Курс «Базовая обработка данных на языке Python»

автор: Киреев В.С., к.т.н., доцент

Лабораторная работа № 1

Тема: «Работа с библиотеками requests, pandas, pyod. Базовые конструкции в Python»

Цель работы: изучить основы работы с библиотеками requests, pandas, pyod.

Теоретическая справка

Определение функций в Python

Функции в Python определяются с помощью ключевого слова def, за которым следует имя функции и круглые скобки (). Внутри этих скобок вы можете указать любые параметры, которые функция должна принимать. Затем после скобок ставится двоеточие : для начала блока кода, который будет выполняться при вызове функции.

Вот базовый синтаксис для определения функции:

```
def имя_функции(параметры):
    # Код функции
```

Вызов функции осуществляется путем написания имени функции с круглыми скобками и передачи аргументов (если они есть). Например:

```
result = add_numbers(5, 3)
print(result)  # Выведет: 8
```

Работа с циклами (break, continue)

В Python есть два важных оператора, используемых внутри цикла для управления его выполнением: break и continue.

Оператор break используется для немедленного выхода из цикла. Это может быть полезно, если нужно прервать выполнение цикла при выполнении определенного условия.

```
for i in range(10):
  if i == 5:
    break
  print(i)
```

В этом примере цикл прерывается, когда i равно 5.

Оператор continue, напротив, пропускает оставшуюся часть текущей итерации и немедленно переходит к следующей итерации.

```
for i in range(10):
  if i == 5:
    continue
  print(i)
```

Аргументы функций в Python

Аргументы функций в Python - это значения, которые передаются функции при ее вызове. В Python есть несколько типов аргументов, которые можно использовать:

1. **Позиционные аргументы**: Это наиболее общий тип аргументов. Они передаются в функцию в том порядке, в котором они определены. Например:

```
def greet(name, greeting):
    return f"{greeting}, {name}!"
print(greet("Вася", "Привет"))  # Выведет: "Привет, Вася!"
```

2. **Ключевые аргументы**: Это аргументы, которые передаются по имени. Они могут быть указаны в любом порядке при вызове функции. Например:

```
print(greet(greeting="Привет", name="Вася"))  # Также выведет: "Привет,
Вася!"
```

3. **Аргументы со значениями по умолчанию**: Это аргументы функции, которые имеют значение по умолчанию. Если при вызове функции аргумент не указан, будет использовано значение по умолчанию. Например:

```
def greet(name="друг", greeting="Привет"):
    return f"{greeting}, {name}!"
print(greet())  # Выведет: "Привет, друг!"
```

4. **Произвольные аргументы**: Если вы не знаете, сколько аргументов будет передано вашей функции, вы можете добавить *args и **kwargs в определение вашей функции. *args используется для неименованных аргументов (обычно позиционных), а **kwargs - для именованных аргументов (ключевых). Например:

```
def print_args(*args, **kwargs):
    for i, arg in enumerate(args):
        print(f"Позиционный аргумент {i} = {arg}")
    for key, value in kwargs.items():
        print(f"Ключевой аргумент {key} = {value}")
print_args("Вася", "Петя", greeting="Привет", time="утро")
```

Область видимости переменных в Python

Область видимости переменной определяет часть программы, где эта переменная может быть доступна. В Python есть два основных типа области видимости переменных:

1. **Глобальная область видимости**: Переменные, определенные в основном теле программы, находятся в глобальной области видимости. Они доступны в любой части кода, включая функции (если они не были переопределены внутри функции).

2. **Локальная область видимости**: Переменные, определенные внутри функции, находятся в локальной области видимости. Они доступны только внутри этой функции.

Вот пример:

```
x = 10  # Глобальная переменная
def my_func():
    y = 5  # Локальная переменная
    print(x)  # Может получить доступ к глобальной переменной
    print(y)  # Может получить доступ к локальной переменной
my_func()
print(x)  # Может получить доступ к глобальной переменной
print(y)  # Ошибка! Не может получить доступ к локальной переменной
```

Возвращаемые значения в Python

В Python функция может возвращать значение с помощью ключевого слова return. Это значение затем можно использовать в другой части программы. Если функция не содержит инструкции return, она по умолчанию возвращает None.

Вот пример функции, которая возвращает значение:

```
def add_numbers(x, y):
    return x + y
result = add_numbers(5, 3)
print(result)  # Выведет: 8
```

Лямбда-функции в Python

Лямбда-функции в Python - это маленькие анонимные функции, которые объявляются с помощью ключевого слова lambda. Они могут принимать любое количество аргументов, но могут иметь только одно выражение. Лямбда-функции могут использоваться везде, где требуются объекты функций. Они очень полезны в качестве входных данных для функций высшего порядка, которые принимают функции в качестве аргументов, таких как map() и filter().

