МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Систем обработки информации и управления»

Лабораторная работа №2

по дисциплине

«Методы машинного обучения»

ИСПОЛНИТЕЛЬ: Сукач Е.А.
ФИО

группа ИУ5-23М

____
подпись

"__"_________2020 г.

ПРЕПОДАВАТЕЛЬ: Гапанюк Ю. Е.
ФИО

____
подпись

"__"_________2020 г.

Москва – 2020

```
Первая часть
```

Unique values of features (for more information please see the link 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

In [6]:

```
import numpy as np
import pandas as pd
```

In [7]:
```
data = pd.read_csv('/Users/elizavetasukach/Desktop/MachineLearning/
data.head()
```

Out[7]:

	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

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

In [8]:
```
data['sex'].value_counts()
```

Out[8]:
```
Male      21790
Female    10771
Name: sex, dtype: int64
```

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

In [25]:
```
#data.loc[data['sex'] == 'Female', 'age'].mean()

data[data['sex']=='Female']['age'].mean()
```

Out[25]:
```
36.85823043357163
```

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

In [32]:
```

a = float((data['native-country'] == 'Germany').sum()) / data.shape
print("The proportion of German citizens =", "%.2f" % a, "%")

```

```
The proportion of German citizens = 0.42 %
```

**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 less than 50K per year? **

In [35]:

```
ages1 = data[data['salary']=='>50K']['age']
#ages1 = data.loc[data['salary'] == '>50K', 'age']
#ages2 = data.loc[data['salary'] == '<=50K', 'age']
ages2 = data[data['salary']=='<=50K']['age']
print("The average age of the rich: {0} +- {1} years, poor - {2} +-
    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 y
ears.
```

6. Is it true that people who receive more than 50k have at least high school education? (*education - Bachelors, Prof-school, Assoc-acdm, Assoc-voc, Masters* or *Doctorate* feature)

In [48]:

```
mas = data[data['salary']=='>50K']['education'].unique()
element = ['Some-college','11th','HS-grad','9th','7th-8th','12th','
flag = 0
for i in element:
    if i in mas:
        flag = 1
if (flag==1):
     print("NO")
else:
     print("YES")
```

```
NO
```

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.

In [19]:

```
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
```

8. 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 [87]:

```
data1 = data.loc[(data['sex'] == 'Male') &
     (data['marital-status'].isin(['Never-married',
                                    'Separated',
                                    'Divorced',
                                    'Widowed'])), 'salary'].value_co

proc1 = data1['>50K'] / data.shape[0] *100
data2 = data.loc[(data['sex'] == 'Male') &
     (data['marital-status'].str.startswith('Married')), 'salary'].

proc2 = data2['>50K'] / data.shape[0] *100

print("Процент женатых людей, зарабатывающих больше 50K = ",round(p
print("Процент холостых людей, зарабатывающих больше 50K = ",round(
```

```
Процент женатых людей, зарабатывающих больше 50K =  18.3 %
Процент холостых людей, зарабатывающих больше 50K =  2.1 %
```

In [22]:

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

Out[22]:

```
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
```

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 among them?

In [88]:

```
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_workaholic

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

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

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

Simple method:

```
In [24]: for (country, salary), sub_df in data.groupby(['native-country', 's
             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 method:

In [25]:

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

Out[25]:

native-country	?	Cambodia	Canada	China	Columbia	Cuba	Dominican-Republic	E
salary								
<=50K	40.164760	41.416667	37.914634	37.381818	38.684211	37.985714	42.338235	38
>50K	45.547945	40.000000	45.641026	38.900000	50.000000	42.440000	47.000000	48

2 rows × 42 columns

In [ ]:

Вторая часть

In [4]:
```
pip install pandasql
```

In [7]:
```
# Import
import pandas as pd
import pandasql as ps
from datetime import datetime
import seaborn
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'svg'
from pylab import rcParams
rcParams['figure.figsize'] = 8, 5
```

In [14]:
```
user_usage = pd.read_csv("/Users/elizavetasukach/Desktop/MachineLea
user_device = pd.read_csv("/Users/elizavetasukach/Desktop/MachineLe
devices = pd.read_csv("/Users/elizavetasukach/Desktop/MachineLearni
devices.rename(columns={"Retail Branding": "manufacturer"}, inplace
```

In [4]:
```
user_usage.head()
```

Out[4]:

	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 [5]:
```
user_device.head()
```

Out[5]:

	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 [6]:
```
devices.head(10)
```

