Тема 2.2.1. Работа с данными в форматах CSV и JSON.

Формат CSV (Comma Separated Values – значения, разделенные запятыми)

Создадим или скачаем из сети файл, в котором хранятся данные по месяцам о минимальных и максимальных значениях температур какого-либо населенного пункта, например, города N:

“месяц год”,“min”,“max”

“январь 2021”,-34,20

“февраль 2021”,-16,8

“март 2021”,-12,10

“апрель 2021”,-8,15

“май 2021”,-3,22

“июнь 2021”,8,25

“июль 2021”,18,32

“август 2021”,15,24

“сентябрь 2021”,7,21

“октябрь 2021”,-3,16

“ноябрь 2021”,-22,-4

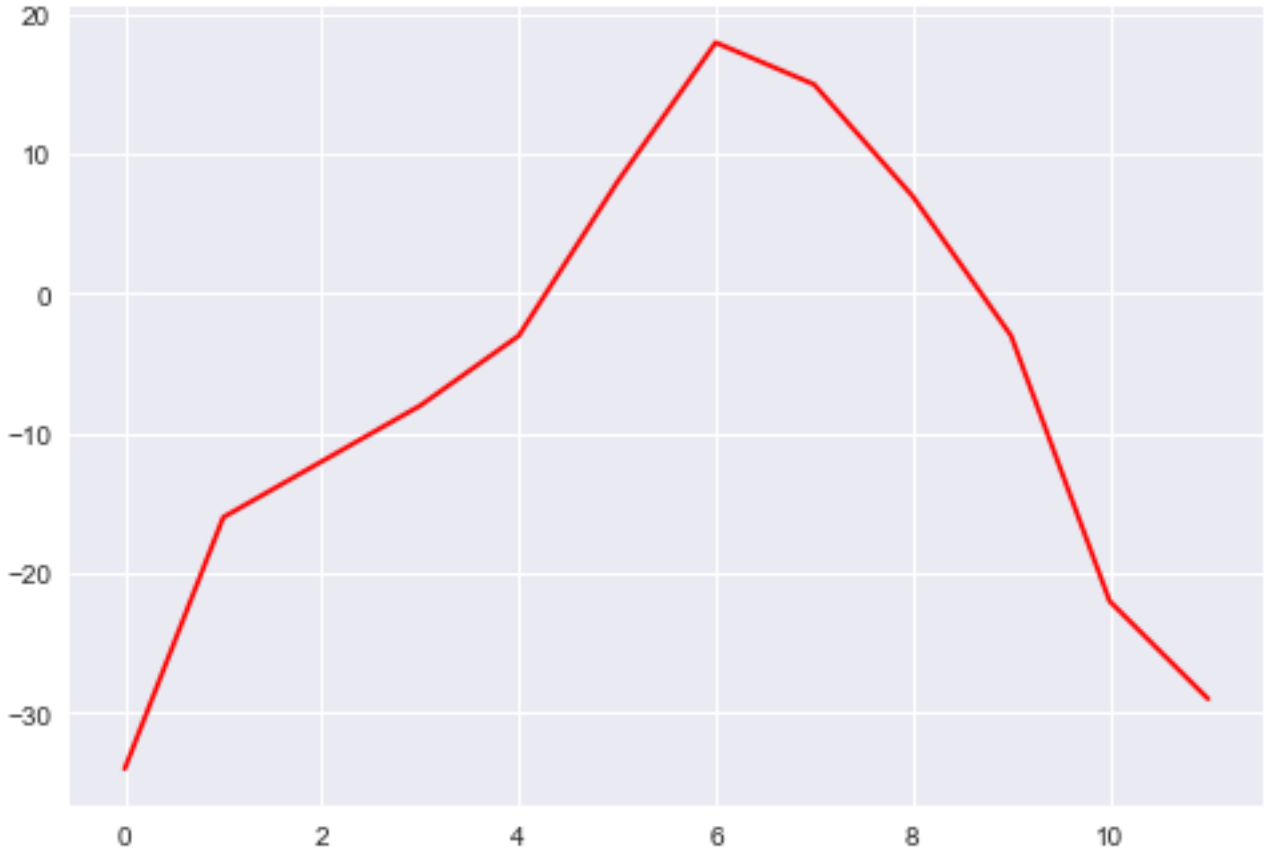
“декабрь 2021”,-29,18

На основе данных файла программно сформируем два массива для хранения данных о температуре минимальной (min_t) и максимальной (max_t):

```
import csv

filename = 'z1.txt'
min_t = []
max_t = []

with open(filename) as f:
    reader = csv.reader(f)
    header_row = next(reader)
    print(header_row)

    for row in reader:
        print(row)
        z = int(row[1])
        z1 = int(row[2])
        min_t.append(z)
        max_t.append(z1)



print(min_t)
print(max_t)
```

