**Харківський національний економічний університет**

**імені Семена Кузнеця**

**ЗВІТ**

**З ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 16**

**за дисципліною: *“*Технології програмування**”

**на тему: “Обробка даних”**

**Варіант: 4**

**Виконав: студент факультету Інформаційних технологій**

**2 курсу, спец. Кібербезпека,**

**групи 6.04.125.010.21.2**

**Бойко Вадим Віталійович**

**Перевірив:**

**Венгріна Олена Сергіївна**

**ХНЕУ ім. С. Кузнеця**

**2023**

**Мета:** Метою роботи є набуття навичок технологій розробки програмного коду для обробки масивів даних із використання бібліотек NumPy та Pandas.

**Завданя:** Відповідно до варіанту завдання розробити клас для обробки даних. Розробити конструктор, що приймає шлях до файлу, роздільник, що використовується в CSV файлі, флаг наявності заголовків стовпців в файлі. (роздільник та флаг зробити параметрами із значеннями за замовчуванням). При неможливості прочитати файл із даними генерувати виняток із відповідним повідомленням користувачеві. Конструктор повинен ініціалізувати внутрішнє поле датафрейму, що буде зберігати дані та використовуватися іншими методами. Розробити методи, що узагальнюють завдання, наведені у варіантах (якщо в завданні вказане конкретне значення, то його слід приймати у якості параметру методу). Методи повинні повертати датафрейми із відповідними даними.

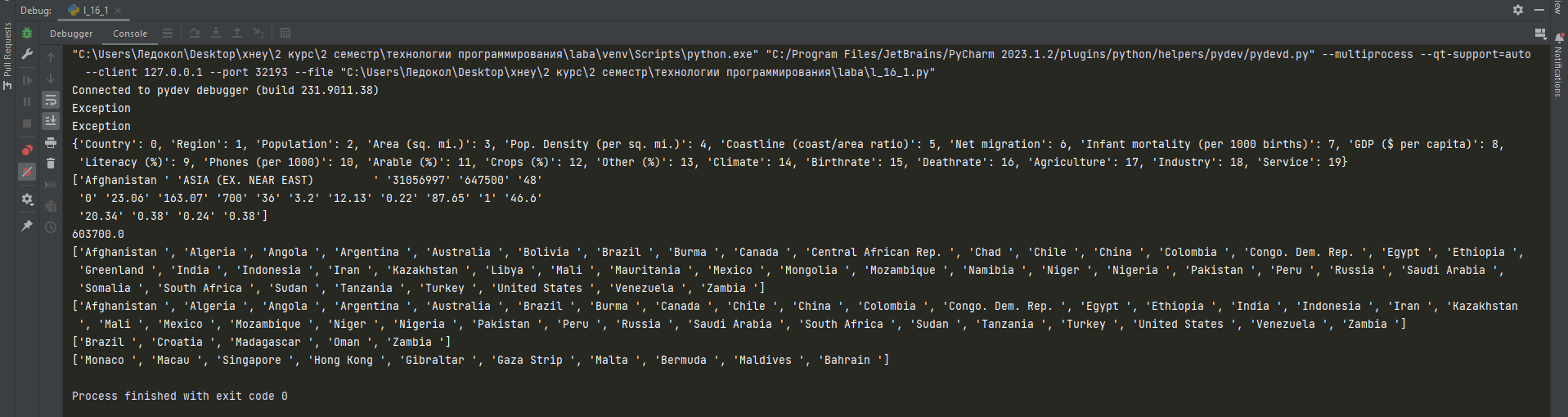
Варінт 4

* Список всіх країн
* Вибрати країни у яких площа більше ніж в Україні
* Вибрати країни з населенням понад 10 млн та у яких площа більше ніж в Україні
* Вибрати країни, які не межують з морем
* Вибрати топ 10 країн найбільш густонаселених (кількість людей на кв. км.)

Я написав наступне:

*import* numpy *as* np  
  
*class* DataParser:  
 *def \_\_init\_\_*(self, *path*: str = *None*, *separator* = ","):  
 *if not path*:  
 *raise* Exception('Path must be a string')  
 *try*:  
 self.path = *path* file\_array = np.loadtxt(*path*, delimiter=*separator*, dtype=str)  
 self.headers = file\_array[0][0:]  
 self.data\_array = np.array(file\_array[1:])  
 *except*:  
 *raise* Exception('Can\'t open a file')  
 self.data\_enum = {}  
 *for* i *in* range(len(self.headers)):  
 self.data\_enum[self.headers[i]] = i  
 print(self.data\_enum)  
  
 *def* country\_list(self):  
 countries = []  
 *for* row *in* self.data\_array:  
 countries.append(row[0])  
 *return* countries  
  
 *def* area\_more\_than\_another\_country\_area(self, *country\_name* = *None*):  
 *if not country\_name*:  
 *raise* Exception('Country name must be provided')  
 countries = []  
 compare\_row = []  
 *for* row *in* self.data\_array:  
 *if country\_name in* row[self.data\_enum['Country']]:  
 compare\_row = row  
 *break* compared\_area = float(compare\_row[self.data\_enum['Area (sq. mi.)']])  
 *for* row *in* self.data\_array:  
 *if* float(row[self.data\_enum['Area (sq. mi.)']]) > compared\_area:  
 countries.append(row[self.data\_enum['Country']])  
 print(compared\_area)  
 *return* countries  
  
 *def* countries\_which\_has\_population\_more\_than(self, *population* = *None*):  
 *if not population*:  
 *raise* Exception('Population must be provided')  
 countries = []  
 *for* row *in* self.data\_array:  
 *if* int(row[self.data\_enum['Population']]) > *population*:  
 countries.append(row[self.data\_enum['Country']])  
 *return* countries  
  
