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

Отчёт

“Методы машинного обучения”

Лабораторная работа № 2

“Изучение библиотек обработки данных”

ИСПОЛНИТЕЛЬ:

Студент группы ИУ5-21М

Коростелёв В.М. _____________

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е._____________

Москва – 2019

Цель работы

Изучить библиотеки обработки данных Pandas и PandaSQL.

Задание

1. Требуется выполнить первое демонстрационное задание под названием «Exploratory data analysis with Pandas» со страницы курса mlcourse.ai.
2. Требуетсявыполнитьследующиезапросысиспользованиемдвухразличныхбиблиотек — Pandas и PandaSQL:
   - один произвольный запрос на соединение двух наборов данных,
   - один произвольный запрос на группировку набора данных с использованием функций агрегирования. Также требуется сравнить время выполнения каждого запроса в Pandas и PandaSQL.

Ход выполнения работы

Часть 1

Ниже приведён демонстрационный Jupyter-ноутбук «Exploratory data analysis with Pandas» курса mlcourse.ai (файл assignment01_pandas_uci_adult.ipynb). Все пояснения приведены на исходном языке ноутбука — на английском.

Assignment #1 (demo)

Exploratory data analysis with Pandas

In this task you should use Pandas to answer a few questions about the Adult dataset.

Unique values of all features (for more information, please see the links above):

- age: continuous.
- workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
- fnlwgt: continuous.
- education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
- education-num: continuous.
- marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- sex: Female, Male.
- capital-gain: continuous.

- capital-loss: continuous.
- hours-per-week: continuous.
- native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.
- salary: >50K,<=50K

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

In [2]:
```
data = pd.read_csv('adult.data.csv')
data.head()
```

Out[2]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	salary
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

1. How many men and women (sex feature) are represented in this dataset?

In [3]:
```
data['sex'].value_counts()
```

Out[3]:
```
Male      21790
Female    10771
Name: sex, dtype: int64
```

2. What is the average age (age feature) of women?

In [4]:
```
data.loc[data['sex'] == 'Female', 'age'].mean()
```

Out[4]:
```
36.85823043357163
```

3. What is the percentage of German citizens (native-country feature)?

In [6]:
```
print("{0:%}".format(data[data["native-country"] == "Germany"] .shape[0] / data.shape[0]))
```

```
0.420749%
```

4-5. What are the mean and standard deviation of age for those who earn more than 50K per year (salary feature) and those who earn less than 50K per year?

In [8]:
```
ages1 = data[data["salary"] == "<=50K"]["age"]
ages2 = data[data["salary"] == ">50K"]["age"]
print("<=50K: = {0} ± {1} years".format(ages1.mean(), ages1.std()))
print(" >50K: = {0} ± {1} years".format(ages2.mean(), ages2.std()))
```

```
<=50K: = 36.78373786407767 ± 14.02008849082488 years
 >50K: = 44.24984058155847 ± 10.519027719851826 years
```

6. Is it true that people who earn more than 50K have at least high school education? (education – Bachelors, Prof-school, Assoc-acdm, Assoc-voc, Masters or Doctorate feature)

In [9]:
```
high_educations = set(["Bachelors", "Prof-school", "Assoc-acdm", "Assoc-voc", "Masters", "Doctorate"])
def high_educated(e):
    return e in high_educations
data[data["salary"] == ">50K"]["education"].map(high_educated).all()
```

Out[9]:
```
False
```

7. Display age statistics for each race (race feature) and each gender (sex feature). Use groupby() and describe(). Find the maximum age of men of Amer-Indian-Eskimo race.

In [10]:
```
data.groupby(["race", "sex"])["age"].describe()
```

Out[10]:

race	sex	count	mean	std	min	25%	50%	75%	max
Amer-Indian-Eskimo	Female	119.0	37.117647	13.114991	17.0	27.0	36.0	46.00	80.0
	Male	192.0	37.208333	12.049563	17.0	28.0	35.0	45.00	82.0
Asian-Pac-Islander	Female	346.0	35.089595	12.300845	17.0	25.0	33.0	43.75	75.0
	Male	693.0	39.073593	12.883944	18.0	29.0	37.0	46.00	90.0
Black	Female	1555.0	37.854019	12.637197	17.0	28.0	37.0	46.00	90.0
	Male	1569.0	37.682600	12.882612	17.0	27.0	36.0	46.00	90.0
Other	Female	109.0	31.678899	11.631599	17.0	23.0	29.0	39.00	74.0
	Male	162.0	34.654321	11.355531	17.0	26.0	32.0	42.00	77.0
White	Female	8642.0	36.811618	14.329093	17.0	25.0	35.0	46.00	90.0
	Male	19174.0	39.652498	13.436029	17.0	29.0	38.0	49.00	90.0