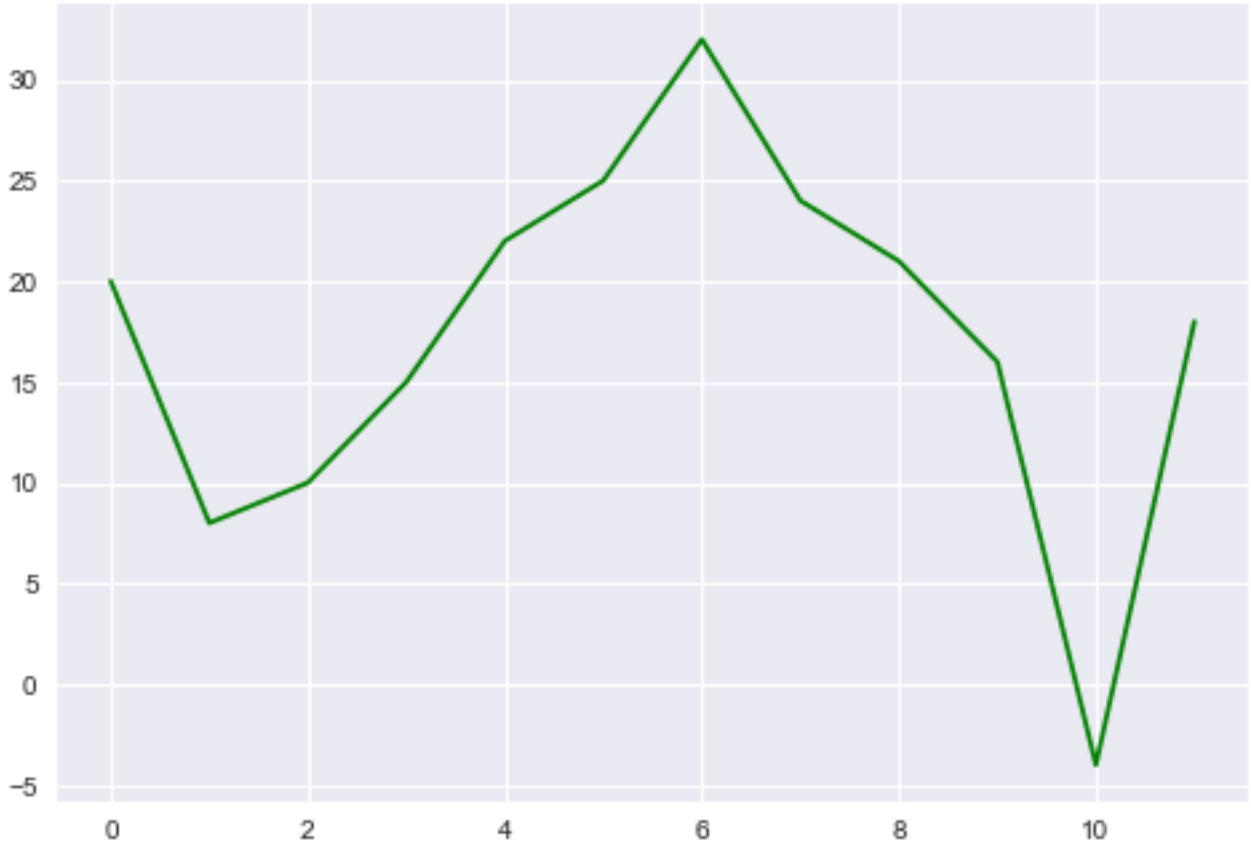
С использованием matplotlib создадим простую диаграмму для отображения минимальных температур:

```
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.plot(min_t, c = 'red')
```