 *def* inner\_join(self, *arr\_1* = [], *arr\_2* = []):  
 countries = []  
 *for* data *in arr\_1*:  
 *if* data *in arr\_2*:  
 countries.append(data)  
 *return* countries  
  
 *def* countries\_without\_sea(self):  
 countries = []  
 *for* row *in* self.data\_array:  
 arable = row[self.data\_enum['Arable (%)']]  
 *if* arable == '':  
 *continue* crops = row[self.data\_enum['Crops (%)']]  
 *if* crops == '':  
 *continue* other = row[self.data\_enum['Other (%)']]  
 *if* other == '':  
 *continue  
 if* float(arable) + float(crops) + float(other) != 100:  
 countries.append(row[self.data\_enum['Country']])  
 *return* countries  
  
 *def* top\_population\_countries(self, *limit* = 10):  
 *if limit* < 1:  
 *raise* Exception('Limit must be a positive number')  
 arr = np.copy(self.data\_array)  
 *for* i *in* range(len(arr) - 1):  
 *for* j *in* range(i, len(arr) - 1):  
 *if* arr[i][self.data\_enum['Pop. Density (per sq. mi.)']] == '' *or* arr[j][self.data\_enum['Pop. Density (per sq. mi.)']] == '':  
 *continue  
 if* float(arr[i][self.data\_enum['Pop. Density (per sq. mi.)']]) < float(arr[j][self.data\_enum['Pop. Density (per sq. mi.)']]):  
 arr[i], arr[j] = np.copy(arr[j]), np.copy(arr[i])  
 countries = []  
 limit = *limit if limit* <= len(arr) *else* len(arr)  
 *for* i *in* range(*limit*):  
 countries.append(arr[i][self.data\_enum['Country']])  
 *return* countries  
  
  
  
  
*try*:  
 dp\_error = DataParser()  
*except*:  
 print('Exception')  
*try*:  
 dp\_not\_correct\_file = DataParser("qwe")  
*except*:  
 print('Exception')  
dp = DataParser("countries\_of\_the\_world.csv", ",")  
print(dp.data\_array[0])  
# print(dp.data\_array)  
  
# print(dp.country\_list())  
area\_more\_than\_Ukraine\_area = dp.area\_more\_than\_another\_country\_area('Ukraine')  
print(area\_more\_than\_Ukraine\_area)  
  
countries\_which\_has\_population\_more\_than\_10\_millions = dp.countries\_which\_has\_population\_more\_than(10000000)  
print(dp.inner\_join(area\_more\_than\_Ukraine\_area, countries\_which\_has\_population\_more\_than\_10\_millions))  
  
print(dp.countries\_without\_sea())  
  
print(dp.top\_population\_countries())

Результат



Exception

Exception

{'Country': 0, 'Region': 1, 'Population': 2, 'Area (sq. mi.)': 3, 'Pop. Density (per sq. mi.)': 4, 'Coastline (coast/area ratio)': 5, 'Net migration': 6, 'Infant mortality (per 1000 births)': 7, 'GDP ($ per capita)': 8, 'Literacy (%)': 9, 'Phones (per 1000)': 10, 'Arable (%)': 11, 'Crops (%)': 12, 'Other (%)': 13, 'Climate': 14, 'Birthrate': 15, 'Deathrate': 16, 'Agriculture': 17, 'Industry': 18, 'Service': 19}

['Afghanistan ' 'ASIA (EX. NEAR EAST) ' '31056997' '647500' '48'

'0' '23.06' '163.07' '700' '36' '3.2' '12.13' '0.22' '87.65' '1' '46.6'

'20.34' '0.38' '0.24' '0.38']

603700.0

['Afghanistan ', 'Algeria ', 'Angola ', 'Argentina ', 'Australia ', 'Bolivia ', 'Brazil ', 'Burma ', 'Canada ', 'Central African Rep. ', 'Chad ', 'Chile ', 'China ', 'Colombia ', 'Congo. Dem. Rep. ', 'Egypt ', 'Ethiopia ', 'Greenland ', 'India ', 'Indonesia ', 'Iran ', 'Kazakhstan ', 'Libya ', 'Mali ', 'Mauritania ', 'Mexico ', 'Mongolia ', 'Mozambique ', 'Namibia ', 'Niger ', 'Nigeria ', 'Pakistan ', 'Peru ', 'Russia ', 'Saudi Arabia ', 'Somalia ', 'South Africa ', 'Sudan ', 'Tanzania ', 'Turkey ', 'United States ', 'Venezuela ', 'Zambia ']

['Afghanistan ', 'Algeria ', 'Angola ', 'Argentina ', 'Australia ', 'Brazil ', 'Burma ', 'Canada ', 'Chile ', 'China ', 'Colombia ', 'Congo. Dem. Rep. ', 'Egypt ', 'Ethiopia ', 'India ', 'Indonesia ', 'Iran ', 'Kazakhstan ', 'Mali ', 'Mexico ', 'Mozambique ', 'Niger ', 'Nigeria ', 'Pakistan ', 'Peru ', 'Russia ', 'Saudi Arabia ', 'South Africa ', 'Sudan ', 'Tanzania ', 'Turkey ', 'United States ', 'Venezuela ', 'Zambia ']

['Brazil ', 'Croatia ', 'Madagascar ', 'Oman ', 'Zambia ']

['Monaco ', 'Macau ', 'Singapore ', 'Hong Kong ', 'Gibraltar ', 'Gaza Strip ', 'Malta ', 'Bermuda ', 'Maldives ', 'Bahrain ']

Process finished with exit code 0