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:
студентка группы ИУ5-21М
Попова И.А.

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

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.

3.1 Часть 1

Разведочный анализ данных с Pandas

In [41]:

```
import pandas as pd
import numpy as np
pd.set_option('display.max.columns', 100)
%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')
```

In [43]:

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

Out[43]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black

In [11]:

```
#1. How many men and women (sex feature) are represented in this dataset?
data['sex'].value_counts()
```

Out[11]:

```
Male      21790
Female    10771
Name: sex, dtype: int64
```

In [12]:

```
#2. What is the average age (age feature) of women?
female_data = data[data['sex'] == 'Female']
```

In [17]:

```
female_data['age'].mean()
```

Out[17]:

```
36.85823043357163
```

In [18]:

```
#alt
data.loc[data['sex'] == 'Female', 'age'].mean()
```

Out[18]:

```
36.85823043357163
```

In [21]:

```
#3. What is the proportion of German citizens (native-country feature)?
float((data['native-country'] == 'Germany').sum()) / data.shape[0]
```

Out[21]:

```
0.004207487485028101
```

In [26]:

```
#4-5. What are mean value and standard deviation of the age
#of those who recieve more than 50K per year (salary feature) and those who receive les
s than 50K per year?
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: 44.0 +- 10.5 years, poor - 37.0 +- 14.0 year
s.
```

In [27]:

```
#6. Is it true that people who receive more than 50k have at least high school educatio
n?
#(education - Bachelors, Prof-school, Assoc-acdm, Assoc-voc, Masters or Doctorate featu
re)
data.loc[data['salary'] == '>50K', 'education'].unique()
```

Out[27]:

```
array(['HS-grad', 'Masters', 'Bachelors', 'Some-college', 'Assoc-voc',
       'Doctorate', 'Prof-school', 'Assoc-acdm', '7th-8th', '12th',
       '10th', '11th', '9th', '5th-6th', '1st-4th'], dtype=object)
```

In [31]:

```
#7. Display statistics of age for each race (race feature) and each gender.
#Use groupby() and describe(). Find the maximum age of men of Amer-Indian-Eskimo race.
#data.groupby(['race', 'sex']).describe()
for (race, sex), sub_df in data.groupby(['race', 'sex']):
    print('Race: {0}, sex {1}'.format(race, sex))
    print(sub_df['age'].describe())
```

```
Race: Amer-Indian-Eskimo, sex Female
count    119.000000
mean      37.117647
std       13.114991
min       17.000000
25%       27.000000
50%       36.000000
75%       46.000000
max       80.000000
Name: age, dtype: float64
Race: Amer-Indian-Eskimo, sex Male
count    192.000000
mean      37.208333
std       12.049563
min       17.000000
25%       28.000000
50%       35.000000
75%       45.000000
max       82.000000
Name: age, dtype: float64
Race: Asian-Pac-Islander, sex Female
count    346.000000
mean      35.089595
std       12.300845
min       17.000000
25%       25.000000
50%       33.000000
75%       43.750000
max       75.000000
Name: age, dtype: float64
Race: Asian-Pac-Islander, sex Male
count    693.000000
mean      39.073593
std       12.883944
min       18.000000
25%       29.000000
50%       37.000000
75%       46.000000
max       90.000000
Name: age, dtype: float64
Race: Black, sex Female
count    1555.000000
mean       37.854019
std        12.637197
min        17.000000
25%        28.000000
50%        37.000000
75%        46.000000
max        90.000000
Name: age, dtype: float64
Race: Black, sex Male
count    1569.000000
mean       37.682600
std        12.882612
min        17.000000
25%        27.000000
50%        36.000000
75%        46.000000
max        90.000000
Name: age, dtype: float64
Race: Other, sex Female
```

```
count     109.000000
mean       31.678899
std        11.631599
min        17.000000
25%        23.000000
50%        29.000000
75%        39.000000
max        74.000000
Name: age, dtype: float64
Race: Other, sex Male
count     162.000000
mean       34.654321
std        11.355531
min        17.000000
25%        26.000000
50%        32.000000
75%        42.000000
max        77.000000
Name: age, dtype: float64
Race: White, sex Female
count     8642.000000
mean        36.811618
std         14.329093
min         17.000000
25%         25.000000
50%         35.000000
75%         46.000000
max         90.000000
Name: age, dtype: float64
Race: White, sex Male
count     19174.000000
mean         39.652498
std          13.436029
min          17.000000
25%          29.000000
50%          38.000000
75%          49.000000
max          90.000000
Name: age, dtype: float64
```

