Работа с excel.

Табличные форматы файлов.

Табличные форматы файлов — это форматы, предназначенные для хранения данных в виде таблиц. Они упрощают управление, анализ и обмен данными. К числу популярных табличных форматов файлов относятся:

Явные форматы файлов:

- **CSV (Comma-Separated Values)**: Формат, в котором данные разделены запятыми. Каждая строка представляет собой запись, а каждое значение в строке отделено запятой. CSV широко используется благодаря своей простоте и поддержке в большинстве приложений и языков программирования.
- **TSV (Tab-Separated Values)**: Формат, аналогичный CSV, но в котором значения разделены табуляцией. Он также прост в использовании и поддерживается многими инструментами для работы с данными.
- **Excel (XLS/XLSX)**: Форматы файлов, используемые Microsoft Excel. XLS — это старый бинарный формат, в то время как XLSX — современный формат, основанный на XML и поддерживающий больше функций. Эти форматы позволяют хранить сложные таблицы с форматированием, формулами, диаграммами и другими элементами.
- **ODS (OpenDocument Spreadsheet)**: Формат, используемый в OpenOffice и LibreOffice. Это открытый стандарт, основанный на XML, который поддерживает большинство функций, аналогичных Excel.

Неявные форматы файлов:

- **JSON (JavaScript Object Notation)**: Формат, который можно использовать для хранения табличных данных в виде объектов и массивов. Хотя JSON не является чисто табличным форматом, он часто используется для передачи структурированных данных, в том числе и табличных.
- **HTML (HyperText Markup Language)**: Таблицы в HTML часто используются для отображения данных на веб-страницах. Хотя это формат для разметки веб-страниц, он также может служить для представления табличных данных.

Графические примеры стандартной таблицы.

Таблица 4

Размер заработной платы учителя по регионам России, *тыс. руб.*

Регион	Средняя зарплата		
	По экономике региона, в целом	Учителей, сентябрь, 2015	Учителей, июль, 2016
Волгоградская область	19,0	16,8	17,2
Краснодарский край	16,5	18,3	21,4
Москва	39,0	42,8	56,6
Алтайский край	12,3	14,0	15, 0
Камчатский край	35,1	34,0	43,0
Санкт-Петербург	27,3	27,5	30,6

	A	B	C	D
1	Продукт	Кв. 1	Кв. 2	Общий ито
2	Шоколад	7 446,00 ₽	1 625,60 ₽	9 071,60 ₽
3	Мармелад	50 796,00 ₽	12 492,00 ₽	63 288,00 ₽
4	Багет	12 675,00 ₽	10 625,00 ₽	23 300,00 ₽
5	Булочки	14 180,00 ₽	7 560,00 ₽	21 740,00 ₽
6	Сахарный пирог	47 280,00 ₽	45 479,20 ₽	92 759,20 ₽
7	Шоколадное печенье	9 438,90 ₽	3 496,00 ₽	12 934,90 ₽
8	Всего	141 815,90 ₽	81 277,80 ₽	223 093,70 ₽

Формат CSV.

Формат CSV (Comma-Separated Values) является одним из наиболее популярных форматов для хранения и обмена табличными данными. В этом формате каждая строка файла представляет собой запись, а значения в каждой строке разделены запятыми (или другим разделителем, таким как точка с запятой или табуляция). CSV используется благодаря своей простоте и поддержке большинством языков программирования и инструментов для работы с данными.

Python предоставляет несколько библиотек для работы с CSV-файлами, включая стандартную библиотеку csv и сторонние библиотеки, такие как pandas.

Стандартная библиотека `csv` предоставляет основные инструменты для чтения и записи CSV-файлов.

Библиотека `pandas` предоставляет более мощные и удобные средства для работы с табличными данными, включая CSV-файлы.

CSV. Методы - Чтение/Запись.

