Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

Е.Ю. Агеев

АНАЛИЗ БОЛЬШИХ ДАННЫХ

Методические указания к лабораторным работам,
и организации самостоятельной работы
для студентов направлений «Бизнес-информатика» и «Программная инженерия»
(уровень магистратуры)

Томск
2021

УДК 004.62
ББК 22.18

Рецензент:
Гриценко Ю.Б., доцент кафедры автоматизированной обработки информации ТУСУР, канд. техн. наук

Агеев, Евгений Юрьевич
Методические указания к лабораторным работам, и организации самостоятельной работы для студентов направлений «Бизнес-информатика» и «Программная инженерия» (уровень магистратуры)/ Е.Ю. Агеев. – Томск: Томский государственный университет систем управления и радиоэлектроники, 2021. – 57 с.

Настоящее учебно-методическое пособие содержит описание лабораторных работ, а также рекомендации по организации самостоятельной работы по анализу и обработке данных с помощью инструментария языка программирования Python, предназначено для студентов технических направлений подготовки и специальностей.

Одобрено на заседании кафедры АОИ, протокол № 1 от 20.01.2022

УДК 004.62
ББК 22.18

Оглавление

1 МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

1.1 Лабораторная работа №1. «Анализ криминальной обстановки в Сан-Франциско»

Цели и задачи

Выполнить полный цикла анализ данных, используя заданный набор данных и инструменты, Python и Jupyter Notebook. В частности, произвести импорт необходимых пакетов Python, выполнить загрузку данных, выполнить подготовку данных, произвести анализ данных, осуществить визуализацию полученных результатов.

Сценарий¶

В этой лабораторной работе вы импортируете некоторые пакеты Python, необходимые для анализа набора данных, содержащего информацию о преступлениях в Сан-Франциско. Затем вы будете использовать Python и Jupyter Notebook, чтобы подготовить эти данные для анализа, проанализировать их, построить графики и сообщить свои результаты.

Требуемые ресурсы

ПК с доступом в Интернет

Библиотеки Python: панды, numpy, matplotlib, folium, datetime и csv

Файлы данных: Map-Crime_Incidents-Previous_Three_Months.csv

Импорт необходимых дополнительных пакетов Python

Порядок выполнения работы.

Выполним импорт следующих пакетов Python, необходимых для этой лабораторной работы: NumPy, Pandas, Matplotlib, Folium.

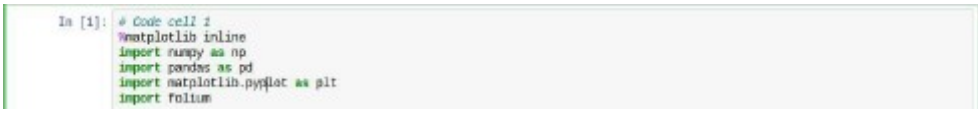
NumPy - это базовый пакет для математических вычислений на Python. Он содержит среди прочего средства работы с массивами объектов и сложные математические функции.

Pandas - это библиотека с открытым исходным кодом, лицензированная BSD, предоставляющая высокопроизводительные, простые в использовании структуры данных и инструменты анализа данных для языка программирования Python.

Matplotlib - это библиотека построения графиков для языка программирования Python и его числового математического расширения NumPy.

Folim - это библиотека для наложения данных на карту местности, создания интерактивной карты.

Импорт дополнительных пакетов и библиотек выполняется следующими командами:



Загрузка данных будет выполняться из ранее подготовленного и предоставленного для выполнения работы файла Map-Crime_Incidents-Previous_Three_Months.csv. Мы определяем переменную dataset_path в которую заносится значение пути до файла с данными и затем используем функцию Pandas read_csv() для чтения данных во вновь создаваемый датафрейм SF:

Иногда перед загрузкой информации из файла может потребоваться просмотреть содержимое файла. Для просмотра первых пяти строк файла csv удобно использовать команду head Linux. Для вызова этой команды из Jupyter Notebook ее необходимо предварить символом «!».

```
In [3]: # Code cell 3
        !head -n 5 ./Data/Map-Crime_Incidents-Previous_Three_Months.csv

IncidntNum,Category,Descript,DayOfWeek,Date,Time,PdDistrict,Resolution,Address,X,Y,Location
,LARCENY/THEFT,GRAND THEFT FROM UNLOCKED AUTO,Sunday,08/31/2014 07:00:00 AM +0000,20:30,CENTRAL,NONE,HYDE ST / CALIFORNI
A ST,-122.417392838334,37.7909741243888,"(37.7909741243888, -122.417392838334)"
,LARCENY/THEFT,GRAND THEFT FROM LOCKED AUTO,Sunday,08/31/2014 07:00:00 AM +0000,14:30,CENTRAL,NONE,COLUMBUS AV / JACKSON
ST,-122.404417628748,37.7963018730836,"(37.7963018730836, -122.404417628748)"
,LARCENY/THEFT,GRAND THEFT FROM LOCKED AUTO,Sunday,08/31/2014 07:00:00 AM +0000,11:30,CENTRAL,NONE,SUTTER ST / STOCKTON
ST,-122.406958660602,37.7894347630337,"(37.7894347630337, -122.406958660602)"
,DRUG/NARCOTIC,POSSESSION OF METH-AMPHETAMINE,Sunday,08/31/2014 07:00:00 AM +0000,17:49,MISSION,"ARREST, BOOKED",16TH ST
/ MISSION ST,-122.419671788296,37.7656581214965,"(37.7656581214965, -122.419671788296)"
```

Просмотр импортированных данных. Датафрейм позволяет иметь структурированные данных, легко обрабатываемые с помощью большого числа встроенных функций Pandas.

Введя имя переменной фрейма данных в ячейку, можно отобразить содержимое фрейма. По умолчанию отображается несколько первых и последних строк. Параметр числа отображаемых строк можно настраивать:

```
In [5]: # Code cell 4
        pd.set_option('display.max_rows', 10) #Visualize 10 rows
        SF
```

Out[5]:

	IncidntNum	Category	Descript	DayOfWeek	Date	Time	PdDistrict	Resolution	Address	X	Y	
0	NaN	LARCENY/THEFT	GRAND THEFT FROM UNLOCKED AUTO	Sunday	08/31/2014 07:00:00 AM +0000	20:30	CENTRAL	NONE	HYDE ST / CALIFORNIA ST	-122.417393	37.790974	(37.7 -122.4
1	NaN	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Sunday	08/31/2014 07:00:00 AM +0000	14:30	CENTRAL	NONE	COLUMBUS AV / JACKSON ST	-122.404418	37.796302	(37.7 -122.4
2	NaN	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Sunday	08/31/2014 07:00:00 AM +0000	11:30	CENTRAL	NONE	SUTTER ST / STOCKTON ST	-122.406959	37.789435	(37.7 -122.4
3	NaN	DRUG/NARCOTIC	POSSESSION OF METH-AMPHETAMINE	Sunday	08/31/2014 07:00:00 AM +0000	17:49	MISSION	ARREST, BOOKED	16TH ST / MISSION ST	-122.419672	37.765050	(37.7 -122.4
4	NaN	DRUG/NARCOTIC	POSSESSION OF COCAINE	Sunday	08/31/2014 07:00:00	18:05	NORTHERN	ARREST, BOOKED	LARKIN ST / O'FARRELL	-122.417904	37.785167	(37.7 -122.4

Для того чтобы отобразить число и наименование столбцов в датафрейме воспользуемся функцией Pandas columns:

```
In [6]: # Code cell 5
        SF.columns

Out[6]: Index(['IncidntNum', 'Category', 'Descript', 'DayOfWeek', 'Date', 'Time',
               'PdDistrict', 'Resolution', 'Address', 'X', 'Y', 'Location'],
              dtype='object')
```

Для отображения числа строк датафрейма можно использовать функцию len():

```
In [7]: # Code cell 6
        len(SF)

Out[7]: 90760
```

Поле “Дата” датафрейма содержит унифицированное представление даты, содержащее день, месяц и год в цифровом виде. Для нашей работы удобнее иметь отдельные колонки для дня и месяца, год может быть опущен, т.к. все операции выполняются над данными в пределах одного года.

Для преобразования поля “Дата” воспользуемся специальной анонимной функцией lambda языка Python. Лямбда позволяет указывать функцию в одной строке кода, без использования def и без определения конкретного имени для нее. Синтаксис лямбда-выражения:

Lambda parameters : expression

В нашем случае лямбда-функция используется для создания встроенной функции, которая выбирает только цифры месяца из переменной Date, и использует тип int для преобразования строкового представления в целое число. Затем применяется функция Pandas apply, чтобы применить эту функцию ко всему столбцу (на практике apply неявно определяет цикл

for и передает по очереди строки лямбда-функции). Аналогичная процедура выполняется для Дня.

```
In [8]: # Code cell 7
        SF['Month'] = SF['Date'].apply(lambda row: int(row[0:2]))
        SF['Day'] = SF['Date'].apply(lambda row: int(row[3:5]))
```

Чтобы убедиться, что эти две переменные были добавлены во фрейм данных SF, используйте функцию печати print, чтобы распечатать некоторые значения из этих столбцов, и введите функцию type, чтобы убедиться, что эти новые столбцы действительно содержат числовые значения:

```
In [9]: # Code cell 8
        print(SF['Month'][0:2])
        print(SF['Day'][0:2])

        0    8
        1    8
        Name: Month, dtype: int64
        0    31
        1    31
        Name: Day, dtype: int64

In [10]: # Code cell 9
         print(type(SF['Month'][0]))

         <class 'numpy.int64'>
```

Столбец датафрейма IncidntNum содержит много ячеек с NaN. В этом случае данные отсутствуют. Кроме того, IncidntNum не предоставляет никакого значения для анализа. Такой столбец лучше удалить из фрейма данных. Один из способов удаления нежелательных переменных во фрейме данных - использование функции del.

```
In [11]: # Code cell 10
         del SF['IncidntNum']
```

Точно так же атрибут Location не будет использоваться в этом анализе. Его можно удалить из фрейма данных. Для удаления столбца Location применим другой способ. Задействуем функцию `drop` для фрейма данных, указав, что axis = 1 (0 для строк), и что команде не требуется присваивать другое значение для сохранения результата (* inplace = True*).

```
In [12]: # Code cell 11
         SF.drop('Location', axis=1, inplace=True )
```

Проверим какие колонки остались в датафрейме и были ли удалены IncidntNum и Location:

```
In [13]: # Code cell 12
         SF.columns

Out[13]: Index(['Category', 'Descript', 'DayOfWeek', 'Date', 'Time', 'PdDistrict',
                'Resolution', 'Address', 'X', 'Y', 'Month', 'Day'],
               dtype='object')
```

Анализ данных. Проведем статистический анализ данных в датафрейме. Используя функцию value_counts, просуммируем количество преступлений каждого типа и выведем результат переменной CountCategory.

```
In [14]: # Code cell 13
         CountCategory = SF['Category'].value_counts()
         print(CountCategory)

         LARCENY/THEFT                  8205
         OTHER OFFENSES                 4804
         NON-CRIMINAL                   3653
         ASSAULT                        2518
         VEHICLE THEFT                  1805
                                        ...
         SEX OFFENSES, NON FORCIBLE        5
         BAD CHECKS                        3
         BRIBERY                           1
         GAMBLING                          1
         PORNOGRAPHY/OBSCENE MAT           1
         Name: Category, Length: 36, dtype: int64
```

По умолчанию данные упорядочены в порядке убывания. Значение необязательного параметра ascending может быть установлено в True, чтобы изменить это поведение.

```
In [15]: # Code cell 14
         SF['Category'].value_counts(ascending=True)

Out[15]: PORNOGRAPHY/OBSCENE MAT          1
         GAMBLING                        1
         BRIBERY                         1
         BAD CHECKS                      3
         SEX OFFENSES, NON FORCIBLE      5
                                       ...
         VEHICLE THEFT                1885
         ASSAULT                      2518
         NON-CRIMINAL                 3653
         OTHER OFFENSES               4904
         LARCENY/THEFT                8285
         Name: Category, Length: 36, dtype: int64
```

Какой тип преступления совершался чаще всего?

Вложив две функции в одну команду, можно достичь того же результата с помощью одной строки кода.

```
In [16]: # Code cell 15
         print(SF['Category'].value_counts(ascending=True))

         PORNOGRAPHY/OBSCENE MAT          1
         GAMBLING                         1
         BRIBERY                          1
         BAD CHECKS                       3
         SEX OFFENSES, NON FORCIBLE       5
                                        ...
         VEHICLE THEFT                 1885
         ASSAULT                       2518
         NON-CRIMINAL                  3653
         OTHER OFFENSES                4904
         LARCENY/THEFT                 8205
         Name: Category, Length: 36, dtype: int64
```

Рассмотрим, как определить количество преступлений по административным округам.

```
In [17]: # code cell 16
         # Possible code for the challenge question
         print(SF['PdDistrict'].value_counts(ascending=True))

         RICHMOND      1622
         PARK          1809
         TARAVAL       2038
         TENDERLOIN    2449
         INGLESIDE     2613
         BAYVIEW       2970
         NORTHERN      3205
         CENTRAL       3857
         MISSION       4011
         SOUTHERN      6185
         Name: PdDistrict, dtype: int64
```

Выделим из фрейма данных подфрейм, задав определенные условия. Например, извлечем только преступления, совершенные в августе. Результат сохраним в новом датафрейме.

```
In [18]: # Code cell 17
         AugustCrimes = SF[SF['Month'] == 8]
         AugustCrimes

Out[18]:
```

	Category	Descript	DayOfWeek	Date	Time	PdDistrict	Resolution	Address	X	Y	Month	Day
0	LARCENY/THEFT	GRAND THEFT FROM UNLOCKED AUTO	Sunday	08/31/2014 07:00:00 AM +0000	20:30	CENTRAL	NONE	HYDE ST / CALIFORNIA ST	-122.417393	37.793974	8	31
1	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Sunday	08/31/2014 07:00:00 AM +0000	14:30	CENTRAL	NONE	COLUMBUS AV / JACKSON ST	-122.404418	37.796932	8	31

Сколько было всего преступлений за август?

Чтобы определить число преступлений определенного типа нужно добавить фильтрацию по выбранной категории, например, количество ограблений:

```
In [19]: # code cell 18
         # Possible code for the question: How many burglaries were reported in the month of August?
         AugustCrimes = SF[SF['Month'] == 8]
         AugustCrimesB = SF[SF['Category'] == 'BURGLARY']
         len(AugustCrimesB)

Out[19]: 1257
```

Мы можем выделить из исходного фрейма подфрейм с числом преступлений в конкретный день выбранного месяца. Чтобы создать подмножество фрейма данных SF для определенного дня, мы используем операнд запроса:

```
In [20]: # Code cell 19
         Crime0704 = SF.query('Month == 7 and Day == 4')
         Crime0704
```

Out[20]:

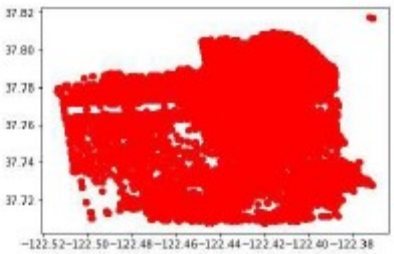
	Category	Descript	DayOfWeek	Date	Time	PdDistrict	Resolution	Address	X	Y	M
19067	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Friday	07/04/2014 07:00:00 AM +0000	22:30	SOUTHERN	NONE	8TH ST / MISSION ST	-122.413161	37.777457	
19068	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Friday	07/04/2014 07:00:00 AM +0000	18:15	SOUTHERN	NONE	CLEMENTINA ST / 9TH ST	-122.412174	37.774291	
19069	BURGLARY	BURGLARY,RESIDENCE UNDER CONSTRT, FORCIBLE ENTRY	Friday	07/04/2014 07:00:00 AM +0000	00:53	TARAVAL	NONE	0 0 Block of MENDOSA AV	-122.466414	37.748011	

Представление данных

Визуализация и представление данных обеспечивает такой взгляд на данные, который позволяет увидеть их полностью и иногда в новом свете, что позволяет сделать вывод, который может быть не очевиден, если просто посмотреть на необработанные данные. Исходный датафрейм SF содержит координаты долготы и широты, которые можно использовать для построения графика данных.

Используем функцию plot () для графического отображения фрейма данных SF. Зададим необязательный параметр, чтобы построить график красным цветом и установить форму маркера в виде круга - ro:

```
In [22]: # Code cell 21
         plt.plot(SF['X'],SF['Y'], 'ro')
         plt.show()
```



Полученное отображение не показывает привязку криминальных событий к административным округам. Для того чтобы реализовать такую привязку присвоим административным округам численные идентификаторы:

```
In [23]: # Code cell 22
         pd_districts = np.unique(SF['PdDistrict'])
         pd_districts_levels = dict(zip(pd_districts, range(len(pd_districts))))
         pd_districts_levels
```

Out[23]:

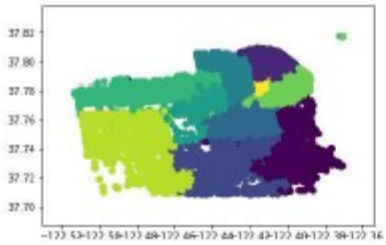
```
{'BAYVIEW': 0,
 'CENTRAL': 1,
 'INGLESIDE': 2,
 'MISSION': 3,
 'NORTHERN': 4,
 'PARK': 5,
 'RICHMOND': 6,
 'SOUTHERN': 7,
 'TARAVAL': 8,
 'TENDERLOIN': 9}
```

С помощью функций apply и lambda создадим новый столбец целочисленного идентификатора административного округа и добавим его в датафрейм.

```
In [24]: # Code cell 23
         SF['PdDistrictCode'] = SF['PdDistrict'].apply(lambda row: pd_districts_levels[row])
```

Теперь используем вновь полученный столбец для автоматического изменения цвета при построении изображения:

```
In [25]: # Code cell 24
plt.scatter(SF['X'], SF['Y'], c=SF['PdDistrictCode'])
plt.show()
```



Мы создали простое графическое изображение, которое показывает, где в округах Сан-Франциско произошли преступления. Этот график полезен, но гораздо более информативным было бы наложение наших данных на реальную карту местности. В библиотеке folium есть функции, которые позволят нам это сделать. Мы будем использовать картографический сервис Open Street Map.

