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 ИУ5-23М
Умряев Д. Т.

Москва — 2020 г.

1. Цель лабораторной работы

Изучить библиотеки обработки данных Pandas и PandaSQL [1].

2. Задание

Задание состоит из двух частей [1].

2.1. Часть 1

Требуется выполнить первое демонстрационное задание “demo assignment” под названием “Exploratory data analysis with Pandas” со страницы курса https://mlcourse.ai/assignments

Условие задания - assignment01_pandas_uci_adult

Официальный датасет находится здесь, но данные и заголовки хранятся отдельно, что неудобно для анализа - https://archive.ics.uci.edu/ml/datasets/Adult

Поэтому готовый набор данных для лабораторной работы удобнее скачать здесь - https://raw.githubusercontent.com/Yorko/mlcourse.ai/master/data/adult.data.csv (удобнее всего нажать на данной ссылке правую кнопку мыши и выбрать в контекстном меню пункт “сохранить ссылку”, будет предложено сохранить файл в формате CSV)

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

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 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/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 - pandasql_example

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

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 Kashnitsky. Translated and edited by Sergey Isaev, Artem Trunov, Anastasia Manokhina, and Yuanyuan Pao. All content is distributed under the Creative Commons CC BY-NC-SA 4.0 license.

Assignment #1 (demo)

Exploratory data analysis with Pandas

Same assignment as a Kaggle Kernel + solution.
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

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

```
pd.set_option('display.max.columns', 100)
# to draw pictures in jupyter notebook
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
# we don't like warnings
# you can comment the following 2 lines if you'd like to
import warnings
warnings.filterwarnings('ignore')
```

Setting maximum display width for text report [2]:

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

Loading data:

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

[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?

[4]:
```
data['sex'].value_counts()
# data.count()
```

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

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

[5]:
```
data.loc[data['sex'] == 'Female', 'age'].mean()
```

[5]:
```
36.85823043357163
```

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

[6]:
```
# data.loc[data['native-country'] == 'Germany'].count()[0]/data.shape[0]
(data['native-country'] == 'Germany').sum()/data.shape[0]
```

[6]:
```
0.004207487485028101
```

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?

[7]:
```
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} years.".format(
    round(ages1.mean()), round(ages1.std(), 1),
    round(ages2.mean()), round(ages2.std(), 1)))
```

```
The average age of the rich: 37 ± 14.0 years,poor - 44 ± 10.5 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)

[8]:
```
educations_more50K = data.loc[data["salary"] == ">50K", "education"]
high_educations = ["Bachelors", "Prof-school", "Assoc-acdm",
                   "Assoc-voc", "Masters", "Doctorate"]
num_high_edus_more50K = educations_more50K.isin(high_educations).sum()
num_edus_more50K = educations_more50K.shape[0]
num_high_edus_more50K == num_edus_more50K
```

[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.

[9]:
```
data.groupby(['race', 'sex'])['age'].describe()
```

[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
```

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

[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.

[11]:
```
married = ["Married-civ-spouse", "Married-spouse-absent",
           "Married-AF-spouse"]
not_married = ["Divorced", "Never-married", "Separated", "Widowed"]
men_more50K = data.loc[(data["salary"] == ">50K") &
                       (data["sex"] == "Male"), "marital-status"]
print("Married: {0}".format(men_more50K.isin(married).sum()))
print("Single: {0}".format(men_more50K.isin(not_married).sum()))
```

```
Married: 5965
Single: 697
```

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?

[12]:
```
max_hours = data["hours-per-week"].max()
print("Maximum number of hours a person works per week: "
      "{0}".format(max_hours))

num_people_works_max_hours = data[data["hours-per-week"] ==
                                  max_hours].shape[0]
print("Number of people who works such a number of hours: "
      "{0}".format(num_people_works_max_hours))

num_earn_more50K = data[(data["hours-per-week"] == max_hours) &
```

```
(data["salary"] == ">50K")].shape[0] / num_people_works_max_hours

print("Percentage of those who earn a lot (>50K) among them: "
      "{0:%}".format(num_earn_more50K))
```

```
Maximum number of hours a person works per week: 99
Number of people who works such a number of hours: 85
Percentage of those who earn a lot (>50K) among them: 29.411765%
```

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?

