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

Наинг Ко Ко Линн

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 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:

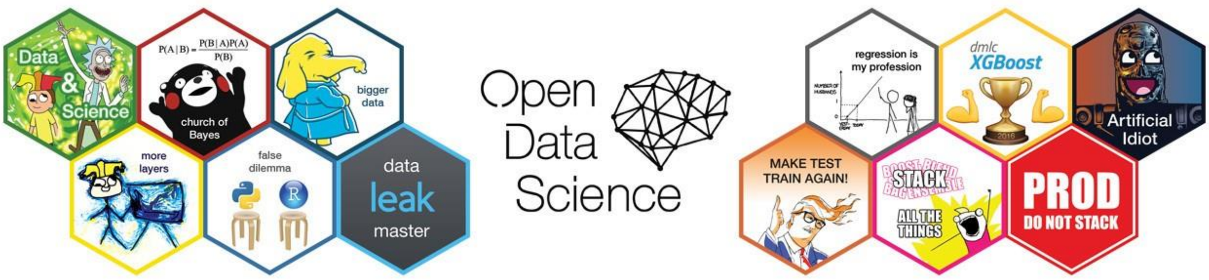
- один произвольный запрос на соединение двух наборов данных,
- один произвольный запрос на группировку набора данных с использованием функций агрегирования.

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 noncommercial 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 [2]:

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

Loading data:

```
In [3]: data = pd.read_csv('adult11.csv')
              data.head()
```

```
data = pd.read_csv('adult11.csv')
data.head()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	salary
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0.0	0.0	40.0	United-States	<=50K
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0.0	0.0	50.0	United-States	<=50K
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0.0	0.0	40.0	United-States	>50K
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688.0	0.0	40.0	United-States	>50K
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0.0	0.0	30.0	United-States	<=50K

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

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

```
data["gender"].value_counts()
```

```
Male      19301
Female     9553
Name: gender, dtype: int64
```

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

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

```
data[data["gender"] == "Female"]["age"].mean()
```

```
36.98230922223385
```

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]))
```

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

```
0.440132%
```

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()))
```

```
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: = 36.90999590890495 ± 14.166242134563927 years
 >50K: = 44.1671772428884 ± 10.52773245753095 years
```

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

```
In [8high_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()
```

```
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()
```

```
False
```

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()
```

```python
data.groupby(["race", "gender"])["age"].describe()
```

race	gender	count	mean	std	min	25%	50%	75%	max
Amer-Indian-Eskimo	**Female**	115.0	34.930435	12.322937	17.0	25.00	33.0	41.5	63.0
	Male	160.0	37.125000	11.781955	17.0	29.00	35.0	44.0	82.0
Asian-Pac-Islander	**Female**	300.0	36.350000	13.155139	17.0	26.00	35.0	45.0	81.0
	Male	564.0	38.843972	12.891560	18.0	29.00	37.0	47.0	90.0
Black	**Female**	1338.0	38.354260	12.967195	17.0	28.00	37.0	47.0	81.0
	Male	1422.0	38.059775	13.239203	17.0	28.00	36.0	46.0	90.0
Other	**Female**	80.0	30.737500	10.861317	17.0	22.75	28.0	36.0	64.0
	Male	153.0	34.954248	11.800613	17.0	26.00	32.0	41.0	67.0
White	**Female**	7720.0	36.864378	14.488046	17.0	25.00	35.0	46.0	90.0
	Male	17002.0	39.650629	13.475728	17.0	29.00	38.0	49.0	90.0

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

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

```
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, Marriedspouse-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["gender"] == "Male") & (data["salary"] == ">50K")]
    ["married"].value_counts())
```

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

```
True     5237
False     567
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))
```

```python
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 99.0 hours/week.
75 people work this time at week.
32.000000% 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). What will these be for Japan?

```
In [13]: p = pd.crosstab(data["native-country"], data["salary"],

values=data['hours-per-week'], aggfunc="mean")p
p
```

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

salary	<=50K	>50K
native-country		
?	38.637883	45.384058
Cambodia	41.307692	47.000000
Canada	36.411765	46.456522
China	35.900000	44.380952
Columbia	40.513514	56.250000
Cuba	40.825397	40.937500
Dominican-Republic	40.962963	42.800000
Ecuador	36.875000	47.333333
El-Salvador	35.231707	43.600000
England	38.148936	46.384615
France	41.100000	39.166667
Germany	38.223404	45.969697
Greece	44.333333	56.428571
Guatemala	38.062500	40.000000

Haiti	34.386364	39.166667
Honduras	32.090909	50.000000
Hong	38.181818	40.000000
Hungary	36.000000	42.666667
India	39.207547	44.470588
Iran	39.458333	48.000000
Ireland	40.933333	44.111111
Italy	38.333333	44.500000
Jamaica	39.780000	43.909091
Japan	38.972973	43.666667
Laos	37.777778	NaN
Mexico	39.870229	46.214286
Nicaragua	36.760000	40.000000
Outlying-US(Guam-USVI-etc)	41.666667	40.000000
Peru	37.318182	40.000000
Philippines	39.096000	44.585366
Poland	38.795455	38.300000
Portugal	40.000000	47.100000
Puerto-Rico	39.101852	39.866667
Scotland	40.000000	NaN
South	42.721311	51.666667
Taiwan	39.600000	44.000000
Thailand	46.500000	53.750000
Trinadad&Tobago	39.266667	NaN
United-States	38.883042	45.430669
Vietnam	39.523810	40.000000
Yugoslavia	36.500000	32.500000

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

```
p.loc["Japan"]
```

```
salary
<=50K     38.972973
>50K      43.666667
Name: Japan, dtype: float64
```

3.2. Часть 2

Импортируем pandasql:

```
In [15]: !pip install pandasql

from pandasql import sqldf
import pandas as pd
In [16]: from pandasql import sqldf

pysqldf = lambda q: sqldf(q, globals())
```

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

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

```
wind.head()
```

	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
```

```
[ ]  1  wind.dtypes
```

```
row          int64
UNIX         int64
date        object
time        object
speed      float64
dtype: object
```

```
In [19]: temp.head()
```

```
1  temp.head()
```

	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
```

```
[ ]  1  temp.dtypes
```

```
row                int64
UNIX               int64
date              object
time              object
temperature        int64
dtype: object
```

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

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

	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")
```

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

```
100 loops, best of 3: 10.9 ms per loop
```

```
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()
```

```
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()
```

	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
            """)
```

```
%%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
          """)
```

```
1 loop, best of 3: 505 ms per loop
```

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()
```

```
1  wind.groupby("date")["speed"].mean().head()
```

```
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
    ()
```

```
[ ]  1  %%timeit
     2  wind.groupby("date")["speed"].mean()
```

```
100 loops, best of 3: 2.72 ms per loop
```

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

```
[ ]  1  pysqldf("""SELECT date, AVG(speed)
     2  FROM wind
     3  GROUP BY date
     4  """).head()
```

	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
                """)
```

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

```
1 loop, best of 3: 202 ms per loop
```

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).