1. Among whom the proportion of those who earn a lot(>50K) is more: among married or single men (marital-status feature)? Consider 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 [118]:

```
data.loc[
    (data['sex'] == 'Male') &
    (data['marital-status'].isin(['Never-married', 'Separated', 'Divorced','Widowed'
])), 'salary'
].value_counts()
```

Out[118]:

```
<=50K    7552
>50K      697
Name: salary, dtype: int64
```

In [36]:

```
data.loc[(data['sex'] == 'Male') &
     (data['marital-status'].str.startswith('Married')), 'salary'].value_counts()
```

Out[36]:

```
<=50K    7576
>50K     5965
Name: salary, dtype: int64
```

In [37]:

```
data['marital-status'].value_counts()
```

Out[37]:

```
Married-civ-spouse       14976
Never-married            10683
Divorced                  4443
Separated                 1025
Widowed                    993
Married-spouse-absent      418
Married-AF-spouse           23
Name: marital-status, dtype: int64
```

1. 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 among them?

In [38]:

```
max_load = data['hours-per-week'].max()
print("Max time - {0} hours./week.".format(max_load))

num_workaholics = data[data['hours-per-week'] == max_load].shape[0]
print("Total number of such hard workers {0}".format(num_workaholics))

rich_share = float(data[(data['hours-per-week'] == max_load)
                 & (data['salary'] == '>50K')].shape[0]) / num_workaholics
print("Percentage of rich among them {0}%".format(int(100 * rich_share)))
```

```
Max time - 99 hours./week.
Total number of such hard workers 85
Percentage of rich among them 29%
```

1. Count the average time of work (hours-per-week) those who earning a little and a lot (salary) for each country (native-country).

In [39]:

```
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
```

In [42]:

```
pd.crosstab(data['native-country'], data['salary'],
           values=data['hours-per-week'], aggfunc=np.mean).T
```

Out[42]:

native-country	?	Cambodia	Canada	China	Columbia	Cuba	Dominican-Republic	Ec
salary								
<=50K	40.164760	41.416667	37.914634	37.381818	38.684211	37.985714	42.338235	38.0
>50K	45.547945	40.000000	45.641026	38.900000	50.000000	42.440000	47.000000	48.7

3.2 Часть 2

Использование Pandas для запросов на соединение и группировку данных

In [44]:

```
user_usage = pd.read_csv("user_usage.csv")
user_device = pd.read_csv("user_device.csv")
devices = pd.read_csv("android_devices.csv")
```

In [45]:

```
user_usage.head()
```

Out[45]:

	outgoing_mins_per_month	outgoing_sms_per_month	monthly_mb	use_id
0	21.97	4.82	1557.33	22787
1	1710.08	136.88	7267.55	22788
2	1710.08	136.88	7267.55	22789
3	94.46	35.17	519.12	22790
4	71.59	79.26	1557.33	22792

In [48]:

```
user_device.head(3)
```

Out[48]:

	use_id	user_id	platform	platform_version	device	use_type_id
0	22782	26980	ios	10.2	iPhone7,2	2
1	22783	29628	android	6.0	Nexus 5	3
2	22784	28473	android	5.1	SM-G903F	1

In [47]:

```
devices.head(10)
```

Out[47]:

	Retail Branding	Marketing Name	Device	Model
0	NaN	NaN	AD681H	Smartfren Andromax AD681H
1	NaN	NaN	FJL21	FJL21
2	NaN	NaN	T31	Panasonic T31
3	NaN	NaN	hws7721g	MediaPad 7 Youth 2
4	3Q	OC1020A	OC1020A	OC1020A
5	7Eleven	IN265	IN265	IN265
6	A.O.I. ELECTRONICS FACTORY	A.O.I.	TR10CS1_11	TR10CS1
7	AG Mobile	AG BOOST 2	BOOST2	E4010
8	AG Mobile	AG Flair	AG_Flair	Flair
9	AG Mobile	AG Go Tab Access 2	AG_Go_Tab_Access_2	AG_Go_Tab_Access_2

