МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Н.Э. Баумана

Кафедра «Систем обработки информации и управления»

ОТЧЕТ

Лабораторная работа №2

по курсу «Машинное обучение»

Тема: «Изучение библиотек обработки данных»

ИСПОЛНИТЕЛЬ: Григорьев Е.А

ФИО

группа ИУ5-21М

____

подпись

"__"_________201_ г.

Москва - 2019

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.

В качестве примеров можно использовать следующие статьи:

- https://www.shanelynn.ie/summarising-aggregation-and-grouping-data-in-python-pandas/
- https://www.shanelynn.ie/merge-join-dataframes-python-pandas-index-1/ (в разделе "Example data" данной статьи содержится рекомендуемый набор данных для проведения экспериментов).

Пример сравнения Pandas и PandaSQL - https://github.com/miptgirl/udacity_engagement_analysis/blob/master/pandasql_example.ipynb

Набор упражнений по Pandas с решениями - https://github.com/guipsamora/pandas_exercises

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.

В качестве примеров можно использовать следующие статьи:

- https://www.shanelynn.ie/summarising-aggregation-and-grouping-data-in-python-pandas/
- https://www.shanelynn.ie/merge-join-dataframes-python-pandas-index-1/ (в разделе "Example data" данной статьи содержится рекомендуемый набор данных для проведения экспериментов).

Пример сравнения Pandas и PandaSQL - https://github.com/miptgirl/udacity_engagement_analysis/blob/master/pandasql_example.ipynb

Набор упражнений по Pandas с решениями - https://github.com/guipsamora/pandas_exercises

```python
import numpy as np
import pandas as pd
import pandasql as ps
import seaborn as sns
import warnings
import time

%matplotlib inline
import matplotlib.pyplot as plt


warnings.filterwarnings('ignore')
pd.set_option('display.max.columns', 100)
```

```python
data = pd.read_csv('./sample_data/adult.txt')
data.head()
```

	age	workClass	fnlwgt	education	educationNum	maritalStatus	occupation	relationship	race	sex	capitalGain	capitalLoss	hoursPerWeek	nativeCountry	salary
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

```python
#all dataframe columns
data.columns
```

```
Index(['age', 'workClass', 'fnlwgt', 'education', 'educationNum',
       'maritalStatus', 'occupation', 'relationship', 'race', 'sex',
       'capitalGain', 'capitalLoss', 'hoursPerWeek', 'nativeCountry',
       'salary'],
      dtype='object')
```

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

```
Male      21789
Female    10771
Name: sex, dtype: int64
```

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

```
36.85682451253482
```

```python
#3. What is the percentage of German citizens (native-country feature)?
germans = data.loc[data.nativeCountry == 'Germany']
germanPercentage = float(germans.age.sum() / data.shape[0]) * 100
germanPercentage
```

```
16.517199017199015
```


```python
import numpy as np
import pandas as pd
import pandasql as ps
import seaborn as sns
import warnings
import time

%matplotlib inline
import matplotlib.pyplot as plt


warnings.filterwarnings('ignore')
pd.set_option('display.max.columns', 100)
```

```python
data = pd.read_csv('./sample_data/adult.txt')
data.head()
```

	age	workClass	fnlwgt	education	educationNum	maritalStatus	occupation	relationship	race	sex	capitalGain	capitalLoss	hoursPerWeek	nativeCountry	salary
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

```python
#all dataframe columns
data.columns
```

```
Index(['age', 'workClass', 'fnlwgt', 'education', 'educationNum',
       'maritalStatus', 'occupation', 'relationship', 'race', 'sex',
       'capitalGain', 'capitalLoss', 'hoursPerWeek', 'nativeCountry',
       'salary'],
      dtype='object')
```

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

```
Male      21789
Female    10771
Name: sex, dtype: int64
```

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

```
36.85682451253482
```

```python
#3. What is the percentage of German citizens (native-country feature)?
germans = data.loc[data.nativeCountry == 'Germany']
germanPercentage = float(germans.age.sum() / data.shape[0]) * 100
germanPercentage
```

```
16.517199017199015
```

[ ]
```
#4. What are the mean and standard deviation of age for those who earn more than 50K per year (salary feature)?
richPeoples = data.loc[data.salary == '>50K']
richAges = richPeoples.age
richAges.std()
```

```
10.519868523717
```

[ ]
```
#5. What are the mean and standard deviation of age for those less than 50K per year (salary feature)?
poorPeoples = data.loc[data.salary == '<=50K']
poorAges = poorPeoples.age
poorAges.std()
```

```
14.020161692826626
```

[ ]
```
#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)
richPeoples.education.unique()
```

```
array([], dtype=object)
```