- **csv.reader(file) -** Чтение CSV-файла построчно.
- **csv.writer(file) -** Запись в CSV-файл построчно.
- **csv.DictReader(file) -** Чтение CSV-файла в виде словарей.
- **csv.DictWriter(file, fieldnames) -** Запись в CSV-файл в виде словарей.
- **reader.next() -** Чтение следующей строки из CSV.
- **writer.writerow(row) -** Запись одной строки в CSV.
- **writer.writerows(rows) -** Запись нескольких строк в CSV.
- **dictreader.next() -** Чтение следующей строки как словаря.
- **dictwriter.writeheader() -** Запись заголовка в CSV.
- **dictwriter.writerow(rowdict) -** Запись одной строки-словаря в CSV.
- **dictwriter.writerows(rowdicts) -** Запись нескольких строк-словарей в CSV.

!!! метод проверки, что значение в определенном столбце является нормализованным числовым значением (а не строка или логический тип) - row[i].isdigit() !!!

CSV. Пример - чтение файла.

```
import csv


# Открываем CSV-файл для чтения
with open('data.csv', mode='r', newline='') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

CSV. Пример - запись в файл.

```
import csv

# Данные для записи
data = [
    ['Name', 'Age', 'City'],
    ['Alice', 30, 'New York'],
    ['Bob', 25, 'San Francisco'],
    ['Charlie', 35, 'Los Angeles']
]

# Открываем CSV-файл для записи
with open('output.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)
```

CSV. Пример - объединение 2-х .csv файлов в один.

Если оба CSV-файла имеют одинаковую структуру (одинаковые заголовки столбцов), вы можете просто прочитать содержимое обоих файлов и записать их в новый файл.

```
# Имена исходных и выходного файлов
file1 = 'file1.csv'
file2 = 'file2.csv'
output_file = 'combined.csv'

# Открываем файлы для чтения и записи
with open(file1, mode='r', newline='') as f1, open(file2, mode='r', newline='') as f2, open(output_file,
mode='w', newline='') as outf:
    reader1, reader2 = csv.reader(f1), csv.reader(f2)
    writer = csv.writer(outf)
    # Читаем и записываем заголовки из первого файла
    header = next(reader1)
    writer.writerow(header)
    # Копируем содержимое первого файла
    for row in reader1:
        writer.writerow(row)
    # Пропускаем заголовки второго файла
    next(reader2)
    # Копируем содержимое второго файла
    for row in reader2:
        writer.writerow(row)
```

CSV. Диалекты. Параметры диалектов.

диалекты представляют собой способ конфигурации параметров чтения и записи CSV-файлов. Диалекты позволяют настроить различные аспекты форматирования CSV, такие как символ-разделитель, символ для заключения строк в кавычки, способ экранирования символов и другие параметры. Это полезно для работы с CSV-файлами, которые могут иметь различные форматирования.

Основные параметры диалектов

- **delimiter**: Символ, используемый для разделения полей. По умолчанию — запятая (,).
- **quotechar**: Символ, используемый для заключения строк в кавычки. По умолчанию — двойная кавычка (").
- **escapechar**: Символ, используемый для экранирования.
- **doublequote**: Логическое значение, указывающее, удваиваются ли кавычки в строках. По умолчанию — True.
- **skipinitialspace**: Логическое значение, указывающее, пропускать ли пробелы после разделителя. По умолчанию — False.
- **lineterminator**: Строка, используемая для разделения строк. По умолчанию — '\r\n'.
- **quoting**: Константа, определяющая, как обрабатывать кавычки. Возможные значения: csv.QUOTE_ALL, csv.QUOTE_MINIMAL, csv.QUOTE_NONNUMERIC, csv.QUOTE_NONE.

CSV. Регистрация диалектов.

Процесс регистрации диалектов - позволяет создать и сохранить набор параметров, определяющих формат CSV-файлов. Особенно полезно, если вы работаете с различными CSV-файлами, которые могут иметь разные настройки форматирования (например, различные разделители, символы кавычек и т.д.). Регистрация диалекта упрощает повторное использование этих настроек без необходимости указывать их каждый раз.

Основные шаги процесса регистрации диалектов

1. **Определение параметров диалекта**: указываете параметры форматирования, такие как разделитель полей, символ кавычек, правила экранирования и т.д.
2. **Регистрация диалекта**: Используете функцию csv.register_dialect() для регистрации нового диалекта с указанными параметрами и присваиваете ему имя.
3. **Использование зарегистрированного диалекта**: При чтении или записи CSV-файлов вы можете ссылаться на зарегистрированный диалект по его имени.

CSV. Методы регистрации диалектов;

- **csv.field_size_limit(new_limit) -** Установка нового предела размера поля.
- **csv.get_dialect(name) -** Получение диалекта по имени.
- **csv.list_dialects() -** Список всех зарегистрированных диалектов.
- **csv.register_dialect(name, dialect) -** Регистрация нового диалекта.
- **csv.unregister_dialect(name) -** Удаление зарегистрированного диалекта.