In [49]:

```
result = pd.merge(user_usage,
                  user_device[['use_id', 'platform', 'device']],
                  on='use_id')
result.head()
```

Out[49]:

	outgoing_mins_per_month	outgoing_sms_per_month	monthly_mb	use_id	platform	devic
0	21.97	4.82	1557.33	22787	android	GT I950
1	1710.08	136.88	7267.55	22788	android	SM G930
2	1710.08	136.88	7267.55	22789	android	SM G930
3	94.46	35.17	519.12	22790	android	D230
4	71.59	79.26	1557.33	22792	android	SM G361

In [50]:

```
print("user_usage dimensions: {}".format(user_usage.shape))
print("user_device dimensions: {}".format(user_device[['use_id', 'platform', 'device']]
.shape))
```

```
user_usage dimensions: (240, 4)
user_device dimensions: (272, 3)
```

In [51]:

```
user_usage['use_id'].isin(user_device['use_id']).value_counts()
```

Out[51]:

```
True     159
False     81
Name: use_id, dtype: int64
```

In [52]:

```
result = pd.merge(user_usage,
                  user_device[['use_id', 'platform', 'device']],
                  on='use_id', how='left')
print("user_usage dimensions: {}".format(user_usage.shape))
print("result dimensions: {}".format(result.shape))
print("There are {} missing values in the result.".format(
        result['device'].isnull().sum()))
```

```
user_usage dimensions: (240, 4)
result dimensions: (240, 6)
There are 81 missing values in the result.
```

In [53]:

```
result.head()
```

Out[53]:

	outgoing_mins_per_month	outgoing_sms_per_month	monthly_mb	use_id	platform	devic
0	21.97	4.82	1557.33	22787	android	GT I950
1	1710.08	136.88	7267.55	22788	android	SM G930
2	1710.08	136.88	7267.55	22789	android	SM G930
3	94.46	35.17	519.12	22790	android	D230
4	71.59	79.26	1557.33	22792	android	SM G361

In [54]:

```
result.tail()
```

Out[54]:

	outgoing_mins_per_month	outgoing_sms_per_month	monthly_mb	use_id	platform	dev
235	260.66	68.44	896.96	25008	NaN	N
236	97.12	36.50	2815.00	25040	NaN	N
237	355.93	12.37	6828.09	25046	NaN	N
238	632.06	120.46	1453.16	25058	NaN	N
239	488.70	906.92	3089.85	25220	NaN	N

In [55]:

```
result = pd.merge(user_usage,
                  user_device[['use_id', 'platform', 'device']],
                  on='use_id', how='right')
print("user_device dimensions: {}".format(user_device.shape))
print("result dimensions: {}".format(result.shape))
print("There are {} missing values in the 'monthly_mb' column in the result.".format(
        result['monthly_mb'].isnull().sum()))
print("There are {} missing values in the 'platform' column in the result.".format(
        result['platform'].isnull().sum()))
```

```
user_device dimensions: (272, 6)
result dimensions: (272, 6)
There are 113 missing values in the 'monthly_mb' column in the result.
There are 0 missing values in the 'platform' column in the result.
```

In [56]:

```
print("There are {} unique values of use_id in our dataframes.".format(
        pd.concat([user_usage['use_id'], user_device['use_id']]).unique().shape[0]))
result = pd.merge(user_usage,
                 user_device[['use_id', 'platform', 'device']],
                 on='use_id', how='outer', indicator=True)

print("Outer merge result has {} rows.".format(result.shape))

print("There are {} rows with no missing values.".format(
    (result.apply(lambda x: x.isnull().sum(), axis=1) == 0).sum()))
```

```
There are 353 unique values of use_id in our dataframes.
Outer merge result has (353, 7) rows.
There are 159 rows with no missing values.
```

In [57]:

```
result.iloc[[0, 1, 200,201, 350,351]]
```

Out[57]:

	outgoing_mins_per_month	outgoing_sms_per_month	monthly_mb	use_id	platform	
0	21.97	4.82	1557.33	22787	android	GT
1	1710.08	136.88	7267.55	22788	android	
200	28.79	29.42	3114.67	23988	NaN	
201	616.56	99.85	5414.14	24006	NaN	
350	NaN	NaN	NaN	23050	ios	iPh
351	NaN	NaN	NaN	23051	ios	iPh