[ ]
```
#7. Display age statistics for each race (race feature) and each gender (sex feature). Use groupby() and 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
```


[ ]
```
#4. What are the mean and standard deviation of age for those who earn more than 50K per year (salary feature)?
richPeoples = data.loc[data.salary == '>50K']
richAges = richPeoples.age
richAges.std()
```

```
10.519868523717
```

[ ]
```
#5. What are the mean and standard deviation of age for those less than 50K per year (salary feature)?
poorPeoples = data.loc[data.salary == '<=50K']
poorAges = poorPeoples.age
poorAges.std()
```

```
14.020161692826626
```

[ ]
```
#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)
richPeoples.education.unique()
```

```
array([], dtype=object)
```

[ ]
```
#7. Display age statistics for each race (race feature) and each gender (sex feature). Use groupby() and 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
```

```
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    19173.000000
mean        39.652793
```


```
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    19173.000000
mean        39.652793
```

```
#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.
notMariedStatuses = ['Never-married', 'Separated', 'Divorced', 'Widowed']
notMariedMen = data.loc[(data.sex == 'Male') & (data.maritalStatus.isin(notMariedStatuses)), 'salary']
notMariedMen.value_counts()
```

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

```
mariedMen = data.loc[(data.sex == 'Male') & data.maritalStatus.str.startswith('Married'), 'salary']
mariedMen.value_counts()
```

```
<=50K    7574
>50K     5964
=50K        1
Name: salary, 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?
maxLoad = data.hoursPerWeek.max()
print("Max time - {0} hours./week.".format(maxLoad))

numWorkaholics = data[data.hoursPerWeek == maxLoad].shape[0]
print("Total number of such hard workers {0}".format(numWorkaholics))

richWorkaholics = data[(data.hoursPerWeek == maxLoad) & (data.salary == '>50K')]
richShare = float(richWorkaholics.shape[0]) / numWorkaholics
print("Percentage of rich among them {0}%".format(100 * richShare))
```

```
Max time - White hours./week.
Total number of such hard workers 1
Percentage of rich among them 0.0%
```

```
#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?
pd.crosstab(data.nativeCountry, data.salary, values=data.hoursPerWeek, aggfunc=np.mean).T
```

nativeCountry	?	Cambodia	Canada	China	Columbia	Cuba	Dominican-Republic	Ecuador	El-Salvador	England	France	Germany	Greece	Guatemala	Haiti	Holand-Netherlands	Honduras
salary																	
<=50K	40.164760	41.416667	37.914634	37.381818	38.684211	37.985714	42.338235	38.041667	36.030928	40.483333	41.058824	39.139785	41.809524	39.360656	36.325	40.0	34.333333
>50K	45.547945	40.000000	45.641026	38.900000	50.000000	42.440000	47.000000	48.750000	45.000000	44.533333	50.750000	44.977273	50.625000	36.666667	42.750	NaN	60.000000

```
data_devices = pd.read_csv('./sample_data/user_device.csv')
data_devices.head()
```

	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


```
#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.
notMariedStatuses = ['Never-married', 'Separated', 'Divorced', 'Widowed']
notMariedMen = data.loc[(data.sex == 'Male') & (data.maritalStatus.isin(notMariedStatuses)), 'salary']
notMariedMen.value_counts()
```

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

```
mariedMen = data.loc[(data.sex == 'Male') & data.maritalStatus.str.startswith('Married'), 'salary']
mariedMen.value_counts()
```

```
<=50K    7574
>50K     5964
=50K        1
Name: salary, 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?
maxLoad = data.hoursPerWeek.max()
print("Max time - {0} hours./week.".format(maxLoad))

numWorkaholics = data[data.hoursPerWeek == maxLoad].shape[0]
print("Total number of such hard workers {0}".format(numWorkaholics))

richWorkaholics = data[(data.hoursPerWeek == maxLoad) & (data.salary == '>50K')]
richShare = float(richWorkaholics.shape[0]) / numWorkaholics
print("Percentage of rich among them {0}%".format(100 * richShare))
```

```
Max time - White hours./week.
Total number of such hard workers 1
Percentage of rich among them 0.0%
```

```
#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?
pd.crosstab(data.nativeCountry, data.salary, values=data.hoursPerWeek, aggfunc=np.mean).T
```

nativeCountry	?	Cambodia	Canada	China	Columbia	Cuba	Dominican-Republic	Ecuador	El-Salvador	England	France	Germany	Greece	Guatemala	Haiti	Holand-Netherlands	Honduras
salary																	
<=50K	40.164760	41.416667	37.914634	37.381818	38.684211	37.985714	42.338235	38.041667	36.030928	40.483333	41.058824	39.139785	41.809524	39.360656	36.325	40.0	34.333333
>50K	45.547945	40.000000	45.641026	38.900000	50.000000	42.440000	47.000000	48.750000	45.000000	44.533333	50.750000	44.977273	50.625000	36.666667	42.750	NaN	60.000000

```
data_devices = pd.read_csv('./sample_data/user_device.csv')
data_devices.head()
```