In [11]:
```
data[(data["race"] == "Amer-Indian-Eskimo")
     & (data["sex"] == "Male")]["age"].max()
```

Out[11]:
```
82
```

8. Among whom is the proportion of those who earn a lot (>50K) greater: married or single men (marital-status feature)? Consider as married those who have a marital-status starting with Married (Married-civ-spouse, Married-spouse-absent or Married-AF-spouse), the rest are considered bachelors.

In [13]:
```
def is_married(m):
    return m.startswith("Married")
data["married"] = data["marital-status"].map(is_married)
(data[(data["sex"] == "Male") & (data["salary"] == ">50K")]
    ["married"].value_counts())
```

Out[13]:
```
True     5965
False     697
Name: married, dtype: int64
```

9. What is the maximum number of hours a person works per week (hours-per-week feature)? How many people work such a number of hours, and what is the percentage of those who earn a lot (>50K) among them?

```
In [15]: m = data["hours-per-week"].max()
         print("Maximum is {} hours/week.".format(m))
         people = data[data["hours-per-week"] == m]
         c = people.shape[0]
         print("{} people work this time at week.".format(c))
         s = people[people["salary"] == ">50K"].shape[0]
         print("{0:%} get >50K salary.".format(s / c))

         Maximum is 99 hours/week.
         85 people work this time at week.
         29.411765% get >50K salary.
```

10. Count the average time of work (hours-per-week) for those who earn a little and a lot (salary) for each country (native-country). What will these be for Japan?

```
In [18]: p = pd.crosstab(data["native-country"], data["salary"],
                         values=data['hours-per-week'], aggfunc="mean")
         p

Out[18]:
```

```
In [19]: p.loc["Japan"]

Out[19]: salary
         <=50K    41.000000
         >50K     47.958333
         Name: Japan, dtype: float64
```

salary	<=50K	>50K
native-country		
?	40.164760	45.547945
Cambodia	41.416667	40.000000
Canada	37.914634	45.641026
China	37.381818	38.900000
Columbia	38.684211	50.000000
Cuba	37.985714	42.440000
Dominican-Republic	42.338235	47.000000
Ecuador	38.041667	48.750000
El-Salvador	36.030928	45.000000
England	40.483333	44.533333
France	41.058824	50.750000
Germany	39.139785	44.977273
Greece	41.809524	50.625000
Guatemala	39.360656	36.666667
Haiti	36.325000	42.750000
Holand-Netherlands	40.000000	NaN
Honduras	34.333333	60.000000
Hong	39.142857	45.000000
Hungary	31.300000	50.000000
India	38.233333	46.475000
Iran	41.440000	47.500000
Ireland	40.947368	48.000000
Italy	39.625000	45.400000
Jamaica	38.239437	41.100000
Japan	41.000000	47.958333
Laos	40.375000	40.000000
Mexico	40.003279	46.575758
Nicaragua	36.093750	37.500000
Outlying-US(Guam-USVI-etc)	41.857143	NaN
Peru	35.068966	40.000000
Philippines	38.065693	43.032787
Poland	38.166667	39.000000
Poland	38.166667	39.000000
Portugal	41.939394	41.500000
Puerto-Rico	38.470588	39.416667
Scotland	39.444444	46.666667
South	40.156250	51.437500
Taiwan	33.774194	46.800000
Thailand	42.866667	58.333333
Trinadad&Tobago	37.058824	40.000000
United-States	38.799127	45.505369
Vietnam	37.193548	39.200000
Yugoslavia	41.600000	49.500000

Часть 2

Импортируем pandasql:

```
In [2]: from pandasql import sqldf
        pysqldf = lambda q: sqldf(q, globals())
```

