Лабораторная работа 1. Введение в Python

1. Цели

Приобрести навыки работы с наборами данных с помощью языка Python. Освоить среду разработки Jupyter.

2. Задачи

1. Установить на устройство среду разработки
2. Подготовить набор данных к дальнейшей обработке

3. Теоретические сведения

Методические указания для решения поставленного задания

3.1. Установка среды разработки

Данный курс рассчитан на изучение языка Python для работы с данными, поэтому используемой средой разработки будет JupyterLab, продвинутая версией JupyterNotebook, отличием которой от других сред является интерактивный режим, позволяющий проводить эксперименты с данными без необходимости повторного выполнения кода.

Установка JupyterLab происходит следующим образом:

1. Скачать Anaconda – оболочку для запуска средств связанных с обработкой данных
2. Установить Anaconda
3. Запустить Anaconda
4. Во вкладке `Home` найти JupyterLab, нажать `launch` под его иконкой. Откроется браузер по умолчанию, вкладка с адресом `localhost:8888/lab`, открывшаяся вкладка – среда разработки, в которой будет писаться код. JupyterLab работает в браузере потому, что он, как и все средства на основе Interactive Python – веб-приложения разворачивающиеся локально

3.2. Работа в JupyterLab

Работа с проектами происходит следующим образом:

1. Если IPython Notebook еще не создан
   1.1. Во вкладке `Launcher` найти раздел `Notebook`, в нем выбрать `Python 3 (ipykernel)` или в меню сверху `File` → `New` → `Notebook`
2. Если нужно открыть существующий IPython Notebook
   2.1. В меню сверху `File` → `Open from Path`, далее указать путь к проекту. Можно указать путь как к папке, так и сразу к ноутбуку.

Код пишется в ячейках и запускается кнопкой `Run the selected cells and advance`, или сочетанием клавиш `Shift + Enter`. Код в каждой ячейке выполняется отдельно, в том порядке, в котором запускаются ячейки, а не в том, в котором они расположены. Одну и ту же ячейку можно запустить несколько раз, каждый раз код будет выполняться заново – это может приводить к ошибкам.

Результаты вычисления доступны для обработки и хранятся в оперативной памяти все время работы среды разработки. Для запуска всех ячеек в порядке их расположения нажмите кнопку `Restart Kernel and Run All Cells…`, это действие перезагрузит среду, очистит память от хранящихся в ней данных и после выполнит весь имеющийся код.

Ячейки в JupyterLab могут быть трех типов:

1. `Code` – используются для написания кода и его выполнения

2. `Markdown` – используются для написания заметок, инструкций и объяснений с помощью языка разметки Markdown
3. `Raw` – просто хранит текст

3.3. Введение в Python

Предполагается, что студенты, приступившие к курсу, имеют базовые понятия программирования, либо опыт в других языках, и потому вполне достаточно провести параллели между алгоритмическими конструкциями и синтаксисом Python.

3.3.1. Последовательные действия

Последовательные действия описываются последовательными строками программы. Стоит, правда, добавить, что в программах важны отступы, поэтому все операторы, входящие в последовательность действий, должны иметь один и тот же отступ

```
a = 1
b = 2
a = a + b
b = a - b
a = a - b
print(a, b)
```

3.3.2. Списки

В Python нет массивов с произвольным типом элемента. Вместо них используются списки. Их можно задать с помощью литералов, записываемых в квадратных скобках, или посредством списковых включений. Варианты задания списка приведены ниже

```
list_1 = [1, 2, 3]
list_2 = [number ** 2 for number in range(10) if number % 2 == 1]
list_3 = list("abcde")
```

3.3.3. Оператор условия и выбора

Разумеется, одними только последовательными действиями в программировании не обойтись, поэтому при написании алгоритмов используется еще и ветвление

```
a = 10
if a > 5:
    b = 5
else:
    b = 10
```

Оператор ветвления имеет в данном случае две части, команды для которых записываются с отступом вправо относительно оператора ветвления. Более общий случай можно записать с помощью следующего кода

```
a = 0
if a < 0:
    b = -1
elif a == 0:
    b = 0
else:
    b = 1
```

3.3.4. Операторы циклов

Третьей необходимой алгоритмической конструкцией является цикл. С помощью цикла можно описать повторяющиеся действия. В Python имеются два вида циклов: `while` и `for`

```
string = "Hello world!"
while string != "":
    print(string)
    string = string[:-1]
```

Оператор `while` работает по принципу «пока верно условие цикла, выполнять тело цикла». Тело цикла выделяется отступом. Каждое исполнение тела цикла называется итерацией. В приведенном примере удаляется последний символ строки до тех пор, пока не останется пустая строка.