Вот пример использования лямбда-функции:

```
# Определение лямбда-функции
multiply = lambda x, y: x * y
# Использование лямбда-функции
result = multiply(3, 4)  # Возвращает 12
```

Методы библиотеки requests

Библиотека Requests в Python является одной из неотъемлемых частей Python для выполнения HTTP-запросов к указанному URL. Будь то REST API или веб-скрапинг, запросы необходимо изучить для дальнейшего использования этих технологий. Когда кто-то делает запрос к URI, он возвращает ответ. Запросы Python предоставляют встроенные функции для управления как запросом, так и ответом.

Модуль запросов Python имеет несколько встроенных методов для выполнения HTTP-запросов к указанному URI с использованием запросов GET, POST, PUT, PATCH или HEAD. HTTP-запрос предназначен либо для извлечения данных из указанного URI, либо для отправки данных на сервер. Он работает как протокол запрос-ответ между клиентом и сервером. Метод GET отправляет закодированную информацию о пользователе, прикрепленную к запросу страницы. Страница и закодированная информация разделяются символом «?». Например:

```
import requests
response = requests.get('https://api.github.com/')
print(response.url)
print(response.status_code)
```

Методы ответа на запрос к сайту

Метод	Описание
response.headers	response.headers возвращает словарь заголовков ответа.

response.encoding	response.encoding возвращает кодировку, используемую для декодирования response.content.
response.elapsed	response.elapsed возвращает объект timedelta со временем, прошедшим с момента отправки запроса до получения ответа.
response.close()	response.close() закрывает соединение с сервером.
response.content	response.content возвращает содержимое ответа в байтах.
response.json()	response.json() возвращает объект JSON результата (если результат был записан в формате JSON, в противном случае возникает ошибка).
response.url	response.url возвращает URL ответа.
response.text	response.text возвращает содержимое ответа в юникоде.

Библиотека PyOD

PyOD - это инструментарий на базе Python для выявления нестандартных объектов в данных с использованием как неконтролируемых, так и контролируемых подходов. Он стремится предоставить унифицированный API для различных алгоритмов обнаружения аномалий. Обнаружение аномалий («выбросов») в большинстве случаев осложняется отсутствием соответствующих меток для таких данных и поэтому представляет собой задачу обучения без учителя. Библиотека включает следующие классы алгоритмов:

- линейные модели, в частности PCA и одноклассовая SVM;
- модели на основе близости, измеряющие расстояния между элементами данных: данные, расположенные близко друг к другу, с большей вероятностью являются нормальными, а данные, расположенные далеко, с большей вероятностью являются аномальными;
- вероятностные модели, использующие статистические распределения для выявления провалов;
- ансамблевые модели, использующие ансамбли моделей для выявления изолированных точек (один из таких алгоритмов - Isolation Forest);
- нейронные сети: автоэнкодеры, в том числе вариативные, могут быть обучены распознавать аномалии в неразмеченных данных.

Например:

```
# обучение детектора COPOD
from pyod.models.copod import COPOD
clf = COPOD()
clf.fit(X_train)
# получить оценки за выбросы
y_train_scores = clf.decision_scores_ # необработанные оценки выбросов
```

```
y_test_scores = clf.decision_function(X_test) # оценки за выбросы
```

Самостоятельное задание

1. Скачать json с 2000 вакансий по специальности Разработчик применяя api.hh.ru, без ограничения региона, с указанием зп (параметр запроса only_with_salary=True). Нужно использовать метод **REQUESTS.GET**.
2. На основе полученного json создать таблицу Pandas DataFrame df из полей - id, name, area.name, salary.from, salary,to, salary.gross, salary.currency, snippet.requirements, experience.name. Можно использовать метод **PD.JSON_NORMALIZE**
3. создать новое поле clean_sal, усредняющее предлагаемую в вакансии зп, на основе известных сумм "от" и "до".
4. с использованием boxplot определить наличие аномальных значений в полученном столбце. Можно использовать **MATPLOLIB.PYPLOT.BOXPLOT**
5. с использованием библиотеки pyod, выбрать 3 метода без учителя, определения аномалий, и на основе их результатов, добавить в датафрейм df поля - anomaly1, anomaly2, anomaly3.
6. с использованием голосования большинством получить столбец anomaly, по 3-м полям выше
7. для каждой из аномалий, ориентируясь на название вакансии, требуемый опыт, регион, требования к вакансии, получить объяснение и в виде категории с кратким текстом, описывающим ее суть, добавить эти объяснения отдельным столбцов в датафрейм. Можно использовать метод **DF.MAP**