1. Given a dataset in a CSV file, how would you read it into a Pandas DataFrame? And how would you handle missing values? import pandas as pd

```
import pandas as pd
df = pd.read_csv('your_dataset.csv')
# Check for missing values
missing_data = df.isna()
# Count missing values in each column
missing_count = df.isna().sum()
# Remove rows with missing values
df_clean = df.dropna()
# Fill missing values with the mean of the column
df_filled = df.fillna(df.mean())
```

2.Describe the difference between a list, a tuple, and a dictionary in Python. Provide an example for each.

Ans).the key differences between lists, tuples, and dictionaries in Python are their mutability, how they are defined, and how elements are accessed. Lists are mutable and defined with [], tuples are immutable and defined with (), and dictionaries are collections of key-value pairs defined with {} or dict().

```
# Creating a list of numbers
my_list = [1, 2, 3, 4, 5]
# Modifying an element
my_list[0] = 10
# Adding an element
my_list.append(6)
# Removing an element
my_list.remove(3)
print(my_list)
```

```
     [10, 2, 4, 5, 6]
```

```
# Creating a tuple of colors
my_tuple = ('red', 'green', 'blue')

# Accessing an element
color = my_tuple[0]  # Accessing 'red'

# Attempting to modify a tuple (this will raise an error)
# my_tuple[0] = 'yellow'  # TypeError: 'tuple' object does not support item assignment
```

```
 #Creating a dictionary of person's information
person = {
    'name': 'Alice',
    'age': 30,
    'city': 'New York'
}

# Accessing values by keys
person_name = person['name']  # Accessing 'Alice'

# Modifying a value
person['age'] = 31

# Adding a new key-value pair
person['job'] = 'Engineer'
print(person)
```

```
     {'name': 'Alice', 'age': 31, 'city': 'New York', 'job': 'Engineer'}
```

3.Imagine you are provided with two datasets, 'sales_data' and 'product_data', both in the form of Pandas DataFrames. How would you merge these datasets on a common column named 'ProductID'?

Ans) To merge two Pandas DataFrames, 'sales_data' and 'product_data,' on a common column named 'ProductID,' you can use the merge() function code:

```
import pandas as pd
merged_data = pd.merge(sales_data, product_data, on='ProductID')
```

4.How would you handle duplicate rows in a Pandas DataFrame? Write a Python code snippet to demonstrate.

Ans) we can use duplicated() and drop_duplicates() methods to find and remove duplicates

```
import pandas as pd
# Create a sample DataFrame with duplicate rows
data = {'Name': ['Alice', 'Bob', 'Alice', 'Charlie', 'Bob'],
        'Age': [25, 30, 25, 35, 30],
        'City': ['New York', 'Los Angeles', 'New York', 'Chicago', 'Los Angeles']}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
# Check for duplicate rows based on all columns
duplicates = df[df.duplicated()]
# Display the duplicate rows
print("\nDuplicate Rows:")
print(duplicates)
# Remove duplicate rows and keep the first occurrence
df_no_duplicates = df.drop_duplicates()
# Display the DataFrame with duplicates removed
print("\nDataFrame with Duplicates Removed:")
print(df_no_duplicates)
```

```
Original DataFrame:
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
2    Alice   25     New York
3  Charlie   35      Chicago
4      Bob   30  Los Angeles

Duplicate Rows:
    Name  Age         City
2  Alice   25     New York
4    Bob   30  Los Angeles

DataFrame with Duplicates Removed:
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
3  Charlie   35      Chicago
```

5.Describe the difference between '.iloc[] and '.loc[]' in the context of Pandas.

ANs) .iloc[] and .loc[] are both used for indexing and selecting data from a DataFrame or Series, but they have different ways of selecting data based on the index and label, iloc[] (integer-location based indexing):iloc[] is used for selecting data by row and column indices, where the indices are specified as integer positions. code:

```
import pandas as pd
data = {'A': [1, 2, 3],
        'B': [4, 5, 6],
        'C': [7, 8, 9]}
df = pd.DataFrame(data, index=['X', 'Y', 'Z'])

# Using .iloc[] to select data by integer positions
value_iloc = df.iloc[1, 2]  # Selects the value 8 based on row 1 and column 2 (zero-based indexing)

# Using .loc[] to select data by labels
value_loc = df.loc['Y', 'C']  # Selects the value 8 based on row label 'Y' and column label 'C'

print("Using .iloc[]:", value_iloc)
print("Using .loc[]:", value_loc)
```

```
Using .iloc[]: 8
Using .loc[]: 8
```

6. In Python's Matplotlib library, how would you plot a line chart to visualize monthly sales? Assume you have a list of months and a list of corresponding sales numbers.