Folium требует, чтобы цвет маркера был указан с использованием шестнадцатеричного значения. Поэтому нам потребуется заново определить, как будут отображаться цвета.

```
In [26]: # Code cell 25
from matplotlib import colors
districts = np.unique(SF['PdDistrict'])
print(list(colors.cnames.values())[0:len(districts)])
```

```
['#F0F8FF', '#FAEBD7', '#00FFFF', '#7FFFD4', '#F0FFFF', '#F5F5DC', '#FFE4C4', '#000000', '#FFEBCD', '#0000FF']
```

Создадим словарь Python для цветовой маркировки административных округов:

```
In [27]: # Code cell 26
color_dict = dict(zip(districts, list(colors.cnames.values())[0:-1:len(districts)]))
color_dict
```

```
Out[27]: {'BAYVIEW': '#F0F8FF',
 'CENTRAL': '#8A2BE2',
 'INGLESIDE': '#00FFFF',
 'MISSION': '#FF8C00',
 'NORTHERN': '#FF1493',
 'PARK': '#F8F8FF',
 'RICHMOND': '#480082',
 'SOUTHERN': '#FAFAD2',
 'TARAVAL': '#B9C4DE',
 'TENDERLOIN': '#9370DB'}
```

Теперь мы можем создать карту с наложенными на нее данными о криминальных событиях. Для центрирования карты используем среднее значение координат из датафрейма SF (функция mean). Чтобы сократить время вычислений, для параметра plotEvery задается значение 0, таким образом ограничивается объем отображаемых данных и уменьшается время обработки. Установите это значение в 1, чтобы вывести на карту значения всех строк датафрейма (для визуализации карты может потребоваться много времени):

```
In [28]: # Code cell 27
# Create map
map_osm = folium.Map(location=[SF['Y'].mean(), SF['X'].mean()], zoom_start = 12)
plotEvery = 50
obs = list(zip( SF['Y'], SF['X'], SF['PdDistrict']))

for el in obs[0:-1:plotEvery]:

    folium.CircleMarker(el[0:2], color=color_dict[el[2]], fill_color=el[2],radius=10).add_to(map_osm)
```

```
In [29]: # Code cell 28
map_osm
```

1. Сколько всего административных округов в Сан-Франциско?
2. В каких административных округах наибольшая преступность?
3. Какие виды преступлений совершаются чаще всего?
4. Каково назначение функции Лямбда в Python?
5. Какие виды преступлений регистрируются в Сан-Франциско?

1.2 Лабораторная работа «Измерение и анализ данных Интернет-соединения»

Цель работы

Научиться получать и сохранять данные от реальных источников в реальном времени, сохранять и обрабатывать эти данные.

Сценарий

В этой лабораторной работе вы получите статистику скорости интернета и сохраните текущие данные в файле значений, разделенных запятыми (csv). Вы также загрузите сохраненные данные из файла csv в структуру данных Python, Pandas `DataFrame`, и будете использовать их функциональные возможности для исследования данных и манипулирования ими, чтобы они были легко читаемыми. В работе проводятся измерения трех типов: 1) времени двойного оборота (время между отправкой сообщения в сеть и получением ответа на это сообщение) 2) времени загрузки 3) времени выгрузки.

Требуемые ресурсы

ПК с доступом в Интернет

Библиотеки Python: datetime, csv, subprocess, pandas, numpy

Файлы данных: rpi_data_long.csv

Порядок выполнения работы.

Для выполнения работы необходимо установить утилиту Speedtest и импортировать ряд дополнительных библиотек Python.

Speedtest-cli - это скрипт Python, который измеряет скорость загрузки и выгрузки вашего интернет-соединения. Для получения дополнительной информации о speedtest перейдите по ссылке https://github.com/sivel/speedtest-cli.

Установка программного пакета в операционной системе может быть выполнена с помощью установщика пакетов Python pip:

```
In [1]: # Code cell 1
        !pip install speedtest-cli

        Collecting speedtest-cli
          Downloading https://files.pythonhosted.org/packages/61/8b/58d1de9a7fff3e91c5ab956ab4ba72b49f42d9f73d5f3e248c748dfcc816
        /speedtest_cli-2.1.1-py2.py3-none-any.whl
        Installing collected packages: speedtest-cli
        Successfully installed speedtest-cli-2.1.1
```

Утилита Speedtest позволяет подключиться к вебсайту в Интернет, загружать и сохранять данные, одновременно измеряя скорость загрузки.

Следующим шагом нужно установить дополнительные библиотеки Python:

```
In [2]: # Code cell 2
        # Python library to manage date and time data
        import datetime
        # Python library to read and write csv files
        import csv
        # Python library to execute bash commands from the notebook.
        # If you want to know more about this, check this resource:
        # http://www.pythonforbeginners.com/os/subprocess-for-system-administrators
        import subprocess
```

Так как в этой лабораторной работе будут проводиться измерения статистики скорости Интернета, естественно, что важным шагом в сборе данных будет привязка временной метки к измерениям. Чтобы сгенерировать метку времени, мы будем использовать функцию datetime.now пакета datetime. Убедиться в том, что установка пакета datetime прошла успешно можно, вызвав эту функцию:

```
In [3]: # Code cell 3
        date_time = datetime.datetime.now()
        print(date_time, type(date_time))

        2019-05-30 00:02:27.000000 <class 'datetime.datetime'>
```

Экземпляр класса datetime нельзя напрямую записать в текстовой форме. Так как манипуляции с данными времени удобно проводить, когда они представлены именно строковым типом, выполним преобразование данных в строковую форму. Функция strftime преоб-

разует информацию о дате в строку. Аргументы этой функции определяют формат строки вывода. Описание этих параметров можно найти в документации по функции strftime [1]

```
In [4]: # Code cell 4
        date_time.strftime('%a, %d %b %Y %H:%M:%S')
Out[4]: 'Fri, 24 May 2019 23:55:03'
```

Прочитайте документацию и сгенерируйте метку времени с преобразованием результата в строку в следующем формате: ГГГГ-ММ-ДД ЧЧ: ММ: СС.

Команда speedtest-cli, если она запускается из терминала, возвращает строку со скоростями загрузки и выгрузки. Для запуска команды из Jupyter Notebook необходимо использовать модуль подпроцесса Python, который позволяет запускать процесс непосредственно из ячейки кода Jupyter Notebook. Запустите тест скорости, используя команду speedtest-cli из Python. Вывод будет сохранен в переменной process_output.

```
In [4]: # Code cell 6
        # This string contains the command line to interface with speedtest.net
        speedtest_cmd = "speedtest-cli --simple"
        # Execute the process
        process = subprocess.Popen(speedtest_cmd.split(), stdout=subprocess.PIPE)
        # Collect the command output
        process_output = process.communicate()[0]
```

Выведем на экран результат измерений. Обратите внимание на тип переменной process_output:

```
In [5]: # Code cell 7
        print(process_output, type(process_output))
b'Ping: 22.663 ms\nDownload: 5.97 Mbit/s\nUpload: 0.64 Mbit/s\n' <class 'bytes'>
```

Мы видим, что переменная содержит результаты всех требуемых измерений, однако нам требуется добавить к выводу метку измерений, а также необходимо разбить вывод на отдельные элементы, так чтобы удобно было их обрабатывать и сохранять независимо друг от друга:

```
In [6]: # Code cell 8
        # Store the time at which the speedtest was executed
        date_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        process_output = process_output.split()
        process_output.append(date_time)
        print(process_output, type(process_output))
[b'Ping:', b'22.663', b'ms', b'Download:', b'5.97', b'Mbit/s', b'Upload:', b'0.64', b'Mbit/s', '2019-05-30 00:05:12'] <c
lass 'list'>
```

Таким образом, мы получили переменную типа список Python с результатами измерений и меткой времени. Оформим все вышеперечисленные действия по запуску утилиты speedtest-cli и обработке ее вывода для получения результата в требуемом виде с помощью функции speedtest ():

```
In [7]: # Code cell 9
        # function to excute the speed test
        def speedtest():
            # We need to store the time at which the speedtest was executed
            date_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            # This is a string that contains what we would write on the command line
            #to interface with speedtest.net
            speedtest_cmd = "speedtest-cli --simple"
            # We now execute the process:
            process = subprocess.Popen(speedtest_cmd.split(), stdout=subprocess.PIPE)
            process_output = process.communicate()[0]
            process_output = process_output.split()
            # and we add the date and time
            process_output.append(date_time)
            return process_output
```

Вызовем функцию:

```
In [8]: speedtest()

Out[8]: [b'Ping:',
         b'21.788',
         b'ms',
         b'Download:',
         b'6.09',
         b'Mbit/s',
         b'Upload:',
         b'0.73',
         b'Mbit/s',
         '2019-05-30 00:06:39']
```

Что возвращает функция speedtest ()? Какой код используется для просмотра результатов функции speedtest ()?

Для того, чтобы обработка полученных значений давала корректные и объективные результаты хорошо накопить достаточно большой объем данных, а это, в свою очередь, лучше всего делать, сохраняя данные на жесткий диск компьютера для последующей обработки. Файлы со значениями данных, разделенные запятыми (csv), являются наиболее распространенным форматом импорта и экспорта для электронных таблиц и баз данных. Больше информации о работе с CSV в Python в официальной документации [2].

Для того чтобы освоить основы работы с файлами, создадим файл с именем test.txt в каталоге / tmp и сохраним в нем текстовую строку «test_msg»:

```
In [9]: # Code cell 11
        with open("/tmp/test.txt",'w') as f:
            f.write('test_msg')
```

Для проверки того что файл был создан и что он содержит именно ту самую текстовую строку используем команду Linux cat, введем ее прямо в ячейку Jupyter Notebook, предварив символом «!»

```
In [10]: # Code cell 12
         !cat /tmp/test.txt

         test_msg
```

Теперь убедимся в том, что мы можем успешно открыть и прочитать содержимое файла используя инструменты Python:

```
In [11]: # Code cell 13
         with open("/tmp/test.txt",'r') as f:
             str = f.read()
         print(str)

         test_msg
```

Понимание значения оператора with, особенно в сочетании с try и except, не требуется для остальной части этой лабораторной работы, но полезным ресурсом для более подробного изучения этого вопроса является http://effbot.org/zone/python-with-statement.htm

Для записи в файл `csv` необходимо создать объект` csv.writer`. Ознакомьтесь также с документом https://docs.python.org/2/library/csv.html и выясните, какую функцию объекта 'csv.writer' можно использовать для добавления строки в файл `csv`.

Аналогично предыдущему, запишем функцию для записи полученных данных в csv-файл:

```
In [9]: # Code cell 14
        # function to save data to csv
        def save_to_csv(data, filename):
            try:
                # If the file exists, we want to append a new line to it, with the
                #results of the current experiment
                with open(filename + '.csv', 'a') as f:
                    wr = csv.writer(f)
                    wr.writerow(data)
            except:
                # If it does not exist, create the file first
                with open(filename + '.csv', 'w') as f:
                    # Hint: This is similar to appending new lines to a file.
                    # Create a csv writer object
                    # ADD CODE HERE
                    # Save (write) to file
                    # ADD CODE HERE
```

Напишите также функцию, открывающую CSV-файл и выводящую его содержимое на экране. Вы можете найти пример в разделе 13.1.5 https://docs.python.org/2/library/csv.html

Общая структура этой функции будет примерно такой:

```
In [13]: # Code cell 15
         def print_from_csv(filename):
             with open(filename + '.csv', 'r') as f:
                     re = csv.reader(f)
                     # 1. Loop over the rows
                     # 2. print
```

Теперь у нас есть все функции, необходимые для сбора и хранения данных о скорости Интернета.

Процедура проведения измерений и сбора данных.

Напишем цикл for, который вызывает speedtest 5 раз, печатает выходные данные тестов и сохраняет данные в CSV-файле:

```
In [ ]: # Code cell 16
        for i in range(5):
            speedtest_output = speedtest()
            print('Test number {}'.format(i))
            print(speedtest_output)
            save_to_csv(speedtest_output, '/tmp/rpi_data_test')
```

Откроем файл, чтобы убедиться, что данные были сохранены правильно, используем при этом нашу функцию:

```
In [ ]: # Code cell 17
        print_from_csv('/tmp/rpi_data_test')
```

Если требуется больший набор данных, то для ускорения можно использовать скоростное тестирование в фоновом режиме. Как бы вы изменили код, если хотите выполнить код, например 100 раз?

Один из вариантов решения:

```
In [ ]: # Code cell 18
        # Code to run 100 times
        # for i in xrange(100):
        #     speedtest_output = speedtest()
        #     print 'Test number: {}'.format(i)
        #     print speedtest_output
        #     save_to_csv(speedtest_output, '/tmp/rpi_data')
```

Обработка данных.

Библиотека Python Pandas очень эффективна при работе со структурированными данными. Pandas имеет очень хорошую и подробную документацию, доступную онлайн: http://pandas.pydata.org/pandas-docs/version/0.14.1/

Для этой части лабораторной работы будет использоваться большой набор данных, собранных заранее. Данные содержаться в файле - rpi_data_long.csv. Данный файл может быть заменен собственным файлом, примерно равного размера, подготовленным самостоятельно.

Импортируем необходимые дополнительные библиотеки Python:

```
In [ ]: # Code cell 19
        import datetime
        import csv
        import pandas as pd
        # NumPy is a library that adds support for large, multi-dimensional arrays and matrices
        # along with high-level mathematical functions to operate on these arrays
        import numpy as np
```

Загрузим файл csv в объект DataFrame, используя Pandas. DataFrame Pandas - это двумерная структура данных со столбцами разных типов. Датафрейм подобен электронной таблице или таблице SQL. Функция библиотеки pandas read_csv автоматически преобразует CSV-файл в объект DataFrame.

Прочитайте документацию по функции read_csv по адресу http://pandas.pydata.org/pandas-docs/version/0.14.1/generated/pandas.read_csv.html. Эта функция содержит много параметров. Единственным обязательным является путь к файлу, то есть местоположение файла CSV. Все остальные параметры являются необязательными.

Будем считать, что работа выполняется с файлом rpi_data_long.csv. Этот CSV-файл находится в том же каталоге, что и корневая папка Jupyter Notebook.

Выполним несколько предварительных операций. Для того чтобы убедиться, что файл на месте и мы можем из него читать, выполним команду head Linux для просмотра первых 5 строк файла csv:

```
In [ ]: # Code cell 21
        !head -n 5 ./Data/rpi_data_long.csv
```

Присвоим переменной data_file значение пути к файлу rpi_data_long.csv:

```
In [ ]: # Code cell 20
        data_file = './Data/rpi_data_long.csv'
```

Колонки в файле сейчас содержат только результаты измерений. С датафреймом удобнее работать, когда каждая колонка имеет название. Подготовим список с названиями колонок:

```
In [ ]: # Code cell 22
        column_names = [ 'Type A', 'Measure A', 'Units A',
                         'Type B', 'Measure B', 'Units B',
                         'Type C', 'Measure C', 'Units C',
                         'Datetime']
```

Используем функцию read_csv для чтения из файла данных и присвоим значения переменной column_names в качестве имен столбцов в датафрейме:

```
In [ ]: # Code cell 23
        with open(data_file, 'r') as f:
            df_redundant = pd.read_csv(f, names = column_names)
```

Метод head () отображает первые несколько строк датафрейма:

```
In [ ]: # Code cell 24
        # You can specify the number of rows you want to print to screen:
        # you do so passing the number as an argument to the function
        # (e.g., head(10))
        df_redundant.head()
```

Как нужно модифицировать строку использования метода head () для отображения первых 20 строк csv файла?

Создадим более компактное представление, используя копию фрейма данных df_redundant. Скопируем df_redundant в другой фрейм данных с именем df_compact, используя метод copy ():

```
In [ ]: # Code cell 25
        df_compact = df_redundant.copy()
```

Переименуем столбцы, дав им имена, соответствующие конкретным измерениям:

- Measure A -> Ping (ms)
- Measure B -> Download (Mbit/s)
- Measure C -> Upload (Mbit/s)

Для этого используем метод rename():

```
In [ ]: # Code cell 26
        df_compact.rename(columns={'Measure A':'Ping (ms)',
                                   'Measure B': 'Download (Mbit/s)',
                                   'Measure C': 'Upload (Mbit/s)'}, inplace=True)
        df_compact.head(3)
```

Поскольку, после переименования, столбцы Type и Units больше не нужны, эти столбцы могут быть удалены:

```
In [ ]: # Code cell 27
        df_compact.drop(['Type A', 'Type B', 'Type C',
                'Units A', 'Units B', 'Units C'], axis=1, inplace=True)
        df_compact.head()
```

Разделим столбец Datetime на два столбца Date и Time, скопировав в эти столбцы соответствующие данные из столбца Datetime. Используем для этого функцию `lambda`, позволяющую строить логические ясные и компактные выражения. В данном случае мы создадим две анонимные функции, которые извлекают только дату и время из объекта` datetime` соответственно. Затем мы применим функцию Pandas apply, чтобы применить указанные анонимные функции ко всему столбцу (на практике `apply` неявно определяет цикл` for` и передает строки по одной нашей функции `lambda`). Результат сохраним в двух новых столбцах DataFrame.

```
In [ ]: # Code cell 28
        df_compact['Date'] = df_compact['Datetime'].apply(lambda dt_str: pd.to_datetime(dt_str).date())
In [ ]: # Code cell 29
        # Please note, this requires an intermediate step, because of how NaT are treated by the time() function.
        # Reference: https://github.com/pandas-dev/pandas/issues/11453
        temp = df_compact['Datetime'].apply(lambda dt_str: pd.to_datetime(dt_str))
        df_compact['Time'] = temp.dt.time
In [ ]: # Code cell 29
        # Please note, this requires an intermediate step, because of how NaT are treated by the time() function.
        # Reference: https://github.com/pandas-dev/pandas/issues/11453
        temp = df_compact['Datetime'].apply(lambda dt_str: pd.to_datetime(dt_str))
        df_compact['Time'] = temp.dt.time
```

Поскольку вся информация из столбца Datetime теперь скопирована в столбцы Date и Time, сам столбец Datetime теперь не нужен. Удалим этот столбец из фрейма данных. Напишите самостоятельно код удаления столбца Datetime. Для того чтобы убедиться что изменения проведены успешно, выведите первые 3 строки датафрейма df_compact.

Чтобы вывести тип значений в столбцах Date и Time используйте метод type:

```
In [ ]: # Code cell 32
        print(df_compact['Date'][0], type(df_compact['Date'][0]) )
        print(df_compact['Time'][0], type(df_compact['Time'][0]) )
```

Сохраните датафрейм df_compact в виде файла csv с именем rpi_data_compact:

```
In [ ]: # Code cell 33
        df_compact.to_csv('./Data/rpi_data_compact.csv')
```

Контрольные вопросы:

1. Что такое анонимная функция Python, как она работает
2. Что возвращает функция speedtest ()? Какой код используется для просмотра результатов функции speedtest ()?
3. Каким образом выполняется переименование столбцов датафрейма?
4. Для чего в лабораторной работе импортируется библиотека NumPy?
5. Как выглядит код удаления столбца Datetime?

1.3 Лабораторная работа «Измерение и анализ Интернет-соединения с хранилищем данных в виде SQL базы данных»

Цель работы

Познакомиться с методами создания, подключения, формирования запросов к SQL-базе данных с помощью функций Python..

Сценарий

В этой лабораторной работе вы узнаете, как подключиться к реляционной базе данных, выполнять запросы и получать данные из базы данных в Jupyter Notebook. Предоставленные вам данные содержат различные измерения проверки связи, загрузки и загрузки. Скорость Интернет-подключения в разных регионах по всей Англии. Данные хранятся в CSV-файле, но вы создадите базу данных и заполните ее данными измерений.

Требуемые ресурсы

ПК с доступом в интернет. Библиотеки Python: sqlite3, pandas, matplotlib, дополнительные библиотеки: csvkit. Файлы данных: LA_wifi_speed_UK.csv, LA_population.csv.

Порядок выполнения работы.

