МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Систем обработки информации и управления»

# ОТЧЕТ

**Лабораторная работа №2**
по курсу «Методы машинного обучения»

Тема: «Изучение библиотек обработки данных»

ИСПОЛНИТЕЛЬ: Ромичева Е.В.

группа ИУ5-22М

____________________
подпись

"__"_________2019 г.

ПРЕПОДАВАТЕЛЬ: Гапанюк Ю.Е.

____________________
подпись

"__"_________2019 г.

Москва - 2019

## Цель работы

**Цель лабораторной работы:** изучение различных методов визуализация данных.

## Задание

**Часть 1.**

Выполните первое демонстрационное задание "demo assignment" под названием "Exploratory data analysis with Pandas" со страницы курса https://mlcourse.ai/assignments

Условие задания - https://nbviewer.jupyter.org/github/Yorko/mlcourse_open/blob/master/jupyter_english/assignments_demo/assignment01_pandas_uci_adult.ipynb?flush_cache=true

Набор данных можно скачать здесь - https://archive.ics.uci.edu/ml/datasets/Adult

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

**Часть 2.**

Выполните 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.

### *Часть 1.*

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

```
In [88]: # 1
         data['sex'].value_counts()

Out[88]: Male      21790
         Female    10771
         Name: sex, dtype: int64
```

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

```
In [4]: # 2
        # data.loc[data['sex'] == 'Female', 'age'].mean()
        # data.groupby(['sex'])['age'].mean()
        data['age'][data['sex'] == 'Female'].mean()

Out[4]: 36.85823043357163
```

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

```
In [5]: # 3
        # float((data['native-country'] == 'Germany').sum()) / data.shape[0]
        # data['native-country'].eq('Germany').value_counts(normalize = True)
        100.*data['native-country'].eq('Germany').sum()/data.shape[0]

Out[5]: 0.42074874850281013
```

**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 [6]:
```
# 4-5
a = data.groupby(['salary']).agg({'age': 'mean'}).reset_index()
b = data.groupby(['salary']).agg({'age': 'std'}).reset_index()
print("The average age of the rich: {0} +- {1} years, poor - {2} +- {3} years.".format(
    round(a['age'][0]), round(b['age'][0],1),
    round(a['age'][1]), round(b['age'][1],1)))
```

```
The average age of the rich: 37.0 +- 14.0 years, poor - 44.0 +- 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)**

In [7]:
```
deg = {'Bachelors', 'Prof-school', 'Assoc-acdm', 'Assoc-voc', 'Masters', 'Doctorate' }
sal_big = data.loc[data['salary'] == '>50K']
good_edu = sal_big.loc[
    lambda x : x['education'].isin(deg)
]
good_edu['age'].size/sal_big['age'].size
# посчитали отношение образованных людей с хорошей зп и людей с хорошей зп
```

Out[7]: `0.5783701058538452`

**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.**

In [8]:
```
# 7
# for (race, sex), sub_df in data.groupby(['race', 'sex']):
#     print("Race: {0}, sex: {1}".format(race, sex))
#     print(sub_df['age'].describe())
sub_df = data.groupby(['race', 'sex'])
sub_df['age'].describe()
```

Out[8]:

| race | sex | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|---|
| Amer-Indian-Eskimo | Female | 119.0 | 37.117647 | 13.114991 | 17.0 | 27.0 | 36.0 | 46.00 | 80.0 |
| | Male | 192.0 | 37.208333 | 12.049563 | 17.0 | 28.0 | 35.0 | 45.00 | 82.0 |
| Asian-Pac-Islander | Female | 346.0 | 35.089595 | 12.300845 | 17.0 | 25.0 | 33.0 | 43.75 | 75.0 |
| | Male | 693.0 | 39.073593 | 12.883944 | 18.0 | 29.0 | 37.0 | 46.00 | 90.0 |
| Black | Female | 1555.0 | 37.854019 | 12.637197 | 17.0 | 28.0 | 37.0 | 46.00 | 90.0 |
| | Male | 1569.0 | 37.682600 | 12.882612 | 17.0 | 27.0 | 36.0 | 46.00 | 90.0 |
| Other | Female | 109.0 | 31.678899 | 11.631599 | 17.0 | 23.0 | 29.0 | 39.00 | 74.0 |
| | Male | 162.0 | 34.654321 | 11.355531 | 17.0 | 26.0 | 32.0 | 42.00 | 77.0 |
| White | Female | 8642.0 | 36.811618 | 14.329093 | 17.0 | 25.0 | 35.0 | 46.00 | 90.0 |
| | Male | 19174.0 | 39.652498 | 13.436029 | 17.0 | 29.0 | 38.0 | 49.00 | 90.0 |

**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.**

