Изменение типов данных

Pandas (Dataset)

Чтение таблицы из файла Excel

In
```
df = pd.read_excel('file.xlsx', sheet_name='Лист 1')

# первый аргумент – строка с именем файла
# второй аргумент (sheet_name) – имя листа
```

In
```
df = pd.read_excel('file.xlsx')
# если второй аргумент пропущен, то будет прочитан первый по счёту лист
```

Слияние двух датасетов

In
```
data.merge(d, on, how)
# d   – датасет, с которым сливают
# on  – колонка, по значениям которой сливают
# how – тип слияния:

data.merge(data2, on='merge_column', how='left')
## left – обязательно присутствуют все значения из таблицы data,
##        вместо значений из data2 могут быть NaN

data.merge(data2, on='merge_column', how='right')
## right – обязательно присутствуют все значения из таблицы data2,
##         вместо значений из data могут быть NaN
```

Формирование сводной таблицы

In
```
data_pivot = data.pivot_table(index = ['column1', 'column2'], columns = 'source',
values = 'column_pivot', aggfunc = 'function')

# index   – столбец или столбцы, по которым происходит группировка данных
# columns – столбец по значениям которого будет происходить группировка
# values  – значения, по которым мы хотим увидеть сводную таблицу
# aggfunc – функция, которая будет применяться к значениям
```

Pandas (Column)

Перевод значений столбца из строкового типа str в вещественный тип float

In
```
pd.to_numeric(data['column'])
# первый аргумент – колонка из датафрейма
# второй аргумент (errors) – метод обработки ошибок


pd.to_numeric(data['column'], errors='raise' )
# если errors='raise' (значение по умолчанию), то при встрече с некорректным
# значением выдается ошибка, операция перевода в числа прерывается;

pd.to_numeric(data['column'], errors='coerce')
# если errors='coerce', то некорректные значения принудительно заменяются на NaN;

pd.to_numeric(data['column'], errors='ignore')
# если errors='ignore', то некорректные значения игнорируются, но остаются.
```

In
```
data['column'] = pd.to_numeric(data['column'])
# Возвращает новую колонку, не заменяя предыдущую.
# Для замены, нужно выполнить присваивание.
```

Перевод значений столбца в другой тип данных

In
```
data['column'].astype('type') # например int для целых чисел, а str для строк
```

In
```
data['column'] = data['column'].astype('type')
# Возвращает новую колонку, не заменяя предыдущую.
# Для замены, нужно выполнить присваивание.
```

Перевод из строки в дату и время

In
```
pd.to_datetime(data['date_time_column'], format='%d.%m.%Y %H:%M:%S')
# обязательный второй агрумент – строка формата

'''
Формат строится с использованием следующих обозначений для частей даты и времени:
• %d – день месяца (от 01 до 31)
• %m – номер месяца (от 01 до 12)
• %Y – год с указанием столетия (например, 2019)
• %H – номер часа в 24-часовом формате
• %I – номер часа в 12-часовом формате
• %M – минуты (от 00 до 59)
• %S – секунды (от 00 до 59)
'''

# например, если даты выглядят так:
20.03.2017 11:00:50    # то формат:
'%d.%m.%Y %H:%M:%S'
```

In
```
data['date_time_column'] = pd.to_datetime(data['date_time_column'],
format='%d.%m.%Y %H:%M:%S')
# Возвращает новую колонку, не заменяя предыдущую.
# Для замены, нужно выполнить присваивание.
```

Получение отдельных частей даты и времени

In
```
# Получение из столбца с датой и временем...
pd.DatetimeIndex(data['time']).year   # года
pd.DatetimeIndex(data['time']).month  # месяца
pd.DatetimeIndex(data['time']).day    # дня
pd.DatetimeIndex(data['time']).hour   # часа
pd.DatetimeIndex(data['time']).minute # минуты
pd.DatetimeIndex(data['time']).second # секунды
```

Python

Обработка исключений

In
```
try:
    ... # код, где может быть ошибка

except:
    ... # действия если возникла ошибка
```

Словарь

Unix time (формат времени)
количеством секунд, прошедших с момента 00:00:00 1 января 1970 года

Сводная таблица
инструмент обработки данных для их обобщения