В работе будет использоваться база данных SQLite, которая реализует автономную, бессерверную, транзакционную базу данных SQL. Для выполнения работы понадобятся утилиты sqlite3 и csvkit. Необходимо убедиться, что они установлены в операционной системе. Выполнить команды запуска утилит можно прямо из Jupyter Notebook, для этого в исполняемой ячейке нужно ввести эти команды, предварив их символом восклицательного знака, а затем запустить ячейку на исполнение. Например, так:

```
In [1]: !sqlite -version
        2.8.17
```

Если это не так, и утилиты еще не установлены в системе, то команды установки можно также выполнить прямо из Jupyter Notebook.

```
In [1]: # Code Cell 1
        !apt-get update
        !apt-get -y install sqlite3
```

После завершения процесса установки создадим базу данных:

```
In [2]: # Code Cell 2
        #Create the InternetSpeed database
        !sqlite3 ./Data/InternetSpeed.db ".databases"
        main: /home/pi/notebooks/myfiles/BOA_Python3_sandbox/Ch2/./Data/InternetSpeed.db
```

Этот код создает базу данных с именем InternetSpeed.db в подкаталоге Data текущего каталога.

Таким образом, база данных с указанным именем создана, но она пуста и даже не имеет структуры (наименований таблиц, столбцов и строк). Необходимо импортировать данные из имеющегося файла csv с измерениями скорости Wi-Fi подключения, зарегистрированными в 300 населенных пунктах в Соединенном Королевстве Великобритания. Файл структурирован таким образом, что каждый столбец содержит измерения для местоположения одной из трех статистик (ping, download и upload). Это означает, что таблица, в которой мы нуждаемся, имеет по 3 столбца (плюс дата и время и столбец индекса) и по 300 записей в каждом. Для автоматизации импорта данных из csv-файла в SQL-базу данных используем программный пакет csvkit , который содержит функцию csvsql, с помощью которой можно легко импортировать данные в базу данных с одновременным созданием в этой базе соответствующей структуры. При отсутствии данного пакета его необходимо установить. Это можно сделать с помощью менеджера пакетов pip для языка Python:

```
In [3]: # Code Cell 3
        #Installing csvkit
        !pip install csvkit
```

Импорт данных в базу данных

Используем csvsql, чтобы создать новую таблицу в базе данных InternetSpeed и записать в нее данные, содержащиеся в файле csv. Однако, мы уже создали пустую базу данных с таким именем, при работе с csvsql мы вновь будем создавать базу данных и поэтому нам

потребуется сначала удалить ранее созданную базу и убедиться, что нет другой базы данных с таким же именем. Если она существует, то удалим ее:

```
In [25]: # Code Cell 4
         # Removing database if already existing
         !test -e InternetSpeed.db && rm InternetSpeed.db
         # Importing the LA_wifi_speed_UK.csv file into a sql database file InternetSpeed.db
         !csvsql --db sqlite:///InternetSpeed.db --insert ./Data/LA_wifi_speed_UK.csv
         # This will take a while
```

Команда test –e проверяет наличие файла InternetSpeed.db в текущем каталоге. Символ двойного амперсанда реализует операцию логического “И”, таким образом, если файла с именем InternetSpeed.db не будет в текущем каталоге, то выполнение команды завершится, и мы перейдем к следующей строке, если же такой файл будет, то выполнится следующая за символом двойного амперсанда команда удаления файла InternetSpeed.db. Следующая строка выполняет команду создания базы данных и занесения в нее информации из файла LA_wifi_speed_UK.csv

Подключение к базе данных и выполнение запросов.

Подключитесь к базе данных, которую вы только что создали, а затем создайте курсор, указывающий на базу данных:

```
In [27]: # Code Cell 6
         # what is the name of the database you want to connect to?
         # conn = sqlite3.connect(??)
         #SOLUTION:
         conn = sqlite3.connect('InternetSpeed.db')
         cur = conn.cursor()
```

Чтобы выполнить запрос к базе данных, нам нужно будет использовать только что созданный курсор и метод execute (). Этот метод принимает в качестве входных данных строку, содержащую интересующий нас запрос. Мы можем сохранить строку в переменной, назовем ее запросом. Вызовем метод следующим образом:

cur.execute (запрос)

Таблица, которую мы хотим запросить, не очень хороша для визуализации, поэтому мы собираемся ВЫБРАТЬ только столбец DateTime, который содержит дату и время измерений. Мы ограничим количество строк, извлекаемых запросом, десятью.

```
In [28]: # Code Cell 7
         # complete the query with the field you want to select, the table you want to select the field from,
         # and the field you want your results to be ordered by
         #query = 'SELECT ? FROM ? ORDER BY ? LIMIT ?;'

         # SOLUTION:
         query = 'SELECT DateTime FROM LA_wifi_speed_UK ORDER BY DateTime LIMIT 10;'

         cur.execute(query)
```

Результат запроса сохраняется в переменной cur. Так как это таблица, чтобы вывести результат построчно можно использовать цикл:

```
In [29]: # Code Cell 8
         for row in cur:
             print(row)

('2016-11-23 00:00:00.000000',)
('2016-11-23 00:30:00.000000',)
('2016-11-23 01:00:00.000000',)
('2016-11-23 01:30:00.000000',)
('2016-11-23 02:00:00.000000',)
('2016-11-23 02:30:00.000000',)
('2016-11-23 03:00:00.000000',)
('2016-11-23 03:30:00.000000',)
('2016-11-23 04:00:00.000000',)
('2016-11-23 04:30:00.000000',)
```

В SQL нет внутреннего порядка строк, и мы имеем дело с временными рядами, которые без правильного порядка не имеют никакого смысла. Выбор случайной выборки строк приведет к удалению временного компонента. Поэтому нужно отсортировать таблицу, содержащую измерение, по столбцу DateTime.

```
In [30]: # Code Cell 9
query = 'SELECT DateTime FROM LA_wifi_speed_UK ORDER BY RANDOM() LIMIT 10'
cur.execute(query)

for row in cur:
    print(row)
```

```
('2016-11-25 03:00:00.000000',)
('2016-12-03 17:00:00.000000',)
('2016-11-26 01:00:00.000000',)
('2016-11-24 19:00:00.000000',)
('2016-12-09 10:00:00.000000',)
('2016-11-26 11:00:00.000000',)
('2016-11-23 10:30:00.000000',)
('2016-12-08 17:00:00.000000',)
('2016-12-04 04:00:00.000000',)
('2016-12-04 21:30:00.000000',)
```

Сфокусируемся на порции данных, отобранных по определенному признаку. Выясним наименование столбцов. Для этого сформируем запрос и сохраним результат в переменной типа «список». Эта переменная пригодится нам позже. Запрос выполним по всем столбцам, но так как наименование столбцов находится в первой строке, то ограничим количество строк до 1.

```
In [31]: # Code Cell 10
# we need to have a cursor pointing to a table for this operation
query = 'SELECT * FROM LA_wifi_speed_UK LIMIT 1'
cur.execute(query)
```

Названия столбцов таблицы хранятся в поле description курсора. Выведем значение этого поля:

```
In [32]: # Code Cell 11
#visualizing the first 10 rows of the field description
cur.description[:10]
```

```
Out[32]: (('E07000223_p', None, None, None, None, None, None),
 ('E07000026_p', None, None, None, None, None, None),
 ('E07000032_p', None, None, None, None, None, None),
 ('E07000224_p', None, None, None, None, None, None),
 ('E07000170_p', None, None, None, None, None, None),
 ('E07000105_p', None, None, None, None, None, None),
 ('E07000004_p', None, None, None, None, None, None),
 ('E07000200_p', None, None, None, None, None, None),
 ('E09000002_p', None, None, None, None, None, None),
 ('E09000003_p', None, None, None, None, None, None))
```

Для создания списка с наименованиями столбцов можно организовать цикл по полю description и внутри этого цикла каждое значение поля присвоить соответствующему элементу списка:

```
In [33]: # Code Cell 12
# loop over the header and store them in a list
columns = [member[0] for member in cur.description]

# ignore index column
columns = columns[1:]
```

Столбцы содержат измерения трех скоростей для каждой географической области. Они в такой форме: <имя области> _ <суффикс>, где суффикс p для ping, u для загрузки и d для загрузки.

Нужно избавиться от суффиксов, так как нас интересуют названия области, и на данный момент каждое имя повторяется три раза. Названия областей используйте для визуализации различных измерений и создания новой таблицы.

Можно заменить заключительную часть строки пустым символом. Для этого используйте метод replace () (см. документацию, https://docs.python.org/2/library/string.html). Этот метод принимает в качестве входных данных часть строки, подлежащую замене, и новую часть строки, подлежащую замене. Например, чтобы удалить символы e и s из строки chestnut, используйте этот код: 'chestnut'.replace('es', '').

Замените наименования столбцов:

```
In [34]: # Code Cell 13
         # remove suffix '_p'
         columns = [c.replace('_p', '') for c in columns]
         # remove suffix '_d'
         columns = [c.replace('_d', '') for c in columns]
         # remove suffix '_u'
         # columns = ??

         # SOLUTION:
         columns = [c.replace('_u', '') for c in columns]

         # this operation is to remove duplicates
         columns = list(set(columns))
```

Выведем значения первых 10 строк в первом столбце:

```
In [35]: # Code Cell 14
         # visualize the first 10 areas
         columns[:10]

Out[35]: ['E07000223',
          'E08000006',
          'E07000143',
          'E09000014',
          'E09000026',
          'E07000092',
          'E07000031',
          'E07000166',
          'E07000064',
          'E07000035']
```

Лучший способ понять данные в базе данных - визуализировать их часть. В частности, необходимо выбрать три статистики для каждой области. Выберите первую область из набора данных:

```
In [36]: # Code Cell 15
         area = columns[0]
         print(area)

         E07000223
```

Предположим, что мы хотим визуализировать данные для области E07000101. Нам нужно выбрать столбцы, содержащие три соответствующих измерения для этой области. Для этого нам нужно снова манипулировать строками.

Один из способов форматирования строки – использование форматирования, например, конструкция: «Мы рассматриваем {} область и {} статистики». format (1, 3)

Вернет результат: «Мы рассматриваем 1 область и 3 статистики».

Чтобы получить правильные имена столбцов, нужно добавить суффиксы обратно. Для этого нужно создать словарь, который связывает полное имя для измерения с суффиксом. Также будет удобно вывести корректные наименования в качестве легенды на графике.

```
In [37]: # Code Cell 16
         suffix = {'_p':'ping', '_d':'download', '_u':'upload'}
         # we have the suffixes as the keys of the dictionary
         print(suffix.keys())

         # we have the complete names as the values of the dictionary
         print(suffix.values())

         dict_keys(['_d', '_u', '_p'])
         dict_values(['download', 'upload', 'ping'])
```

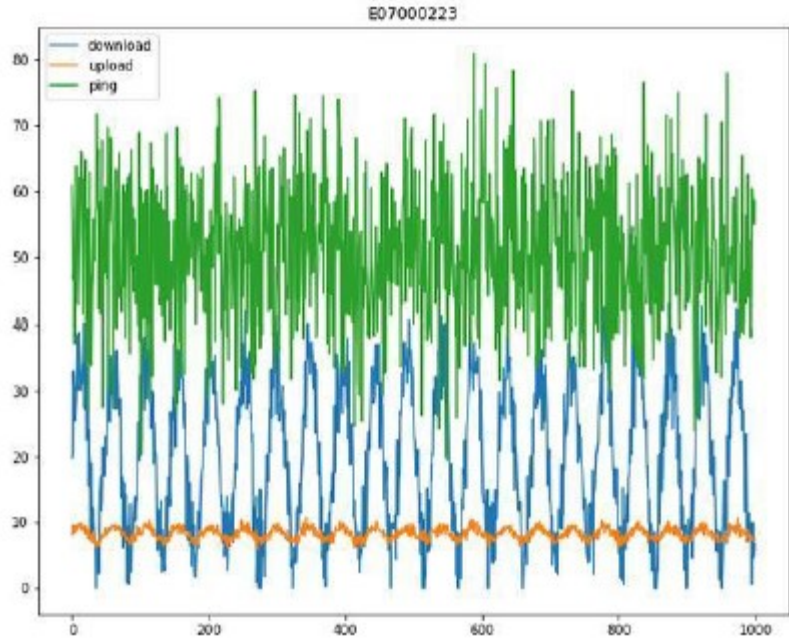
Для того чтобы построить данные на графике используйте этот код:

plt.plot (cur.fetchall (), label = суффикс [s])

Метод fetchall () извлекает все строки результата запроса. Метка поля будет использоваться для легенды на графике. Мы также можем добавить заголовок к графику, используя метод title ().

```
In [38]: # Code Cell 17
         area = columns[0]
         plt.figure(figsize=(10, 8))

         # Plot each variable in suffix.keys() for each area
         for s in suffix.keys():
             # query = ??
             # SOLUTION:
             query = 'SELECT "{}{}" FROM LA_wifi_speed_UK ORDER BY DateTime'.format(area, s)
             cur.execute(query)
             plt.plot(cur.fetchall(), label=suffix[s])
         plt.legend()
         plt.title(area)
```

Результат должен выглядеть примерно так:



Обработка данных с помощью Pandas

Если мы хотим произвести какую-то дополнительную обработку данных, содержащихся в базе данных, например рассчитать средние значения для каждого региона, то это легко сделать с помощью библиотеки Pandas.

Создайте пустой датафрейм с конкретными столбцами:

['Area', 'Average_p', 'Average_d', 'Average_u']

Обратите внимание что порядок создаваемых столбцов датафрейма совпадает с порядком ключей в выходных данных suffix.keys (), то есть ['_p', '_d', '_u'].

In [39]:
```
# Code Cell 18
#new_columns = ?
#df = pd.DataFrame(columns = ?)

# SOLUTION:
new_columns = ['Area', 'Average_p', 'Average_d', 'Average_u']
df = pd.DataFrame(columns = new_columns)
```

Средние значения скорости пинга, загрузки и скачивания для каждой области достаточно интересные и информативные показатели. Чтобы вычислить их нужно перебрать столбцы, и для каждой области, для каждого типа измерения вычислить среднее значение. Затем вставить его в созданный пустой фрейм данных.

Для решения этой задачи нужно сначала создать временный список и заполнить его тремя типами измерений для каждой области. Далее этот список используйте для расчета средних значений:

In [40]:
```
# Code Cell 19
# this will take a while...
for i in range(len(columns)-1): #EDL : replace xrange with range
    tmp_list = []
    tmp_list.append(columns[i])
    for s in suffix.keys():
        # query = ??
        # SOLUTION:
        query = 'SELECT AVG("{}{}") FROM LA_wifi_speed_UK'.format(columns[i], s)

        cur.execute(query)

        mean = cur.fetchone()
        tmp_list.append(mean[0])
    #append the columns to the empty DataFrame
    df = df.append(pd.Series(tmp_list, index=new_columns), ignore_index=True)
# visualize the head of the dataframe here
# SOLUTION:
df.head()
```

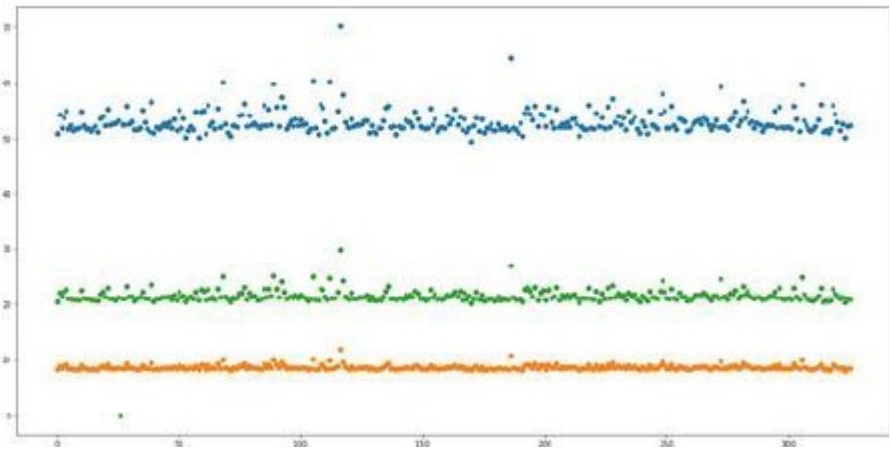
Out[40]:

	Area	Average_p	Average_u	Average_d
0	E07000223	20.453309	8.241653	50.929824
1	E06000006	22.234556	8.674736	54.314151
2	E07000143	21.421846	8.495484	51.887536
3	E06000014	22.446388	8.940971	53.680858

Можно визуализировать средние значения для каждой области, используя функцию plot(). Это покажет, является ли средняя скорость одинаковой для разных областей или нет.

```
In [41]: # Code Cell 20
         plt.figure(figsize=(20,10))
         plt.plot(df.index, df[['Average_d','Average_u','Average_p']], 'o')
         plt.legend(['Average Download', 'Average Upload', 'Average ping'])
```

```
Out[41]: [<matplotlib.lines.Line2D at 0x7dffff50>,
          <matplotlib.lines.Line2D at 0x7df84070>,
          <matplotlib.lines.Line2D at 0x7df84270>]
```



Теперь средние скорости хранятся в фрейме данных Pandas. Но нам нужно сохранить их в таблицу в нашей базе данных, и Pandas предлагает метод to_sql() для этого. Сначала проверьте, существует ли таблица (т.к. если такая таблица уже есть, это может создать проблемы). Если таблица существует, удалим ее, а затем заново создадим. В противном случае просто создадим таблицу.

Конструкция try ... except будет захватывать исключение, возникающее в том случае, если таблица не существует, и перейдет к созданию таблицы (для получения справки см. Раздел 8.3 https://docs.python.org/2.7/tutorial/.errors.html).

```
In [42]: # Code Cell 21
         # Make sure to drop the table first if it already exists
         try:
             cur.execute('DROP TABLE average_speed')
         except:
             pass

         # what is the name of the table you want to save? To what connection?
         # Save the dataframe table into a SQL table
         # df.to_sql(?,?)

         # SOLUTION:
         df.to_sql('average_speed', conn)
```

После занесения данных в базу выведите небольшую порцию этих данных из базы для того, чтобы убедиться в том, что все прошло успешно.

```
In [43]: # Code Cell 22
         # just checking everything worked fine: select everything from the table we just created
         # and then print the first two rows of the result

         query_2 = 'SELECT * FROM average_speed'
         cur.execute(query_2)

         # print the first fetched record
         print(cur.fetchone())
         # print the next fetched record
         print(cur.fetchone())
```

```
(0, 'E07000223', 20.453398977533574, 8.241652932874452, 50.92362409985345)
(1, 'E08000006', 22.234557614959748, 8.874737563433312, 54.31418898967002)
```

Чтобы понять, почему и как меняется средняя скорость интернета по регионам, нам нужно больше информации о них. В частности, можно предположить, что в более населенных районах скорость будет хуже, т.к. число одновременно подключенных пользователей в среднем больше.

Информация о численности населения по районам есть в дополнительном файле LA_population.csv. Часто бывает так что данные, которые необходимо обработать, находятся не в одном, а в нескольких разных файлах. Итак, необходимо объединить информацию, содержащуюся в этом файле, с информацией о средней скорости интернет-подключения, вычисленной ранее.