In [58]:

```
# First, add the platform and device to the user usage.
result = pd.merge(user_usage,
                 user_device[['use_id', 'platform', 'device']],
                 on='use_id',
                 how='left')

# Now, based on the "device" column in result, match the "Model" column in devices.
devices.rename(columns={"Retail Branding": "manufacturer"}, inplace=True)
result = pd.merge(result,
                  devices[['manufacturer', 'Model']],
                  left_on='device',
                  right_on='Model',
                  how='left')

result.head()
```

Out[58]:

	outgoing_mins_per_month	outgoing_sms_per_month	monthly_mb	use_id	platform	devic
0	21.97	4.82	1557.33	22787	android	GT I950
1	1710.08	136.88	7267.55	22788	android	SM G930
2	1710.08	136.88	7267.55	22789	android	SM G930
3	94.46	35.17	519.12	22790	android	D230
4	71.59	79.26	1557.33	22792	android	SM G361

In [59]:

```
devices[devices.Model == 'SM-G930F']
```

Out[59]:

	manufacturer	Marketing Name	Device	Model
10381	Samsung	Galaxy S7	herolte	SM-G930F

In [60]:

```
devices[devices.Device.str.startswith('GT')]
```

Out[60]:

	manufacturer	Marketing Name	Device	Model
1095	Bitmore	GTAB700	GTAB700	NID_7010
1096	Bitmore	GTAB900	GTAB900	S952
2402	Grundig	GTB1050	GTB1050	GTB 1050
2403	Grundig	GTB850	GTB850	GTB 850
2404	Grundig	TC69CA2	GTB801	GTB 801
9125	Samsung	NaN	GT-I5510M	GT-I5510M
9126	Samsung	NaN	GT-I5510T	GT-I5510T
9127	Samsung	NaN	GT-I5800L	GT-I5800L
9128	Samsung	NaN	GT-N7000B	GT-N7000B
9129	Samsung	NaN	GT-P7300B	GT-P7300B
9130	Samsung	NaN	GT-P7320T	GT-P7320T
9131	Samsung	NaN	GT-P7500M	GT-P7500M
9132	Samsung	NaN	GT-P7500R	GT-P7500R
9133	Samsung	NaN	GT-P7500V	GT-P7500V
9134	Samsung	NaN	GT-S5698	GT-S5698
9135	Samsung	NaN	GT-S5820	GT-S5820
9136	Samsung	NaN	GT-S5830V	GT-S5830V
9173	Samsung	Absolute	GT-B9120	GT-B9120
9184	Samsung	Europa	GT-I5500B	GT-I5500B
9185	Samsung	Europa	GT-I5500L	GT-I5500L
9186	Samsung	Europa	GT-I5500M	GT-I5500M
9187	Samsung	Europa	GT-I5503T	GT-I5503T
9188	Samsung	Europa	GT-I5510L	GT-I5510L
9191	Samsung	Galaxy (China)	GT-B9062	GT-B9062
9288	Samsung	Galaxy Ace	GT-S5830	GT-S5830
9289	Samsung	Galaxy Ace	GT-S5830B	GT-S5830B
9290	Samsung	Galaxy Ace	GT-S5830C	GT-S5830C
9291	Samsung	Galaxy Ace	GT-S5830D	GT-S5830D
9292	Samsung	Galaxy Ace	GT-S5830F	GT-S5830F
9293	Samsung	Galaxy Ace	GT-S5830G	GT-S5830G
...	...	...	...	...
10480	Samsung	Galaxy Tab 7.7	GT-P6800	GT-P6800
10481	Samsung	Galaxy Tab 7.7	GT-P6810	GT-P6810
10484	Samsung	Galaxy Tab 8.9	GT-P7300	GT-P7300
10485	Samsung	Galaxy Tab 8.9	GT-P7310	GT-P7310
10486	Samsung	Galaxy Tab 8.9	GT-P7320	GT-P7320
10778	Samsung	Galaxy W	GT-I8150	GT-I8150

