Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Лабораторная работа №2
по курсу «Методы машинного обучения»

«Изучение библиотек обработки данных»

Выполнила: Чекулина М.Ю.
Группа ИУ5-22М

Москва – 2019

1. Задание:

Часть 1.

Выполните первое демонстрационное задание "demo assignment" под названием "Exploratory data analysis with Pandas" со страницы курса https://mlcourse.ai/assignments (https://mlcourse.ai/assignments)

Условие задания - https://nbviewer.jupyter.org/github/Yorko/mlcourse_open/blob/master/jupyter_english/assignm flush_cache=true (https://nbviewer.jupyter.org/github/Yorko/mlcourse_open/blob/master/jupyter_english/assignr flush_cache=true)

Набор данных можно скачать здесь - https://archive.ics.uci.edu/ml/datasets/Adult (https://archive.ics.uci.edu/ml/datasets/Adult)

Пример решения задания - https://www.kaggle.com/kashnitsky/a1-demo-pandas-and-uci-adult-dataset-solution (https://www.kaggle.com/kashnitsky/a1-demo-pandas-and-uci-adult-dataset-solution)

Часть 2.

Выполните следующие запросы с использованием двух различных библиотек - Pandas и PandaSQL:

один произвольный запрос на соединение двух наборов данных один произвольный запрос на группировку набора данных с использованием функций агрегирования Сравните время выполнения каждого запроса в Pandas и PandaSQL.

В качестве примеров можно использовать следующие статьи:

https://www.shanelynn.ie/summarising-aggregation-and-grouping-data-in-python-pandas/ (https://www.shanelynn.ie/summarising-aggregation-and-grouping-data-in-python-pandas/) https://www.shanelynn.ie/merge-join-dataframes-python-pandas-index-1/ (https://www.shanelynn.ie/merge-join-dataframes-python-pandas-index-1/) (в разделе "Example data" данной статьи содержится рекомендуемый набор данных для проведения экспериментов). Пример сравнения Pandas и PandaSQL - https://github.com/miptgirl/udacity_engagement_analysis/blob/master/pandasql_example.ipy (https://github.com/miptgirl/udacity_engagement_analysis/blob/master/pandasql_example.ipy

Набор упражнений по Pandas с решениями - https://github.com/guipsamora/pandas_exercises (https://github.com/guipsamora/pandas_exercises)

2. Реализация

In [1]:
```
import numpy as np
import pandas as pd

data = pd.read_csv('Data/adult.data.csv')
data.head()
```

Out[1]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black

In [2]:
```
data['sex'].value_counts()
```

Out[2]:
```
Male      21790
Female    10771
Name: sex, dtype: int64
```

In [3]:
```
data.loc[data['sex'] == 'Female', 'age'].mean()
```

Out[3]:
```
36.85823043357163
```

In [4]:
```
float((data['native-country'] == 'Germany').sum()) / data.shape[0]
```

Out[4]:
```
0.004207487485028101
```

In [5]:
```
ages1 = data.loc[data['salary'] == '>50K', 'age']
ages2 = data.loc[data['salary'] == '<=50K', 'age']
print("The average age of the rich: {0} +- {1} years, poor - {2} +- {3
    round(ages1.mean()), round(ages1.std(), 1),
    round(ages2.mean()), round(ages2.std(), 1)))
```

```
The average age of the rich: 44 +- 10.5 years, poor - 37 +- 14.0 yea
rs.
```

In [6]:
```
data.loc[data['salary'] == '>50K', 'education'].unique()
```

Out[6]:
```
array(['HS-grad', 'Masters', 'Bachelors', 'Some-college', 'Assoc-voc
',
       'Doctorate', 'Prof-school', 'Assoc-acdm', '7th-8th', '12th',
       '10th', '11th', '9th', '5th-6th', '1st-4th'], dtype=object)
```

