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 ИУ5-21М
Коробко Д. О.

_____

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`).

```
In [89]: import numpy as np
         import pandas as pd
         pd.set_option('display.max.columns', 100)
         pd.set_option("display.width", 70)
         import matplotlib.pyplot as plt
         import seaborn as sns
```

Подключение предлагаемой выборки и проверка выводом первых пяти записей:

```
In [90]: data = pd.read_csv('data/adult.data.csv')
         data.head()
```

```
Out[90]:    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 [91]: data["sex"].value_counts()

Out[91]:  Male      21790
          Female    10771
         Name: sex, dtype: int64
```

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

```
In [92]: data[data["sex"] == "Female"]["age"].mean()

Out[92]: nan
```

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

```
In [93]: print("{0:%}".format(data[data["native-country"] == "Germany"]
                              .shape[0] / data.shape[0]))

0.000000%
```

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 [94]: 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: = nan ± nan years
 >50K: = nan ± nan 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 [95]: 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[95]: True
```

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

```
Out[96]:                                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 [97]: data[(data["race"] == "Amer-Indian-Eskimo")
              & (data["sex"] == "Male")]["age"].max()
```

```
Out[97]: nan
```

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 [98]: 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[98]: Series([], 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 [99]: 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.
0.000000% 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`).

```
In [100]: p = pd.crosstab(data["native-country"], data["salary"],
                          values=data['hours-per-week'], aggfunc="mean")
          p
```

```
Out[100]: 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
```

3.2. Часть 2

Импорт pandasql:

```
In [101]: import pandasql as ps
          from timeit import default_timer as timer

          pandasqldf = lambda q: ps.sqldf(q, globals())
```

Загрузка двух наборов данных:

```
In [102]: device = pd.read_csv('data/user_device.csv')
          usage = pd.read_csv('data/user_usage.csv')

In [103]: device.head()

Out[103]:    use_id  user_id platform  platform_version     device  \
          0   22782    26980      ios              10.2  iPhone7,2
          1   22783    29628  android               6.0    Nexus 5
          2   22784    28473  android               5.1   SM-G903F
          3   22785    15200      ios              10.2  iPhone7,2
          4   22786    28239  android               6.0  ONE E1003
```

```
   use_type_id
0            2
1            3
2            1
3            3
4            1
```

```
In [104]: usage.head()
```

```
Out[104]:    outgoing_mins_per_month  outgoing_sms_per_month  monthly_mb  \
0                            21.97                    4.82     1557.33
1                          1710.08                  136.88     7267.55
2                          1710.08                  136.88     7267.55
3                            94.46                   35.17      519.12
4                            71.59                   79.26     1557.33

   use_id
0   22787
1   22788
2   22789
3   22790
4   22792
```

Проведем два теста, в которых мы сравним время выполнения запросов в Pandas и PandasSQL. В первом тесте производится выполнение запроса SQL с соединением таблиц (наборов данных).

```
In [105]: t1_1 = timer()
          test1_1 = pandasqldf("""
                        SELECT *
                        FROM device JOIN usage
                        ON device.use_id==usage.use_id
                        """)
          elapsed1_1 = timer() - t1_1
          test1_1.head()
```

```
Out[105]:    use_id  user_id platform  platform_version    device  \
0   22787    12921  android               4.3  GT-I9505
1   22788    28714  android               6.0  SM-G930F
2   22789    28714  android               6.0  SM-G930F
3   22790    29592  android               5.1     D2303
4   22792    28217  android               5.1  SM-G361F

   use_type_id  outgoing_mins_per_month  outgoing_sms_per_month  \
0            1                    21.97                    4.82
1            1                  1710.08                  136.88
2            1                  1710.08                  136.88
3            1                    94.46                   35.17
4            1                    71.59                   79.26
```

```
   monthly_mb  use_id
0     1557.33   22787
1     7267.55   22788
2     7267.55   22789
3      519.12   22790
4     1557.33   22792
```

```
In [106]: t1_2 = timer()
          test1_2 = device.merge(usage)
          elapsed1_2 = timer() - t1_2

In [107]: print("Тест 1:\nPandasSQL - {0:.6f} с.\nPandas - {1:.6f} с.".format(

Тест 1:
PandasSQL - 0.014057 с.
Pandas - 0.003904 с.
```

Из результатов первого теста видно, что выполнение запроса SQL с соединением таблиц происходит значительно быстрее при использовании Pandas. Для полученной после соединения таблицы выполним запрос SQL с группировкой:

```
In [108]: t2_1 = timer()
          test2_1 = pandasqldf("""
                        SELECT DISTINCT
                        device, avg(monthly_mb) as avg_mb
                        FROM test1_1
                        GROUP BY device
                        """)
          elapsed2_1 = timer() - t2_1
          test2_1.head()

Out[108]:   device    avg_mb
          0  A0001  15573.33
          1  C6603   1557.33
          2  D2303    519.12
          3  D5503   1557.33
          4  D5803   1557.33

In [109]: t2_2 = timer()
          test2_2 = test1_2.groupby('device').monthly_mb.mean()
          elapsed2_2 = timer() - t2_2

In [110]: print("Тест 2:\nPandasSQL - {0:.6f} с.\nPandas - {1:.6f} с.".format(

Тест 2:
PandasSQL - 0.008167 с.
Pandas - 0.002052 с.
```

Второй тест также показывает, что при выполнении запроса с группировкой Pandas опережает PandasSQL.