Out[6]:

	manufacturer	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 [7]:
```
result = pd.merge(user_usage,
                  user_device[['use_id', 'platform', 'device']],
                  on='use_id')
result.head()
```

Out[7]:

	outgoing_mins_per_month	outgoing_sms_per_month	monthly_mb	use_id	platform	devi
0	21.97	4.82	1557.33	22787	android	G I95
1	1710.08	136.88	7267.55	22788	android	SI G93
2	1710.08	136.88	7267.55	22789	android	SI G93
3	94.46	35.17	519.12	22790	android	D23
4	71.59	79.26	1557.33	22792	android	SI G36

Выдать id пользователя и девайс, для которого outgoing_mins_per_month >1000

In [20]:
```
def example1_pandas(user_usage,user_device):
    return pd.merge(user_usage[user_usage.outgoing_mins_per_month >
                  user_device[['use_id', 'device']],
                  on='use_id') [['use_id','device','outgoing_mins_pe
```

In [21]:

```
example1_pandas(user_usage,user_device)
```

Out[21]:

	use_id	device	outgoing_mins_per_month
0	22788	SM-G930F	1710.08
1	22789	SM-G930F	1710.08
2	22858	ONEPLUS A3003	1221.85

Выдать группировку по полю monthly_mb

In [23]:

```
def example2_pandas(user_usage):
    return user_usage.groupby(['monthly_mb']).size()
```

In [24]:

```
example2_pandas(user_usage)
```

Out[24]:

```
monthly_mb
0.00        1
11.68       1
33.79       1
74.40       1
212.64      1
           ..
15573.33    9
16611.55    1
20764.45    2
25955.55    1
31146.67    1
Length: 83, dtype: int64
```

In [26]:

```
user_usage.shape
```

Out[26]:

```
(240, 4)
```

In [39]:

```
def example1_pandasql(user_usage,user_device):
    simple_query = '''
        SELECT
        user_usage.use_id,
        user_device.device,
        user_usage.outgoing_mins_per_month
        FROM user_usage JOIN user_device ON user_usage.use_id = use
        WHERE user_usage.outgoing_mins_per_month >1000
        '''
    return ps.sqldf(simple_query, locals())
```

In [40]:

```
example1_pandasql(user_usage,user_device)
```

Out[40]:

	use_id	device	outgoing_mins_per_month
0	22788	SM-G930F	1710.08
1	22789	SM-G930F	1710.08
2	22858	ONEPLUS A3003	1221.85

In [25]:

```
def example2_pandasql(user_usage):
    simple_query = '''
        SELECT
        monthly_mb,
        count(*)
        FROM user_usage
        GROUP BY monthly_mb
        '''
    return ps.sqldf(simple_query, locals())


  #  user_usage.groupby(['monthly_mb']).size()
```

In [26]:

```
example2_pandasql(user_usage)
```

Out[26]:

	monthly_mb	count(*)
0	0.00	1
1	11.68	1
2	33.79	1
3	74.40	1
4	212.64	1
...	...	...
78	15573.33	9
79	16611.55	1
80	20764.45	2
81	25955.55	1
82	31146.67	1

83 rows × 2 columns

In [41]:

```
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 [42]:

```
ex1_times = []
for count in range(1000, 137000, 1000):
    pandasql_time = count_mean_time(example1_pandasql, [user_usage,
    pandas_time = count_mean_time(example1_pandas, [user_usage,user
    ex1_times.append({'count': count, 'pandasql_time': pandasql_tim
```

In [43]:

```
ex1_times_df = pd.DataFrame(ex1_times)
ex1_times_df.columns = ['number of rows in daily_engagements', 'pan
ex1_times_df = ex1_times_df.set_index('number of rows in daily_enga
```

In [44]:

```
ax = ex1_times_df.plot(title = 'Example #1 time elapsed (seconds)',
```

```
<Figure size 576x360 with 2 Axes>
```

In [47]:

```
ex2_times = []
for count in range(1000, 137000, 1000):
    pandasql_time = count_mean_time(example2_pandasql, [user_usage]
    pandas_time = count_mean_time(example2_pandas, [user_usage])
    ex2_times.append({'count': count, 'pandasql_time': pandasql_tim
```

In [49]:

```
ex2_times_df = pd.DataFrame(ex2_times)
ex2_times_df.columns = ['number of rows in daily_engagements', 'pan
ex2_times_df = ex2_times_df.set_index('number of rows in daily_enga
```

In [50]:

```
ax = ex2_times_df.plot(title = 'Example #2 time elapsed (seconds)',
```

```
<Figure size 576x360 with 2 Axes>
```

In [ ]:

```
```