Ans)

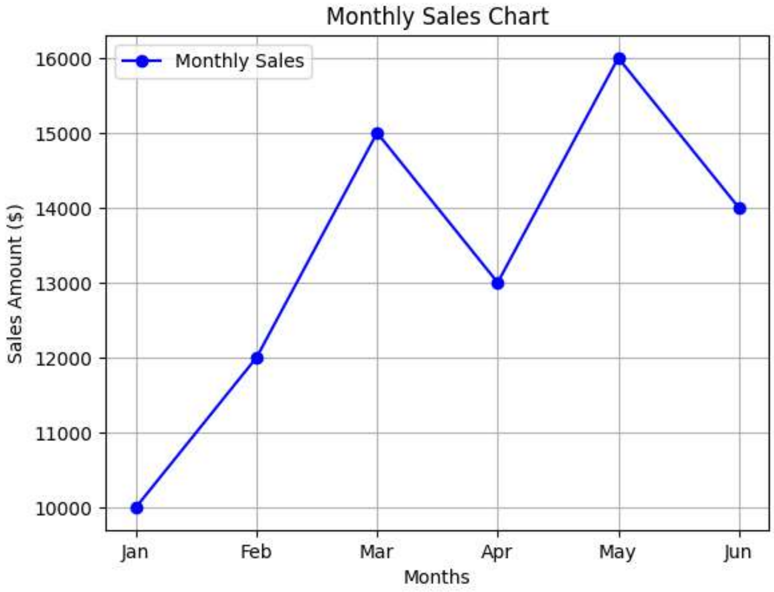
```
import matplotlib.pyplot as plt
```

```
# Data
months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun"]
sales = [10000, 12000, 15000, 13000, 16000, 14000]

# Create the line chart
plt.plot(months, sales, marker='o', linestyle='-', color='b', label='Monthly Sales')

# Customize the chart
plt.xlabel("Months")
plt.ylabel("Sales Amount ($)")
plt.title("Monthly Sales Chart")
plt.grid(True)
plt.legend()

# Display the chart or save it to a file
plt.show()
```



7. How would you use Python to connect to a SQL database and fetch data into a Pandas DataFrame?

ANs) We need to install sqlite3 library or even pyodbc library

```
import sqlite3
conn = sqlite3.connect('your_database.db')
cursor = conn.cursor()
import pandas as pd

# Define your SQL query
sql_query = "SELECT * FROM your_table_name"
df = pd.read_sql_query(sql_query, conn)
#close the database when done.
cursor.close()
conn.close()
```

8.Explain the concept of list comprehensions in Python. Can you provide an example where it's useful for data analysis?

Ans) List comprehensions are known for their readability and are often used in Python for tasks like data processing and analysis. List comprehensions are a concise way to create lists. They allow you to generate a new list by applying an expression to each item in an existing iterable, optionally applying a condition to filter items

It is useful where there is more complex data to analyize. They help to filter data, extract specific elements from a dataset, or transform data from one format to another, which can be especially valuable when working with large datasets

```
original_numbers = [1, 2, 3, 4, 5]
squares = [x ** 2 for x in original_numbers]

print(squares)
```

```
[1, 4, 9, 16, 25]
```

9.How would you reshape a long-format DataFrame to a wide format using Pandas? Explain with an example.

Ans) we can reshape a long format data frame using the "pivot" or "pivot_table" function

this involves: 1. converting data fro a tall 2. narrow structure where values for different categories

```
import pandas as pd

data = {'Month': ['Jan', 'Jan', 'Feb', 'Feb', 'Mar', 'Mar'],
        'Product': ['A', 'B', 'A', 'B', 'A', 'B'],
        'Sales': [100, 200, 150, 250, 120, 180]}

df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)
```

```
Original DataFrame:
  Month Product  Sales
0   Jan       A    100
1   Jan       B    200
2   Feb       A    150
3   Feb       B    250
4   Mar       A    120
5   Mar       B    180
```

```
# Reshape the DataFrame using pivot
wide_df = df.pivot(index='Month', columns='Product', values='Sales')

print("\nWide-Format DataFrame:")
print(wide_df)
```

```
Wide-Format DataFrame:
Product    A    B
Month
Feb      150  250
Jan      100  200
Mar      120  180
```

10.What are lambda functions in Python? How are they beneficial in data wrangling tasks?

ANs) Lambda functions, also known as anonymous functions or lambda expressions, are small, one-line functions in Python that can have any number of arguments but can only have one expression. They are defined using the "lambda" keyword

In data wrangling tasks, lambda functions are beneficial for several reasons:

1. Conciseness
2. Readability
3. Functional Programming like map() and filter

```
# Using lambda with map() to apply a function to each element of a list
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))
print(squared)


# Using lambda with filter() to filter elements from a list
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)

# Using lambda with sorted() to sort a list of dictionaries by a specific key
data = [{'name': 'Alice', 'age': 30}, {'name': 'Bob', 'age': 25}, {'name': 'Charlie', 'age': 35}]
sorted_data = sorted(data, key=lambda x: x['age'])

print(sorted_data)
```

```
[1, 4, 9, 16, 25]
[2, 4]
[{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}, {'name': 'Charlie', 'age': 35}]
```

11. Describe a scenario where you would use the 'groupby()' method in Pandas. How would you aggregate data after grouping?