In [7]:
```
for (race, sex), sub_df in data.groupby(['race', 'sex']):
    print("Race: {0}, sex: {1}".format(race, sex))
    print(sub_df['age'].describe())
```

```
Race: Amer-Indian-Eskimo, sex: Female
count    119.000000
mean      37.117647
std       13.114991
min       17.000000
25%       27.000000
50%       36.000000
75%       46.000000
max       80.000000
Name: age, dtype: float64
Race: Amer-Indian-Eskimo, sex: Male
count    192.000000
mean      37.208333
std       12.049563
min       17.000000
25%       28.000000
50%       35.000000
75%       45.000000
max       82.000000
Name: age, dtype: float64
Race: Asian-Pac-Islander, sex: Female
count    346.000000
mean      35.089595
std       12.300845
min       17.000000
25%       25.000000
50%       33.000000
75%       43.750000
max       75.000000
Name: age, dtype: float64
Race: Asian-Pac-Islander, sex: Male
count    693.000000
mean      39.073593
std       12.883944
min       18.000000
25%       29.000000
50%       37.000000
75%       46.000000
max       90.000000
Name: age, dtype: float64
Race: Black, sex: Female
count    1555.000000
```

```
mean       37.854019
std        12.637197
min        17.000000
25%        28.000000
50%        37.000000
75%        46.000000
max        90.000000
Name: age, dtype: float64
Race: Black, sex: Male
count    1569.000000
mean       37.682600
std        12.882612
min        17.000000
25%        27.000000
50%        36.000000
75%        46.000000
max        90.000000
Name: age, dtype: float64
Race: Other, sex: Female
count    109.000000
mean      31.678899
std       11.631599
min       17.000000
25%       23.000000
50%       29.000000
75%       39.000000
max       74.000000
Name: age, dtype: float64
Race: Other, sex: Male
count    162.000000
mean      34.654321
std       11.355531
min       17.000000
25%       26.000000
50%       32.000000
75%       42.000000
max       77.000000
Name: age, dtype: float64
Race: White, sex: Female
count    8642.000000
mean       36.811618
std        14.329093
min        17.000000
25%        25.000000
50%        35.000000
75%        46.000000
max        90.000000
Name: age, dtype: float64
Race: White, sex: Male
count    19174.000000
mean        39.652498
std         13.436029
min         17.000000
25%         29.000000
```

```
50%          38.000000
75%          49.000000
max          90.000000
Name: age, dtype: float64
```

In [8]:
```
data.loc[(data['sex'] == 'Male') &
     (data['marital-status'].isin(['Never-married',
                                   'Separated',
                                   'Divorced',
                                   'Widowed'])), 'salary'].value_count
```

Out[8]:
```
<=50K    7552
>50K      697
Name: salary, dtype: int64
```

In [9]:
```
data.loc[(data['sex'] == 'Male') &
     (data['marital-status'].str.startswith('Married')), 'salary'].val
```

Out[9]:
```
<=50K    7576
>50K     5965
Name: salary, dtype: int64
```

In [10]:
```
data['marital-status'].value_counts()
```

Out[10]:
```
Married-civ-spouse       14976
Never-married            10683
Divorced                  4443
Separated                 1025
Widowed                    993
Married-spouse-absent      418
Married-AF-spouse           23
Name: marital-status, dtype: int64
```

In [11]:
```
max_load = data['hours-per-week'].max()
print("Max time - {0} hours./week.".format(max_load))

num_workaholics = data[data['hours-per-week'] == max_load].shape[0]
print("Total number of such hard workers {0}".format(num_workaholics))

rich_share = float(data[(data['hours-per-week'] == max_load)
                 & (data['salary'] == '>50K')].shape[0]) / num_workaho
print("Percentage of rich among them {0}%".format(int(100 * rich_share
```

```
Max time - 99 hours./week.
Total number of such hard workers 85
Percentage of rich among them 29%
```