Создайте в базе данных новую таблицу и сохраните данные о численности населения в базе данных InternetSpeed. Действия по занесению информации в базу данных уже известны, их надо повторить. Используя csvsql, импортируйте файл LA_population.csv в базу данных InternetSpeed. Однако, перед этим нужно закрыть открытое ранее подключение к базе данных:

```
In [44]: # Code Cell 23
         #first close the database file, otherwise the external csvsql command cannot use
         conn.close()
```

Теперь добавьте в базу новую таблицу:

```
In [45]: # Code Cell 24
         # use the external csvsql command to add to the DB file data from the CSV file
         #!csvsql --db sqlite:///...

         #SOLUTION:
         !csvsql --db sqlite:///InternetSpeed.db --insert ./Data/LA_population.csv
```

Вновь подключитесь к базе данных:

```
In [46]: # Code Cell 25
         #reopen the DB file
         #conn = sqlite3.connect(...)

         # SOLUTION:
         conn = sqlite3.connect('InternetSpeed.db')
         cur = conn.cursor()
```

Проверьте, что все сделано верно и посмотрите, что находится внутри этой новой таблицы, выделите первые 10 строк и выведите на экран:

```
In [47]: # Code Cell 26
         #query = ?
         # SOLUTION:
         query = 'SELECT * FROM LA_population LIMIT 10'

         #cur.execute(?)
         # SOLUTION:
         cur.execute(query)

         for row in cur:
             # print the first fetched record
             # SOLUTION:
             print(row)
```

```
('Aberdeen City', 'S12000033', 222793)
('Aberdeenshire', 'S12000034', 252973)
('Adur', 'E07000223', 61182)
('Allerdale', 'E07000026', 96422)
('Amber Valley', 'E07000032', 122309)
('Angus', 'S12000041', 115978)
('Antrim and Newtownabbey', 'N09000001', 138567)
('Ards and North Down', 'N09000011', 156672)
('Argyll and Bute', 'S12000035', 88166)
('Armagh City, Banbridge and Craigavon', 'N09000002', 199693)
```

Мы можем объединить две таблицы, чтобы получить всю необходимую информацию. Идея состоит в том, что две таблицы имеют общее поле, название области. Это общий ключ, который будет использован для объединения. Нам нужна новая таблица, в которой есть строка для каждого имени области, которая содержит как информацию о средней скорости, так и информацию о населенности. Для этого SQL сравнивает строку за строкой и объединяет строки с одинаковым значением ключа (т. е. с одинаковым именем области).

```
In [48]: # Code Cell 27
         query = 'SELECT * FROM average_speed JOIN LA_population ON LA_population."LA_code"=average_speed.Area'

         cur.execute(query)
         k = 0
         for row in cur:
             if k>10:
                 break
             #print ?

             # SOLUTION:
             print(row)

             k+=1
```

```
(0, 'E07000223', 20.453390977533574, 8.241652932074452, 50.92262409905345, 'Adur', 'E07000223', 61182)
(1, 'E08000006', 22.234557614959748, 8.874737563433312, 54.31418090967982, 'Salford', 'E08000006', 233933)
(2, 'E07000143', 21.42184629756809, 8.495484016531023, 51.8875361982357, 'Breckland', 'E07000143', 138401)
(3, 'E09000014', 22.44638822021056, 8.940970870767864, 53.6885773299926, 'Haringey', 'E09000014', 254926)
(4, 'E08000026', 22.68436838582302, 9.061575987134813, 54.93980145898435, 'Redbridge', 'E08000026', 278870)
(5, 'E07000092', 21.88355153349529, 8.328357128304762, 51.67870697781834, 'Rushmoor', 'E07000092', 93807)
(6, 'E07000031', 21.899158811548214, 8.398174084063603, 52.30746638319433, 'South Lakeland', 'E07000031', 10
```

Можете ли вы сейчас ответить на вопрос о зависимости средней скорости Интернет-подключения от населенности района?

Контрольные вопросы;

1. Что такое csvsql, как она работает?

2. Можно ли объединить таблицы в базе данных, как это сделать?
3. Какой метод Pandas позволяет добавлять информацию в базу данных?
4. Что делает метод fetchall()?
5. Почему для обработки данных используется библиотека Pandas, можно ли выполнить такую же обработку другим способом?

1.4 Лабораторная работа «Описательная статистика в Python»

Цель работы

Познакомиться со средствами статистической обработки данных в библиотеке Pandas, такими как оценка размера данных, вычисление среднего значения и медианного значения, среднеквадратичного отклонения, максимального и минимального значения в наборе данных, разброса значений, а также средствами визуализации информации.

Сценарий

В этой лабораторной работе вы импортируете набор данных в датафрейм Pandas и получите описательную статистику для этих данных. Затем построите график исследуемых данных и поэкспериментируете с параметрами графика. Познакомитесь с построением графиков данных с использованием библиотеки Matplotlib.PyPlot.

Требуемые ресурсы

ПК с доступом к Интернет

Библиотеки Python: Pandas и Matplotlib.PyPlot

Файл данных: rpi_describe.csv

Порядок выполнения работы

Перед началом работы необходимо импортировать используемые модули Python и настроить Jupyter Notebook для отображения вывода Matplotlib. Для обработки будет использоватся библиотека Pandas. Средствами этой же библиотеки будет выполнен импорт данных из CSV-файла в датафрейм.

Файл данных содержит образцы измерения веса для коробок на 20 унций пищевого продукта. Данные используются для проверки точности машин, которые загружают ящики. Импортируйте необходимые для работы модули:

```
In [1]: # Code cell 1
        # pandas pd
        # matplotlib.pyplot as plt

        # solution
        import pandas as pd
        import matplotlib.pyplot as plt
        # matplotlib.use('qt5agg')

        # given
        %matplotlib inline
        # import matplotlib
         # required on some Jupter Notebook installations
```

Импортируйте данные из файла rpi_describe.csv, используя метод Pandas read_csv. В качестве имени переменной для фрейма данных задайте data:

```
In [2]: # Code cell 2
        # import the csv into the dataframe

        # SOLUTION:
        data = pd.read_csv("./Data/rpi_describe.csv")
```

Используя методы Pandas head() и tail(), убедитесь, что файл импортирован правильно:

```
In [3]: # Code cell 3
        # view the contents of the first five rows in the dataframe

        # SOLUTION:
        data.head()
```

Out[3]:

	weight
0	20.440
1	20.244
2	20.549
3	20.755
4	20.740

```
In [4]: # Code cell 4
        # view the contents of the last five rows in the dataframe

        # SOLUTION:
        data.tail()
```

Out[4]:

	weight
9995	20.419
9996	20.554
9997	20.768
9998	20.829
9999	20.667

Из вывода метода tail можно видеть, что в файле содержится 10000 строк данных. Хотя это только один столбец, Pandas очень эффективно обрабатывает этот файл.

Pandas включает в себя ряд методов для статистической обработки наборов данных.

Метод dataframe.describe () отображает основные статистические данные для фрейма данных:

```
In [5]: # Code cell 5
        # use the describe method of the new dataframe to view the table of statistics

        # SOLUTION:
        data.describe()
```

Out[5]:

	weight
count	10000.000000
mean	20.499212
std	0.199874
min	19.752000
25%	20.365000
50%	20.500000
75%	20.635000
max	21.171000

Достаточно часто не требуется иметь записи значений с множеством знаков после запятой. Чтобы работать с округленными значениями, можно добавить новый столбец к данным для округленных значений. В Pandas доступ к столбцам осуществляется по их заголовкам. Для создания нового столбца используется имя нового столбца в кавычках в квадратных скобках в качестве индекса для кадра данных. Метод round() используется для округления значений в столбце:

In [6]:
```
# Code cell 6
# Add a new column to the dataframe and populate it with rounded weights.
# Verify that values were added.

# SOLUTION:
data['rounded'] = data.weight.round(2)
data.head()
```

Out[6]:

	weight	rounded
0	20.440	20.44
1	20.244	20.24
2	20.549	20.55
3	20.755	20.76
4	20.740	20.74

Столбец датафрейма можно заполнить в том числе рассчитанными значениями. Например, было бы интересно видеть размер отклонения от заданной величины в 20 унций для каждого взвешивания. Создайте новый столбец в кадре данных с именем «diff» и заполните этот столбец величиной отклонения, на сколько больше или меньше целевого значения в 20 унций оказался конкретный экземпляр. Используйте округленное значение для расчета.

In [7]:
```
# Code cell 7
# Create the new column named "diff" and fill it with values.
# Check the result.

#SOLUTION:
data['diff'] = data['rounded'] - 20
data.head()
```

Out[7]:

	weight	rounded	diff
0	20.440	20.44	0.44
1	20.244	20.24	0.24
2	20.549	20.55	0.55
3	20.755	20.76	0.76
4	20.740	20.74	0.74

Иногда нам требуется создать отчеты, содержащие результаты статистической обработки в текстовом виде. Создайте переменные для хранения значений описательной статистики, а затем создайте текстовые строки, использующие эти значения.

In [8]:
```
# Code cell 8
# Create a variables to hold values for the dataset rounded column
# count =
# mean =
# median =
# std =
# rng =

# SOLUTION:
count = data['rounded'].count()
mean = data['rounded'].mean()
median = data['rounded'].median()
std = data['rounded'].std()
rng = data['rounded'].max() - data.rounded.min()
```

Чтобы создать текстовые строки, использующие переменные для отчета о статистике для набора данных, используйте строковый метод format (). Формат использует заполнители {}, чтобы указать, куда следует вставлять значения переменных.

Составьте предложения, такие как «Среднее значение распределения ...» для каждой созданной переменной. Для последнего оператора включите значения min (), max () и range в одни и те же предложения, чтобы на практике использовать несколько заполнителей. Вы можете комбинировать и другие переменные в том же предложении.

In [9]:
```
# Code cell 9
# Create variables to hold your statements.
# countstring =
# meanstring =
# stdstring =
# rangestring =

# SOLUTION:
countstring = "There are {} records in the data.".format(count)
meanstring = 'The mean of the distribution is {} and the median is {}.'.format(mean, median)
stdstring = 'The standard deviation of the distribution is {}.'.format(std)
rangestring = 'The minimum value is {}, the maximum value is {}, and the range is {}.'.format(data.rounded.mi
```

Выведите на экран полученные текстовые строки с помощью функции print:

```
In [18]: # Code cell 18
         # Print all of your statements

         # SOLUTION:
         print(countstring)
         print(meanstring)
         print(stdstring)
         print(rangestring)
```

```
There are 10000 records in the data.
The mean of the distribution is 20.499159999999998 and the median is 20.5.
The standard deviation of the distribution is 0.19990740530823192.
The minimum value is 19.75, the maximum value is 21.17, and the range is <class 'range'>.
```

Обратите внимание на вывод значения для стандартного отклонения. Он содержит слишком большое число цифр после запятой. Формат вывода можно отформатировать, чтобы было проще понять результаты. Например, вы можете изменить вывод стандартного отклонения, чтобы отображать только первые 2 цифры после десятичной точки [4, 5].

Как вывести значение оставив только 2 цифры после десятичной точки?

Создайте частотное распределение значений в наборе данных. Создайте новый фрейм данных, который будет содержать значения частоты для конкретных значений веса, используя метод value_counts (). Этот метод создает объект типа «серия данных», а не фрейм данных. Столбец индекса этой серии задается уникальными значениями данных, а для имени столбца автоматически устанавливается значение 0. Чтобы преобразовать серию данных в датафрейм, можно использовать функцию to_frame (). Дополнительно вызов метода reset_index () в результирующем датафрейме преобразует предыдущий столбец данных в новый, для имени которого автоматически устанавливается значение index.

```
In [12]: # Code cell 12
         # Create a variable called 'freq' to hold the weight values and their frequencies

         # Convert the freq object to a data frame. Use to_frame().

         # SOLUTION:
         freq = data['rounded'].value_counts()
         freq = freq.to_frame().reset_index()
```

Используйте функцию head(), чтобы посмотреть содержимое нового фрейма данных. Столбцы во фрейме данных не имеют корректных названий. Переименуйте их в «value» и «freq», используя атрибут columns столбца данных. Пример:

```
In [14]: # Code cell 14
         # Rename the columns in the dataframe. Verify the result.

         # SOLUTION:
         freq.columns = ['value','freq']
         freq.head()
```

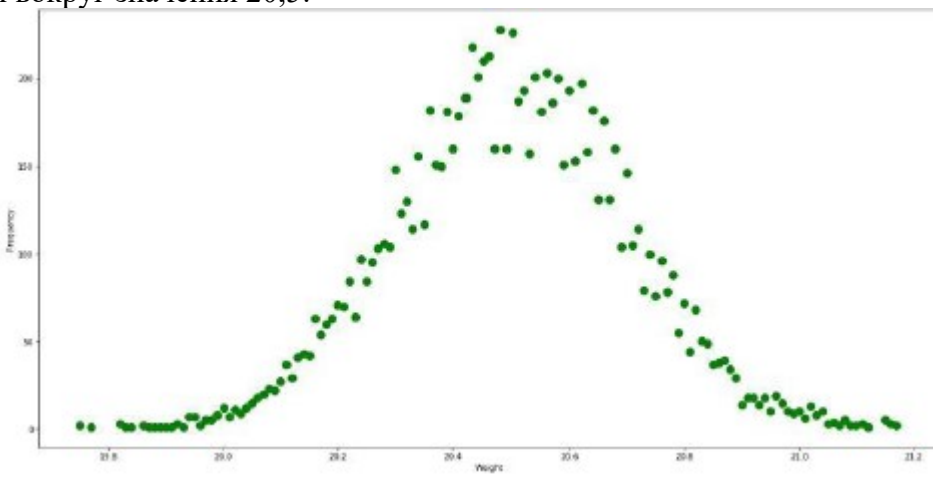
Out[14]:

	value	freq
0	20.48	228
1	20.50	226
2	20.43	218
3	20.46	213
4	20.45	210

Постройте график распределения значений с помощью модуля Matplotlib.PyPlot.

```
In [15]: # Code cell 15

         # Set a size for the graph
         plt.figure(figsize=(20,10))

         # Add axis labels
         plt.ylabel('Frequency')
         plt.xlabel('Weight')

         # Plot the graph using a round symbol "o" of size 10
         plt.plot(freq.value,freq.freq, "o", markersize = 10, color = 'g')
```

Частотный график значений напоминает график распределения Гаусса, сосредоточенный вокруг значения 20,5:



Поэкспериментируйте с отображением данных с различными размерами, маркерами, размерами маркеров и цветами [6, 7].

Контрольные вопросы:

1. Какие статистические значения для датафрейма позволяет получить метод describe ()?
2. Как добавить в датафрейм новый столбец с данными?
3. Что такое частотное распределение данных?
4. Что делает функция to_frame (), почему необходимо преобразовать серию данных в датафрейм?
5. Как при выводе значения переменной в текстовой строке оставить только 2 цифры после десятичной точки, если исходное значение содержит больше цифр?

1.5. Лабораторная работа «Корреляционный анализ в Python»

Цель занятия

Получение практических навыков выполнения корреляционного анализа.

Сценарий

Корреляция является важным статистическим отношением, которое может указывать, связаны ли значения переменных. В этой лабораторной работе вы узнаете, как использовать Python для вычисления корреляции, настроите набор данных. Узнаете, как определить, являются ли переменные в наборе данных связанными. Практически используете Python для вычисления корреляции между двумя наборами переменных.

Требуемые ресурсы

ПК с доступом к Интернет

Библиотеки Python: Pandas, Numpy, Matplotlib, Seaborn

Порядок выполнения работы.

Вы будете использовать набор данных, который содержит выборку персональных данных 40 студентов-правоведов правшей, изучающих английский язык, в большом Юго-западном университете. Испытуемые проходили четыре теста Векслера (словарный запас, сходства, блочный дизайн и завершение изображения), переработанных для взрослых. Исследователи использовали магнитно-резонансную томографию (МРТ) для определения размера мозга испытуемых. Информация о поле и размере тела (рост и вес) также включены. Исследователи не предоставили часть информации из соображений конфиденциальности. Две простые модификации были применены к набору данных:

1. Заменены вопросительные знаки, используемые для представления скрытых данных, описанных выше, строкой «NaN». Замена нужна, потому что Pandas неправильно обрабатывает вопросительные знаки.

2. Заменены все символы табуляции запятыми и набор данных преобразован в набор данных CSV.

Подготовленный набор данных сохранен в файле brainsize.txt

Прежде чем набор данных можно будет использовать, он должен быть загружен в память. Ниже приведен код, показывающий, как это сделать. Первая строка импортирует модули Pandas и определяет pd как дескриптор, который ссылается на модуль. Вторая строка загружает файл CSV набора данных в переменную с именем brainFile.

Третья строка использует read_csv (), метод Pandas, для преобразования набора данных CSV, хранящегося в brainFile, в датафрейм. Фрейм данных затем сохраняется в переменной brainFrame.

```
In [1]: # Code cell 1
        import pandas as pd
        brainFile = './Data/brainsize.txt'
        brainFrame = pd.read_csv(brainFile)
```

Чтобы убедиться, что фрейм данных был правильно загружен и создан, используйте метод head (). Этот метод отображает первые пять записей кадра данных.

```
In [2]: # Code cell 2
        brainFrame.head()
```

Out[2]:

	Gender	FSIQ	VIQ	PIQ	Weight	Height	MRI_Count
0	Female	133	132	124	118.0	64.5	816932
1	Male	140	150	124	NaN	72.5	1001121
2	Male	139	123	150	143.0	73.3	1038437
3	Male	133	129	128	172.0	68.8	965353
4	Female	137	132	134	147.0	65.0	951545

Модуль Pandas включает метод describe(), который выполняет общие статистические вычисления для данного набора данных. Помимо предоставления результатов статистической обработки, describe() также является отличным способом быстрой проверки достоверности значений в кадре данных. Примените метод describe(), для кадра данных brainFrame.

```
In [3]: # Code cell 3
        brainFrame.describe()
```

Out[3]:

	FSIQ	VIQ	PIQ	Weight	Height	MRI_Count
count	40.000000	40.000000	40.00000	38.000000	39.000000	4.000000e+01
mean	113.450000	112.350000	111.02500	151.052632	68.525641	9.087550e+05
std	24.082071	23.616107	22.47105	23.478509	3.994649	7.228205e+04
min	77.000000	71.000000	72.00000	106.000000	62.000000	7.906190e+05
25%	89.750000	90.000000	88.25000	135.250000	66.000000	8.559185e+05
50%	116.500000	113.000000	115.00000	146.500000	68.000000	9.053990e+05
75%	135.500000	129.750000	126.00000	172.000000	70.500000	9.500780e+05
max	144.000000	150.000000	150.00000	192.000000	77.000000	1.079549e+06

Построение графика распределения данных или диаграммы рассеяния (scatterplot graph).

Диаграммы рассеяния удобны при работе с корреляциями, поскольку они позволяют быстро визуально проверить характер взаимосвязи между переменными. В этой лабораторной работе используется коэффициент корреляции Пирсона, который чувствителен только к линейной зависимости между двумя переменными. Существуют и другие методы поиска корреляционной зависимости, но они выходят за рамки этой лабораторной работы.

Прежде чем строить графики, необходимо импортировать модули Numpy и Matplotlib:

```
In [4]: # Code cell 4
        import numpy as np
        import matplotlib.pyplot as plt
```

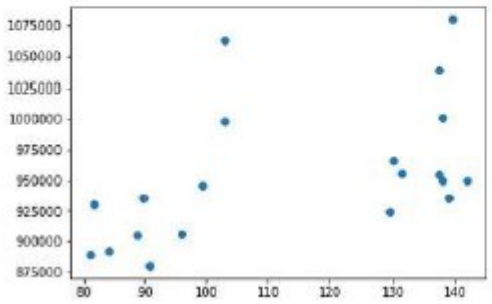
Чтобы гарантировать, что результаты не будут искажены из-за различий в мужском и женском телах, исходный дата-фрейм нужно разделить на два кадра данных: один содержит все мужские записи, а другой - только женские экземпляры.