Для выполнения данного задания возьмём два набора данных:

```
In [8]: wind = (pd.read_csv('wind speed.csv', header=None,
                            names=["row", "UNIX", "date",
                                   "time", "speed", "text"])
                            .drop("text", axis=1))
        temp = (pd.read_csv('temperature.csv', header=None,
                            names=["row", "UNIX", "date",
                                   "time", "temperature", "text"])
                            .drop("text", axis=1))
```

```
In [14]: wind.head()
```

Out[14]:

	row	UNIX	date	time	speed
0	1	1475315718	2016-09-30	23:55:18	7.87
1	2	1475315423	2016-09-30	23:50:23	7.87
2	3	1475315124	2016-09-30	23:45:24	9.00
3	4	1475314821	2016-09-30	23:40:21	13.50
4	5	1475314522	2016-09-30	23:35:22	15.75

```
In [15]: wind.dtypes

Out[15]: row        int64
         UNIX       int64
         date      object
         time      object
         speed    float64
         dtype: object
```

```
In [17]: temp.head()
```

Out[17]:

	row	UNIX	date	time	temperature
0	1	1475315718	2016-09-30	23:55:18	48
1	2	1475315423	2016-09-30	23:50:23	48
2	3	1475315124	2016-09-30	23:45:24	48
3	4	1475314821	2016-09-30	23:40:21	48
4	5	1475314522	2016-09-30	23:35:22	48

```
In [18]: temp.dtypes

Out[18]: row             int64
         UNIX            int64
         date           object
         time           object
         temperature     int64
         dtype: object
```

Объединим эти наборы и проверим время выполнения:
Pandas:

In [19]:
```
wind.merge(temp[["UNIX", "temperature"]], on="UNIX").head()
```
Out[19]:

	row	UNIX	date	time	speed	temperature
0	1	1475315718	2016-09-30	23:55:18	7.87	48
1	2	1475315423	2016-09-30	23:50:23	7.87	48
2	3	1475315124	2016-09-30	23:45:24	9.00	48
3	4	1475314821	2016-09-30	23:40:21	13.50	48
4	5	1475314522	2016-09-30	23:35:22	15.75	48

In [20]:
```
%%timeit
wind.merge(temp[["UNIX", "temperature"]], on="UNIX")
```
```
19.5 ms ± 890 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Pandasql:

In [22]:
```
pysqldf("""SELECT w.row, w.UNIX, w.date, w.time,
        w.speed, t.temperature
        FROM wind AS w JOIN temp AS t
        ON w.UNIX = t.UNIX """).head()
```
Out[22]:

	row	UNIX	date	time	speed	temperature
0	1	1475315718	2016-09-30	23:55:18	7.87	48
1	2	1475315423	2016-09-30	23:50:23	7.87	48
2	3	1475315124	2016-09-30	23:45:24	9.00	48
3	4	1475314821	2016-09-30	23:40:21	13.50	48
4	5	1475314522	2016-09-30	23:35:22	15.75	48

In [23]:
```
%%timeit
pysqldf("""SELECT w.row, w.UNIX, w.date, w.time,
        w.speed, t.temperature
        FROM wind AS w JOIN temp AS t
        ON w.UNIX = t.UNIX """).head()
```
```
809 ms ± 46 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Pandas в 40 раз быстрее, чем pandasql.

Сгруппируем набор данных с использованием функций агрегирования:
Pandas:

In [24]:
```
wind.groupby("date")["speed"].mean().head()
```
Out[24]:
```
date
2016-09-01    6.396560
2016-09-02    5.804086
2016-09-03    4.960248
2016-09-04    5.184571
2016-09-05    5.830676
Name: speed, dtype: float64
```

In [25]:
```
%%timeit
wind.groupby("date")["speed"].mean().head()
```
```
3.47 ms ± 146 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Pandasql:

```
In [26]: pysqldf("""SELECT date, AVG(speed) FROM wind GROUP BY date """).head()
```

Out[26]:

	date	AVG(speed)
0	2016-09-01	6.396560
1	2016-09-02	5.804086
2	2016-09-03	4.960248
3	2016-09-04	5.184571
4	2016-09-05	5.830676

```
In [27]: %%timeit
pysqldf("""SELECT date, AVG(speed) FROM wind GROUP BY date """).head()
```

```
322 ms ± 55.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Pandas быстрее в 93 раза.