In [12]:
```
for (country, salary), sub_df in data.groupby(['native-country', 'sala
    print(country, salary, round(sub_df['hours-per-week'].mean(), 2))
```

```
? <=50K 40.16
? >50K 45.55
```

```
Cambodia <=50K 41.42
Cambodia >50K 40.0
Canada <=50K 37.91
Canada >50K 45.64
China <=50K 37.38
China >50K 38.9
Columbia <=50K 38.68
Columbia >50K 50.0
Cuba <=50K 37.99
Cuba >50K 42.44
Dominican-Republic <=50K 42.34
Dominican-Republic >50K 47.0
Ecuador <=50K 38.04
Ecuador >50K 48.75
El-Salvador <=50K 36.03
El-Salvador >50K 45.0
England <=50K 40.48
England >50K 44.53
France <=50K 41.06
France >50K 50.75
Germany <=50K 39.14
Germany >50K 44.98
Greece <=50K 41.81
Greece >50K 50.62
Guatemala <=50K 39.36
Guatemala >50K 36.67
Haiti <=50K 36.33
Haiti >50K 42.75
Holand-Netherlands <=50K 40.0
Honduras <=50K 34.33
Honduras >50K 60.0
Hong <=50K 39.14
Hong >50K 45.0
Hungary <=50K 31.3
Hungary >50K 50.0
India <=50K 38.23
India >50K 46.48
Iran <=50K 41.44
Iran >50K 47.5
Ireland <=50K 40.95
Ireland >50K 48.0
Italy <=50K 39.62
Italy >50K 45.4
Jamaica <=50K 38.24
Jamaica >50K 41.1
Japan <=50K 41.0
Japan >50K 47.96
Laos <=50K 40.38
Laos >50K 40.0
Mexico <=50K 40.0
Mexico >50K 46.58
Nicaragua <=50K 36.09
Nicaragua >50K 37.5
Outlying-US(Guam-USVI-etc) <=50K 41.86
```

```
Peru <=50K 35.07
Peru >50K 40.0
Philippines <=50K 38.07
Philippines >50K 43.03
Poland <=50K 38.17
Poland >50K 39.0
Portugal <=50K 41.94
Portugal >50K 41.5
Puerto-Rico <=50K 38.47
Puerto-Rico >50K 39.42
Scotland <=50K 39.44
Scotland >50K 46.67
South <=50K 40.16
South >50K 51.44
Taiwan <=50K 33.77
Taiwan >50K 46.8
Thailand <=50K 42.87
Thailand >50K 58.33
Trinadad&Tobago <=50K 37.06
Trinadad&Tobago >50K 40.0
United-States <=50K 38.8
United-States >50K 45.51
Vietnam <=50K 37.19
Vietnam >50K 39.2
Yugoslavia <=50K 41.6
Yugoslavia >50K 49.5
```

In [13]:

```
pd.crosstab(data['native-country'], data['salary'],
            values=data['hours-per-week'], aggfunc=np.mean).T
```

Out[13]:

native-country	?	Cambodia	Canada	China	Columbia	Cuba	Dominican-Republic	Ecu
salary								
<=50K	40.164760	41.416667	37.914634	37.381818	38.684211	37.985714	42.338235	38.04
>50K	45.547945	40.000000	45.641026	38.900000	50.000000	42.440000	47.000000	48.75

2 rows × 42 columns

In [14]:
```
import numpy as np
import pandas as pd

dictionary = pd.read_csv('Data/lab_2_part_2/dictionary.csv')
dictionary.head()
```

Out[14]:

	Country	Code	Population	GDP per Capita
0	Afghanistan	AFG	32526562.0	594.323081
1	Albania	ALB	2889167.0	3945.217582
2	Algeria	ALG	39666519.0	4206.031232
3	American Samoa*	ASA	55538.0	NaN
4	Andorra	AND	70473.0	NaN

In [15]:
```
import numpy as np
import pandas as pd
summer = pd.read_csv('Data/lab_2_part_2/summer.csv')
summer.head()
```

Out[15]:

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze
3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold
4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver

In [16]:
```
import numpy as np
import pandas as pd
winter = pd.read_csv('Data/lab_2_part_2/winter.csv')
winter.head()
```

Out[16]:

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
0	1924	Chamonix	Biathlon	Biathlon	BERTHET, G.	FRA	Men	Military Patrol	Bronze
1	1924	Chamonix	Biathlon	Biathlon	MANDRILLON, C.	FRA	Men	Military Patrol	Bronze
2	1924	Chamonix	Biathlon	Biathlon	MANDRILLON, Maurice	FRA	Men	Military Patrol	Bronze
3	1924	Chamonix	Biathlon	Biathlon	VANDELLE, André	FRA	Men	Military Patrol	Bronze
4	1924	Chamonix	Biathlon	Biathlon	AUFDENBLATTEN, Adolf	SUI	Men	Military Patrol	Gold

In [17]:
```
# соединение таблиц
def connection_pandas(dictionary,summer):
        result = pd.merge(dictionary, summer, left_on = 'Code', right_
        return result

connection_pandas(dictionary, summer).head()
```

Out[17]:

	Country_x	Code	Population	GDP per Capita	Year	City	Sport	Discipline	
0	Afghanistan	AFG	32526562.0	594.323081	2008	Beijing	Taekwondo	Taekwondo	
1	Afghanistan	AFG	32526562.0	594.323081	2012	London	Taekwondo	Taekwondo	
2	Algeria	ALG	39666519.0	4206.031232	1984	Los Angeles	Boxing	Boxing	N
3	Algeria	ALG	39666519.0	4206.031232	1984	Los Angeles	Boxing	Boxing	N N
4	Algeria	ALG	39666519.0	4206.031232	1992	Barcelona	Athletics	Athletics	BOUI

In [18]:
```
import pandasql as ps
pysql = lambda a: ps.sqldf(a, globals())
def connection_pandasql(dictionary,summer):
    query = "select * from dictionary,summer where dictionary.Code = s
    join_result = pysql(query)
    return join_result
abc = connection_pandasql(dictionary, summer)
connection_pandasql(dictionary, summer).head()
```

Out[18]:

	Country	Code	Population	GDP per Capita	Year	City	Sport	Discipline	Athlete
0	Russia	RUS	144096812.0	9092.580536	2012	London	Aquatics	Diving	KUZNETSOV, Evgeny
1	Russia	RUS	144096812.0	9092.580536	2012	London	Aquatics	Diving	ZAKHAROV, Ilya
2	Russia	RUS	144096812.0	9092.580536	2012	London	Aquatics	Diving	ZAKHAROV, Ilya
3	Russia	RUS	144096812.0	9092.580536	2012	London	Aquatics	Swimming	EFIMOVA, Iuliia
4	Russia	RUS	144096812.0	9092.580536	2012	London	Aquatics	Swimming	FESIKOV, Sergei

In [19]:
```
# сравнение времени выполнения запросов

import time
class Profiler(object):
    def __enter__(self):
        self._startTime = time.time()

    def __exit__(self, type, value, traceback):
        print("Elapsed time: {:.3f} sec".format(time.time() - self._st

with Profiler() as p:
    connection_pandas(dictionary, summer)
```

```
Elapsed time: 0.010 sec
```

In [20]:
```
with Profiler() as p:
    connection_pandas(dictionary, winter)
```

```
Elapsed time: 0.006 sec
```

In [21]:
```
with Profiler() as p:
    connection_pandasql(dictionary, summer)
```

```
Elapsed time: 0.361 sec
```

In [22]:
```
with Profiler() as p:
    connection_pandasql(dictionary, winter)
```

```
Elapsed time: 0.332 sec
```

In [23]:
```
# Вывод: соединение с помощью pandas работает в 30 быстрее, чем pandas

# Агрегирование: произвольный запрос на группировку набора данных
# с использованием функций агрегирования
def aggregation_pandas(dictionary,summer):
    result = pd.merge(dictionary, summer, left_on = 'Code', right_on =
    final_0 = result[result['Year'] == 2012]
    final = final_0[final_0['Medal'] == 'Gold'].groupby("Country_x").a
        "Medal": "count",
        'Discipline' : 'nunique',
        'Gender' : 'nunique',
    })
    return final

aggregation_pandas(dictionary, summer).head(10)
```

Out[23]:

	Medal	Discipline	Gender
Country_x			
Algeria	1	1	1
Argentina	1	1	1
Australia	19	5	2
Azerbaijan	2	1	1
Bahamas	4	1	1
Belarus	3	2	2
Brazil	14	3	2
Canada	1	1	1
China	56	13	2
Colombia	1	1	1

In [24]:
```
def aggregation_pandasql(summer):
    query = '''
    SELECT Country, count(Medal), count(DISTINCT Discipline), count(DI
    WHERE Medal == 'Gold' and Year == 2012 and Country != 'None'
    GROUP BY Country
    '''
    return ps.sqldf(query,locals())

aggregation_pandasql(summer).head(10)
```

Out[24]:

	Country	count(Medal)	count(DISTINCT Discipline)	count(DISTINCT Gender)
0	ALG	1	1	1
1	ARG	1	1	1
2	AUS	19	5	2
3	AZE	2	1	1
4	BAH	4	1	1
5	BLR	3	2	2
6	BRA	14	3	2
7	CAN	1	1	1
8	CHN	56	13	2
9	COL	1	1	1

In [25]:
```
# сравнение времени выполнения запросов агрегирования
import seaborn
import matplotlib.pyplot as plt
with Profiler() as p:
    aggregation_pandas(dictionary,summer)
```

```
Elapsed time: 0.026 sec
```

In [26]:
```
with Profiler() as p:
    aggregation_pandasql(summer)
```

```
Elapsed time: 0.428 sec
```