	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

```
data_usage = pd.read_csv('./sample_data/user_usage.csv')
data_usage.head()
```

	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

```
def timing(f):
    def wrap(*args):
        time1 = time.time()
        ret = f(*args)
        time2 = time.time()
        print('{:s} function took {:.3f} ms'.format(f.__name__, (time2-time1)*1000.0))

        return ret
    return wrap

@timing
def pandas_merge():
  merged_data = data_devices.merge(data_usage, 'inner', on='use_id')
  return merged_data

@timing
def pandasql_merge(devices,usage):
  simple_query = '''
    SELECT *
    FROM devices JOIN usage
    WHERE devices.use_id==usage.use_id
    '''
  ps.sqldf(simple_query, locals())

@timing
def pandas_group(devices_usage):
  devices_usage.groupby('device').monthly_mb.mean()

@timing
def pandasql_group(devices_usage):
  aggr_query = '''
    SELECT distinct device, avg(monthly_mb) as avg_mb
    FROM devices_usage
    GROUP BY device
  '''
  return ps.sqldf(aggr_query, locals())

devices_usage = pandas_merge()
pandasql_merge(data_devices, data_usage)
pandas_group(devices_usage)
pandasql_group(devices_usage)
```

```
pandas_merge function took 6.392 ms
pandasql_merge function took 22.178 ms
pandas_group function took 1.835 ms
pandasql_group function took 11.202 ms
```

	device	avg_mb
0	A0001	15573.330000
1	C6603	1557.330000
2	D2303	519.120000
3	D5503	1557.330000
4	D5803	1557.330000


```
data_usage = pd.read_csv('./sample_data/user_usage.csv')
data_usage.head()
```

	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

```
def timing(f):
    def wrap(*args):
        time1 = time.time()
        ret = f(*args)
        time2 = time.time()
        print('{:s} function took {:.3f} ms'.format(f.__name__, (time2-time1)*1000.0))

        return ret
    return wrap

@timing
def pandas_merge():
  merged_data = data_devices.merge(data_usage, 'inner', on='use_id')
  return merged_data

@timing
def pandasql_merge(devices,usage):
  simple_query = '''
    SELECT *
    FROM devices JOIN usage
    WHERE devices.use_id==usage.use_id
    '''
  ps.sqldf(simple_query, locals())

@timing
def pandas_group(devices_usage):
  devices_usage.groupby('device').monthly_mb.mean()

@timing
def pandasql_group(devices_usage):
  aggr_query = '''
    SELECT distinct device, avg(monthly_mb) as avg_mb
    FROM devices_usage
    GROUP BY device
  '''
  return ps.sqldf(aggr_query, locals())

devices_usage = pandas_merge()
pandasql_merge(data_devices, data_usage)
pandas_group(devices_usage)
pandasql_group(devices_usage)
```

```
pandas_merge function took 6.392 ms
pandasql_merge function took 22.178 ms
pandas_group function took 1.835 ms
pandasql_group function took 11.202 ms
```

	device	avg_mb
0	A0001	15573.330000
1	C6603	1557.330000
2	D2303	519.120000
3	D5503	1557.330000
4	D5803	1557.330000

5	D6603	7267.550000
6	E6653	5191.120000
7	EVA-L09	1557.330000
8	F3111	2076.450000
9	GT-I8190N	407.010000
10	GT-I9195	1211.260000
11	GT-I9300	464.185000
12	GT-I9505	5564.726364
13	GT-I9506	803.240000
14	GT-I9515	1557.330000
15	GT-N7100	11939.560000
16	HTC Desire 510	12562.488000
17	HTC Desire 530	1557.330000
18	HTC Desire 620	74.400000
19	HTC Desire 626	519.120000
20	HTC Desire 825	5498.970000
21	HTC One M9	2362.070000
22	HTC One S	1038.210000
23	HTC One mini 2	13842.956667
24	HTC One_M8	6577.120000
25	HUAWEI CUN-L01	11.680000
26	HUAWEI VNS-L31	3114.670000
27	LG-H815	1557.330000
28	Lenovo K51c78	1557.330000
29	Moto G (4)	5191.120000
30	MotoE2(4G-LTE)	212.640000