Создайте два новых фрейма данных menDf и womenDf, каждый из которых содержит соответствующие записи.

```
In [5]: # Code cell 5
        menDf = brainFrame[(brainFrame.Gender == 'Male')]
        womenDf = brainFrame[(brainFrame.Gender == 'Female')]
```

Расположим на графике результаты измерения интеллектуальных способностей от размера мозга. Поскольку набор данных включает три результата различных тестов измерения показателя интеллекта (PIQ, FSIQ и VIQ), а график двумерный, используем прежде метод Pandas mean() для вычисления среднего значения между всеми тремя результатами тестов и сохраним результат в переменной menMeanSmarts.

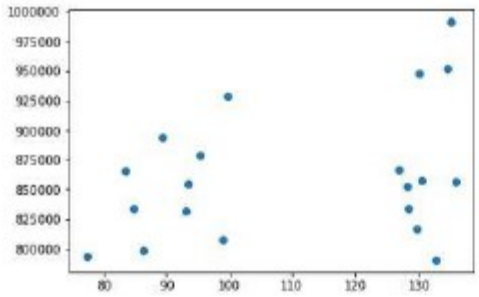
```
In [6]: # Code cell 6
        menMeanSmarts = menDf[["PIQ", "FSIQ", "VIQ"]].mean(axis=1)
        plt.scatter(menMeanSmarts, menDf["MRI_Count"])
        plt.show()
        %matplotlib inline
```



Вторая строка использует метод matplotlib scatter() для создания графика диаграммы рассеяния между переменной menMeanSmarts и MRI_Countattribute. MRI_Count в этом наборе данных можно рассматривать как меру физического размера мозга испытуемых. Третья строка просто отображает график. Четвертая строка используется для обеспечения отображения графика не на отдельном листе, а прямо в этом блокноте.

Аналогичным образом, постройте график диаграммы рассеяния для отфильтрованного только для женщин информационного кадра.

```
In [7]: # Code cell 7
        # SOLUTION:
        womenMeanSmarts = womenDf[["PIQ", "FSIQ", "VIQ"]].mean(axis=1)
        plt.scatter(womenMeanSmarts, womenDf["MRI_Count"])

        plt.show()
        %matplotlib inline
```



Расчет корреляции для дата-фрейма.

Метод Pandas corr() обеспечивает простой способ вычисления корреляции для фрейма данных. Простым вызовом метода для фрейма данных можно получить корреляцию между всеми переменными одновременно.

```
In [8]: # Code cell 8
        brainFrame.corr(method='pearson')
```

Out[8]:

	FSIQ	VIQ	PIQ	Weight	Height	MRI_Count
FSIQ	1.000000	0.946639	0.934125	-0.051483	-0.086002	0.357641
VIQ	0.946639	1.000000	0.778135	-0.076088	-0.071068	0.337478
PIQ	0.934125	0.778135	1.000000	0.002512	-0.076723	0.386817
Weight	-0.051483	-0.076088	0.002512	1.000000	0.699614	0.513378
Height	-0.086002	-0.071068	-0.076723	0.699614	1.000000	0.601712
MRI_Count	0.357641	0.337478	0.386817	0.513378	0.601712	1.000000

Используя тот же метод corr(), легко вычислить корреляцию переменных, содержащихся в кадре данных только для женщин:

```
In [9]: # Code cell 9
        womenDf.corr(method='pearson')
```

Out[9]:

	FSIQ	VIQ	PIQ	Weight	Height	MRI_Count
FSIQ	1.000000	0.955717	0.939382	0.038192	-0.059011	0.325697
VIQ	0.955717	1.000000	0.802652	-0.021889	-0.146453	0.254933
PIQ	0.939382	0.802652	1.000000	0.113901	-0.001242	0.396157
Weight	0.038192	-0.021889	0.113901	1.000000	0.552357	0.446271
Height	-0.059011	-0.146453	-0.001242	0.552357	1.000000	0.174541
MRI_Count	0.325697	0.254933	0.396157	0.446271	0.174541	1.000000

И только для мужчин:

```
In [10]: # Code cell 10
         # SOLUTION:
         menDf.corr(method='pearson')
```

Out[10]:

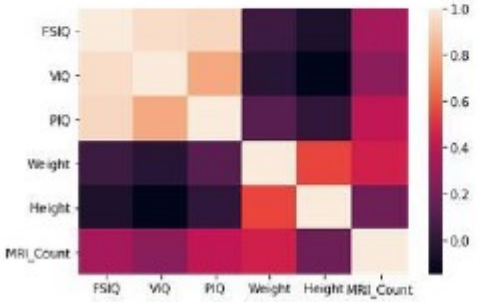
	FSIQ	VIQ	PIQ	Weight	Height	MRI_Count
FSIQ	1.000000	0.944400	0.930694	-0.278140	-0.356110	0.498369
VIQ	0.944400	1.000000	0.786021	-0.350453	-0.355588	0.413105
PIQ	0.930694	0.786021	1.000000	-0.156863	-0.287676	0.568237
Weight	-0.278140	-0.350453	-0.156863	1.000000	0.406542	-0.076875
Height	-0.356110	-0.355588	-0.287676	0.406542	1.000000	0.301543
MRI_Count	0.498369	0.413105	0.568237	-0.076875	0.301543	1.000000

Визуализация помогает увидеть в данных закономерности, которые могут быть не так заметны в табличном представлении. Чтобы было проще визуализировать корреляции данных, можно использовать графики тепловых карт. Графики тепловых карт, основанные на цветных квадратах, могут помочь быстро определить корреляции. Модуль Python с именем Seaborn позволяет очень легко строить графики тепловых карт. Сначала нужно убедиться, что этот модуль установлен в системе, если нет, то потребуется загрузить и установить модуль Seaborn.

```
In [11]: # Code cell 11
         !pip install seaborn

         Requirement already satisfied: seaborn in /srv/pl-app/lib/python3.5/site-packages
         Requirement already satisfied: pandas in /srv/pl-app/lib/python3.5/site-packages (from seaborn)
         Requirement already satisfied: python-dateutil>=2 in /usr/lib/python3/dist-packages (from pandas->seaborn
```

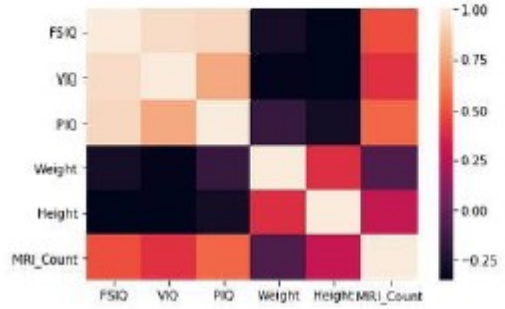
Если модуль Seaborn установлен, то тепловые карты могут быть построены.

```
In [13]: # Code cell 12
         import seaborn as sns

         wcorr = womenDf.corr()
         sns.heatmap(wcorr)
         #plt.savefig('attribute_correlations.png', tight_layout=True)

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x6cd27c10>
```



Аналогичным образом можно построить тепловую карту для информационного кадра только для мужчин.

```
In [14]: # Code cell 14
         mcorr = menDf.corr()
         sns.heatmap(mcorr)
         #plt.savefig('attribute_correlations.png', tight_layout=True)

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x6ccc53d0>
```



Контрольные вопросы.

1. Обратите внимание на диагональ слева направо в таблице корреляции. Почему диагональ заполнена единицами, это совпадение?
2. Можно видеть, также что значения в таблице являются зеркальными, значения ниже единичной диагонали имеют зеркальный аналог выше этой диагонали. Почему?
3. Многие переменные пары представляют корреляцию, близкую к нулю. Что это обозначает?

4. Зачем было разделять данные по полу?

5. Какие переменные имеют более сильную корреляцию с размером мозга (MRI_Count)? Это ожидаемо?

1.6 Лаюораторная работа «Измерение и анализ Интернет-соединения визуализация результатов»

Цель занятия

Развитие навыков предварительной обработки, очистки и анализа данных, методов визуализации результатов.

Сценарий

В этой лабораторной работе вы узнаете, как использовать библиотеку Pandas для выполнения предварительных шагов, необходимых перед выполнением любого анализа данных. Это включает удаление недостающих значений, изменение формата данных и выполнение предварительного статистического анализа. После очистки данных вы будете использовать Matplotlib для исследования и визуализации данных.

Требуемые ресурсы

ПК с доступом к Интернет

Библиотеки Python: datetime, csv, subprocess, pandas, numpy

Файл данных: data_long.csv.

Порядок выполнения работы.

Иногда говорят, что специалисты по машинному обучению тратят 80% своего времени на очистку данных. Хотя это утверждение, возможно, является преувеличением, верно, что очистка данных является фундаментальной задачей, необходимой для обеспечения успеха проекта машинного обучения. Как и во многих других сферах деятельности, действует закон «мусор на входе, мусор на выходе». Провести полноценный анализ грязных данных просто невозможно.

Часть 1 Очистка, подготовка и анализ данных

Очистка данных.

Загрузите данные из файла rpi_data_compact.csv. Этот файл содержит измерения скорости интернета, полученные во время лабораторной работы Internet_Speed_Data_Acquisition. В частности, основное внимание уделяется трем параметрам: время пинга (мс), скорость загрузки (Мбит/с) и скорость загрузки (Мбит/с).

In [1]:
```
# Code Cell 1
import pandas as pd
import numpy as np
```

In [2]:
```
# Code Cell 2
# Import data from csv file, and visualize the first rows
# SOLUTION:
df_compact = pd.read_csv('./Data/rpi_data_compact.csv')
df_compact.head(3)
```

Out[2]:

	Unnamed: 0	Ping (ms)	Download (Mbit/s)	Upload (Mbit/s)	Date	Time
0	0	26.992	91.80	14.31	2016-11-24	13:36:25
1	1	24.532	88.19	14.12	2016-11-24	13:36:55
2	2	20.225	59.86	14.11	2016-11-24	13:37:25

Удалите ненужные столбцы.

Как вы могли заметить, фрейм данных df_compact имеет дополнительный столбец. Используйте команду df.drop, чтобы удалить этот столбец. Обратитесь за помощью в лабораторию Internet_Speed_Data_Acquisition.

```
In [3]: # Code Cell 3
        # Remove extra index columns
        # SOLUTION:
        df_compact.drop(['Unnamed: 0'], axis -1, inplace-True)
        df_compact.head(3)
```

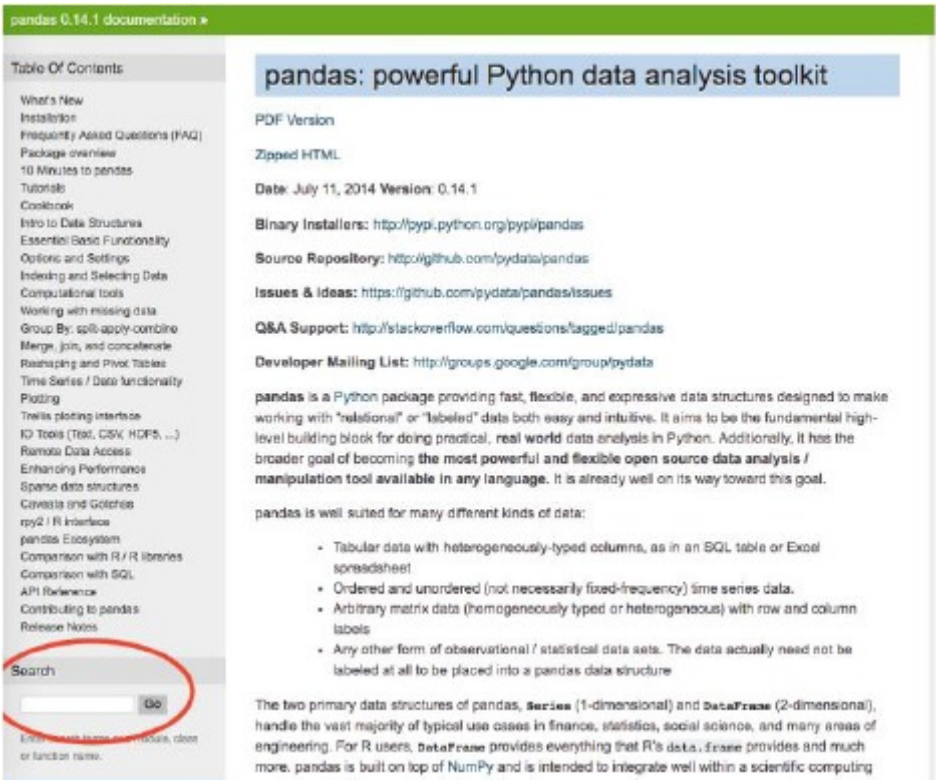
Out[3]:

	Ping (ms)	Download (Mbit/s)	Upload (Mbit/s)	Date	Time
0	26.992	91.80	14.31	2016-11-24	13:36:25
1	24.532	88.19	14.12	2016-11-24	13:36:55
2	20.225	59.86	14.11	2016-11-24	13:37:25

Удалите NaN.

Распространенной проблемой, влияющей на качество данных, является наличие значений NaN. Это может привести к тому, что функции анализа данных внезапно прервут вычисление, выдадут ошибку или выдадут неверные результаты. Обычно значения NaN представляют собой часть информации, которая должна содержаться в наборе данных, но отсутствует в нем. В этом примере значения NaN в df_compact могут представлять измерения в местах падения Интернет-соединения или запросы, на которые сервер Speedtest.net не смог ответить. Приведенный ниже код определяет, сколько значений NaN содержится в нашем фрейме данных.

Сначала перейдите на http://pandas.pydata.org/pandas-docs/version/0.14.1/index.html и найдите isnull в поле поиска.



Документация по функции isnull немного сложна для понимания. К счастью, интерактивный интерфейс Jupyter позволяет нам вызывать эту функцию и быстро проверять ее вывод.

In [4]:
```
# Code Cell 4
NaNs_in_df = df_compact.isnull()
print(type(NaNs_in_df))
NaNs_in_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[4]:

	Ping (ms)	Download (Mbit/s)	Upload (Mbit/s)	Date	Time
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False

Результатом функции isnull является новый фрейм данных, который содержит True или False, если соответствующий элемент DataFrame равен NaN или нет. Использование функции sum в этом DataFrame автоматически преобразует значения True в 1 и False в 0.

In [5]:
```
# Code Cell 5
NaNs_per_column = NaNs_in_df.sum()
print(type(NaNs_per_column))
NaNs_per_column.head()
```

```
<class 'pandas.core.series.Series'>
```

Out[5]:
```
Ping (ms)           1
Download (Mbit/s)   1
Upload (Mbit/s)     7
Date                7
Time                7
dtype: int64
```

Результирующий NaN_per_column представляет собой объект серии pandas, который можно рассматривать как отдельный столбец DataFrame (DataFrame на самом деле является нарицательным для Series, где ключи - это имена столбцов). Объект Series содержит почти все функции DataFrame. Используйте функцию суммы в серии Nan_per_column и отобразите результат.

In [ ]:
```
# Code Cell 6
NaNs_total = NaNs_per_column.sum()
NaNs_total
```

Можно объединить обе операции в одну строку следующим образом:

In [ ]:
```
# Code Cell 7
df_compact.isnull().sum().sum()
```

Вычислите количество пропущенных значений в процентах от всех элементов в фрейме данных (округлите результат до второго десятичного знака, используя numpy.round. Справку по этой функции см. по ссылке https://docs.scipy.org/doc/numpy/reference/generated/numpy.round_.html).

Используйте функцию pandas dropna(), чтобы удалить значения NaN из df_compact

In [ ]:
```
# Code Cell 8
NaNs_pct = np.round(df_compact.isnull().sum().sum()/float(len(df_compact)*len(df_compact.columns))*100, decimals = 4)
print('The DataFrame contains : {} NaNs, equal to {} of the measurements'.format(NaNs_total, NaNs_pct)) #EOL : moved parenthesis
```

Функция dropna, если она вызывается с параметрами по умолчанию, удаляет все строки DataFrame, если любое из значений в строке равно «NaN».

```
In [ ]: # Code Cell 9
        # Remove NaN values
        df_compact_clean = df_compact.dropna()
```

Сравните длину датафрейма до и после использования dropna. Вы заметили что-то странное? Что именно и как это объяснить?

```
In [ ]: # Code Cell 10
```

Покажите, как вы выводили длину фрейма и добавьте ваше пояснение по результатам.

Изменение формата значений.

Столбцы для ping, upload и download в фрейме данных df_compact содержат числовые значения. Поэтому разумно ожидать, что все они имеют один тип данных, например float64. Однако это не так, что легко проверить с помощью dtypes:

```
In [ ]: # Code Cell 11
        # Ping and Download are not floats
        df_compact.dtypes
```

Используйте функцию Python float() для преобразования значения записей в числовой формат с плавающей запятой. Вот, например, как можно преобразовать строковое значение:

```
In [ ]: # Code Cell 12
        str_val = '10.56'
        float_val = float(str_val)
        print(str_val, type(str_val), float_val, type(float_val))
```

Подсказка: используйте apply и lambda. Загляните в предыдущие лабораторные работы для справки.

```
In [ ]: # Code Cell 14
        # this disables a notebook warning that is not relevant for our use case
        pd.options.mode.chained_assignment = None

        # Convert Ping and Download to float
        #df_compact_clean['Ping (ms)_float'] = ...
        #df_compact_clean['Download (Mbit/s)_float'] = ...

        #Check that the types have been successfully converted
        # ...
```

Удалите исходные столбцы Ping (ms) и Download (Mbit/s) и переименуйте новые Ping (ms) _float и Download (Mbit/s) _float в Ping (ms) и Download (Mbit/s). Используйте df.drop и df.rename.

```
In [ ]: # Code Cell 15
        # Remove the original 'Ping (ms)' and 'Download (Mbit/s)' columns


        # Rename the new 'Ping (ms) float' and 'Download (Mbit/s) float ' to Ping (ms)' and 'Download (Mbit/s)
```

Проверьте полученный результат, так как изменялся формат, а не значения, данные должны выглядеть по-прежнему:

```
In [ ]: # Code Cell 16
        df_compact_clean.head()
```

Обратите внимание, что последовательность столбцов в датафрейме выглядит нелогично, первым идет столбец скорости выгрузки. Перед сохранением датафрейма имеет смысл расставить столбцы в нем в правильном порядке. Этого можно добиться с помощью функции переиндексации:

```
In [ ]: # Code Cell 17
df_compact_clean = df_compact_clean.reindex(columns = ['Date', 'Time', 'Ping (ms)','Download (Mbit/s)','Upload (Mbit/s)']);
df_compact_clean.head()
```

Теперь, когда набор данных в датафрейме окончательно оформлен, сохраните его в файле csv и с заданным вами именем:

```
In [ ]: # Code Cell 18
# Let's save the new cleaned dataframe to a csv
df_compact_clean.to_csv('./rpi_data_processed.csv', index=False)

df_clean = df_compact_clean
```

Выполнение базовых статистических операций над датафреймом.

Новые данные требуют не только очистки и подготовки, их нужно «покрутить и пощупать». Когда вы начинаете проект по анализу данных, стоит потратить усилия на изучение и вычисление некоторых основных статистических свойств имеющегося набора (наборов) данных. Лучше понять свойства имеющихся данных можно рассчитав базовые статистические параметры, такие как средние значения, стандартные отклонения и корреляции.

а) Рассчитайте среднее значение и стандартное отклонение с помощью Pandas.

Среднее значение и стандартное отклонение всех столбцов DataFrame можно вычислить с помощью функций mean() и std(). Найдите их в документации библиотеки pandas и примените к датафрейму df_clean. Выведите результаты как величина = среднее_значение ± стандартное_отклонение. Не забудьте указать единицы измерения, связанные с каждой величиной.