S1mple:

[13]:
```
for (country, salary), sub_df in data.groupby(['native-country',
                                               'salary']):
    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
```

Elegant:

```
[14]: pd.crosstab(data['native-country'], data['salary'],
                 values=data['hours-per-week'], aggfunc=np.mean)
```

```
[14]: 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

Импортируем pandasql:

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

Для выполнения данного задания возьмём два набора данных из приложения KillBiller и некоторые загруженные данные, содержащиеся в двух файлах CSV [3]:

```
[16]: user_usage = pd.read_csv('user_usage.csv')
      user_device = pd.read_csv('user_device.csv')
```

```
[17]: user_usage.head()
```

```
[17]:    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
```

```
[18]: user_usage.dtypes
```

```
[18]: outgoing_mins_per_month    float64
      outgoing_sms_per_month     float64
      monthly_mb                 float64
      use_id                       int64
      dtype: object
```

```
[19]: user_device.head()
```

```
[19]:    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
```

```
[20]: user_device.dtypes
```

```
[20]: use_id                int64
      user_id               int64
      platform             object
      platform_version    float64
      device               object
      use_type_id           int64
      dtype: object
```

Объединим эти наборы данных различными способами, проверяя время их выполнения [2, 4, 5]:

```
[21]: user_usage.merge(user_device[["use_id", "platform", "device"]],
                       on="use_id").head()
```

```
[21]:    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 platform    device
      0   22787  android  GT-I9505
      1   22788  android  SM-G930F
      2   22789  android  SM-G930F
      3   22790  android     D2303
      4   22792  android  SM-G361F
```

```
[22]: %%timeit
      user_usage.merge(user_device[["use_id", "platform", "device"]],
                       on="use_id")
```

```
8.24 ms ± 1.29 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
[23]: pysqldf("""SELECT u_u.outgoing_mins_per_month,
                        u_u.outgoing_sms_per_month,
                        u_u.monthly_mb, u_u.use_id,
                        u_d.platform, u_d.device
                 FROM user_usage AS u_u JOIN user_device AS u_d
                 ON u_u.use_id = u_d.use_id
              """).head()
```

```
[23]:    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 platform    device
0   22787  android  GT-I9505
1   22788  android  SM-G930F
2   22789  android  SM-G930F
3   22790  android     D2303
4   22792  android  SM-G361F
```

[24]:
```
%%timeit
pysqldf("""SELECT u_u.outgoing_mins_per_month,
                  u_u.outgoing_sms_per_month,
                  u_u.monthly_mb, u_u.use_id,
                  u_d.platform, u_d.device
           FROM user_usage AS u_u JOIN user_device AS u_d
           ON u_u.use_id = u_d.use_id
        """)
```

```
31.9 ms ± 3.32 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Видно, что pandasql в 5 раз медленнее, чем pandas.

Сгруппируем набор данных с использованием функций агрегирования различными способами:

[25]:
```
user_device.groupby("platform")["platform"].count().head()
```

[25]:
```
platform
android    184
ios         88
Name: platform, dtype: int64
```

[26]:
```
%%timeit
user_device.groupby("platform")["device"].count()
```

```
1.4 ms ± 34.5 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

[27]:
```
pysqldf("""SELECT Count(platform) AS "num devices", platform
           FROM user_device
           GROUP BY platform
        """).head()
```

[27]:
```
   num devices platform
0          184  android
1           88      ios
```

[28]:
```
%%timeit
pysqldf("""SELECT Count(platform) AS "num devices", platform
           FROM user_device
           GROUP BY platform
        """)
```

```
17.7 ms ± 506 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Здесь разница уже более чем в 10 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. — 2020. — Режим доступа: https://github.com/ugapanyuk/ml_course_2020/wiki/LAB_MMO__PANDAS (дата обращения: 13.03.2020).

[2] pandas 1.0.1 documentation [Electronic resource] // PyData. — 2020. — Access mode: http://pandas.pydata.org/pandas-docs/stable/ (online; accessed: 13.03.2020).

[3] KillBiller [Electronic resource] // KillBiller. — 2018. — Access mode: https://www.shanelynn.ie/merge-join-dataframes-python-pandas-index-1/ (online; accessed: 13.03.2020).

[4] yhat/pandasql: sqldf for pandas [Electronic resource] // GitHub. — 2017. — Access mode: https://github.com/yhat/pandasql (online; accessed: 13.03.2020).

[5] Team The IPython Development. IPython 7.13.0 Documentation [Electronic resource] // Read the Docs. — 2020. — Access mode: https://ipython.readthedocs.io/en/stable/ (online; accessed: 13.03.2020).