

1. Describe the functionality of the 'describe()' function and its variations.

The describe() function is used to get a summary of some data. The variations of describe() come from where it is imported from.

When imported from Pandas, describe() is used for a dataframe and when it's imported from SciPy it's used for numpy arrays.

2. Specify the maximum character limit permissible for file names and file paths in the Windows operating system. Additionally, enumerate the special characters that are prohibited in file names.

File name character limit – 255

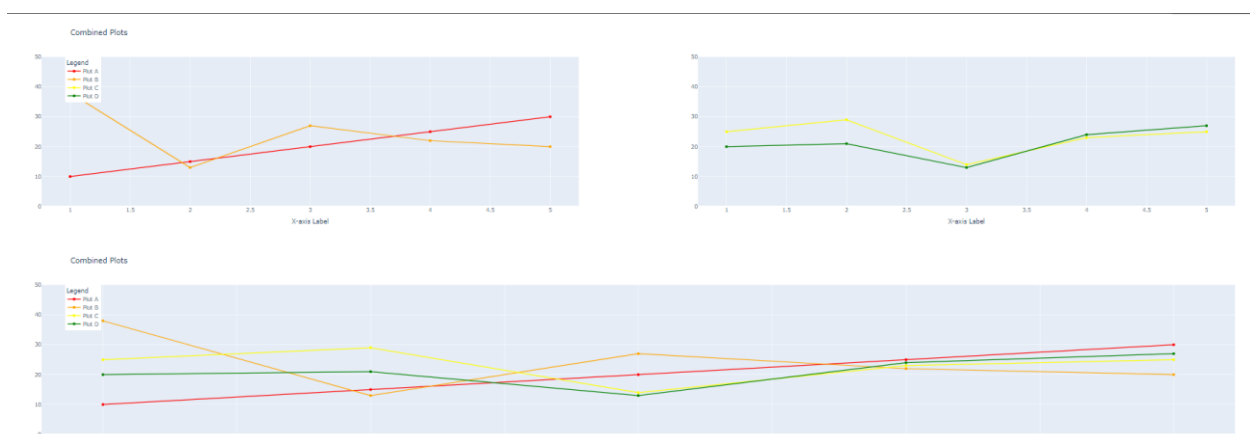
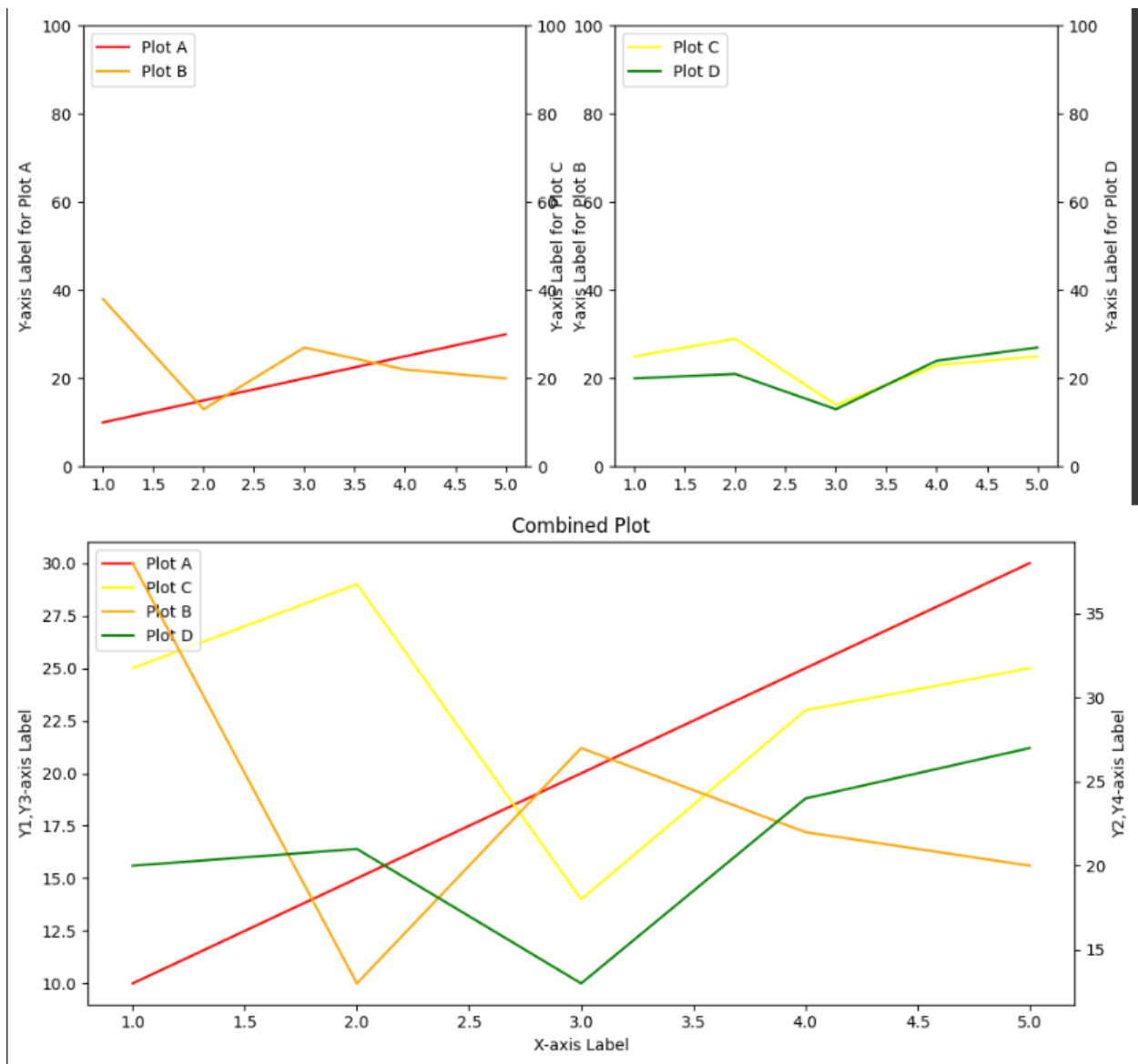
File path character limit - 260