```
import csv


# Получение зарегистрированного диалекта
dialect = csv.get_dialect('my_dialect')
print(dialect.delimiter)  # Выводит: ;


# Удаление зарегистрированного диалекта
csv.unregister_dialect('my_dialect')
```

CSV. Пример - регистрация нового диалекта.

```
import csv

# Регистрация нового диалекта
csv.register_dialect('my_dialect', delimiter=';', quotechar='"',
quoting=csv.QUOTE_ALL)

# Использование диалекта для записи
with open('output.csv', mode='w', newline='') as file:
    writer = csv.writer(file, dialect='my_dialect')
    writer.writerow(['Name', 'Age', 'City'])
    writer.writerow(['Alice', 30, 'New York'])
    writer.writerow(['Bob', 25, 'San Francisco'])
    writer.writerow(['Charlie', 35, 'Los Angeles'])
```

CSV. Пример - чтение с диалектами.

```
import csv


# Чтение CSV-файла с использованием зарегистрированного диалекта
with open('file.csv', mode='r', newline='') as file:
    reader = csv.reader(file, dialect='my_dialect')
    for row in reader:
        print(row)
```

PANDAS-CSV

Основные шаги работы с CSV-файлами в pandas

1. Чтение CSV-файлов в DataFrame
2. Объединение DataFrame'ов
3. Запись объединённого DataFrame в новый CSV-файл

Функционал pandas:

A. **Функция pd.read_csv** позволяет легко загрузить данные из CSV-файла в DataFrame.
B. **Функция to_csv** записывает содержимое DataFrame в CSV-файл.
C. **Функция pd.concat** используется для объединения DataFrame'ов. Она принимает список DataFrame'ов и объединяет их вдоль указанной оси (по умолчанию — вдоль оси строк, то есть сверху вниз).

Дополнительные возможности pandas:

A. **Фильтрация данных**: Вы можете фильтровать строки DataFrame на основе условий.
B. **Обработка отсутствующих значений**: Вы можете управлять отсутствующими значениями в DataFrame.
C. **Группировка данных**: Вы можете группировать данные по значениям одного или нескольких столбцов.
D. **Сортировка данных**: Вы можете сортировать строки DataFrame по значениям в столбцах.

PANDAS-CSV. Пример-1.

```
import pandas as pd


# Чтение CSV-файлов в DataFrame
df1 = pd.read_csv('file1.csv')
df2 = pd.read_csv('file2.csv')
# Объединение DataFrame'ов
combined_df = pd.concat([df1, df2])
# Запись объединённого DataFrame в новый CSV-файл
combined_df.to_csv('combined.csv', index=False)
```

PANDAS-CSV. Пример-2.

