Московский государственный технический университет имени Н.Э.Баумана

Кафедра «Системы обработки информации и управления»

О Т Ч Е Т

Лабораторная работа №2

по курсу «Методы машинного обучения»

« Изучение библиотек обработки данных.»

Исполнитель: Соболева Е.Д.

группа ИУ5-11М

Проверил: Гапанюк Ю.Е.

Москва, 2020

In [5]:

```
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 [6]:

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

Out[6]:

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

In [4]:

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

Out[4]:

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

In [5]:

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

In [6]:

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

Out[6]:

```
36.85823043357163
```

In [7]:

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

Out[7]:

```
36.85823043357163
```

In [8]:

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

Out[8]:

```
0.004207487485028101
```

In [9]:

```
#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 rece
ive less 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 +- 10.5 years, poor - 37 +- 14.0 yea
rs.
```

In [10]:

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

Out[10]:

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

In [11]:

```
#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 [12]:

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

Out[12]:

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

In [13]:

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

Out[13]:

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

In [14]:

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

Out[14]:

```
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 [15]:

```
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 [16]:

```
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 [17]:

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

Out[17]:

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

PART 2

In [7]:

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

In [24]:

```
android_devices.head()
```

Out[24]:

	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

In [25]:

```
user_usage.head()
```

Out[25]:

	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 [26]:

```
user_device.head()
```

Out[26]:

	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
3	22785	15200	ios	10.2	iPhone7,2	3
4	22786	28239	android	6.0	ONE E1003	1

In [27]:

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

In [28]:

```
merged.head()
```

Out[28]:

	outgoing_mins_per_month	outgoing_sms_per_month	monthly_mb	use_id	device
0	21.97	4.82	1557.33	22787	GT-I9505
1	1710.08	136.88	7267.55	22788	SM-G930F
2	1710.08	136.88	7267.55	22789	SM-G930F
3	94.46	35.17	519.12	22790	D2303
4	71.59	79.26	1557.33	22792	SM-G361F

In [29]:

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

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

In [30]:

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

Out[30]:

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

In [34]:

```
merged = pd.merge(user_usage,
                  user_device[['use_id', 'device', 'platform']],
                  on='use_id',
                   how='left')
print("user_usage dimensions: {}".format(user_usage.shape))
print("merged dimensions: {}".format(merged.shape))
print("Missing values: {}".format(merged['device'].isnull().sum()))
```

```
user_usage dimensions: (240, 4)
merged dimensions: (240, 6)
Missing values: 81
```

In [33]:

```
merged.tail()
```

Out[33]:

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

In [35]:

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

```
user_device dimensions: (272, 6)
merged dimensions: (272, 6)
Missing values in monthly_mb: 113
Missing values in platform: 0
```

In [37]:

```
merged = pd.merge(user_usage,
                  user_device[['use_id', 'device', 'platform']],
                  on='use_id',
                  how='outer',
                  indicator=True)

print("Rows in outer merge: {}".format(merged.shape))

print("No missing values: {}".format(
    (merged.apply(lambda x: x.isnull().sum(), axis=1) == 0).sum()))
```

```
Rows in outer merge: (353, 7)
No missing values: 159
```

In [44]:

```
# First, add the platform and device to the user usage.
merged = 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 devic
es.
android_devices.rename(columns={"Retail Branding": "manufacturer"},
                         inplace=True)
merged = pd.merge(result,
                   android_devices[['manufacturer', 'Model']],
                   left_on='device',
                   right_on='Model',
                   how='left')

merged.head()
```

Out[44]:

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

In [46]:

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

Out[46]:

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

In [47]:

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

Out[47]:

	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
...	...	...	...	...
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 [41]:

```
merged.head()
```

Out[41]:

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

In [48]:

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

Out[48]:

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

Pandasql

In [ ]:

```
import pandasql as ps
```

In [22]:

```
# pandasql code
def lj_pandasql(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())
lj_pandasql(user_usage, user_device).tail()
```

Out[22]:

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

Агрегирование

pandasql

In [112]:

```
from datetime import datetime
def aggr_pandasql(user_usage, user_device):
    aggreg_query = '''
       SELECT avg(user_usage.monthly_mb) as avg_monthly_mb,
               user_device.platform
        FROM user_usage
        LEFT JOIN user_device
        ON user_usage.use_id = user_device.use_id
        GROUP BY user_device.platform;
    '''
    return ps.sqldf(aggreg_query, locals())
aggr_pandasql(user_usage, user_device)
```

Out[112]:

	avg_monthly_mb	platform
0	2545.485062	None
1	4221.387834	android
2	961.155000	ios

In [51]:

```
```

Out[51]:

```
datetime.timedelta(microseconds=28319)
```

pandas

In [41]:

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

Out[41]:

	outgoing_mins_per_month	outgoing_sms_per_month	monthly_mb	use
platform				
android	201.258535	85.354586	4221.387834	22922.350
ios	366.060000	293.975000	961.155000	22920.500

TIME

In [125]:

```
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

lj_ps_mean = count_mean_time(lj_pandasql,
                             [user_usage, user_device], N=40)
lj_ps_mean
```

Out[125]:

0.014299607276916504

In [126]:

```
def pd_merge_group(user_usage, user_device):
    merged = pd.merge(user_usage,
                 user_device[['use_id', 'device', 'platform']],
                 on='use_id',
                  how='left')
    res = merged.groupby('platform').mean()
    return res
pd_merge_group_mean = count_mean_time(pd_merge_group,
                             [user_usage, user_device], N=40)
pd_merge_group_mean
```

Out[126]:

0.007319676876068115

In [127]:

```
def pd_merge(user_usage, user_device):
    merged = pd.merge(user_usage,
                 user_device[['use_id', 'device', 'platform']],
                 on='use_id',
                  how='left')
    return merged
pd_merge_mean = count_mean_time(pd_merge,
                                [user_usage, user_device], N=40)
pd_merge_mean
```

Out[127]:

0.004346024990081787

In [128]:

```
aggr_ps_mean = count_mean_time(aggr_pandasql,
                               [user_usage, user_device], N=40)
aggr_ps_mean
```

Out[128]:

```
0.01447572112083435
```

In [129]:

```
merge_delta = lj_ps_mean - pd_merge_mean
merge_delta
```

Out[129]:

```
0.009953582286834718
```

In [130]:

```
aggr_delta = aggr_ps_mean - pd_merge_group_mean
aggr_delta
```

Out[130]:

```
0.0071560442447662345
```

Вывод: pandasql дольше работает