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

Лабораторная работа ⊠2
по дисциплине
«Методы машинного обучения»
на тему
«Изучение библиотек обработки данных»

Выполнил:
студент группы ИУ5-24М
Голубев И. И.

Москва — 2019 г.

1. Цель лабораторной работы

Изучить библиотеки обработки данных Pandas и PandaSQL [?].

2. Задание

Задание состоит из двух частей.

2.1. Часть 1

Требуется выполнить первое демонстрационное задание под названием «Exploratory data analysis with Pandas» со страницы курса mlcourse.ai.

2.2. Часть 2

Требуется выполнить 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.

Также требуется сравнить время выполнения каждого запроса в Pandas и PandaSQL.

3. Ход выполнения работы

3.1. 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.

#

mlcourse.ai – Open Machine Learning Course

Author: Yury Kashnitskiy. Translated and edited by Sergey Isaev, Artem Trunov, Anastasia Manokhina, and Yuanyuan Pao This material is subject to the terms and conditions of the Creative Commons CC BY-NC-SA 4.0 license. Free use is permitted for any non-commercial purpose.

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`.

Importing all required packages:

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

Setting maximum display width for text report [?]:

```
In [2]: pd.set_option("display.width", 70)
```

Loading data:

```
In [3]: data = pd.read_csv('adult.data.csv')
        data.head()
```

```
Out[3]:    age         workclass  fnlwgt  education  education-num  \
        0   39         State-gov   77516  Bachelors             13
        1   50  Self-emp-not-inc   83311  Bachelors             13
        2   38           Private  215646    HS-grad              9
        3   53           Private  234721       11th              7
        4   28           Private  338409  Bachelors             13

               marital-status         occupation   relationship   race  \
```

```
0       Never-married       Adm-clerical  Not-in-family  White
1  Married-civ-spouse    Exec-managerial        Husband  White
2            Divorced  Handlers-cleaners  Not-in-family  White
3  Married-civ-spouse  Handlers-cleaners        Husband  Black
4  Married-civ-spouse     Prof-specialty           Wife  Black

      sex  capital-gain  capital-loss  hours-per-week  \
0    Male          2174             0              40
1    Male             0             0              13
2    Male             0             0              40
3    Male             0             0              40
4  Female             0             0              40

  native-country salary
0  United-States  <=50K
1  United-States  <=50K
2  United-States  <=50K
3  United-States  <=50K
4           Cuba  <=50K
```

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

```
In [4]: data["sex"].value_counts()
```

```
Out[4]: Male      21790
        Female    10771
        Name: sex, dtype: int64
```

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

```
In [5]: data[data["sex"] == "Female"]["age"].mean()
```

```
Out[5]: 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 [7]: 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 [8]: 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[8]: 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 [9]: data.groupby(["race", "sex"])["age"].describe()
```

```
Out[9]:                               count       mean        std   min  \
        race               sex
        Amer-Indian-Eskimo Female    119.0  37.117647  13.114991  17.0
                           Male      192.0  37.208333  12.049563  17.0
        Asian-Pac-Islander Female    346.0  35.089595  12.300845  17.0
                           Male      693.0  39.073593  12.883944  18.0
        Black              Female   1555.0  37.854019  12.637197  17.0
                           Male     1569.0  37.682600  12.882612  17.0
        Other              Female    109.0  31.678899  11.631599  17.0
                           Male      162.0  34.654321  11.355531  17.0
        White              Female   8642.0  36.811618  14.329093  17.0
                           Male    19174.0  39.652498  13.436029  17.0

                                    25%   50%    75%   max
        race               sex
        Amer-Indian-Eskimo Female  27.0  36.0  46.00  80.0
                           Male    28.0  35.0  45.00  82.0
        Asian-Pac-Islander Female  25.0  33.0  43.75  75.0
                           Male    29.0  37.0  46.00  90.0
        Black              Female  28.0  37.0  46.00  90.0
                           Male    27.0  36.0  46.00  90.0
        Other              Female  23.0  29.0  39.00  74.0
                           Male    26.0  32.0  42.00  77.0
        White              Female  25.0  35.0  46.00  90.0
                           Male    29.0  38.0  49.00  90.0
```

```
In [10]: data[(data["race"] == "Amer-Indian-Eskimo")
              & (data["sex"] == "Male")]["age"].max()
```

```
Out[10]: 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 [11]: 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[11]: 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 [12]: 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 [13]: p = pd.crosstab(data["native-country"], data["salary"],
                         values=data['hours-per-week'], aggfunc="mean")
         p
```

```
Out[13]: 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
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
```