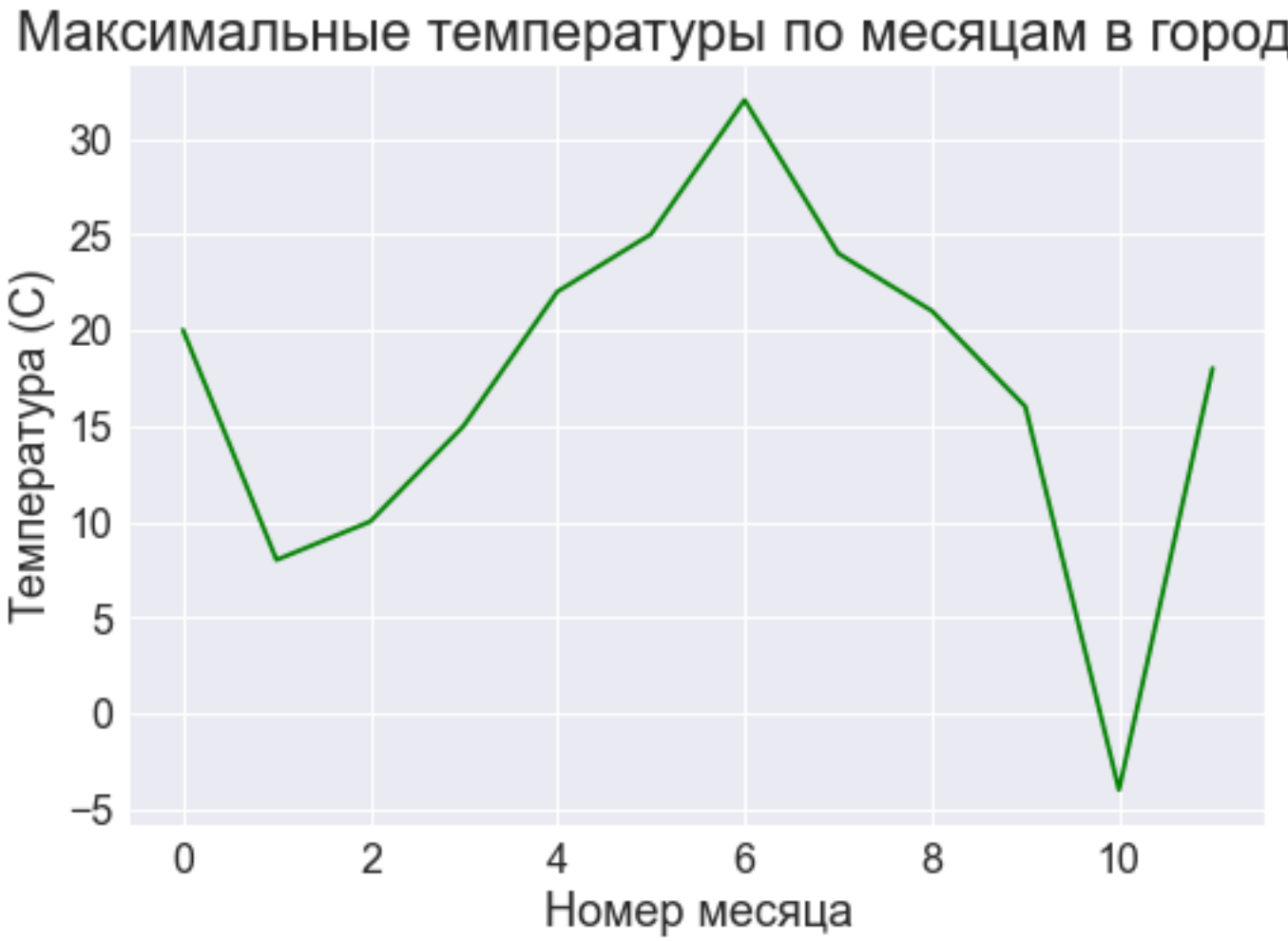
и максимальных температур:

```
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.plot(max_t, c = 'green')
```