Some of the characters prohibited in file names are:

- < (less than)
- > (greater than)
- : (colon)
- " (double quote)
- / (forward slash)
- \ (backslash)
- | (vertical bar or pipe)
- ? (question mark)
- *** (asterisk)**
- . (indicates a hidden file)
- \0 (null)

These characters are prohibited due to their reserved meanings within the file system

3. Demonstrate how to achieve a matplotlib operation using plotly.



4. Identify alternatives to the itertools module in Python.

- more-itertools: Extends the functionality of itertools with extra functions
- toolz: Provides functions for working with iterables
- fancy: Also provides functions for working with iterables but has more features than the toolz library
- pyrsistent: Is a library primarily focused on immutable data structures, but has some functionalities for iterables

5. Provide a Python script demonstrating database connectivity, CSV loading, CRUD operations, and encoding handling between CSV (utf-8) and a database (latin-9). Furthermore, ensure the CSV dataset contains over 100,000 rows, encompassing words from diverse languages.

```
import configparser
import pandas as pd
import pyodbc
import argparse
import time

config = configparser.ConfigParser()
config.read('config.ini')

def create_connection(server, database, username, password):
    conn = None
    try:
        conn = pyodbc.connect('DRIVER={SQL Server};SERVER=' + server +
                              ';DATABASE=' + database +
                              ';UID=' + username +
                              ';PWD=' + password)
        print("Database connection established.")
    except pyodbc.Error as e:
        print(e)
    return conn

def create_table_from_dataframe(conn, df, table_name):
    try:
        cursor = conn.cursor()

        # Drop the table if it exists
        drop_table_sql = f"DROP TABLE IF EXISTS {table_name}"
        cursor.execute(drop_table_sql)
```

```

# Generate SQL column definitions dynamically based on DataFrame column
types
columns_sql = ', '.join([f'[{col}] NVARCHAR(MAX)' for col in df.columns])

# Create the table with UTF-8 collation
create_table_sql = f"CREATE TABLE {table_name} ({columns_sql})"
cursor.execute(create_table_sql)
conn.commit()
print(f"Table '{table_name}' created successfully based on DataFrame
schema.")
except pyodbc.Error as e:
    print(e)

def bulk_insert(conn, target_table, data_file, encoding='utf-8',
field_terminator=',', row_terminator='\n', batch_size=None, max_errors=None):
    try:
        cursor = conn.cursor()

        # Construct the BULK INSERT statement
        bulk_insert_query = f"""
            BULK INSERT {target_table}
            FROM '{data_file}'
            WITH (
                DATAFILETYPE = 'char',
                CODEPAGE = '65001', -- UTF-8 encoding
                FIELDTERMINATOR = '{field_terminator}',
                ROWTERMINATOR = '{row_terminator}',
                FIRSTROW = 2
                {'', BATCHSIZE = ' + str(batch_size) if batch_size is not None
else ''} -- Only add BATCHSIZE if not None
                {'', MAXERRORS = ' + str(max_errors) if max_errors is not None
else ''} -- Only add MAXERRORS if not None
            )
        """

        # Execute the BULK INSERT statement
        cursor.execute(bulk_insert_query)
        # Commit the transaction
        conn.commit()
        print("Data loaded from DataFrame to database successfully.")
    except pyodbc.Error as e:
        print(e)
        conn.rollback()