In [14]:
```
p.loc["Japan"]
```

Out[14]:
```
salary
<=50K    41.000000
>50K     47.958333
Name: Japan, dtype: float64
```

3.2. Часть 2

Импортируем pandasql:

In [15]:
```
from pandasql import sqldf
pysqldf = lambda q: sqldf(q, globals())
```

Для выполнения данного задания возьмём два набора данных из исходных данных, представленных в открытом доступе:

```
In [16]: 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))
```

Посмотрим на эти наборы данных:

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

Out[17]:    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 [18]: wind.dtypes

Out[18]: row         int64
         UNIX        int64
         date       object
         time       object
         speed     float64
         dtype: object

In [19]: temp.head()

Out[19]:    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 [20]: temp.dtypes

Out[20]: row             int64
         UNIX            int64
         date           object
         time           object
         temperature     int64
         dtype: object
```

Объединим эти наборы данных различными способами, проверяя время их выполнения [?, ?, ?]:

```
In [21]: wind.merge(temp[["UNIX", "temperature"]], on="UNIX").head()
```

```
Out[21]:    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 [22]: %%timeit
         wind.merge(temp[["UNIX", "temperature"]], on="UNIX")
```

```
15.8 ms ± 856 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
In [23]: 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[23]:    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 [24]: %%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
                    """)
```

```
695 ms ± 35.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Видно, что `pandasql` в 50 раз медленнее, чем `pandas`.

Сгруппируем набор данных с использованием функций агрегирования различными способами:

```
In [25]: wind.groupby("date")["speed"].mean().head()
```

```
Out[25]: 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 [26]: %%timeit
         wind.groupby("date")["speed"].mean()
```

```
2.72 ms ± 124 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

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

```
Out[27]:          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 [28]: %%timeit
         pysqldf("""SELECT date, AVG(speed)
                    FROM wind
                    GROUP BY date
                 """)
```

```
257 ms ± 6.72 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Здесь разница уже более чем в 100 раз. Таким образом для таких простых запросов проще использовать Pandas.

Список литературы

[1] Гапанюк Ю. Е. Лабораторная работа «Изучение библиотек обработки данных» [Электронный ресурс] // GitHub. — 2019. — Режим доступа: https://github.com/ugapanyuk/ml_course/wiki/LAB_PANDAS (дата обращения: 20.02.2019).

[2] pandas 0.24.1 documentation [Electronic resource] // PyData. — 2019. — Access mode: http://pandas.pydata.org/pandas-docs/stable/ (online; accessed: 20.02.2019).

[3] You are my Sunshine [Electronic resource] // Space Apps Challenge. — 2017. — Access mode: https://2017.spaceappschallenge.org/challenges/earth-and-us/you-are-my-sunshine/details (online; accessed: 22.02.2019).

[4] yhat/pandasql: sqldf for pandas [Electronic resource] // GitHub. — 2017. — Access mode: https://github.com/yhat/pandasql (online; accessed: 22.02.2019).

[5] Team The IPython Development. IPython 7.3.0 Documentation [Electronic resource] // Read the Docs. — 2019. — Access mode: https://ipython.readthedocs.io/en/stable/ (online; accessed: 20.02.2019).