Удаление строк и столбцов DataFrame

```
df.drop([col1, col2], axis=1, inplace=True)
```

Математические операции со столбцами

`df[col1]+df[col2]`	Поэлементное сложение
`df[col1]-df[col2]`	Поэлементное вычитание
`df[col1]*df[col2]`	Поэлементное умножение
`df[col1]/df[col2]`	Поэлементное деление
`df[col1]**n`	Поэлементное возведение в степень *n*

Формат datetime

```
#год, месяц, день, час, минута, секунда
YYYY-MM-DD HH:MM:SS
```

Приведение столбца к формату datetime

```
df[date] = pd.to_datetime(df[date])
```

Основные атрибуты аксессора dt для datetime

`date`	Дата
`year`	Год
`month`	Месяц
`day`	День месяца
`dayofweek`	Номер дня недели (от 0 до 6)

`weekday_name`	Название дня недели (от Monday до Sunday)
`quarter`	Квартал (интервал в три месяца)
`time`	Время
`hour`	Час
`minute`	Минуты
`second`	Секунды

Выделение временных составляющих из формата datetime

```
df['Year'] = df[date].dt.year
```

Интервалы между датами (формат timedelta)

```
df['dur'] = df['start']-df['stop']
```

Основные атрибуты аксессора dt для timedelta

`days`	Интервал в днях
`seconds`	Интервал в секундах

Применение функций к столбцам DataFrame

```
def function(arg):
   ....
   return processed arg
df[col].apply(function)
```

Применение функций с несколькими аргументами к столбцам DataFrame

```
# первый аргумент – элемент столбца, остальные передаются в параметре args метода apply.

def function(a, b, c):
   ....
   return processed arg
df[col].apply(simple, args=(10, 15))
```

Применение lambda-функций к столбцам DataFrame

```
df[col].apply(lambda arg: processed_arg ....)
```

Категориальные и числовые признаки

Числовые	Отражают измеримую или количественную меру. Могут иметь неограниченный набор значений. *Примеры: масса, площадь, цена, координата*
Категориальные	Отражают принадлежность к определённой категории. Имеют ограниченный набор значений. *Примеры: национальность, страна, уровень образования*

Преобразование к типу данных category

```
df[col] = df[col].astype('category')
```

Достоинства и недостатки category

Достоинства	Недостатки
Значительное уменьшение объёма памяти, занимаемого таблицей	Необходимость в дополнительном преобразовании

Повышение производительности при работе со столбцами	Есть вероятность потерять данные при обновлении категорий таблицы

Best Practice: уменьшение числа категорий

```
popolar_values = df[col].value_counts().nlargest(n).index
df[col].apply(lambda x: x if x in popular_values else: 'other')
```