```
...
# Фильтрация строк, где значение в столбце 'Age' больше 30
filtered_df = combined_df[combined_df['Age'] > 30]
...
# Замена отсутствующих значений в столбце 'Age' на среднее значение
combined_df['Age'].fillna(combined_df['Age'].mean(), inplace=True)
...
# Группировка данных по столбцу 'City' и вычисление среднего значения столбца 'Age'
grouped_df = combined_df.groupby('City')['Age'].mean()
...
# Сортировка строк по значению в столбце 'Name'
sorted_df = combined_df.sort_values(by='Name')
...
```

Анализ данных. Графическое представление данных.

Шаги для построения графиков зависимостей

1. Чтение данных из CSV-файлов в DataFrame
2. Анализ данных и подготовка их для визуализации
3. Построение графиков с использованием `matplotlib` и `pandas`

```
# Чтение данных из CSV-файла
df = pd.read_csv('data.csv')
# Преобразование столбца 'Date' в формат datetime
df['Date'] = pd.to_datetime(df['Date'])

# Построение линейного графика
plt.figure(figsize=(10, 5))
plt.plot(df['Date'], df['Value'], marker='o')
plt.title('Зависимость Value от Date')
plt.xlabel('Date')
plt.ylabel('Value')
plt.grid(True)
plt.show()
```

Формат Excel.

Формат Excel (обычно файлы с расширениями .xls или .xlsx) является одним из наиболее распространенных форматов для хранения и обмена табличными данными. Он используется в основном программой Microsoft Excel и поддерживается многими другими программами для работы с электронными таблицами.

Плюсы Excel

1. **Богатый набор функций и формул**:
    - Excel поддерживает сложные вычисления и встроенные функции, такие как статистические, математические, логические и финансовые функции.
2. **Форматирование данных**:
    - Поддержка форматирования ячеек, таких как шрифты, цвета, стили границ, объединение ячеек и др.
3. **Многолистовые рабочие книги**:
    - Excel позволяет работать с несколькими листами в одной книге, что упрощает организацию данных.
4. **Графики и диаграммы**:
    - Встроенные инструменты для создания разнообразных графиков и диаграмм для визуализации данных.
5. **Встроенные таблицы и сводные таблицы**:
    - Поддержка создания и использования сводных таблиц для анализа больших объемов данных.
6. **Макросы и сценарии VBA**:
    - Возможность автоматизации задач с помощью макросов и Visual Basic for Applications (VBA).

Минусы Excel

1. **Больший размер файлов**:
    - Excel-файлы могут быть значительно больше по размеру по сравнению с CSV-файлами, особенно если они содержат сложное форматирование и графики.
2. **Сложность работы с большими объемами данных**:
    - Excel может быть медленным и менее производительным при работе с очень большими наборами данных (миллионы строк).
3. **Проприетарный формат**:
    - Формат Excel является собственностью Microsoft, что может вызывать проблемы совместимости с некоторыми программами или системами.
4. **Зависимость от конкретного ПО**:
    - Для полного использования всех возможностей Excel часто требуется Microsoft Excel или совместимое программное обеспечение.

Excel. Методы работы.

используется библиотека pandas. Методы:

Методы чтения и записи

1. **pd.read_excel()**: Чтение Excel файла в DataFrame.
2. **DataFrame.to_excel()**: Запись DataFrame в Excel файл.

Методы работы с листами

3. **ExcelFile.sheet_names**: Получение списка всех листов.
4. **pd.ExcelFile()**: Загрузка Excel файла с возможностью работы с несколькими листами.
5. **pd.read_excel(sheet_name='Sheet1')**: Чтение данных с указанного листа.

Методы работы с данными

6. **DataFrame.head()**: Вывод первых нескольких строк DataFrame.
7. **DataFrame.tail()**: Вывод последних нескольких строк DataFrame.
8. **DataFrame.describe()**: Статистическое описание DataFrame.
9. **DataFrame.info()**: Информация о DataFrame.
10. **DataFrame.shape**: Получение размера DataFrame (строки, столбцы).

Методы фильтрации и сортировки

11. **DataFrame.sort_values()**: Сортировка DataFrame по значению.
12. **DataFrame.filter()**: Фильтрация DataFrame по условиям.
13. **DataFrame.groupby()**: Группировка данных в DataFrame.

Excel. Методы работы.

Методы работы с ячейками и столбцами

14. **DataFrame.at[]**: Доступ к элементу по метке.
15. **DataFrame.iat[]**: Доступ к элементу по позиции.
16. **DataFrame.loc[]**: Доступ к строкам и столбцам по меткам.
17. **DataFrame.iloc[]**: Доступ к строкам и столбцам по позициям.

Методы объединения и изменения структуры

18. **pd.concat()**: Объединение DataFrame вдоль указанной оси.
19. **pd.merge()**: Слияние DataFrame на основе общих столбцов.
20. **DataFrame.pivot_table()**: Создание сводной таблицы.

Дополнительные методы для обработки данных

21. **DataFrame.fillna()**: Заполнение пропущенных значений.
22. **DataFrame.dropna()**: Удаление строк/столбцов с пропущенными значениями.
23. **DataFrame.replace()**: Замена значений в DataFrame.
24. **DataFrame.apply()**: Применение функции к элементам DataFrame.
25. **DataFrame.duplicated()**: Обнаружение дублирующихся строк.

Excel. Более сложные операции;

Более сложные операции с файлами Excel в Python можно выполнить с помощью библиотеки openpyxl, которая предоставляет низкоуровневые инструменты для работы с Excel-файлами, такие как доступ к ячейкам, листам, формулам и стилям.

сложные операции, которые можно выполнить с помощью openpyxl:

- **Создание нового Excel-файла**: Можно создать новую книгу Excel и добавить в неё листы, заполнив их данными.
- **Добавление новых листов**: В существующую книгу Excel можно добавить новые листы и работать с ними.
- **Работа с формулами**: Можно добавить формулы в ячейки и вычислить их значения.
- **Применение стилей**: Можно применять различные стили к ячейкам и диапазонам ячеек.
- **Обработка графиков и диаграмм**: Можно добавить и настроить графики и диаграммы в Excel.
- **и другие..**

Excel. openpyxl - создание нового файла excel.