```
In [ ]: # Code Cell 19
# Compute mean and std for all the columns of df_compact
# SOLUTION:
# means = ...
# stands = ...

# Place mean and std for each column in a tuple
stats_ping = (means['Ping (ms)'], stands['Ping (ms)'])
stats_download = (means['Download (Mbit/s)'], stands['Download (Mbit/s)'])
stats_upload = (means['Upload (Mbit/s)'], stands['Upload (Mbit/s)'])

# Print the mean value ± the standard deviation, including measuring units
print('Average ping time: {} ± {} ms'.format(stats_ping[0],stats_ping[1]))
print('Average download speed: {} ± {} Mbit/s'.format(*stats_download))
print('Average upload speed: {} ± {} Mbit/s'.format(*stats_upload))
```

б) Рассчитайте минимальный и максимальный диапазон значений с помощью Pandas.

Найдите в документации библиотеки pandas, как вычислить минимальное и максимальное значения для всех столбцов в DataFrame.

```
In [ ]: # Code Cell 23
# Compute max and min for all the columns of df_compact
mins = df_clean.min()
maxs = df_clean.max()

# Place mean and std for each column in a tuple
mima_ping = (mins['Ping (ms)'], maxs['Ping (ms)'])
mima_download = (mins['Download (Mbit/s)'], maxs['Download (Mbit/s)'])
mima_upload = (mins['Upload (Mbit/s)'], maxs['Upload (Mbit/s)'])

# Print the mean and max values, including measuring units
print('Min ping time: {} ms. Max ping time: {} ms'.format(*mima_ping))
print('Min download speed: {} Mbit/s. Max download speed: {} Mbit/s'.format(*mima_download))
print('Min upload speed: {} Mbit/s. Max upload speed: {} Mbit/s'.format(*mima_upload))
```

в) Используйте функцию pandas describe().

Обратите внимание, какой объем информации по всему датафрейму вы можете получить с помощью этой функции и всего одной строчки кода!

```
In [ ]: # Code Cell 24
        df_clean.describe()
```

г) Использование argmin, argmax и iloc.

Представьте, что вы хотели бы иметь программный скрипт, который автоматически отправляет отчеты о состоянии вашего интернет-соединения по электронной почте. Отчеты должны включать дату и время, соответствующие минимальной скорости интернета. Временная информация позволит интернет-провайдеру точно определить причину наблюдаемого вами медленного соединения. Используя функции pandas argmin и argmax, найдите дату и время, соответствующие самому длинному и самому короткому времени ping, самой низкой и самой высокой скорости загрузки, а также самой низкой и самой высокой скорости выгрузки.

```
In [ ]: # Code Cell 25
        # Find the min and max ping time
        argmin_ping = df_clean['Ping (ms)'].argmin()
        argmax_ping = df_clean['Ping (ms)'].argmax()

        # Find the min and max download speed
        argmin_download = df_clean['Download (Mbit/s)'].argmin()
        argmax_download = df_clean['Download (Mbit/s)'].argmax()

        # Find the min and max upload speed
        argmin_upload = df_clean['Upload (Mbit/s)'].argmin()
        argmax_upload = df_clean['Upload (Mbit/s)'].argmax()
```

Функции argmin и argmax возвращают индекс относительно строк датафрейма. Чтобы получить доступ к определенной строке с помощью этого индекса, используйте iloc.

```
In [ ]: # Code Cell 26
        # Create a small DataFrame and access its rows using iloc

        # A pandas DataFrame can be initialized passing a dict as a parameter to the constructor pd.DataFrame().
        # The key will represent the column, the values the rows.
        df = pd.DataFrame({'field_1': [0,1], 'field_2': [0,2]})
        df.head()
```

```
In [ ]: # Code Cell 27
        # To access the field_1 of the first row using iloc()
        df.iloc[1]['field_1']
```

Используйте индексы, вычисленные с помощью argmax и argmin в сочетании с iloc, чтобы визуализировать дату и время максимального/минимального Ping, Download и Upload.

```
In [ ]: # Code Cell 28
        #Print the corresponding Date and Time
        #print('Ping measure reached minimum on {} at {}'.format(df_clean.loc[...,
        #                                                        df_clean.loc[...))

        #print('Download measure reached minimum on {} at {}'.format(...

        #print('Upload measure reached minimum on ...

        #print('Ping measure reached maximum on ...

        #print('Download measure reached maximum on ...

        #print('Upload measure reached maximum on ...
```

д) Корреляционный анализ.

Полезно проанализировать, имеет ли скорость загрузки тенденцию к увеличению или уменьшению вместе со скоростью загрузки. Причина этого заключается в том, что использование сети и технические проблемы должны одинаково влиять на загрузку и загрузку. В этом случае скорости загрузки и выгрузки будут называться положительно коррелированными. Это означает, что более быстрая загрузка и выгрузка обычно происходят одновременно. Это, в принципе, общая тенденция, однако случаи быстрой загрузки с медленной выгрузкой также возможны. С другой стороны, если есть ограниченный предел полосы пропускания линии связи, то при увеличении скорости загрузки все меньшая часть доступной полосы пропускания будет оставаться для выгрузки и скорость выгрузки будет снижаться. Скачивание и выгрузка будут соревноваться за доступную полосу пропускания и будут держать друг друга «под контролем». В этом случае скорости загрузки и выгрузки будут отрицательно коррелированными. Это означает, что загрузка обычно будет быстрее, чем медленнее загрузка, и наоборот. Как и в первом случае, это может быть общим наблюдаемым трендом, в то время как в отдельные моменты может быть одновременно хорошая скорость и загрузки, и выгрузки. Для полноты картины, время пинга может быть положительно или отрицательно коррелировано либо с загрузкой, либо с загрузкой. Тогда естественно представить себе таблицу, в которой каждая величина сравнивается со всеми остальными. Такие таблицы являются хорошо известными математическими объектами и называются корреляционными матрицами.

Используйте функцию pandas corr, чтобы получить корреляционную матрицу ping, upload и download. Сохраните результат в переменной df_corr.

```
In [ ]: # Code Cell 29
        # Are these variables correlated?
        df_corr = df_clean.corr()
        df_corr
```

```
In [ ]: # Code Cell 30
        corr = df_corr.values
        print('Correlation coefficient between ping and download: {}'.format(corr[0, 1]))
        print('Correlation coefficient between ping and upload: {}'.format(corr[0, 2]))
        print('Correlation coefficient between upload and download: {}'.format(corr[2, 1]))
```

Идеальная положительная корреляция дает значение +1, тогда как идеальная отрицательная корреляция дает значение -1. В том случае, когда сравниваемые величины оказываются не зависящими друг от друга, результат расчета функции корреляции будет близким к нулю.

Часть 2 Визуализация данных

Как говорит пословица: «Картинка стоит тысячи слов». Информативные, содержательные и интуитивно понятные графики играют решающую роль в исследовании данных. Визуализация полезна и на начальных этапах проекта и на всех последующих. Графики - отличный способ представить результаты и подкрепить сформулированные выводы вашей работы перед аудиторией. Python имеет специальную библиотеку для построения графиков с широкими возможностями, она называется Matplotlib. В качестве дополнительного учебного ресурса, безусловно, стоит взглянуть на официальную документацию Matplotlib и, в частности, на многочисленные примеры.

а) Импорт библиотеки Matplotlib.

Прежде чем использовать возможности библиотеки Matplotlib, ее нужно подключить к коду:

```
In [ ]: # Code Cell 31
        import matplotlib.pyplot as plt
        # The following allows your Jupyter notebook to create plots inside a cell
        %matplotlib inline
```

В интерактивном Python есть набор «магических» команд, признаком которых является символ %. Эти дополнительные команды облегчают процесс разработки и расширяют

возможности программиста. Например, командой %time, введенной в начале строки, можно определить время выполнения этой программной строки. Полный перечень магических команд можно получить командой %lsmagic. Команда %matplotlib inline разрешает вывод графика прямо в текущем блокноте Jupyter, по умолчанию график выводится в отдельном окне.

б) Постройте график статистики для интернета-соединения.

Используйте первую лабораторную работу для справки. Создайте график, на который будут выводиться три параметра интернет-соединения:

1. Пинг (мс) как функция времени,
2. Скорость загрузки (Мбит/с) как функция времени,
3. Скорость выгрузки (Мбит/с) как функция времени.

Используйте функцию legend(), чтобы добавить легенду к вашему графику.

In [ ]:
```
# Code Cell 32
# Initialise figure
fig, ax = plt.subplots(figsize=(10, 5))

# Create x-axis
t = pd.to_datetime(df_clean['Time'])


# Plot three curves of different colors
ax.plot(t, df_clean['Ping (ms)'], label='Ping (ms)')
#ax.plot(...
#ax.plot(...

# Insert legend
ax.legend()
plt.show()
```

б) Измените стиль линий.

Поскольку измерения пинга включают большие и резкие вариации, их лучше визуализировать с помощью точек. В команде ax.plot (...) для отображения данных ping укажите, что эти измерения представлены в виде точек.

In [ ]:
```
# Code Cell 33
# Initialise figure
fig, ax = plt.subplots(figsize=(10, 5))

# Plot three curves. Ping data
# is visualized using dots
t = pd.to_datetime(df_clean['Time'])

#ax.plot(...
#ax.plot(...
#ax.plot(...

# Insert legend
```

в) Добавьте метки осей.

График без меток осей и, возможно, названия, трудно воспринимать, поскольку неизвестно что на нем изображено. Сделайте приведенный выше график совместимым со стандартной практикой, добавив метки осей и заголовок. Укажите размер шрифта около 16, чтобы заголовок и метки осей выглядели красиво и четко.

```
In [ ]: # Code Cell 35
        # Initialise figure
        fig, ax = plt.subplots(figsize=(10, 5))

        # Plot three curves
        t = pd.to_datetime(df_clean['Time'])
        ax.plot(t, df_clean['Ping (ms)'], 'o', label='Ping (ms)')
        ax.plot(t, df_clean['Upload (Mbit/s)'], label='Upload (Mbit/s)')
        ax.plot(t, df_clean['Download (Mbit/s)'], label='Download (Mbit/s)')

        # Insert legend
        ax.legend()

        # Add axis labels and title
        #ax.set_xlabel(...
        #ax.set_ylabel(...
        #ax.set_title(...

        # Change tick size
        ax.tick_params(labelsize=14)
```

г) Измените стиль графика.

Используйте контекст стиля «fivethirtyeight», чтобы сделать предыдущий график более привлекательным. Для этого добавьте в код строку с оператором with перед вызовом функций Matplotlib.

```
In [ ]: # Code Cell 36
        # Use a style context
        #with ...

            # Initialise figure
            #fig, ax =

            # Plot ping as a function of time

            # Add axis labels and title

            # Change tick size
```

д) Представьте данные в виде гистограммы.

Гистограмма — это графическое представление частоты значений числовых данных. Изучите код ниже. Дополнительным уровнем сложности является использование подзаголовков для отображения гистограмм бок о бок.

```
In [ ]: # Code Cell 37
        with plt.style.context('fivethirtyeight'):

            nbins = 100
            # Initialize figure
            fig, ax = plt.subplots(2, 2, figsize=(10, 10))
            ax[0][0].hist(df_clean['Ping (ms)'], nbins)
            ax[0][0].set_xlabel('Ping (ms)', fontsize=16)
            ax[0][0].tick_params(labelsize=14)
            ax[0][1].hist(df_clean['Upload (Mbit/s)'], nbins)
            ax[0][1].set_xlabel('Upload (Mbit/s)', fontsize=16)
            ax[0][1].tick_params(labelsize=14)
            ax[1][0].hist(df_clean['Download (Mbit/s)'], nbins)
            ax[1][0].set_xlabel('Download (Mbit/s)', fontsize=16)
            ax[1][0].tick_params(labelsize=14)
            ax[1][1].set_visible(False)
```

Контрольные вопросы:

1. Какие типы были у записей в df_compact до преобразования?
2. В какое время наблюдалась максимальная и минимальная скорость загрузки, выгрузки данных, задержки отправки эхо-запросов командой ping?

3. Каким получился результат корреляционного анализа?

4. Каков точный формат задания подписей к осям графика, наименования графика, добавления легенды для выводимых на график величин?

5. Какие еще библиотеки применяются для визуализации данных Python, назовите и кратко опишите пять таких библиотек.

1.7 Лабораторная работа «Простая линейная регрессия в Python»

Цель занятия

В этой лабораторной работе вы познакомитесь с концепциями простой линейной регрессии и работы с предоставленными данными для составления прогнозов.

Сценарий

В статистике линейная регрессия - это способ моделирования отношения между зависимой переменной y и независимой переменной x. В этой лабораторной работе вы проанализируете данные о продажах в районе и выполните простую линейную регрессию, чтобы спрогнозировать годовые чистые продажи на основе количества магазинов в районе.

Требуемые ресурсы

ПК с доступом к Интернет

Библиотеки Python: pandas, numpy, scipy, and matplotlib

Файл данных: stores-dist.csv

Порядок выполнения работы

1. Импорт библиотек и загрузка данных

Необходимо импортировать следующие библиотеки:

- matplotlib.pyplot as plt
- numpy as np
- pandas as pd

```
In [ ]: # Code Cell 1
```

Загрузите данные из файла stores-dist.csv и убедитесь, что они загружены успешно.

```
In [ ]: # Code Cell 2

        # Import the file, stores-dist.csv
        salesDist = pd.read_csv('./Data/stores-dist.csv')

        # Verify the imported data
        salesDist.head()
```

Переименуйте заголовки столбцов annual net sales и number of stores in district, чтобы упростить обработку данных.

- annual net sales переименуйте в sales
- number of stores in district переименуйте в stores

```
In [ ]: # Code Cell 3
        # The district column has no relevance at this time, so it can be dropped.
        salesDist = salesDist.rename(columns={'annual net sales':'sales','number of stores in district':'stores'})
        salesDist.head()
```

2. Предварительный анализ данных и выведение их на график.

Определение корреляции.

Хотя корреляционный анализ не устанавливает причинно-следственные связи, он, тем не менее, показывает наличие взаимозависимостей одних данных от других. Это может быть полезно при построении линейной регрессии.

```
In [ ]: # Code Cell 4
        # Check correlation of data prior to doing the analysis
        # # Hint: check lab 3.1.5.5
```

Из коэффициента корреляции видно, что столбец района, district имеет низкую корреляцию с годовым чистым объемом продаж и количеством магазинов в районе. Таким образом, этот столбец не требуется в рамках регрессионного анализа и его можно удалить из фрейма данных.

```
In [ ]: # Code Cell 5
        # The district column has no relevance at this time, so it can be dropped.
        #sales = salesDist.drop(...)

        sales.head()
```

Построение графика данных

Выберем число магазинов как независимую переменную x, а число продаж как зависимую переменную y.

```
In [ ]: # Code Cell 6
        # dependent variable for y axis
        y = sales['sales']
        # independent variable for x axis
        x = sales.stores
```

```
In [ ]: # Code Cell 7
        # Display the plot inline
        %matplotlib inline

        # Increase the size of the plot
        plt.figure(figsize=(20,10))

        # Create a scatter plot: Number of stores in the District vs. Annual Net Sales
        plt.plot(x,y, 'o', markersize = 15)

        # Add axis labels and increase the font size
        plt.ylabel('Annual Net Sales', fontsize = 30)
        plt.xlabel('Number of Stores in the District', fontsize = 30)

        # Increase the font size on the ticks on the x and y axis
        plt.xticks(fontsize = 20)
        plt.yticks(fontsize = 20)

        # Display the scatter plot
        plt.show()
```

3. Выполнение простой линейной регрессии.

Для создания линии регрессии для наших данных будем использовать библиотеку numpy. Эта библиотека для расчета различных математических функций, в частности массивов, позволяет выполнять расчет очень быстро, гораздо быстрее чем это можно сделать средствами самого Python. Много информации о том какие функции доступны и какие особенности применения NumPy существуют можно найти на официальном сайте https://numpy.org. Кроме линии регрессии мы расчитаем центральную точку, “центр тяжести” нашего набора данных, ее называют центроид. Центроид - это среднее значение для набора данных. Сгенерированная линия простой линейной регрессии проходит через центроид.

Расчет угла наклона и точки пересечения оси зависимой переменной y выполняется в одно действие с помощью функции NumPy polyfit():

```
In [ ]: # Code Cell 8
        # Use numpy polyfit for linear regression to fit the data
        # Generate the slope of the line (m)
        # Generate the y-intercept (b)
        m, b = np.polyfit(x,y,1)
        print ('The slope of line is {:.2f}.'.format(m))
        print ('The y-intercept is {:.2f}.'.format(b))
        print ('The best fit simple linear regression line is {:.2f}x + {:.2f}.'.format(m,b))
```

Центроид набора данных вычисляется с использованием функции mean().