Ans) Scenrio) : - Sales Analysis by Product Category and Quarter

Let's say that a sales dataset containing information about sales transactions, including the product category, sales amount, and the date of each transction We want to analyise sales performance by product category and quater(3 month)

We can do this using group() method

1.Load data

2. prepare data
3. group data
4. aggreagte data
5. Create a summary

```
import pandas as pd

data = {'Product Category': ['Electronics', 'Clothing', 'Electronics', 'Clothing'],
        'Sales Amount': [1000, 500, 800, 600],
        'Date': ['2022-01-15', '2022-04-25', '2022-07-10', '2022-10-05']}

sales_data = pd.DataFrame(data)

# Convert 'Date' to datetime and extract 'Quarter'
sales_data['Date'] = pd.to_datetime(sales_data['Date'])
sales_data['Quarter'] = sales_data['Date'].dt.to_period('Q')

# Group by 'Product Category' and 'Quarter' and calculate total sales
grouped_data = sales_data.groupby(['Product Category', 'Quarter'])
total_sales = grouped_data['Sales Amount'].sum()

# Create a summary DataFrame
summary_df = total_sales.unstack().fillna(0)

print(summary_df)
```

```
Quarter           2022Q1  2022Q2  2022Q3  2022Q4
Product Category
Clothing             0.0   500.0     0.0   600.0
Electronics       1000.0     0.0   800.0     0.0
```

12. You are provided with a Pandas DataFrame that contains a column with date strings. How would you convert this column to a datetime format? Additionally, how would you extract the month and year from these datetime objects?

Ans)

```
import pandas as pd

# Sample DataFrame with a 'Date' column containing date strings
data = {'Date': ['2023-01-15', '2023-02-20', '2023-03-10']}
df = pd.DataFrame(data)

# Convert 'Date' to datetime
df['Date'] = pd.to_datetime(df['Date'])

# Extract month and year
df['Month'] = df['Date'].dt.month
df['Year'] = df['Date'].dt.year

print(df)
```

```
        Date  Month  Year
0 2023-01-15      1  2023
1 2023-02-20      2  2023
2 2023-03-10      3  2023
```

13.Explain the purpose of the 'pivot_table' method in Pandas and describe a business scenario where it might be useful.

Ans)

Pivot_table in Pandas is used for reshaping and summarizing data within a DataFrame. It allows you to create a pivot table, which is a two-dimensional table that summarizes data based on one or more columns of the DataFrame

Uses: 1.Data summarization 2.Aggregation 3.Reshaping data 4.Handling missing values

Business Scenario: Inventory Sales Analysis

```
import pandas as pd

# Sample DataFrame with sales data
data = {'Product Category': ['Electronics', 'Clothing', 'Electronics', 'Clothing'],
        'Sales Amount': [1000, 500, 800, 600],
        'Date': ['2023-01-15', '2023-04-25', '2023-07-10', '2023-10-05']}
df = pd.DataFrame(data)

# Convert 'Date' to datetime
df['Date'] = pd.to_datetime(df['Date'])

# Extract 'Quarter' from 'Date'
df['Quarter'] = df['Date'].dt.to_period('Q')

# Create a pivot table to summarize sales by product category and quarter
pivot_table = pd.pivot_table(df, values='Sales Amount', index='Product Category', columns='Quarter', aggfunc='sum', fill_value=0)

print(pivot_table)
```

```
Quarter           2023Q1  2023Q2  2023Q3  2023Q4
Product Category
Clothing               0     500       0     600
Electronics         1000       0     800       0
```

14. How would you handle large datasets that don't fit into memory? Are you familiar with Dask or any similar libraries?

Ans)

Dask is one the popular libraires for handling out memory data sets that dont fit into memory

Other similiar libraries includes:

1. apache kafka
2. asynico

15. In a dataset, you observe that some numerical columns are highly skewed. How can you normalize or transform these columns using Python?

Ans)

Skewness is a measure of the asymmetry of the probability distribution of a random variable In Python, a feature in a dataset is considered "highly skewed" when its distribution is significantly asymmetrical

There are different techniques

1. Log Transformation
2. Boc-cox transformation
3. square root transformation
4. Reciprocal transformation
5. Z- score transformation

```
import numpy as np
df['Skewed_Column_Log'] = np.log1p(df['Skewed_Column'])  #Log transformation
```

```
import numpy as np
df['Skewed_Column_Log'] = np.log1p(df['Skewed_Column'])  #box transformation
```

```
import numpy as np
df['Skewed_Column_Sqrt'] = np.sqrt(df['Skewed_Column']) #Square root transformation
```

```
import numpy as np
df['Skewed_Column_Reciprocal'] = 1 / df['Skewed_Column'] #reciprocal transformation
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['Skewed_Column_Z_Score'] = scaler.fit_transform(df[['Skewed_Column']]) #Z- score transformation
```

✓ 0s completed at 7:12 PM ● ×