Глава 2. Исследовательский анализ данных

В этой Главе мы обсудим с вами действия, с которых нужно начинать анализ данных перед тем, как применять к ним простые и сложные алгоритмы машинного обучения.

В качестве набора данных мы подобрали набор *Cars*. Набор данных представляет собой статистику параметров автомобилей на вторичном рынке. Набор включает ряд категориальных и численных значений, составляющих одну запись (строку). Каждый столбец в записи – это отдельный признак. Среди указанных признаков приведены целевой для задачи предсказания (регрессии) – цена автомобиля. Также среди параметров есть целевой для задачи классификации – тип трансмиссии. Последняя задача может быть рассмотрена, например, как пример задачи на заполнение пропусков (если продавец не указал соответствующий параметр).

Для начального анализа данных крайне рекомендуется использовать библиотеку Pandas [11].

Библиотека Pandas для анализа данных

Для начала работы с любой библиотекой необходимо импортировать её

```
import pandas as pd
```

Теперь давайте загрузим данные в структуру Pandas датафрейм (Dataframe). Эта структура представляет собой двумерную структуру данных с именными столбцами потенциально разных типов. Вы можете думать об этом как о файле Excel, но на языке Python.

Считывание файлов в датафрейм

Для загрузки данных мы можем использовать функцию `pd.read_csv(path,delimiter)`. Для успешного использования необходимо указать путь (`path`) к файлу. В общем случае путь это строка, содержащий полный путь к файлу и название файла.

Мы также можем указать разделитель (`delimiter`), в нашем случае это «,». Но для разных файлов могут использоваться и другие типы разделителей,

такие как «;», « » (пробел), «\t» (табуляция). Вы можете проверить используемый в файле разделитель, предварительно открыв его в текстовом редакторе, например Notepad++.

Так следующая строка загрузит файл с названием `cars.csv`, который лежит в папке `/content/`, разделитель – запятая.

```
df = pd.read_csv('/content/cars.csv', delimiter = ',')
```

Для того чтобы визуализировать содержание загруженного датафрейма в блокнотах Google Colab достаточно просто запустить отдельную ячейку, в которой прописана переменная, которая содержит датафрейм. В других случаях можно воспользоваться функцией `display`. На представлена Рис. 2-1 визуализация датафрейма набора Cars.

	Make	Model	Year	Style	Distance	Engine_capacity(cm3)	Fuel_type	Transmission	Price(euro)
0	Toyota	Prius	2011	Hatchback	195000.0	1800.0	Hybrid	Automatic	7750.0
1	Renault	Grand Scenic	2014	Universal	135000.0	1500.0	Diesel	Manual	8550.0
2	Volkswagen	Golf	1998	Hatchback	1.0	1400.0	Petrol	Manual	2200.0
3	Renault	Laguna	2012	Universal	110000.0	1500.0	Diesel	Manual	6550.0
4	Opel	Astra	2006	Universal	200000.0	1600.0	Metan/Propan	Manual	4100.0
...	...	...	...	...	...	...	...	...	...
41002	Dacia	Logan Mcv	2015	Universal	89000.0	1500.0	Diesel	Manual	7000.0
41003	Renault	Modus	2009	Hatchback	225.0	1500.0	Diesel	Manual	4500.0
41004	Mercedes	E Class	2016	Sedan	50000.0	1950.0	Diesel	Automatic	29500.0
41005	Mazda	6	2006	Combi	370000.0	2000.0	Diesel	Manual	4000.0
41006	Renault	Grand Scenic	2006	Minivan	300000.0	1500.0	Diesel	Manual	4000.0

41007 rows × 9 columns

Рис. 2-1 Визуализация датафрейма набора Cars

Общая информация о датафрейме

Чтобы получить общую информацию о содержимом датафрейма. Для этого мы просто применяем метод `.info()` к нашему датафрейму. Вы должны увидеть следующую информацию (Рис. 2-2): количество столбцов, имена столбцов, тип данных в каждом столбце, количество пропущенных значений для каждого столбца.

```
RangeIndex: 41007 entries, 0 to 41006
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Make                  41007 non-null  object
 1   Model                 41007 non-null  object
 2   Year                  41007 non-null  int64
 3   Style                 41007 non-null  object
 4   Distance              41007 non-null  float64
 5   Engine_capacity(cm3)  41007 non-null  float64
 6   Fuel_type             41007 non-null  object
 7   Transmission          41007 non-null  object
 8   Price(euro)           41007 non-null  float64
dtypes: float64(3), int64(1), object(5)
memory usage: 2.8+ MB
```

Рис. 2-2 Информация о датафрейме набора Cars

Поиск и удаление дубликатов

Давайте разберемся с повторяющимися значениями в наборе данных. Повторяющиеся строки могут возникать по многим причинам: программная ошибка, человеческая ошибка и т. д. При этом никакой пользы от включения повторных строк не будет для алгоритмов машинного обучения.