```
In [ ]: # Code Cell 9
        # y coordinate for centroid
        y_mean = y.mean()
        # x coordinate for centroid
        x_mean = x.mean()
        print ('The centroid for this dataset is x = {:.2f} and y = {:.2f}.'.format(x_mean, y_mean))
```

Теперь мы можем вывести на один общий график картину данных и рассчитанную нами линию регрессии, а также центроид:

```
In [ ]: # Code Cell 10
        # Create the plot inline
        %matplotlib inline

        # Enlarge the plot size
        plt.figure(figsize=(20,10))

        # Plot the scatter plot of the data set
        plt.plot(x,y, 'o', markersize = 14, label = "Annual Net Sales")

        # Plot the centroid point
        plt.plot(x_mean,y_mean, '*', markersize = 30, color = "r")

        # Plot the linear regression line
        plt.plot(x, m*x + b, '-', label = 'Simple Linear Regression Line', linewidth = 4)

        # Create the x and y axis labels
        plt.ylabel('Annual Net Sales', fontsize = 30)
        plt.xlabel('Number of Stores in District', fontsize = 30)

        # Enlarge x and y tick marks
        plt.xticks(fontsize = 20)
        plt.yticks(fontsize = 20)

        # Point out the centroid point in the plot
        plt.annotate('Centroid', xy=(x_mean-0.1, y_mean-5), xytext=(x_mean-3, y_mean-20), arrowprops=dict(facecolor='black',

        # Create legend
        plt.legend(loc = 'upper right', fontsize = 20)
```

Если по каким-то причинам предложенные выше параметры приводят к тому, что размеры графика не укладываются в окно ноутбука Юпитер, поправьте соответствующие значения.

Полученная линия линейной регрессии, является функцией для зависимого параметра, и мы можем с ее помощью предсказать годовые чистые продажи на основе количества магазинов в районе. Давайте создадим специальную функцию, реализующую такое предсказание:

```
In [ ]: # Code Cell 11
        # Function to predict the net sales from the regression line
        def predict(query):
            if query >= 1:
                predict = m * query + b
                return predict
            else:
                print ("You must have at least 1 store in the district to predict the annual net sales.")
```

Проверим эту функцию «в деле». Что она предсказывает если мы указываем 4 магазина в районе?

Контрольные вопросы:

1. Судя по данным коэффициента корреляции, какой тип корреляции наблюдается между годовой выручкой от продаж и количеством магазинов в районе?
2. Линейную регрессию в Python можно выполнить другими способами, назовите несколько?
3. Как вы думаете, является ли линейная регрессия методом машинного обучения?

1.8. Лабораторная работа «Классификация с помощью дерева решений»

Цель занятия

В этой лабораторной работе вы будете использовать модель классификатора дерева решений, чтобы определить какими особыми свойствами обладали те, кто выжил в катастрофе круизного лайнера «Титаник». Это одна из классических задач [10].

Сценарий

В этой лабораторной работе вы создадите классификатор дерева решений, который будет работать с набором данных, содержащим подробную информацию о более чем 1300 пассажиров, которые находились на борту пассажирского лайнера «Титаник» во время его печально известного первого рейса.

Требуемые ресурсы

ПК с доступом к Интернет

Библиотеки Python: pandas, sklearn и IPython.display. Дополнительное приложение: Graphviz Файлы данных: titanic-train.csv, titanic-test.csv, titanic_all.csv

Порядок выполнения работы

Часть 1. Создание классификатора дерева решений

В этой части лабораторной работы вы создадите классификатор дерева решений, который будет учиться на выбранном для обучения наборе данных. Набор данных содержит имена и демографические данные каждого пассажира. Кроме того, включены подробные сведения о поездке пассажиров. Из этих данных мы можем построить дерево решений, которое иллюстрирует факторы, которые способствовали живучести или ее отсутствию во время путешествия. Наборы данных содержат следующие переменные:

Variable	Description
1. PassengerID	Unique identifier for each passenger
2. Survival	Did the passenger survive? (0 = No; 1 = Yes)
3. Pclass	Passenger ticket class. (1 = 1st; 2 = 2nd; 3 = 3rd)
4. Name	Name of the passenger. (last name, first name)
5. Gender	Male or female
6. Age	Age in years. Mostly integers with float values for children under one year.
7. SibSp	Number of siblings or spouse onboard.
8. Parch	Number of parents or children onboard.
9. Ticket	Ticket number
10. Fare	Amount paid for fare in pre-1970 British Pounds
11. Cabin	Cabin number
12. Embarked	Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

Глядя на эти данные, как вы сами думаете, как можно было бы сформулировать вопросы о факторах, которые способствовали выживанию или гибели пассажиров в катастрофе «Титаник»? Например, какой возраст или пол может быть важным фактором?

Создание фрейма данных

а) Импортируем библиотеку pandas

Создадим фрейм данных из набора данных для обучения, который хранится в файле titanic-train.csv.

```
In [29]: #Code cell 1
         #import pandas
         import pandas as pd

         #create a pandas dataframe called "training" from the titanic-train.csv file
         training = pd.read_csv("./Data/titanic-train.csv")
```

б) Убедитесь, что импорт успешен и просмотрите данные.

```
In [30]: #Code cell 2
         #verify the contents of the training dataframe using the pandas info() method.
         # SOLUTION:
         training.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 915 entries, 0 to 914
Data columns (total 12 columns):
PassengerId    915 non-null int64
Survived       915 non-null int64
Pclass         915 non-null int64
Name           915 non-null object
Gender         915 non-null object
Age            738 non-null float64
SibSp          915 non-null int64
Parch          915 non-null int64
Ticket         915 non-null object
Fare           915 non-null float64
Cabin          202 non-null object
Embarked       914 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 85.9+ KB
```

Судя по выводу, полученному при применении метода info, как вы думаете, есть ли в наборе данных недостающие значения?

Выведите несколько строк фрейма данных чтобы посмотреть в каком виде они представлены:

```
In [31]: #Code cell 3
         #view the first few rows of the data
         # SOLUTION:
         training.head()
```

Out[31]:

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	1	Davidson, Mr. Thornton	male	31.0	1	0	F.C. 12750	52.0000	B71	S
1	2	0	3	Asim, Mr. Adola	male	35.0	0	0	SOTON/O.Q. 3101310	7.0500	NaN	S
2	3	0	3	Nankoff, Mr. Minko	male	NaN	0	0	349218	7.8958	NaN	S
3	4	0	1	Thayer, Mr. John Borland	male	49.0	1	1	17421	110.8833	C68	C
4	5	0	3	Strandberg, Miss. Ida Sofia	female	22.0	0	0	7553	9.8375	NaN	S

Подготовка данных для модели дерева решений.

а) Замените строковые данные числовыми метками

Для анализа мы будем использовать специальное расширение библиотеки научных и инженерных расчётов SciPy, scikit-learn для реализации алгоритмов машинного обучения. В том числе, здесь есть возможность создания деревьев решений. Scikit-Learn впервые был предложен David Cournapeau (Дэвид Курнапо) в 2007 году в рамках проекта Google Summer of Code. Позже французский магистрант Matthieu Brucher (Матье Брухер) присоединился к проекту и начал использовать его как часть своей дипломной работы. Результаты его работы заинтересовали сотрудников INRIA - национальный исследовательский институт во Франции, работающий в области компьютерных наук, теории управления и прикладной математики. В итоге в 2010 г. появляется первый публичный релиз. В SciKit-Learn собрано большое количество алгоритмов для задач, связанных с классификацией и машинным обучением в целом.

Модель дерева решений может обрабатывать только числовые данные, поэтому значения переменной «пол» - Gender должны быть преобразованы в числовые представления. 0 будет использоваться для обозначения «мужского пола», а 1 - для «женского».

```
In [45]: #code cell 16
         #replace the Gender labels in the testing dataframe
         # SOLUTION:
         testing["Gender"] = testing["Gender"].apply(lambda toLabel: 0 if toLabel == 'male' else 1)
```

В этом коде лямбда-выражение используется с методом фрейма данных apply(). Это лямбда-выражение представляет функцию, которая использует условный оператор для замены текстовых значений в столбцах соответствующим числовым значением. Лямбда-оператор

можно интерпретировать как «если параметр toLabel равен 'male', вернуть 0, если значение другое, вернуть 1.» Метод apply() выполнит эту функцию для значений в каждой строке столбца «Пол» фрейма данных.

В некоторых записях отсутствует информация о возрасте. Для правильной обработки данных значения должны быть во всех записях. Заменим отсутствующие значения на средний возраст пассажиров. Это не идеальное решение, но в итоге мы получим датафрейм пригодный для обработки методами машинного обучения. Если у вас есть другая идея, примените ее.

Замену можно сделать с помощью метода fillna() для столбца «Возраст» в наборе данных. По умолчанию метод fillna() изменит значения только во временной переменной, связанной с датафреймом. Если мы хотим чтобы данные были изменены в исходном фрейме данных, то это можно сделать с помощью аргумента inplace = True.

```
In [46]: #code cell 17
         #Use the fillna method of the testing dataframe column "Age"
         #to replace missing values with the mean of the age values.
         testing["Age"].fillna(testing["Age"].mean(), inplace=True)
```

б) Проверьте результат. Убедитесь, что значения были заменены.

Убедитесь, что пропущенные значения заполнены, а метки Gender равны 0 и 1.

```
In [47]: #code cell 18
         #verify the data preparation steps.
         # Enter and run both info and head methods
         # from here, by entering and running one and then the other.
         # SOLUTION:
         testing.info()
         testing.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 393 entries, 0 to 392
Data columns (total 12 columns):
PassengerId    393 non-null int64
Survived       393 non-null int64
Pclass         393 non-null int64
Name           393 non-null object
Gender         393 non-null int64
Age            393 non-null float64
SibSp          393 non-null int64
Parch          393 non-null int64
Ticket         393 non-null object
Fare           393 non-null float64
Cabin          93 non-null object
Embarked       392 non-null object
dtypes: float64(2), int64(6), object(4)
memory usage: 36.9+ KB
```

Out[47]:

	Passengerid	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	916	0	2	Coleridge, Mr. Reginald Charles	0	29.0	3	0	W./C. 14263	10.50	NaN	S
1	917	1	1	Spedden, Mrs. Frederic Oakley (Margaretta Corn...	1	40.0	1	1	16966	134.50	E34	C
2	918	0	3	Windelov, Mr. Einar	0	21.0	3	0	SOTON/OQ 3101317	7.25	NaN	S
3	919	1	1	Minahan, Miss. Daisy E	1	33.0	1	0	19928	90.00	C78	Q
4	920	1	3	Wilkes, Mrs. James (Ellen Needs)	1	47.0	1	0	363272	7.00	NaN	S

Обучение и проверка точности модели на основе дерева решений

а) Создайте массив с целевой переменной.

Цель модели - классифицировать пассажиров как потенциально способных выжить или погибнуть. Набор данных идентифицирует выживших и погибших. Модель должна определить, какие значения входных переменных, скорее всего, принадлежат погибшим жертвам, а какие выжившим, а затем использует эту информацию для классификации пассажиров на основе набора тестовых данных.

```
In [37]: #code cell 8
         #create the array for the target values
         y_target = training["Survived"].values
```

б) Создайте массив значений, которые будут входными данными для обучения модели.

Только некоторые данные из полного датафрейма полезны для создания дерева классификатора. Мы создаем список столбцов из данных, которые мы хотим, чтобы классификатор использовал в качестве входных переменных, а затем создаем массив, используя имя

столбца из этой переменной. Переменная X_input содержит значения для всех значений, которые модель будет использовать, чтобы научиться классифицировать пассажиров на категории «способные выжить» и «жертвы». После обучения модели мы будем использовать эту переменную для добавления этих меток к набору тестовых данных и сравнения совпадает ли метка с реальным выживанием или гибелью пассажира.

В предложенном ниже варианте отобраны такие характеристики как стоимость билета – Fare и класс каюты Pclass и то и другое указывает на состоятельность пассажиров. Можно предположить, что у более бедных пассажиров было меньше шансов спастись. Пол – Gender и возраст – Age могли способствовать или препятствовать спасению в смысле физической слабости или, наоборот, большей силы, у более слабых меньше шансов. Последний параметр – SibSp – число родственников. Если человек ехал не один, а с братьями, сестрами, женой, мужем, то шансов спастись будет, вероятно больше. Вы можете предложить свой набор параметров из исходного датасета.

```
In [38]: #code cell 9
         columns = ["Fare", "Pclass", "Gender", "Age", "SibSp"]
         #create the variable to hold the features that the classifier will use
         X_input = training[list(columns)].values
```

в) Обучение модели.

Импортируйте модуль дерева решений из библиотеки машинного обучения sklearn. Создайте объект-классификатор clf_train. Затем используйте метод fit() объекта классификатора с переменными X_input и y_target в качестве параметров для обучения модели.

```
In [39]: #code cell 10
         #import the tree module from the sklearn library
         from sklearn import tree

         #create clf_train as a decision tree classifier object
         clf_train = tree.DecisionTreeClassifier(criterion="entropy", max_depth=3)

         #train the model using the fit() method of the decision tree object.
         #Supply the method with the input variable X_input and the target variable y_target
         clf_train = clf_train.fit(X_input, y_target)
```

г) Оценка точности модели

Используйте метод score() объекта дерева решений, чтобы отобразить процентную точность назначений соответствующих категорий пассажирам, сделанных классификатором. В качестве аргументов он принимает входные и целевые переменные.

```
In [ ]: #code cell 11
        clf_train.score(X_input,y_target)
```

Чем ближе результат к 100%, тем точнее работает модель.

Визуализация дерева решений.

а) Создайте промежуточный файл вывода

Для вывода характеристик дерева решений в файл импортируйте модуль StringIO из sklearn.externals.six. Мы создадим точечный файл Graphviz, который позволит экспортировать результаты работы классификатора в формат, который можно будет потом преобразовать в рисунок.

```
In [ ]: #code cell 12
        from sklearn.externals.six import StringIO
        with open("./Data/titanic.dot", 'w') as f:
          f = tree.export_graphviz(clf_train, out_file=f, feature_names=columns)
```

б) Установите Graphviz

Graphviz – пакет утилит для автоматической визуализации графов, заданных в виде описания на языке DOT, а также дополнительных текстовых и графических программ, виджетов и библиотек, используемых при разработке программного обеспечения для визуализации структурированных данных. Установить Graphviz необходимо из командной строки операционной системы. Для этого нужно открыть окно командной строки. Затем выполнить команду установки. В Debian Linux это можно сделать командой apt-get install graphviz, если вы работаете с пакетом Anaconda в Windows, то установить Graphviz можно, выполнив следующие команды в окне командной строки – Anaconda Prompt:

conda install -c conda-forge python-graphviz

conda install -c conda-forge/label/broken python-graphviz

conda install -c conda-forge/label/cf201901 python-graphviz

conda install -c conda-forge/label/cf202003 python-graphviz

в) Преобразование промежуточного файл графов формата DOT в обычный рисунок

Созданный выше файл точек можно преобразовать в файл .png с помощью средства визуализации точек Graphviz, команды dot. Это команда выполняется интерпретатором командной строки операционной системы, а не Python. В Jupyter Notebook если вы предваряете в программной ячейке команду восклицательным знаком - !, то вы, тем самым, исключаете интерпретацию этой строки Python, и обращаетесь к операционной системе, передавая ей эту команду. В операционной системе Linux команду преобразования dot-файла в png-файл можно выполнить следующим образом:

```
In [ ]: #code cell 13
        #run the Graphviz dot command to convert the .dot file to .png
        !dot -Tpng ./Data/titanic.dot -o ./Data/titanic.png
```

Новый графический файл titanic.png должен появиться в подкаталоге Data, каталога содержащего текущий блокнот Юпитер.

г) Вывод изображения в блокноте Юпитер

Импортируйте модуль Image из библиотеки IPython.display. Это позволит нам открывать и отображать внешний графический файл на странице записной книжки. Функция Image используется для отображения файла с именем файла .png в качестве аргумента.

```
In [ ]: #code cell 14
        #import the Image module from the Ipython.display libary
        from IPython.display import Image

        #display the decison tree graphic
        Image("./Data/titanic.png")
```

Разумеется, посмотреть рисунок можно и без этого, сразу после выполнения предыдущего шага, стандартными средствами операционной системы.

д) Интерпретация полученного дерева решений

В полученном дереве решений мы можем видеть несколько вещей. Во-первых, в корне дерева находится переменная «Пол», что указывает на то, что это единственный наиболее важный фактор при классификации. Ветви слева предназначены для Gender = 0 или мужской. Каждый корневой и промежуточный узел содержит фактор принятия решения, энтропию и количество пассажиров, которые соответствуют критерию в этой точке дерева. Например, корневой узел указывает на то, что набор данных для обучения составляет 891 наблюдение. На следующем уровне мы видим, что 577 человек были мужчинами и 314 женщинами. На третьем уровне, в крайнем правом углу, мы видим, что 415 человек были мужчинами и заплатили за проезд менее 26,2 фунтов 686 человек. Наконец, листовые узлы этого промежуточного узла показывают, что 15 из этих пассажиров были моложе 13,5 лет, а остальные 400 были старше этого возраста.

Наконец, элементы в массиве значений указывают на выживаемость. Первое значение - это количество погибших людей, а второе - количество выживших по каждому критерию. Корневой узел сообщает нам, что из нашей выборки 549 человек погибли и 342 выжили.

Энтропия - это мера шума при принятии решения. Шум можно рассматривать как неопределенность. Например, в узлах, в которых решение приводит к равным значениям в массиве значений выживаемости, энтропия имеет максимально возможное значение, равное 1,0. Это означает, что модель не смогла окончательно принять решение о классификации на основе входных переменных. Для значений очень низкой энтропии решение было гораздо более четким, и разница в количестве выживших и жертв намного выше.

Часть 2: Применение модели дерева решений

В этой части лабораторной работы мы применим обученную модель дерева решений для маркировки немаркированного набора данных пассажиров Титаника. Дерево решений оценит особенности каждого наблюдения и пометит пассажиров как выживших (метка = 1) или погибших (метка = 0).

Импорт и подготовка данных

а) Импортируйте данные.

Назовите фрейм данных "тестирование" и импортируйте файл titanic-test.csv.

```
In [44]: #code cell 15
         #import the file into the 'testing' dataframe.
         testing = pd.read_csv("./Data/titanic-test.csv")
```

Сколько записей в этом наборе данных?

Какие важные переменные не имеют значений и сколько отсутствующих значений?

Как и в предыдущем случае, используйте лямбда-выражение, чтобы заменить значения «мужской» и «женский» на 0 для мужчин и 1 для женщин.

```
In [1]: #code cell 16
        #replace the Gender labels in the testing dataframe
        # Hint: look at code cell 4
```

б) Замените отсутствующие значения возраста средним возрастом:

```
In [46]: #code cell 17
         #Use the fillna method of the testing dataframe column "Age"
         #to replace missing values with the mean of the age values.
         testing["Age"].fillna(testing["Age"].mean(), inplace=True)
```

в) Убедитесь, что значения были заменены.

```
In [ ]: #code cell 18
        #verify the data preparation steps. Enter and run both the info and head
        #methods from here, by entering and running one and then the other.
```

Добавьте маркировку записей к набору данных тестирования

На этом этапе вы примените обученную модель к набору данных тестирования.

а) Создайте массив входных переменных из набора данных тестирования.

```
In [48]: #code cell 19
         #create the variable X_input to hold the features that the classifier will use
         X_input = testing[list(columns)].values
```

б) Примените модель к набору данных тестирования.

Используйте метод predict() объекта clf_train, который был обучен маркировать записи в наборе данных тестирования с наиболее вероятной классификацией выживаемости. Предоставьте массив входных переменных из набора данных тестирования в качестве параметра для этого метода.

```
In [ ]: #code cell 20
        #apply the model to the testing data and store the result in a pandas dataframe.
        #Use X_input as the argument for the predict() method of the clf_train classifier object

        target_labels = clf_train.predict(X_input)

        #convert the target array into a pandas dataframe using the pd.DataFrame() method and target as argument
        target_labels = pd.DataFrame({'Est_Survival':target_labels, 'Name':testing['Name']})

        #display the first few rows of the data set
```

Таким образом, в наборе данных у нас появится еще одна колонка с метками 1 или 0 прогноза выживаемости для соответствующего пассажира.

в) Оценим точность оценочного прогноза.

Точную информацию о выживании каждого пассажира можно найти в другом файле под названием all_data.csv. Чтобы выбрать только пассажиров, содержащихся в наборе данных тестирования, мы объединяем фрейм данных target_labels и фрейм данных all_data по полю Name. Затем мы сравниваем оценочную метку со значением выживаемости в исходном фрейме данных и можем рассчитать точность нашей модели.