	manufacturer	Marketing Name	Device	Model
10779	Samsung	Galaxy W	GT-I8150B	GT-I8150B
10780	Samsung	Galaxy W	GT-I8150T	GT-I8150T
10796	Samsung	Galaxy Xcover	GT-S5690	GT-S5690
10797	Samsung	Galaxy Xcover	GT-S5690L	GT-S5690L
10798	Samsung	Galaxy Xcover	GT-S5690M	GT-S5690M
10799	Samsung	Galaxy Xcover	GT-S5690R	GT-S5690R
10804	Samsung	Galaxy Y	GT-S5360	GT-S5360
10805	Samsung	Galaxy Y	GT-S5360B	GT-S5360B
10806	Samsung	Galaxy Y	GT-S5360L	GT-S5360L
10807	Samsung	Galaxy Y	GT-S5360T	GT-S5360T
10808	Samsung	Galaxy Y	GT-S5363	GT-S5363
10809	Samsung	Galaxy Y	GT-S5368	GT-S5368
10810	Samsung	Galaxy Y	GT-S5369	GT-S5369
10813	Samsung	Galaxy Y Duos	GT-S6102	GT-S6102
10814	Samsung	Galaxy Y Duos	GT-S6102B	GT-S6102B
10815	Samsung	Galaxy Y Duos	GT-S6102E	GT-S6102E
10818	Samsung	Galaxy Y Pop	GT-S6108	GT-S6108
10819	Samsung	Galaxy Y Pro	GT-B5510	GT-B5510
10820	Samsung	Galaxy Y Pro	GT-B5510B	GT-B5510B
10821	Samsung	Galaxy Y Pro	GT-B5510L	GT-B5510L
10822	Samsung	Galaxy Y Pro Duos	GT-B5512	GT-B5512
10823	Samsung	Galaxy Y Pro Duos	GT-B5512B	GT-B5512B
10824	Samsung	Galaxy Y TV	GT-S5367	GT-S5367
10979	Sharp	AQUOS SERIE mini SHV38	GTQ	SHV38

164 rows × 4 columns

In [61]:

```
result.head()
```

Out[61]:

	outgoing_mins_per_month	outgoing_sms_per_month	monthly_mb	use_id	platform	devic
0	21.97	4.82	1557.33	22787	android	GT I950
1	1710.08	136.88	7267.55	22788	android	SM G930
2	1710.08	136.88	7267.55	22789	android	SM G930
3	94.46	35.17	519.12	22790	android	D230
4	71.59	79.26	1557.33	22792	android	SM G361

In [62]:

```
result.groupby("manufacturer").agg({
        "outgoing_mins_per_month": "mean",
        "outgoing_sms_per_month": "mean",
        "monthly_mb": "mean",
        "use_id": "count"
    })
```

Out[62]:

	outgoing_mins_per_month	outgoing_sms_per_month	monthly_mb	use_id
manufacturer				
HTC	299.842955	93.059318	5144.077955	44
Huawei	81.526667	9.500000	1561.226667	3
LGE	111.530000	12.760000	1557.330000	2
Lava	60.650000	261.900000	12458.670000	2
Lenovo	215.920000	12.930000	1557.330000	2
Motorola	95.127500	65.666250	3946.500000	16
OnePlus	354.855000	48.330000	6575.410000	6
Samsung	191.010093	92.390463	4017.318889	108
Sony	177.315625	40.176250	3212.000625	16
Vodafone	42.750000	46.830000	5191.120000	1
ZTE	42.750000	46.830000	5191.120000	1

Использование Pandasql для запросов на соединение и группировку данных

In [63]:

```
import pandasql as ps
```

In [110]:

```
# pandasql code
def left_join_ps(user_usage, user_device):
    join_query = '''
    SELECT user_usage.outgoing_mins_per_month,
    user_usage.outgoing_sms_per_month,
    user_usage.monthly_mb,
    user_device.use_id,
    user_device.device,
    user_device.platform

    FROM user_usage
    LEFT JOIN user_device
    ON user_usage.use_id = user_device.use_id;
    '''
    return ps.sqldf(join_query, locals())
left_join_ps(user_usage, user_device).tail()
```