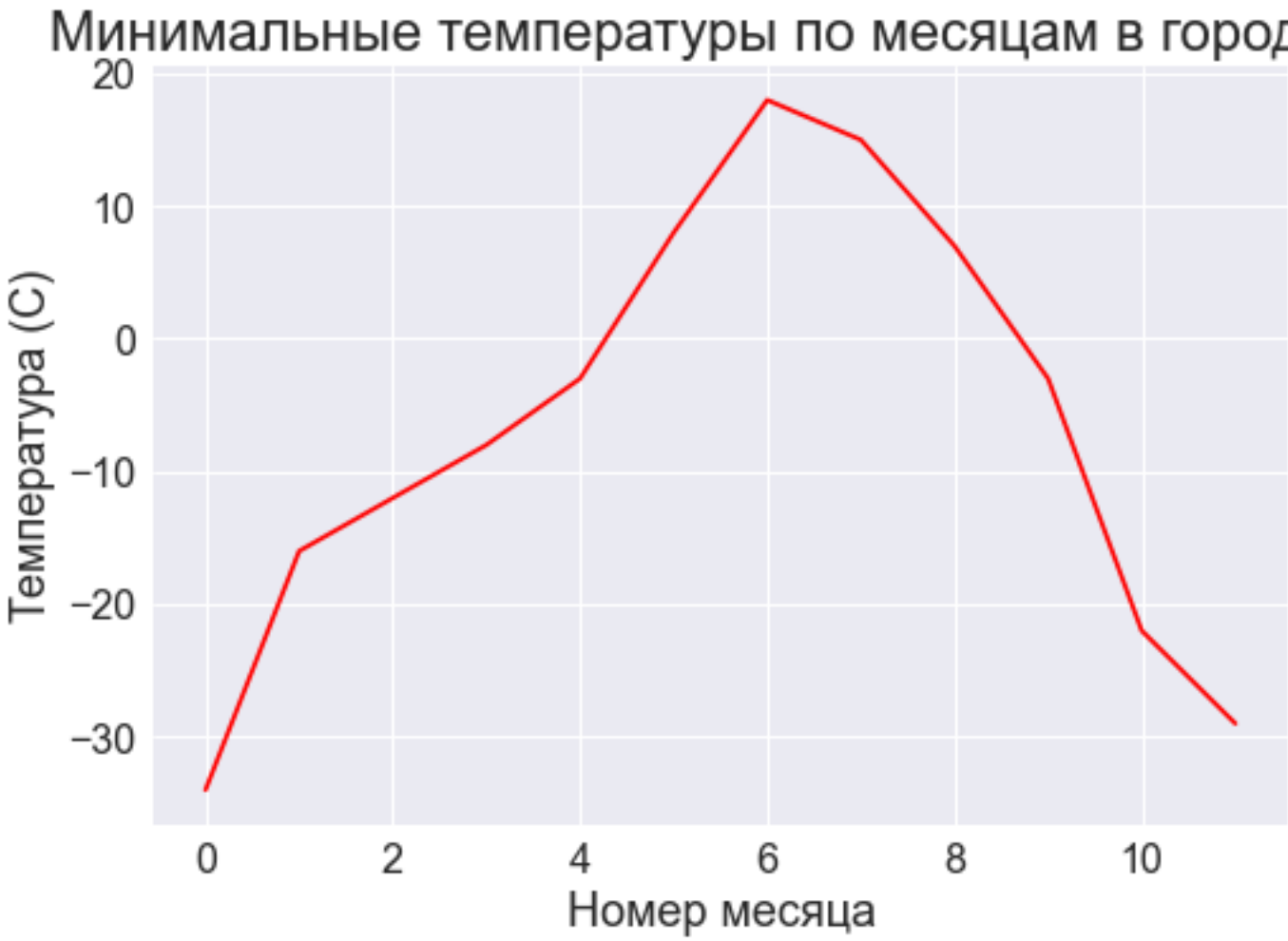
Отформатируем диаграмму для отображения максимальных температур:

```
plt.title('Максимальные температуры по месяцам в городе N', fontsize = 22)

plt.xlabel('Номер месяца' , fontsize = 18)

plt.ylabel('Температура (C)' , fontsize = 18)

plt.tick_params(axis = "both", which = "major", labelsize = 16)
```



И для отображения минимальных температур сделаем аналогично:

```
plt.title('Минимальные температуры по месяцам в городе N',
fontsize = 22)

plt.xlabel('Номер месяца' , fontsize = 18)

plt.ylabel('Температура (C)' , fontsize = 18)

plt.tick_params(axis = "both", which = "major", labelsize = 16)
```



Формат данных JSON

JSON (JavaScript Object Notation) - это текстовый формат, в основном применяется для передачи данных между сервером и веб-приложением.