```python
from openpyxl import Workbook

# Создание нового Excel-файла
wb = Workbook()

# Создание нового листа
ws = wb.active
ws.title = "Sheet1"

# Заполнение ячеек данными
ws['A1'] = 42
ws['B1'] = "Hello"

# Сохранение книги (excel файла)
wb.save("new_excel_file.xlsx")
```

Excel. openpyxl - добавление новых листов.

```
from openpyxl import load_workbook

# Загрузка существующего Excel-файла
wb = load_workbook("existing_excel_file.xlsx")

# Добавление нового листа
new_ws = wb.create_sheet("NewSheet")

# Запись данных в новый лист
new_ws['A1'] = "Data"

# Сохранение изменений
wb.save("existing_excel_file.xlsx")
```

Excel. openpyxl - работа с формулами.

```
from openpyxl.utils import FORMULAE


# Добавление формулы в ячейку
ws['C1'] = "=SUM(A1:B1)"


# Получение значения ячейки с формулой
value = ws['C1'].value


# Вычисление значения формулы
ws['C1'] = ws['C1'].value


# Сохранение изменений
wb.save("existing_excel_file.xlsx")
```

Excel. openpyxl - работа со стилями.

```
from openpyxl.styles import Font, Alignment

# Применение стиля к ячейке
ws['A1'].font = Font(bold=True, color="FF0000")
ws['A1'].alignment = Alignment(horizontal="center", vertical="center")

# Применение стиля к диапазону ячеек
for row in ws.iter_rows(min_row=2, max_row=10, min_col=1, max_col=3):
    for cell in row:
        cell.font = Font(italic=True)

# Сохранение изменений
wb.save("existing_excel_file.xlsx")
```

Excel. openpyxl - работа с графиками.

```
from openpyxl.chart import BarChart, Reference

# Создание объекта диаграммы
chart = BarChart()
chart.type = "col"

# Данные для диаграммы
data = Reference(ws, min_col=2, min_row=1, max_row=5, max_col=3)
chart.add_data(data, titles_from_data=True)