Для большей гибкости при организации циклов применяются операторы `break` и `continue`. Первый позволяет прервать цикл, а второй – продолжить цикл, перейдя к следующей итерации не завершая текущую.

Следующий пример читает строки из файла и выводит те, у которых длина больше пяти

```
with open("файл.csv", "r") as file:
    while True:
        line = file.readline()
        if not line:
            break
        if len(line) > 5:
            print(line)
```

В этом примере организован бесконечный цикл, который прерывается только при получении из файла пустой строки `line`, что обозначает конец файла.

В языке Python логическое значение несет каждый объект – нули, пустые строки и последовательности, специальные объекты вроде `None` и логический литерал `False` имеют значение «ложь», а все остальные значение «истина». Для обозначения истины обычно используется литерал `True`

Цикл `for` выполняет тело цикла для каждого элемента последовательности. В следующем примере выводится таблица умножения

```
numbers = [number for number in range(1, 10)]
for first_number in numbers:
    for seconds_number in numbers:
        print(f"{first_number} * {seconds_number} = {first_number * seconds_number}")
```

Здесь циклы `for` являются вложенными. Метод `range` порождает список целых чисел из полуоткрытого диапазона `[1,10)`. Перед каждой итерацией счетчик цикла получает очередное значение из этого списка. Полуоткрытые диапазоны общеприняты в Python. Считается, что их использование более удобно и вызывает меньше ошибок. В Python первый элемент имеет индекс `0`.

3.3.5. Методы

Программист может определять собственные функции двумя способами: с помощью оператора `def` или прямо в выражении, посредством `lambda`. Второй способ был применен при заполнение списка `numbers` значениями от `1` до `10` в одну строку, в предыдущем примере кода

```
def price(rubles, penny=0):
    return "%i руб. %i коп." % (rubles, penny)

print(price(8, 50))
print(price(7))
print(price(rubles=23, penny=70))
```

В этом примере определена функция для двух аргументов, из которых второй имеет значение по умолчанию `0`. Вариантов вызова этой функции с конкретными параметрами

также несколько. При вызове функции сначала должны идти позиционные параметры, а затем, именованные. Аргументы со значениями по умолчанию должны следовать после обычных аргументов. Оператор `return` возвращает значение функции. Из функции можно возвратить только один объект, но он может быть списком из нескольких объектов.

3.3.6. Модули

В соответствии с модульным подходом к программированию – большая задача разбивается на несколько более мелких, каждую из которых решает отдельный модуль. Набор классов и функций, имеющий множество связей между своими элементами, было бы логично расположить в одном модуле. Есть и еще одно полезное замечание – модули должно быть легче использовать, чем написать заново. Это значит, что модуль должен иметь удобный интерфейс – набор функций, классов и констант, который он предлагает своим пользователям.

В языке Python набор модулей, посвященных одной проблеме, можно поместить в пакет. Хорошим примером такого пакета является пакет `pandas`, в котором собраны модули для различных аспектов обработки табличных данных.

Пример работы библиотеки `datetime`

```
import datetime
date = datetime.date(2004, 11, 20)
```

В данном примере импортируется модуль `datetime`. В результате работы оператора `import` в текущем пространстве имен появляется объект с именем `datetime`. Модули для использования в программах на языке Python по своему происхождению делятся на обычные – написанные на Python, и модули расширения, написанные на другом языке программирования – как правило, на языке `C`. С точки зрения пользователя они могут отличаться разве что быстродействием. Бывает, что в стандартной библиотеке есть два варианта модуля: на Python и на `C`. Таковы, например, модули `pickle` и `cPickle`.

3.4. Подготовка набора данных к дальнейшей обработке

Данные для обработки в Python, в большинстве случаев хранятся в файлах формата «значения, разделенные запятой» (расширение `.csv`). Часто, значения в таких файлах нужно будет менять для проведения расчетов.

Примеры изменения данных будут приведены ниже, работа будет производится на наборе Unicorn Companies Dataset (рус. Набор данных о компаниях-единорогах).