Out[110]:

	outgoing_mins_per_month	outgoing_sms_per_month	monthly_mb	use_id	device	platf
235	260.66	68.44	896.96	NaN	None	N
236	97.12	36.50	2815.00	NaN	None	N
237	355.93	12.37	6828.09	NaN	None	N
238	632.06	120.46	1453.16	NaN	None	N
239	488.70	906.92	3089.85	NaN	None	N

Использование функций агрегирования

In [147]:

```
# pandasql code
def aggregating_ps(user_usage, user_device):
    aggr_query = '''
        SELECT
            avg(outgoing_mins_per_month) as outgoing_mins_per_month,
            user_device.platform
        FROM user_usage
        LEFT JOIN user_device
        ON user_usage.use_id = user_device.use_id
        GROUP BY platform
        '''
    return ps.sqldf(aggr_query, locals())
```

In [148]:

```
aggregating_ps(user_usage, user_device)
```

Out[148]:

	outgoing_mins_per_month	platform
0	414.376420	None
1	201.258535	android
2	366.060000	ios

In [144]:

```
merged = pd.merge(user_usage,
                  user_device[['use_id', 'platform']],
                  on='use_id',
                  how='left')
merged['platform'] = merged['platform'].astype(str)
merged.groupby('platform').mean()
```

Out[144]:

	outgoing_mins_per_month	outgoing_sms_per_month	monthly_mb	use_id
platform				
android	201.258535	85.354586	4221.387834	22922.350318
ios	366.060000	293.975000	961.155000	22920.500000
nan	414.376420	120.540370	2545.485062	23998.444444

Сравнение времени выполнения каждого запроса в Pandas и PandaSQL.

In [83]:

```
import time

def count_mean_time(func, params, N =5):
    total_time = 0
    for i in range(N):
        time1 = time.time()
        if len(params) == 1:
            tmp_df = func(params[0])
        elif len(params) == 2:
            tmp_df = func(params[0], params[1])
        time2 = time.time()
        total_time += (time2 - time1)
    return total_time/N
```

In [113]:

```
left_join_pandasql_time = count_mean_time(left_join_ps, [user_usage, user_device], N=20
)
left_join_pandasql_time
```

Out[113]:

0.019597113132476807

In [114]:

```
def left_join_pandas(user_usage, user_device):
    result = pd.merge(user_usage,
                      user_device[['use_id', 'platform', 'device']],
                      on='use_id', how='left')
    return result
left_join_pandas_time = count_mean_time(left_join_pandas, [user_usage, user_device], N=
20)
left_join_pandas_time
```

Out[114]:

0.0056846380233764645

In [115]:

```
aggreg_pandasql_time = count_mean_time(aggregating_ps, [user_usage, user_device], N=20)
aggreg_pandasql_time
```

Out[115]:

0.017153775691986083

In [116]:

```
def aggreg_query_pandas(user_usage, user_device):
    merged = pd.merge(user_usage,
                      user_device[['use_id', 'platform']],
                      on='use_id',
                      how='left')
    return merged.groupby('platform').mean()

aggr_pandas_time = count_mean_time(aggreg_query_pandas, [user_usage, user_device], N=20
)
aggr_pandas_time
```

Out[116]:

0.00812966823577881

In [117]:

```
print('Разница во времени при выполнении запроса LEFT JOIN: {0}'.format(left_join_panda
sql_time - left_join_pandas_time))
print('Разница во времени при выполнении запроса с ф-ями агрегирования: {0}'.format(agg
reg_pandasql_time - aggr_pandas_time))
```

```
Разница во времени при выполнении запроса LEFT JOIN: 0.013912475109100342
Разница во времени при выполнении запроса с ф-ями агрегирования: 0.0090241
07456207273
```

Сравнив время выполнения запросов при помощи библиотек Pandas и Pandasql, можно сделать вывод, что для простых запросов лучше использовать Pandas, так как время выполнения запросов к источнику данных меньше, чем у Pandasql. Но при написании сложных запросов Pandasql даст возможность использовать привычные SQL запросы.

Список литературы

1. Гапанюк Ю. Е. Лабораторная работа «Изучение библиотек обработки данных» [Электронный ресурс] // GitHub. – 2019. – Режим доступа: https://github.com/ (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/ (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/ (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 (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/ (https://ipython.readthedocs.io/en/) stable/ (online; accessed: 20.02.2019).