```
In [11]: # 8
mar = {'Married-civ-spouse', 'Married-spouse-absent ', 'Married-AF-spouse'}
men = data.loc[data['sex'] == 'Male']
sal_big = men.loc[data['salary'] == '>50K']
married_salary = sal_big.loc[
    lambda x : x['marital-status'].isin(mar)
]
print ('Отношение богатых замужних мужчин к богатым незамужним = ', married_salary['age'].size/sal_big['age'].size)
sal_big['marital-status'].value_counts()

Отношение богатых замужних мужчин к богатым незамужним =  0.89192434704293

Out[11]: Married-civ-spouse       5938
Never-married             325
Divorced                  284
Separated                  49
Widowed                    39
Married-spouse-absent      23
Married-AF-spouse           4
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 (>50K) among them?**

```
In [14]: # 9
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_workaholics = 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_workaholics)))

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*) for those who earn a little and a lot (*salary*) for each country (*native-country*). What will these be for Japan?**

In [13]:
```
# 10
zz = data.groupby(['native-country', 'salary']).agg({'hours-per-week': 'mean'})
zz
```

Out[13]:

| native-country | salary | hours-per-week |
|---|---|---|
| ? | <=50K | 40.164760 |
| | >50K | 45.547945 |
| Cambodia | <=50K | 41.416667 |
| | >50K | 40.000000 |
| Canada | <=50K | 37.914634 |
| | >50K | 45.641026 |
| China | <=50K | 37.381818 |
| | >50K | 38.900000 |
| Columbia | <=50K | 38.684211 |
| | >50K | 50.000000 |
| Cuba | <=50K | 37.985714 |
| | >50K | 42.440000 |
| Dominican-Republic | <=50K | 42.338235 |
| | >50K | 47.000000 |
| Ecuador | <=50K | 38.041667 |
| | >50K | 48.750000 |
| El-Salvador | <=50K | 36.030928 |
| | >50K | 45.000000 |
| England | <=50K | 40.483333 |
| | >50K | 44.533333 |
| France | <=50K | 41.058824 |
| | >50K | 50.750000 |

## *Часть 2.*

### Соединение таблиц с помощью Pandas

In [9]:
```
def example1_pandas(project_submissions, enrollments):
    merg_res = pd.merge(project_submissions, enrollments, on= 'account_key')
    return merg_res

example1_pandas(project_submissions, enrollments)
```

Out[9]:

| | creation_date | completion_date | assigned_rating | account_key | lesson_key | processing_state | status | join_date | cancel_date | days_to_cancel | is_uda |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2015-01-14 | 2015-01-16 | UNGRADED | 256 | 3176718735 | EVALUATED | canceled | 2014-12-03 | 2015-04-01 | 119.0 | Fa |
| 1 | 2015-01-14 | 2015-01-16 | UNGRADED | 256 | 3176718735 | EVALUATED | canceled | 2015-04-01 | 2015-06-10 | 70.0 | Fa |
| 2 | 2015-01-10 | 2015-01-13 | INCOMPLETE | 256 | 3176718735 | EVALUATED | canceled | 2014-12-03 | 2015-04-01 | 119.0 | Fa |
| 3 | 2015-01-10 | 2015-01-13 | INCOMPLETE | 256 | 3176718735 | EVALUATED | canceled | 2015-04-01 | 2015-06-10 | 70.0 | Fa |
| 4 | 2015-01-20 | 2015-01-20 | PASSED | 256 | 3176718735 | EVALUATED | canceled | 2014-12-03 | 2015-04-01 | 119.0 | Fa |
| 5 | 2015-01-20 | 2015-01-20 | PASSED | 256 | 3176718735 | EVALUATED | canceled | 2015-04-01 | 2015-06-10 | 70.0 | Fa |
| 6 | 2015-03-10 | 2015-03-13 | PASSED | 434 | 3176718735 | EVALUATED | canceled | 2015-01-12 | 2015-06-03 | 142.0 | Fa |

### Соединение таблиц с помощью PandaSQL

```
In [12]: pysql = lambda q: ps.sqldf(q, globals())
         def example1_pandasql(project_submissions, enrollments):
         #     query = """SELECT *
         #     FROM project_submissions p JOIN enrollments e ON p.account_key = e.account_key
         #     GROUP BY p.account_key;"""
             query = "select * from project_submissions, enrollments where project_submissions.account_key = enrollments.account_key;"
             join_res = pysql(query)
             return join_res