```
In [ ]: #code cell 21
        #import the numpy library as np
        import numpy as np
        # load data for all passengers in the variable all_data
        all_data = pd.read_csv("./Data/titanic_all.csv")
        # Merging using the field Name as key, selects only the rows of the two datasets that refer to the same passenger
        testing_results = pd.merge(target_labels, all_data[['Name','Survived']], on=['Name'])

        # Compute the accuracy as a ratio of matching observations to total observations. Store this in in the variable acc.
        acc = np.sum(testing_results['Est_Survival'] == testing_results['Survived']) / float(len(testing_results))
        # Print the result
```

Часть 3: Оценка качества модели дерева решений¶

Библиотека sklearn включает модуль, который можно использовать для оценки качества модели дерева решений. Метод train_test_split() разделит наблюдения во всем наборе данных на два случайно выбранных массива наблюдений, которые составляют наборы данных для тестирования и обучения. После подгонки модели к обучающим данным, обученная модель может быть оценена, а точность прогнозов сравнена как для обучающего, так и для тестового набора данных. Желательно, чтобы две оценки были близки, но точность для набора тестовых данных обычно ниже, чем для набора обучающих данных.

Импорт данных

На этот раз мы импортируем данные из файла csv, но сразу укажем необходимые столбцы, которые должны отображаться во фрейме данных. Мы сделаем это, передав список имен столбцов в виде массива в параметр usecols метода read_csv(). Используйте следующие столбцы: «Выжившие», 'Survived', «Стоимость проезда», 'Fare', «Класс», 'Pclass', «Пол», 'Gender', «Возраст», 'Age' и «Родственники», 'SibSP'. Каждый должен быть заключен в кавычки, а список должен быть заключен в квадратные скобки. Назовите этот фрейм данных all_data.

```
In [ ]: #code cell 22
        #import the titanic_all.csv file into a dataframe called all_data. Specify the list of columns to import.
        all_data = pd.read_csv("./Data/titanic_all.csv", usecols=['Survived','Pclass','Gender','Age','SibSp','Fare'])
        #View info for the new dataframe
```

Выведите информацию по созданному датафрейму и несколько строк из него.

Сколько записей в наборе данных, есть ли отсутствующие данные, сколько записей с отсутствующими данными?

Подготовьте данные.

а) Удалите строковые записи для мужского и женского пола и замените их на 0 и 1 соответственно.

```
In [ ]: #code cell 23
        #Label the gender variable with 0 and 1
```

б) Замените отсутствующие значения возраста средним значением возраста, подсчитанным на всех элементах набора данных, у которых есть этот параметр.

```
In [ ]: #code cell 24
        #replace missing Age values with the mean age

        #display the first few rows of the data set
```

Создание входных и выходных переменных для данных обучения и тестирования.

Если в первом случае мы сами разделяли набор данных на набор для обучения и набор для тестирования и могли сделать это не лучшим образом, то сейчас воспользуемся встроенным инструментарием SciKit-Learn. Библиотека sklearn включает модули, которые помогают с подбором параметров модели. Мы импортируем из sklearn.model_selection метод train_test_split(). Этот метод автоматически разделит весь набор данных, вернув в общей сложности четыре массива numpy, два для функций (тест и проверка) и два для меток (тест и проверка). Один параметр метода определяет долю наблюдений, используемых для тестирования и обучения. Другой параметр указывает начальное значение «соли», которое будет использоваться для рандомизации назначения наблюдения для тестирования или обучения. Это необходимо для того, чтобы другой пользователь мог воспроизвести вашу работу, получая те же назначения наблюдений для наборов данных. Синтаксис метода:

`` train_test_split (input_X, target_y, test_size = 0,4, random_state = 0) ``

40% данных будет использовано для тестирования. Случайное начальное число установлено на 0.

Метод возвращает четыре значения. Эти значения являются входными переменными для данных обучения и тестирования и целевыми переменными для данных обучения и тестирования в указанном порядке.

а) Назначение входных и выходных переменных и генерация массивов.

```
In [ ]: #code cell 25
        #Import train_test_split() from the sklearn.model_selection library
        from sklearn.cross_validation import train_test_split

        #create the input and target variables as uppercase X and lowercase y. Reuse the columns variable.
        X = all_data[list(columns)].values
        y = all_data["Survived"].values

        #generate the four testing and training data arrays with the train_test_split() method
        X_train,X_test,y_train,y_test=train_test_split(X, y, test_size=0.40, random_state=0)
```

б) Обучение модели и подгонка ее под данные тестирования.

Теперь модель снова можно подогнать. Модель будет обучаться с использованием только обучающих данных, выбранных функцией train_test_split.

```
In [ ]: #code cell 26
        #create the training decision tree object
        clf_train = tree.DecisionTreeClassifier(criterion="entropy", max_depth=3)

        #fit the training model using the input and target variables
        clf_train = clf_train.fit(X_train, y_train)
```

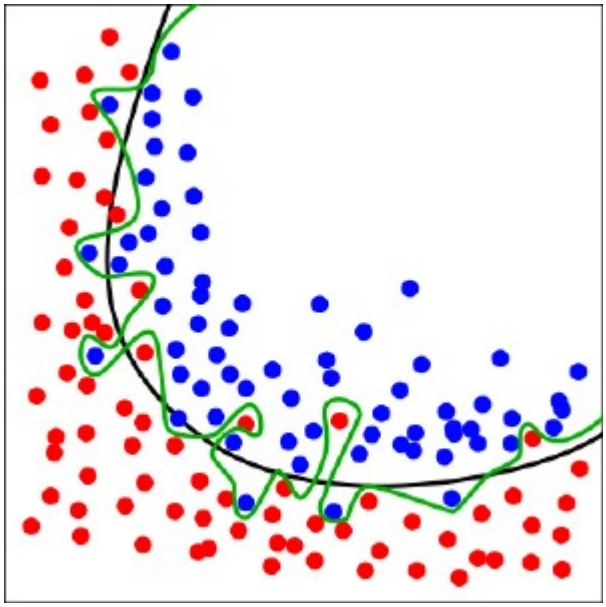
в) Сравнение моделей с оценкой точности каждой из них.

Для генерации оценок применения модели к обучающей и к проверочной выборке используется метод score().

```
In [ ]: #code cell 27
        #score the model on the two datasets and store the scores in variables. Convert the scores to strings using str()
        train_score = str(clf_train.score(X_train,y_train))
        test_score = str(clf_train.score(X_test,y_test))

        #output the values in a test string
        print('Training score = '+ train_score+' Testing score = '+test_score)
```

Теперь мы сравнили оценки обученной модели как на данных тестирования, так и на данных проверки. Лучший результат на данных обучения может быть связан с тем, что модель имеет тенденцию «переобучаться» на обучающих данные, поэтому тестовая оценка является лучшей проверкой того, как модель, в принципе, может работать за пределами обучающей выборки данных.

Переобучение (переподгонка, пере- в значении «слишком», англ. overfitting) в машинном обучении и статистике — явление, когда построенная модель хорошо объясняет примеры из обучающей выборки, но относительно плохо работает на примерах, не участвовавших в обучении (на примерах из тестовой выборки). См. рисунок ниже.



Зелёная линия показывает переобученную модель, а чёрная линия - регуляризированную и более точную модель. Хотя зелёная линия лучше соответствует образцам, по которым проходило обучение, классификация по зелёной линии очень зависит от конкретных данных, и скорее всего новые данные будут плохо соответствовать классификации по зелёной линии и лучше - классификации по чёрной линии.

Часть 4 дополнительная и не обязательная

Если вы заинтересовались, то можете попробовать следующие приемы и посмотреть, как это повлияет на результат прогноза.

1. Удалите наблюдения с отсутствующими значениями возраста.

Использование среднего возраста для замены отсутствующих значений может повлиять на точность модели. Один из альтернативных способов заключается в удалении всех наблюдений с отсутствующими значениями возраста. Хотя это уменьшит размер обучающего набора данных, это может повысить точность.

2. Измените входные переменные.

Другой проблемой является определение того, какие входные переменные или характеристики важны для точности классификатора. Один из способов проверить это -

попробовать запустить классификатор с разными наборами входных переменных, отредактировав список переменных, который используется для обучения модели.

Контрольные вопросы

1. В случаях подобных катастроф команда корабля должна следовать правилу «Женщины и дети в первую очередь» как руководство для того, кому должен быть предоставлен доступ к спасательным шлюпкам. Придерживались ли этого правила в данном случае? Изучите дополнительную информацию и ответьте на вопрос.
2. Какое значение было использовано для замены отсутствующих возрастов?
3. Из вашего анализа дерева решений, что описывает группу, в которой было больше всего смертей по количеству? В какой группе осталось больше всего выживших?
4. Какова была точность прогноза при выполнении части 2 работы?
5. Какова была точность прогноза на обучающей и тестовой выборке данных при выполнении части 3 работы?

1.9 Лабораторная работа «Обработка ошибок при использовании метода линейной регрессии»

Цель работы

В этой лабораторной работе вы познакомитесь с концепциями оценки ошибок подгонки в линейной регрессии.

Сценарий

В статистике линейная регрессия - это способ моделирования отношения между зависимой переменной y и независимой переменной x. Цель регрессии - найти модель, которая как можно точнее описывает данные. В этой лабораторной работе вы будете использовать данные о продажах и результат линейной регрессии из предыдущей лабораторной работы, чтобы оценить точность модели.

Требуемые ресурсы

ПК с доступом к Интернет

Библиотеки Python: pandas, numpy и sklearn.

Файлы данных: stores-dist.csv

Порядок выполнения работы

Часть 1: Импорт библиотек и данных

В этой части вы импортируете необходимые для работы библиотеки и данные из файла store-dist.csv.

Импорт библиотек.

Для выполнения потребуются библиотеки Pandas, NumPy и SciKit-Learn. Так как некоторые библиотеки обновляются чаще, а другие реже, при импорте вы можете увидеть предупреждающие сообщения о том, что некоторые функции импортируемой библиотеки устарели и скоро будут обновлены. Для подавления вывода этих предупреждений в примере ниже показано что можно дополнительно импортировать библиотеку warnings и применить функцию подавления вывода предупреждающих сообщений, т.к. они в общем не влияют на работу импортируемой библиотеки.

```
In [ ]: # Code Cell 1

        # This lab produces some minor warnings that can be ignored.
        # These warnings appear because some libraries are updated more often than others
        # and the system is letting the user know that some function will be depricated soon
        # Use the following code to prevent the warnings from being displayed, or comment them out
        # to see the warnings
        import warnings
        warnings.filterwarnings('ignore')

        import numpy as np
        import pandas as pd
        from sklearn import cross_validation
        from sklearn.linear_model import LinearRegression
```

Импорт данных.

Импортируйте данные из файла store-dist.csv, измените заголовки столбцов, как вы уже делали ранее и убедитесь, что файл импортирован правильно.

Заголовки столбцов annual net sales переименуйте в sales и number of stores in district в stores, чтобы упростить обработку данных.

```
In [ ]: # Code Cell 2

        # Import the file stores-dist.txt
        salesDist = pd.read_csv('./Data/stores-dist.txt')

        # Change the column headings
        salesDist.columns = ['district','sales','stores']

        # Verify the imported data
        # ...
```

Так как столбец district не нужен для оценки точности и ошибок линейной регрессии, то его можно удалить:

```
In [ ]: # Code Cell 3
        # Drop the district column.
        sales = salesDist.drop('district',axis=1)

        # Verify that the district column has been dropped.
        # ...
```

Часть 2: Расчет ошибок

В этой части вы будете использовать numpy для создания линии регрессии для проанализированных данных. Вы также, как и ранее, вычислите центроид для этого набора данных. Сгенерированная линия простой линейной регрессии также должна проходить через центроид. Затем вы примените sklearn.metrics для оценки модели линейной регрессии. Вы рассчитаете Коэффициент детерминации R2 и среднеквадратичную ошибку (MSE) [11-13].

Назначение переменных x и y.

Назначьте продажи из фрейма данных как зависимую переменную y и число магазинов как независимую переменную для оси x.

```
In [ ]: # Code Cell 4
        #dependent variable for y axis
        y = sales.sales
        #independent variable for x axis
        #x = ...
```

Расчет зависимой переменной y в модели

В предыдущей лабораторной работе вы рассчитали компоненты для линейной регрессии, соответствующей полиномиальной модели, используя np.polyfit для вычисления вектора коэффициентов p, который минимизирует квадратичную ошибку. Используя np.poly1d, вы можете вычислить соответствующее значение для каждого значения x в предполагаемой полиномиальной модели.

Для сохранения значений наклона и точки пересечения оси y аппроксимирующей линии, используйте переменную p. Массив p отображает коэффициенты в порядке убывания. Для полинома первого порядка первый коэффициент - это наклон (m), а второй коэффициент - это точка пересечения по оси y (b):

```
In [ ]: # Code Cell 5
        # compute the y values from the polynomial model for each x value
        order = 1
        p = np.poly1d(np.polyfit(x, y ,order))

        print('The array p(x) stores the calculated y value from the polynomial model for each x value,\n\n{}.'.format(p(x)))
        print('\nThe vector of coefficients p describes this regression model:\n{}'.format(p))
        print('\nThe zeroth order term (y-intercept or b) is stored in p[0]: {}.'.format(p[0]))
        print('\nThe first order term (slope or m) is stored in p[1]: {}.'.format(p[1]))
```

Оценка моделей различными способами.

На этом этапе вы будете использовать sklearn для оценки моделей. Sklean предлагает ряд способов измерения ошибок. Вы рассчитаете коэффициент детерминации R2, среднеквадратичную ошибку (MSE) и среднюю абсолютную ошибку (MAE), используя готовые функции в sklearn. Чтобы вычислить значение для соответствующего типа ошибки, укажите значения из y, которые представляют собой наблюдаемые значения из импортированного файла CSV, в качестве первого аргумента store-dist.csv. В качестве второго аргумента используйте значения из p(x), которые были вычислены из вашей полиномиальной модели первого порядка в форме:

$$y=mx+b$$

где m равно p[1], а b равно p[0] в результатах poly1d

Функция оценки регрессии $R2$ (коэффициент детерминации) дает некоторую информацию о степени точности модели. Наилучшая возможная оценка - 1,0. Коэффициент детерминации показывает, насколько хорошо модель объясняет наблюдаемый результат.

```
In [ ]: # Code Cell 6
        from sklearn.metrics import r2_score
        r2 = r2_score(y, p(x))
        r2
```

Среднеквадратичная ошибка (MSE) - это мера того, насколько хорошо модель делает прогноз. Это число всегда неотрицательно. Лучше, когда значения среднеквадратичной ошибки ближе к нулю.

```
In [ ]: # Code Cell 7
        from sklearn.metrics import mean_squared_error
        mse = mean_squared_error(y, p(x))
        mse
```

Средняя абсолютная ошибка (MAE) - это мера того, насколько предсказания близки к конечным результатам по абсолютной величине. MAE - это среднее значение абсолютных ошибок, разности между прогнозом и истинным значением.

```
In [ ]: # Code Cell 8
        from sklearn.metrics import mean_absolute_error
        mae = mean_absolute_error(y, p(x))
        mae
```

Все эти меры позволяют определить, насколько хорошо ваша модель может делать прогнозы. В этой лабораторной работе вы оценили только одну модель - простую линейную регрессию, но соответствующие методы оценки есть для всех типов моделей

Контрольные вопросы:

1. Какое значение вы получили для коэффициента детерминации?
2. Какое значение среднеквадратичной ошибки было рассчитано для вашей модели?
3. Какое значение средней абсолютной ошибки было получено?
4. Как с помощью NumPy найти ковариацию двух переменных?

3 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

3.1 Общие положения

Цель самостоятельной работы по дисциплине – закрепление и углубление теоретических знаний; формирование умения работать с научной и технической литературой и осуществлять самостоятельный поиск информации; развитие научно-исследовательских и творческих способностей; приобретение навыков расчётно-аналитической работы.

Самостоятельная работа студента по дисциплине «Статистика» включает следующие виды его активности:

- проработка лекционного материала;
- изучение тем теоретической части дисциплины, вынесенных для самостоятельной проработки;
- подготовка к лабораторным работам;
- подготовка к зачету с оценкой.

3.2 Проработка лекционного материала

При проработке лекционного материала по каждой теме студент должен внимательно ознакомиться с конспектом лекций, а затем для углубленного изучения материала следует обратиться к литературным источникам (учебникам, учебным пособиям, монографиям, статьям, статистическим сборникам), а также материалам, размещенным в сети Интернет. Для закрепления материала темы необходимо ответить на предлагаемые в пособиях вопросы и прорешать задачи по теме.

При изучении каждой темы целесообразно:

1) ознакомиться с методическим обеспечением изучаемой дисциплины, включающей тематический план и программу курса;

2) руководствоваться рекомендованной нормативной базой и учебной литературой, которая имеется в фондах библиотеки;

3) использовать возможности сайта библиотеки университета и другие информационные ресурсы Интернета;

4) прочитать соответствующую теме главу учебника;

5) доработать конспект лекции.

При изучении учебного материала темы студенту необходимо, прежде всего, разобраться в основанных понятиях и терминах данной темы. Для этого рекомендуется использовать различные источники информации, в том числе учебные пособия, монографии, периодические издания, законодательные и нормативные документы, статистические материалы, информацию государственных органов власти и управления, органов местного самоуправления, переводные издания, а также труды зарубежных авторов.

Изучение рекомендованных источников следует начинать с основных рекомендованных преподавателем глав и разделов учебников и учебных пособий, а затем переходить к научным монографиям и материалам периодических изданий. При этом полезно делать выписки и конспекты наиболее интересных материалов, что способствует более глубокому осмыслению материала и лучшему его запоминанию. Такая практика учит отделять в тексте главное от второстепенного, а также позволяет проводить систематизацию и сравнительный анализ изучаемой информации, что важно в условиях большого количества разнообразных по качеству и содержанию сведений.

Проработка пройденного лекционного материала является наиболее важным видом самостоятельной работы. Чем глубже и полнее проработан материал, тем легче при выполнении других видов самостоятельной работы. Систематическая, регулярная работа над прой-

денным лекционным материалом, начиная с первого занятия, является необходимым условием для понимания материалов последующих лекций и усвоения материалов практических и лабораторных занятий.

3.4 Подготовка к промежуточной аттестации

Подготовка к промежуточной аттестации осуществляется по вопросам, приведенным в рабочей программе дисциплины.

СПИСОК ЛИТЕРАТУРЫ

1. Формат метки времени https://docs.python.org/2/library/time.html
2. Работа с CSV в Python https://docs.python.org/2/library/csv.html
3. Краткое введение в реляционные базы данных (временное): http://searchsqlserver.techtarget.com/definition/relational-database
4. Видео-урок о реляционных базах данных: https://www.youtube.com/watch?v=jyju2P-7hPA&list=PLAwxTw4SYaPm4R6j_wzVOCV9fJaiQDYx4
5. Введение в SQL http://www.w3schools.com/sql/sql_intro.asp
6. Документация по форматированию текстовых строк https://docs.python.org/2/library/string.html
7. Примеры форматирования строк: https://mkaz.tech/code/python-string-format.html
8. Использование различных маркеров на графиках http://matplotlib.org/api/markers_api.html
9. Использование различных цветов на графиках http://matplotlib.org/api/colors_api.html
10. Titanic - Machine Learning from Disaster https://www.kaggle.com/c/titanic
11. Бахрушин В. Е. Методы оценивания характеристик нелинейных статистических связей //Системные технологии. — 2011. — № 2(73). — С. 9—14.
12. Магнус Я.Р., Катышев П.К., Пересецкий А.А. Эконометрика. Начальный курс. — 6,7,8-е изд., доп. и перераб. — Москва: Дело. — 576 с. — ISBN 5-7749-0055-X.
13. Айвазян С.А., Мхитарян В.С. Прикладная статистика. Основы эконометрики (в 2-х т.). —Москва: Юнити-Дана (проект TASIS), 2001. — Т. "1,2". — 1088 с. — ISBN 5-238-00304-8.