JSON построен на двух структурах:

- Набор пар «имя-значение», которые могут быть реализованы как объект, запись, словарь, хеш-таблица, список «ключей-значений» или ассоциативный массив.
- Упорядоченный список значений, реализованный в виде массива, вектора, списка или последовательности.

JSON в Python

В Python для поддержки формата JSON используется модуль json.

Запись в формате JSON схожа с записью словарей Python:

```
my_json_string = """{
"book": [
{
"number": "3321",
"format": "JSON",
"author": "Jim Cati"
},
{
" number": "3322",
"format": "JSON",
"author": "Dim Doggi"
}
],
"blog": [
{
"name": "Datacamp",
"URL":"datacamp.com"
}
]
}
```

"""

Преобразование данных из формата JSON в словарь Python

Преобразование выполняется при помощи метода json.loads() из модуля json.

```
import json

my_json_string = """{

"book": [

{

"number": "3321",

"format": "JSON",

"author": "Jim Cati"

},

{

" number": "3322",

"format": "JSON",

"author": "Dim Doggi"

}

],

"blog":[
```

```
{
"name": "Datacamp",
"URL":"datacamp.com"
}
]
}
"""
to_python = json.loads(my_json_string)
to_python['blog']
[{'URL': 'datacamp.com', 'name': 'Datacamp'}]
print(to_python)
```

Преобразование объекта Python в JSON

Для этого используется json.dumps().

```
blog = {'URL': 'datacamp.com', 'name': 'Datacamp'}
to_json = json.dumps(blog)
to_json
'{"URL": "datacamp.com", "name": "Datacamp"}'
```

Сравнение топов данных в Python и JSON.

Python	JSON
dict	Object
list	Array
tuple	Array
str	String
int	Number
float	Number
True	true
False	false
None	null

Преобразование объектов Python в типы данных JSON.

Кортеж Python — в массив JSON

```
tuple_example = 'Jim', 'Nick', 'Alka'
print(json.dumps(tuple_example))
```

Список Python — в массив JSON

```
list_example = ["Jim", 7, 5, "Bet"]
```

```
print(json.dumps(list_example))
```

Строка Python — в строку JSON

```
string_example = "One,two..."
print(json.dumps(string_example))
```

Булевы значения Python — в булевы значения JSON

```
boolean_value = False
print(json.dumps(boolean_value))
```

Запись в файл JSON

Модуль json позволяет также записывать данные JSON в файл. Такие файлы сохраняют с расширением .json.

```
with open('test_file.json', 'w') as file:
    json.dump(my_json_string, file)
```

Чтение файлов JSON

Для загрузки файла вызовем json.load().

```
with open('test_file.json', 'r') as f:
    json_data = json.load(f)
print(json_data)
```

json.load vs json.loads

json.load используют для загрузки файла, а **json.loads** – для загрузки строки (**loads** расшифровывается как «load string»).

json.dump vs json.dumps

Аналогично, **json.dump** применяется, если нужно сохранить JSON в файл, а **json.dumps** (dump string) – если данные JSON нам нужны в виде строки.

Работа с данными JSON в Data Science

Иногда при работе над проектами, связанными с data science, требуется загрузить данные в формате JSON. Библиотека для анализа данных Pandas предоставляет для этого функцию .read_json.

Как только данные загружены, их можно конвертировать в объект dataframe при помощи атрибута pandas.DataFrame.

```
import pandas as pd
data = pd.read_json("https://api.github.com/users")
df = pd.DataFrame(data)
```

Ограничения имплементации

Процесс кодирования в JSON называется сериализацией, а декодирования – десериализацией.

Некоторые реализации десериализаторов имеют ограничения на:

- размер принимаемых текстов JSON
- максимальный уровень вложенности объектов и массивов JSON
- диапазон точности чисел JSON
- содержание и максимальную длину строк JSON.

И подобные ограничения связаны только с типами данных Python и работой самого интерпретатора Python.