         example1_pandasql(project_submissions, enrollments)
```

Out[12]:

| | creation_date | completion_date | assigned_rating | account_key | lesson_key | processing_state | account_key | status | join_date | cancel_date | days_to_c |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2015-01-14 | 2015-01-16 | UNGRADED | 256 | 3176718735 | EVALUATED | 256 | canceled | 2014-12-03 | 2015-04-01 | |
| 1 | 2015-01-14 | 2015-01-16 | UNGRADED | 256 | 3176718735 | EVALUATED | 256 | canceled | 2015-04-01 | 2015-06-10 | |
| 2 | 2015-01-10 | 2015-01-13 | INCOMPLETE | 256 | 3176718735 | EVALUATED | 256 | canceled | 2014-12-03 | 2015-04-01 | |
| 3 | 2015-01-10 | 2015-01-13 | INCOMPLETE | 256 | 3176718735 | EVALUATED | 256 | canceled | 2015-04-01 | 2015-06-10 | |
| 4 | 2015-01-20 | 2015-01-20 | PASSED | 256 | 3176718735 | EVALUATED | 256 | canceled | 2014-12-03 | 2015-04-01 | |
| 5 | 2015-01-20 | 2015-01-20 | PASSED | 256 | 3176718735 | EVALUATED | 256 | canceled | 2015-04-01 | 2015-06-10 | |
| 6 | 2015-03-10 | 2015-03-13 | PASSED | 434 | 3176718735 | EVALUATED | 434 | canceled | 2015-01-12 | 2015-06-03 | |

Сравнение времени выполнения запросов

```
In [10]: class Profiler(object):
             def __enter__(self):
                 self._startTime = time.time()

             def __exit__(self, type, value, traceback):
                 print ("Elapsed time: {:.3f} sec".format(time.time() - self._startTime))

         with Profiler() as p:
             example1_pandas(project_submissions, enrollments)

         Elapsed time: 0.009 sec
```

```
In [11]: with Profiler() as p:
             example1_pandasql(project_submissions, enrollments)

         Elapsed time: 0.105 sec
```

Видно, что соединение таблиц с помощью Pandas выполняется в 12 раз быстрее, чем соединение с помощью PandaSQL.

Агрегирование таблиц с помощью PandaSQL

```
In [12]: daily_engagements['weekday'] = list(map(lambda x: datetime.strptime(x, '%Y-%m-%d').strftime('%A'),
                                                 daily_engagements.utc_date))

         # pandasql code
         def example2_pandasql(daily_engagements):
             aggr_query = '''
                 SELECT
                     avg(total_minutes_visited) as total_minutes_visited,
                     weekday
                 FROM daily_engagements
                 GROUP BY weekday
                 '''
             return ps.sqldf(aggr_query, locals()).set_index('weekday')

         # pandas code
         def example2_pandas(daily_engagements):
             return pd.DataFrame(daily_engagements.groupby('weekday').total_minutes_visited.mean())

         weekday_engagement = example2_pandasql(daily_engagements)
         weekday_engagement
```

Out[12]:

| weekday | total_minutes_visited |
|---|---|
| Friday | 23.156233 |
| Monday | 26.418982 |
| Saturday | 21.725677 |
| Sunday | 23.539406 |
| Thursday | 24.685176 |
| Tuesday | 26.857676 |
| Wednesday | 25.362789 |

Агрегирование таблиц с помощью Pandas

In [13]:
```
example2_pandas(daily_engagements)
```

Out[13]:

| weekday | total_minutes_visited |
|---|---|
| Friday | 23.156233 |
| Monday | 26.418982 |
| Saturday | 21.725677 |
| Sunday | 23.539406 |
| Thursday | 24.685176 |
| Tuesday | 26.857676 |
| Wednesday | 25.362789 |

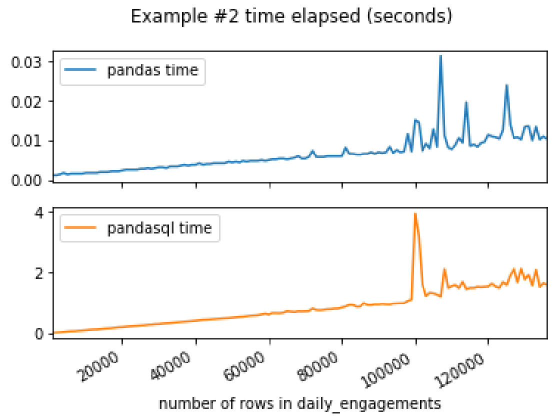
Сравнение времени выполнения запросов

In [5]:
```
ex2_times = []
for count in range(1000, 137000, 1000):
    pandasql_time = count_mean_time(example2_pandasql, [daily_engagements[:count]])
    pandas_time = count_mean_time(example2_pandas, [daily_engagements[:count]])
    ex2_times.append({'count': count, 'pandasql_time': pandasql_time, 'pandas_time': pandas_time})
```

In [6]:
```
ex2_times_df = pd.DataFrame(ex2_times)
```

In [7]:
```
ex2_times_df.columns = ['number of rows in daily_engagements', 'pandas time', 'pandasql time']
ex2_times_df = ex2_times_df.set_index('number of rows in daily_engagements')
```

In [8]:
```
ax = ex2_times_df.plot(title = 'Example #2 time elapsed (seconds)', subplots = True)
```




Видно, что агрегирование таблиц с помощью Pandas выполняется быстрее, чем агрегирование с помощью PandaSQL.