# Добавление диаграммы в лист
ws.add_chart(chart, "E1")

# Сохранение изменений
wb.save("existing_excel_file.xlsx")
```

Google Sheets Excel.

Google Sheets - это веб-приложение для создания, редактирования и совместной работы с электронными таблицами, разработанное компанией Google. В отличие от традиционных программ Excel, Google Sheets доступен в браузере и не требует установки на компьютер. Он предоставляет возможность создания таблиц, редактирования данных, настройки форматирования, использования формул, создания графиков и многое другое.

возможности:

- **Веб-доступность**: Google Sheets доступен через браузер на любом устройстве с подключением к интернету.
- **Совместная работа**: Несколько пользователей могут работать с одним документом одновременно, видя изменения в реальном времени.
- **Облачное хранение**: Все таблицы сохраняются в облаке Google Drive, что обеспечивает доступ к ним с любого устройства.
- **Редактирование на лету**: Данные обновляются автоматически при изменениях, позволяя пользователям работать над таблицами одновременно.
- **Формулы и функции**: Google Sheets поддерживает широкий набор функций и формул, аналогичный тем, что доступен в Excel.
- **Графики и диаграммы**: Встроенные инструменты для создания различных видов графиков и диаграмм для визуализации данных.
- **Импорт и экспорт файлов**: Возможность импорта и экспорта файлов в различных форматах, включая Excel, CSV и другие.

Удаленное изменение таблиц.

В Python удаленное изменение Google Sheets можно выполнить с использованием библиотеки `gspread`, которая предоставляет удобный интерфейс для работы с Google Sheets через API.

установка: pip install gspread oauth2client

возможности:

- **Авторизация в Google Sheets API**;
- **Выбор таблицы для редактирования**;
- **Изменение ячеек**;
- и другие

Авторизация через Google Sheets API

```
import gspread
from oauth2client.service_account import ServiceAccountCredentials

# Указываем путь к файлу JSON с учетными данными
credentials =
ServiceAccountCredentials.from_json_keyfile_name('credentials.json',
['https://spreadsheets.google.com/feeds'])

# Авторизация
client = gspread.authorize(credentials)
```

Выбор таблицы/Изменение ячеек;

```
# Открываем таблицу по её названию
sheet = client.open('Название таблицы').sheet1
# Изменяем значение ячейки A1
sheet.update_cell(1, 1, 'Новое значение')

# Изменяем значение диапазона ячеек
sheet.update('B2:C5', [['Значение B2', 'Значение C2'],
                       ['Значение B3', 'Значение C3'],
                       ['Значение B4', 'Значение C4'],
                       ['Значение B5', 'Значение C5']])
```

Добавление/Удаление данных;

```
# Добавляем новую строку данных
sheet.append_row(['Новое значение столбца A', 'Новое значение столбца B',
'Новое значение столбца C'])
# Удаляем строку с указанным индексом (нумерация с 1)
sheet.delete_row(2)

# Удаляем столбец с указанным индексом (нумерация с 1)
sheet.delete_column(3)
```

чтение данных;

```
# Получаем значение ячейки
value = sheet.cell(1, 1).value

# Получаем данные из диапазона ячеек
data = sheet.get('A1:C5')

# Получаем все данные из таблицы
all_data = sheet.get_all_values()
```

Дополнительные операции;

```
# Получаем список листов в таблице
sheets_list = client.open('Название таблицы').worksheets()

# Создаем новый лист
new_sheet = client.open('Название таблицы').add_worksheet('Новый лист', 100, 20)

# Удаляем лист
client.open('Название таблицы').del_worksheet(new_sheet)
```

общий пример - обновление данных таблицы;

```
import gspread
from oauth2client.service_account  import ServiceAccountCredentials

# Указываем путь к файлу JSON с учетными данными
credentials  = ServiceAccountCredentials. from_json_keyfile_name ('credentials.json' ,
['https://spreadsheets.google.com/feeds' ])

# Авторизация
client  = gspread.authorize(credentials)

# Открываем существующую таблицу по её названию
existing_sheet  = client.open('Существующая таблица' )

# Получаем доступ к нужному листу
worksheet  = existing_sheet.sheet1

# Изменяем значение ячейки
worksheet.update_cell(1, 2, 'Новое значение' )
```