```

```

def main():
    parser = argparse.ArgumentParser(description="Database CRUD operations with
age_gender movie data")
    parser.add_argument("--config-file", type=str, help="Config file containing
database connection information")
    parser.add_argument("--csv-file", type=str, help="CSV file containing
age_gender data")
    parser.add_argument("--create-table", type=str, help="create table table-
name")
    args = parser.parse_args()

    df = pd.read_csv(args.csv_file, encoding='utf-8')
    username = config.get('Settings','username')
    password = config.get('Settings','password')
    driver_name = config.get('Settings','DRIVER_NAME')
    database_name = config.get('Settings','DATABASE_NAME')
    server_name = config.get('Settings','SERVER_NAME')
    conn = create_connection(server_name, database_name, username, password)

    if conn is not None:
        table_name = args.create_table
        create_table_from_dataframe(conn, df, table_name=table_name)

        if table_name in [x.table_name for x in conn.cursor().tables()]:
            start_time = time.time()
            print(args.csv_file)
            filepath = args.csv_file
            bulk_insert(conn, target_table=table_name, data_file=filepath)

            time_taken = time.time() - start_time
            print("time_taken ", time_taken)
        else:
            print(f"Table '{table_name}' was not created successfully. Aborting
bulk insert.")

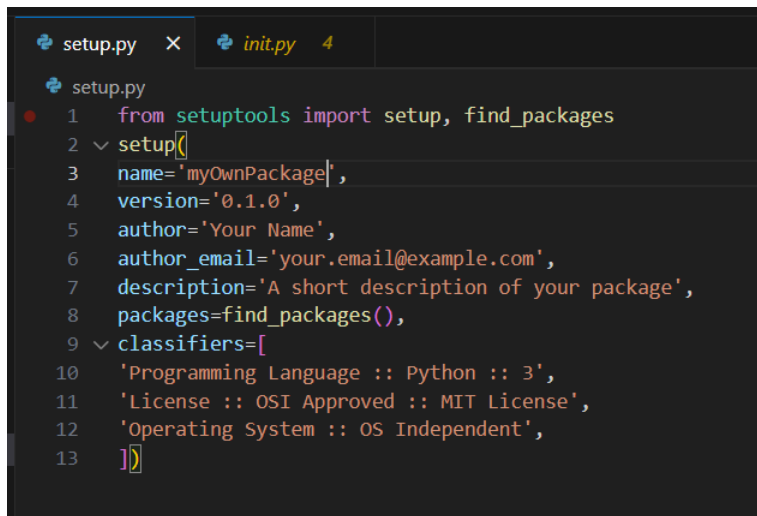
        conn.close()
        print("Database connection closed.")

if __name__ == '__main__':

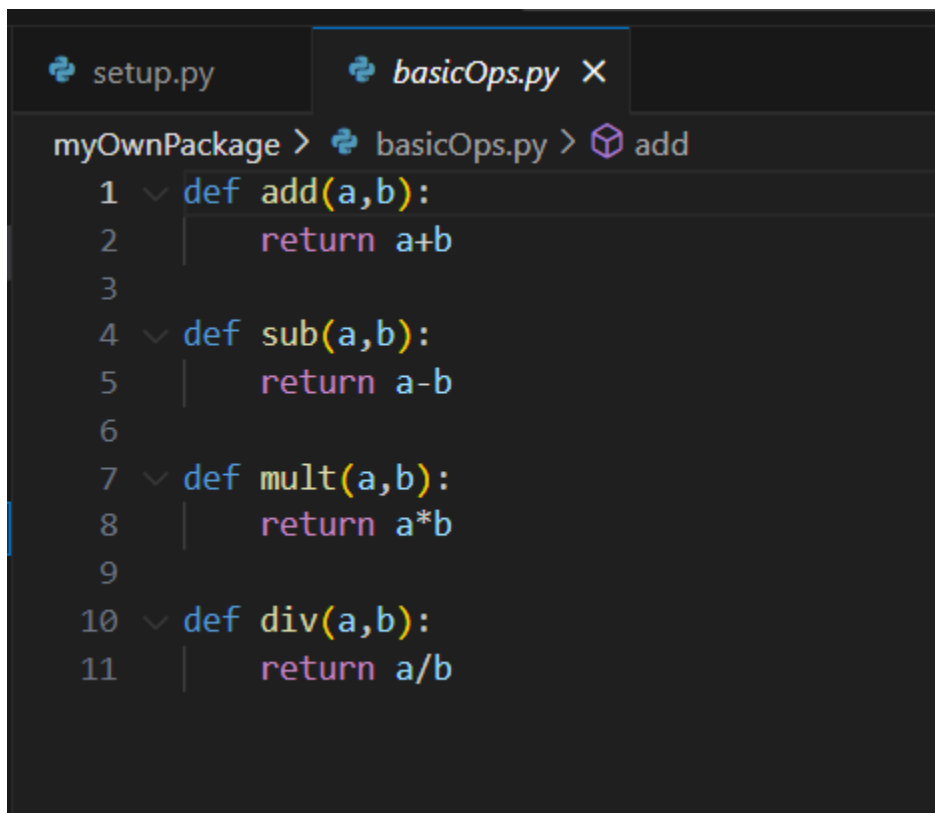
```

```
main()
```

