```
# 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

data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 28],
    'City': ['New York', 'San Francisco', 'Los Angeles', 'Chicago']
}
df = pd.DataFrame(data)
print("Task 1 - Original DataFrame:")
print(df)
```

```
Task 1 - Original DataFrame:
      Name  Age           City
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   35    Los Angeles
3    David   28        Chicago
```

Row and Column Manipulation

```
# Drop the 'City' column from the DataFrame created in Task 1
df_dropped = df.drop('City', axis=1)
print("\nTask 2 - DataFrame after dropping 'City' column:")
print(df_dropped)
```

```
Task 2 - DataFrame after dropping 'City' column:
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   35
3    David   28
```

Handling Null Values

Handling Null Values

```
# Create a new DataFrame with some null values and fill them
df_with_nulls = pd.DataFrame({
    'A': [1, np.nan, 3],
```

```
    'B': [np.nan, 5, 6],
    'C': [7, 8, np.nan]
})
print("\nTask 3 - DataFrame with nulls:")
print(df_with_nulls)
```

```
Task 3 - DataFrame with nulls:
     A    B    C
0  1.0  NaN  7.0
1  NaN  5.0  8.0
2  3.0  6.0  NaN
```

```
# Fill nulls with different methods
df_filled = df_with_nulls.fillna({'A': 0, 'B': df_with_nulls['B'].mean(), 'C': 'missing'})
print("\nDataFrame after filling nulls:")
print(df_filled)
```

```
DataFrame after filling nulls:
     A    B        C
0  1.0  5.5      7.0
1  0.0  5.0      8.0
2  3.0  6.0  missing
```

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("\nTask 4 - Grouped description:")
grouped = df_group.groupby('Category')['Value'].describe()
print(grouped)
```

```
Task 4 - Grouped description:
          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
```

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]})
# Vertical concatenation
df_concat = pd.concat([df1, df2], axis=0)
```

```
print("\nTask 5 - Vertically concatenated DataFrame:")
print(df_concat)
```

```
Task 5 - Vertically concatenated DataFrame:
   A  B
0  1  3
1  2  4
0  5  7
1  6  8
```

```
# Horizontal merge (assuming index alignment)
df_merged = pd.concat([df_concat.reset_index(drop=True), df3], axis=1)
print("\nHorizontally merged DataFrame:")
print(df_merged)
```

```
Horizontally merged DataFrame:
   A  B     C     D
0  1  3   9.0  11.0
1  2  4  10.0  12.0
2  5  7   NaN   NaN
3  6  8   NaN   NaN
```

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.
fruits_tuple = ('apple', 'banana', 'cherry')
numbers_set = {1, 2, 3, 4, 5}

print("\nTask 6:")
print("Original tuple:", fruits_tuple)
print("Original set:", numbers_set)

# Attempt to modify them (will raise error for tuple)
try:
    fruits_tuple += ('orange',)
except TypeError as e:
    print("Cannot modify tuple:", e)

numbers_set.add(6)
print("Modified set:", numbers_set)
```

```
Task 6:
Original tuple: ('apple', 'banana', 'cherry')
Original set: {1, 2, 3, 4, 5}
Modified set: {1, 2, 3, 4, 5, 6}
```

Dictionaries

```
# Create a dictionary of student names and their scores.
# Then, update a student's score and add a new student.
```

```
student_scores = {
    'John': 85,
    'Sarah': 92,
    'Mike': 78
}

print("\nTask 7 - Original dictionary:")
print(student_scores)

# Update a score
student_scores['Mike'] = 82
# Add new student
student_scores['Emma'] = 95

print("Updated dictionary:")
print(student_scores)
```

```
Task 7 - Original dictionary:
{'John': 85, 'Sarah': 92, 'Mike': 78}
Updated dictionary:
{'John': 85, 'Sarah': 92, 'Mike': 82, 'Emma': 95}
```

Functions and Lambda

```
# Create a function to calculate the square of a number.
# Then, use a lambda function to do the same.
def square_func(x):
    return x ** 2

square_lambda = lambda x: x ** 2

print("\nTask 8:")
print("Using function:", square_func(5))
print("Using lambda:", square_lambda(5))
print("Using function:", square_func(3.5))
print("Using lambda:", square_lambda(3.5))
```

```
Task 8:
Using function: 25
Using lambda: 25
Using function: 12.25
Using lambda: 12.25
```

Iterators and Generators

```
# Create an iterator for the first 5 even numbers.
# Then, create a generator for the same.
class EvenNumbersIterator:
    def __init__(self, limit):
        self.current = 0
```

```
        self.limit = limit

    def __iter__(self):
        return self

    def __next__(self):
        if self.current >= self.limit:
            raise StopIteration
        result = self.current
        self.current += 2
        return result

def even_numbers_generator(limit):
    current = 0
    while current < limit:
        yield current
        current += 2

print("\nTask 9 - Using iterator:")
even_iter = EvenNumbersIterator(10)
for num in even_iter:
    print(num, end=' ')

print("\nUsing generator:")
for num in even_numbers_generator(10):
    print(num, end=' ')
```

```
Task 9 - Using iterator:
0 2 4 6 8
Using generator:
0 2 4 6 8
```

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]

squared = list(map(lambda x: x**2, numbers))
product = reduce(lambda x, y: x * y, numbers)
evens = list(filter(lambda x: x % 2 == 0, numbers))

print("\n\nTask 10:")
print("Squared numbers:", squared)
print("Product of numbers:", product)
print("Even numbers:", evens)
```

```
Task 10:
Squared numbers: [1, 4, 9, 16, 25]
```

```
Product of numbers: 120
Even numbers: [2, 4]
```

Object-Oriented Programming - Creating a Class

```
# Create a class called 'Rectangle' with attributes 'length' and 'width'.
# Include methods to calculate area and perimeter.
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)

print("\nTask 11:")
rect1 = Rectangle(5, 3)
rect2 = Rectangle(7, 4)

print(f"Rectangle 1 - Area: {rect1.area()}, Perimeter: {rect1.perimeter()}")
print(f"Rectangle 2 - Area: {rect2.area()}, Perimeter: {rect2.perimeter()}")
```

```
Task 11:
Rectangle 1 - Area: 15, Perimeter: 16
Rectangle 2 - Area: 28, Perimeter: 22
```

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]
})

print("\nTask 12:")
# 1. Average salary by department
avg_salary = df_employees.groupby('Department')['Salary'].mean()
print("\nAverage salary by department:")
print(avg_salary)
```

```
Task 12:

Average salary by department:
Department
Finance    60000.0
```

```
HR         61500.0
IT         62500.0
Name: Salary, dtype: float64
```

. Employees with salary > 60000

```
# Employees with salary > 60000
high_earners = df_employees[df_employees['Salary'] > 60000]['Name']
print("\nEmployees with salary > 60000:")
print(high_earners)

# Add Bonus column
df_employees['Bonus'] = df_employees['Salary'] * 0.1
print("\nDataFrame with Bonus column:")
print(df_employees)
```

```
Employees with salary > 60000:
1    Jane
2     Bob
Name: Name, dtype: object

DataFrame with Bonus column:
      Name Department  Salary   Bonus
0     John         IT   55000  5500.0
1     Jane         HR   65000  6500.0
2      Bob         IT   70000  7000.0
3    Alice    Finance   60000  6000.0
4  Charlie         HR   58000  5800.0
```

https://colab.research.google.com/drive/1I5nJckSAe37SqJSSRHedhUZCrTPg5kPP

https://colab.research.google.com/drive/1I5nJckSAe37SqJSSRHedhUZCrTPg5kPP