3.4.1. Загрузка данных

Для изменения данных, их сначала нужно загрузить в среду. Для этого, в данном курсе рекомендуется использовать библиотеку `pandas`. Загрузка библиотеки `pandas` осуществляется командой `import pandas`. Продолжение строки `as pd` значит, что библиотеке назначается псевдоним `pd` и теперь использовать её нужно через этот псевдоним. Изначально псевдоним использовался для удобства, но сейчас это стандарт обращения к данной библиотеке.

```
import pandas as pd
dataset = pd.read_csv("unicorn.csv")
```

Следующая строка создает переменную с названием `dataset`. В данном случае данные переменной будут равны тому, что вернет вызов метода `read_csv` из библиотеки `pandas`,

который принимает на вход строку с адресом к файлу с расширением `.csv`, а возвращает данные типа `pandas.DataFrame`. Тип `pandas.DataFrame` используется для хранения таблиц.

3.4.2. Удаление строк из таблицы

Удалить строки из набора может понадобится из-за аномальных значений, избыточных данных или особых потребностей. Например, может быть поставлена задача изучить на данном наборе только компании работающие в одной определенной сфере

Удаление строк по индексу происходит с помощью метода `drop`, который принимает на вход индекс строки в виде целого числа. Для внесения изменений в таблицу ей нужно присвоить результат выполнения метода, иначе изменения не сохранится.

```
dataset = dataset.drop(0)
```

Также, можно удалять сразу несколько строк передавая в метод список индексов.

```
dataset = dataset.drop([1, 2, 3])
```

Или можно удалять строки на основе каких-либо условий. Ниже приведен пример для удаления всех строк, где в столбце `Country` строка имеет значение `Sweden`.

```
for index, row in dataset.iterrows():
    if row["Country"] == "Sweden":
        dataset = dataset.drop(index)
```

Псевдокод:

1. Для каждой строки из таблицы
    1.1. Если значение в поле `Country` является строкой и равно `Sweden`
        1.1.1. Удалить из таблицы строку с текущим индексом (индекс подсчитывается автоматически в пункте 1) и сохранить изменение

После выполнения манипуляций над строками может понадобится восстановить их индексы.

```
dataset.index = [index for index in range(len(dataset))]
```

3.4.3. Удаление столбцов из таблицы

Удалить столбцы может понадобится тогда, когда среди них есть мультиколлинеарность, данные в столбце незначимы или их невозможно исследовать.

Удаление столбца по названию происходит с помощью метода `drop`, который принимает на вход название столбца в виде строки и параметр `axis`, равный `1`

```
dataset = dataset.drop("Country", axis=1)
```

Также, можно удалять сразу несколько столбцов передавая в метод список названий и параметр `axis`, равный `1`

```
dataset = dataset.drop(
    [
        "Country",
        "Date Joined",
        "Financial Stage"
    ],
    axis=1
)
```

3.4.4. Замена и удаление значений

Замена значений используется, когда нужно исправить форматирование данных, обработать выбросы или пустые значения, привести значения к одной единице измерений. Например, в данном наборе столбец `Total Raised` имеет все вышеперечисленные проблемы

Замена пустых значений происходит с помощью метода `replace`, который принимает на вход первым параметром «что менять» и вторым параметром «на что менять». Также для этой цели можно использовать более продвинутый метод `fillna` из библиотеки `pandas`, но для данной лабораторной работы будет достаточно метода `replace`

```
dataset["Total Raised"] = dataset["Total Raised"].str.replace("None", "$0B")
```

Также возможно удаление или замена отдельных символов в значениях. В примере ниже происходит удаление символа `$`, удаление символа `B`, удаление символа `M` с последующим делением значения на `1 000` и удаление символа `K` с последующим делением значения на `1 000 000`. После типы полученных записей меняются на числовые.

```
dataset["Total Raised"] = dataset["Total Raised"].str.replace("$", "")
for index, row in dataset.iterrows():
    last_symbol = ["Total Raised"][-1]
    if last_symbol == "B":
        dataset["Total Raised"][index] = dataset["Total Raised"][index][:-1]
    elif last_symbol == "M":
        dataset["Total Raised"][index] = float(dataset["Total Raised"][index][:-1]) / 1000
    elif last_symbol == "K":
        dataset["Total Raised"][index] = float(dataset["Total Raised"][index][:-1]) / 1000000
dataset["Total Raised"] = pd.to_numeric(dataset["Total Raised"])
```

Псевдокод:

1. Удалить из столбца `Total Raised` все символы `$` и сохранить изменения
2. Для каждой строки из таблицы
    2.1. Создаётся переменная значение которой равно последнему символу строки в поле `Total Raised`
    2.2. Если переменная из пункта 2.1 является строкой и равна `B`
        2.2.1. Заменить значение, находящееся в строке с текущим индексом и полем `Total Raised` на это же значение без последнего символа. Сохранить изменение
    2.3. Если переменная из пункта 2.1 является строкой и равна `M`
        2.3.1. Заменить значение, находящееся в строке с текущим индексом и полем `Total Raised` на это же значение без последнего символа, и деленное на `1 000`. Сохранить изменение
    2.4. Если переменная из пункта 2.1 является строкой и равна `K`
        2.4.1. Заменить значение, находящееся в строке с текущим индексом и полем `Total Raised` на это же значение без последнего символа, и деленное на `1 000 000`. Сохранить изменение
3. Сменить тип всех значений в столбце `Total Raised` на числовой и сохранить изменения

3.4.5. Факторизация

Факторизация используется, когда нужно исследовать строковые значения, которые часто повторяются в наборе и находятся в каком-то диапазоне значений. В данном наборе, факторизовать можно столбец `Country`, `City`, `Industry` и `Financial Stage`

Замена строковых значений на категориальные значения (факторизация), производится вызовом метода `factorize` из библиотеки `pandas`. Метод принимает на вход данные, которые нужно факторизовать и возвращает список из двух списков, факторизованные значения хранятся в первом подсписке, поэтому сохраняем только его

```
dataset["Country"] = pd.factorize(dataset["Country"])[0]
```

3.4.6. Нормализация и стандартизация значений

Нормализация и стандартизация значений применяются для смягчения влияния больших значений на результаты обучения моделей машинного обучения. Например, если значения первого столбца варьируются от 1 до 10, а второго от 1000 до 100 000, то значения второго столбца будут влиять на результаты предсказания модели гораздо сильнее чем первого, хотя на самом деле могут оказаться незначимыми вовсе.

Нормализация используется, когда данные нужно привести к единому виду, который позволит сравнивать их между собой или использовать для расчёта схожести объектов. Стандартизацию применяют перед использованием алгоритмов, работающих на основе вычисления расстояния между данными

Нормализация значений в столбце производится с помощью следующего кода. Тип `pandas.DataFrame` позволяет посчитать нормализованное значение без построения циклов. Вручную, значения вычисляется для каждого ячейки отдельно, по формуле приведенной ниже

$$\text{новое значение} = \frac{\text{старое значение} - \text{мин. значение столбца}}{\text{макс. значение столбца} - \text{ мин. значение столбца}}$$

```
dataset["Столбец"] = (dataset["Столбец"] - dataset["Столбец"].min()) / (dataset["Столбец"].max()
- dataset["Столбец"].min())
```

Стандартизация производится теми же методами, но по другой формуле, приведенной ниже

$$\text{новое значение} = \frac{\text{старое значение} - \text{среднее значение столбца}}{\text{стандартное отклонение столбца}}$$

```
dataset["Столбец"] = (dataset["Столбец"] - dataset["Столбец"].mean()) / dataset["Столбец"].std()
```

3.4.7. Сохранение результатов обработки

Обработанный набор данных можно сохранить для последующего использования. Для сохранения в формат `.csv` используется метод `to_csv` принимающий на вход строку, содержащую адрес, по которому будет сохранен файл

```
dataset.to_csv(
    "Набор данных.csv"
)
```

Для сохранения в файл MS Excel используется метод `to_excel` принимающий на вход строку, содержащую адрес, по которому будет сохранен файл. Также для удобства в метод были переданы параметры `sheet_name` – название листа и `index` – сохранение столбца с индексами

```
dataset.to_excel(
    'Набор данных.xlsx',
    sheet_name='Данные',
    index=False,
)
```

4. Задание

Выбрать с сайта `kaggle.com` набор данных в формате `.csv`, загрузить и подготовить его к дальнейшей обработке с помощью описанных ранее методов. Наборы данных не должны повторяться внутри группы. Задание индивидуальное. Требования:

1. Значения подразумевающие числа должны иметь числовой тип
2. Строковые значения должны заменены на числовые значения и иметь числовой тип
3. Индексы строк должны идти последовательно
4. Незначимые данные должны быть удалены
5. Указать какие знания можно получить из набора
6. Сохранить обработанный набор
7. Сохранить `IPython Notebook`

4.1. Продвинутое задание

С помощью студенческого билета БГУИР получить среду разработки JetBrains DataSpell