2: {rect2.area()} | Perimeter: {rect2.perimeter()}")
print("-" * 50)


# Task 12: Pandas Data Analysis
# Using the following DataFrame, perform these tasks:
# 1. Find the average salary by department
# 2. Get the names of employees with salary > 60000
# 3. Add a new column 'Bonus' which is 10% of salary

df_employees = pd.DataFrame({
    'Name': ['John', 'Jane', 'Bob', 'Alice', 'Charlie'],
```

```
    'Department': ['IT', 'HR', 'IT', 'Finance', 'HR'],
    'Salary': [55000, 65000, 70000, 60000, 58000]
})

# Your code here:

# Use of the given sample DataFrame
data = {
    "Name": ["Alice", "Bob", "Charlie", "Diana", "Ethan", "Fiona"],
    "Department": ["HR", "IT", "Finance", "HR", "IT", "Finance"],
    "Salary": [55000, 72000, 68000, 48000, 95000, 61000]
}

df_employees = pd.DataFrame(data)

print("Original Employee DataFrame:")
#print(df_employees)
#print("-" * 60)


# 1) Calculate the average salary by department ---
avg_salary = df_employees.groupby("Department")["Salary"].mean()

print("Average Salary by Department:")
print(avg_salary)
print("-" * 60)

#2) Extract names of employees with salary > 60000 ---
high_salary_empl = df_employees[df_employees["Salary"] > 60000][["Name", "Salary"]]

print("Employees with Salary > 60000:")
print(high_salary_empl)
print("-" * 60)

#3) Add a new column 'Bonus' = 10% of salary ---
df_employees["Bonus"] = df_employees["Salary"] * 0.10

print("Added 'Bonus' Column (10% of Salary):")
print(df_employees)
print("-" * 60)

#4) Verify DataFrame Info ---
print("DataFrame Info:")
print(df_employees.info())
print("-" * 60)
```

```
--------------------------------------------------
<>:85: SyntaxWarning: invalid escape sequence '\ '
<>:85: SyntaxWarning: invalid escape sequence '\ '
/tmp/ipython-input-3603158864.py:85: SyntaxWarning: invalid escape sequence '\ '
  print("\ Method 3: Forward Fill (uses previous row's value):")
```

	Name	Age	City
0	Alice	25	New York
1	Bob	30	San Francisco
2	Charlie	35	Los Angeles
3	David	28	Chicago

```
--------------------------------------------------
Error:'Gender', the column 'Gender' doesn't exist
--------------------------------------------------
 The column 'City' has successfully been dropped
```

	Name	Age	City
0	Alice	25	New York
1	Bob	30	San Francisco
2	Charlie	35	Los Angeles
3	David	28	Chicago

```
--------------------------------------------------

 Columns after dropping 'City': ['Name', 'Age']
--------------------------------------------------

 Updated DataFrame:
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   35
3    David   28
--------------------------------------------------
 Modified DataFrame with Null Values:
```

	Name	Age	City
0	Alice	25.0	New York
1	None	30.0	San Francisco
2	Charlie	NaN	None
3	David	28.0	Chicago

```
--------------------------------------------------

 Method 1: Filled Null Values (Custom Replacement):
      Name   Age           City
0    Alice  25.0       New York
1     None  30.0  San Francisco
2  Charlie   0.0           None
3    David  28.0        Chicago
--------------------------------------------------

 Method 2: Dropped Rows Containing Any Nulls:
<bound method DataFrame.dropna of       Name   Age           City
0    Alice  25.0       New York
1     None  30.0  San Francisco
2  Charlie   0.0           None
3    David  28.0        Chicago>
\ Method 3: Forward Fill (uses previous row's value):
/tmp/ipython-input-3603158864.py:84: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future ve
  df_ffill = data_null_df.fillna(method='ffill')
```

	Name	Age	City
0	Alice	25.0	New York
1	None	30.0	San Francisco
2	Charlie	0.0	None
3	David	28.0	Chicago

```
--------------------------------------------------
--------------------------------------------------
Original DataFrame:
```

	Category	Value
0	A	10

1	B	20
2	A	15
3	B	25
4	A	30
5	C	35

```
--------------------------------------------------
Grouped DataFrame:
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7d5ab8d3e1e0>
--------------------------------------------------
Grouped DataFrame:
```

	count	mean	std	min	25%	50%	75%	max
Category								
A	3.0	18.333333	10.408330	10.0	12.50	15.0	22.50	30.0
B	2.0	22.500000	3.535534	20.0	21.25	22.5	23.75	25.0
C	1.0	35.000000	NaN	35.0	35.00	35.0	35.00	35.0

	Value							
	count	mean	std	min	25%	50%	75%	max
Category								
A	3.0	18.333333	10.408330	10.0	12.50	15.0	22.50	30.0
B	2.0	22.500000	3.535534	20.0	21.25	22.5	23.75	25.0
C	1.0	35.000000	NaN	35.0	35.00	35.0	35.00	35.0

```
--------------------------------------------------
--------------------------------------------------
Step 1: Concatenated DataFrame (df1 + df2):
   ID     Name  Score
0   1    Alice     85
1   2      Bob     90
2   3  Charlie     88
3   4    Diana     92
4   5    Ethan     78
5   6    Fiona     95
--------------------------------------------------
```

	ID	Name	Score	City
	1			
	2			
	3			
2	3	Charlie	88	Vancouver
3	4	Diana	92	Calgary
4	5	Ethan	78	Montreal
5	6	Fiona	95	Halifax

```
 Step 2: Merged DataFrame (with df3):
   ID     Name  Score       City
0   1    Alice     85    Toronto
1   2      Bob     90     Ottawa
2   3  Charlie     88  Vancouver
3   4    Diana     92    Calgary
4   5    Ethan     78   Montreal
5   6    Fiona     95    Halifax
--------------------------------------------------

 Shape of the final DataFrame: (6, 4)
 Columns: ['ID', 'Name', 'Score', 'City']
'Original Tuple:'
('Apple', 'Banana', 'Cherry')
'Original Set:'
{10, 20, 30, 40, 50}
```