6. Develop a module and associated packages within a library, followed by installation using pip.



```
1 from setuptools import setup, find_packages
2 setup(
3     name='myOwnPackage',
4     version='0.1.0',
5     author='Your Name',
6     author_email='your.email@example.com',
7     description='A short description of your package',
8     packages=find_packages(),
9     classifiers=[
10         'Programming Language :: Python :: 3',
11         'License :: OSI Approved :: MIT License',
12         'Operating System :: OS Independent',
13     ]
14 )
```



```
1 def add(a,b):
2     return a+b
3
4 def sub(a,b):
5     return a-b
6
7 def mult(a,b):
8     return a*b
9
10 def div(a,b):
11     return a/b
```

```
PS C:\Users\aryan.devikar\OneDrive - Nitor Infotech Pvt. Ltd\Documents\Training Material\Databricks> pip install newPackage
Requirement already satisfied: newPackage in c:\users\aryan.devikar\appdata\local\programs\python\python311\lib\site-packages (0.1.0)
```

7. Instructions on the process of extracting date-time data from a DataFrame and exporting it to an Excel file using Pandas while preserving the original data type.

If the date-time field is of a data type `datetime`, then there is no need for conversion, and we can extract the appropriate data from that field.

If the date is in the form of a string, then we'd have to perform the `pd.to_datetime()` conversion on that column in the DataFrame. If we insist on preserving the data type, we'll have to specify a format string `"%Y-%m-%d"` (or whatever format the date is in) and then use the `strftime` function to convert it into a string via `strftime(format_string)`

8. Propose alternative methods to replace the usage of the `iloc` function in Python.

- 'at' and 'iat'
 - These methods are used for accessing and modifying individual elements within a DataFrame, with 'at' being label-based and 'iat' being integer-location-based
- Using query
 - You can use the bracket notation for column selection and row slicing
- Using bracket notation
 - Queries are useful for complex conditional filtering

9. Illustrate the procedure for resetting the index in Python.

Index resetting is done by the `reset_index()` function

```
data = {
    'A': [2, 3, 4],
    'B': [4, 9, 16],
    'C': [8, 27, 64]
}

df.set_index('A', inplace= True)

df_reset = df.reset_index()
```