Вы можете проверить количество повторяющихся строк, применив метод `.duplicated()` к датафрейму. В результате получится столбец, который содержит информацию о том, является ли данная строка повторной или нет. Чтобы узнать количество повторных строк мы можем применить метод `.sum()`

```
df.duplicated().sum()
```

Для датафрейма набора Cars имеется 3743 дубликата, почти 1/10 от всех данных. Чтобы удалить дубликаты, мы просто используем следующую строку

```
DF = df.drop_duplicates().reset_index(drop=True)
```

Позвольте мне объяснить, что здесь происходит:

- мы берем наш оригинальный датафрейм `df`;
- мы просим Python использовать метод удаления дубликатов `.drop_duplicates()`;
- мы просим сбросить индексы датафрейма `.reset_index(drop=True)`, в качестве альтернативы вы можете сохранить исходные индексы датафрейма;
- мы сохраняем результаты вышеупомянутого преобразования в новый датафрейм `DF` .

Хорошей практикой является размещение результата значительного преобразования в отдельном датафрейме, чтобы вы всегда могли вернуться к исходному датафрейму. Вы можете проверить, что все сделано правильно, оценив количество строк дубликатов в новом датафрейме.

Сохранение датафрейма в файл

Для сохранения датафрейма в файл можно воспользоваться методом `.to_csv(path, index)`, которому необходимо указать полный путь и название файла, в который вы хотите сохранить данные (`path`). Также можно указать, необходимо ли сохранять индексы строк (булева переменная `index`). Так следующая строка сохранит датафрейм без дубликатов в папку `/content/` с названием `'cars_ no_dup.csv'`

```
DF.to_csv('/content/cars_no_dup.csv',index=False)
```

Представление части датафрейма

Здесь стоит упомянуть два методах: `.head(n)` и `.tail(n)`. Эти два метода можно применить к нашему датафрейму для визуализации первых `n` или последних `n` строк. Это очень полезно, когда ваш датафрейм довольно большой и не может быть легко визуализирован полностью. Попробуйте сами: покажите первые 6, а затем последние 9 строки нашего датафрейма.

Индексация

Теперь поговорим об индексации. Одним из способов получения определенных элементов датафрейма является использование атрибута `.loc`

Ниже приведены некоторые из примеров

• взятие одной ячейки `DF.loc[1437,'Transmission']`

Этот код элемент из строки с индексом 1437 в столбце `'Transmission'`.

• взятие одной колонки в формате серий (Series) `DF.loc[:,'Transmission']`

Этот код вернет все элементы в столбце `'Transmission'`.

• взятие одной колонки в формате датафрейма `DF.loc[:,['Transmission']]`

• взятие нескольких колонок `DF.loc[:,['Transmission','Year']]`

Этот код вернет все элементы в столбцах `'Transmission'` и `'Year'`.

• взятие нескольких колонок подряд (среза столбцов) `DF.loc[:,'Make':'Style']`

Этот код вернет все элементы в столбцах, начиная с `'Make'` по `'Style'` (включительно)

Теперь поговорим о построчной индексации.

• взятие одной конкретной строки в формате серий `DF.loc[69,:]`

Этот код вернет все элементы из строки с индексом 69.

• взятие одной конкретной строки в формате датафрейма `DF.loc[69:69,:]`

• получение среза строк `DF.loc[322:1437,:]`

Этот код вернет все элементы в строках, начиная с индекса 322 и заканчивая индексом 1437 (включительно)

И, конечно же, вы можете комбинировать срезы по строкам и срезы по столбцам `DF.loc[227:229,'Make':'Fuel_type']`. Этот код вернет все элементы в строках, начиная с индекса 227 и заканчивая индексом 229 (включительно), и в столбцах с `'Make'` по `'Fuel_type'`.

Альтернативой использования атрибута `.loc` является использование атрибута `.iloc`. Основное различие между `.loc` и `.iloc` заключается в следующем: `.loc` работает с названиями столбцов, а `.iloc` использует вместо этого целочисленную нумерацию.

Так же доступно логическое индексирование (Boolean Indexing) когда мы хотим взять такую часть датафрейма, которая соответствует некому условию. Например строка `DF[DF['Transmission']=='Manual']` вернет все элементы исходного датафрейма, для которых выполняется условие что признак в столбце `'Transmission'` равен `'Manual'`.

Некоторые функции не могут работать с датафреймами напрямую, а рассчитаны на то, что данные подаются в формате массивов, например numpy

array. Чтобы перейти от датафреймов к массивам достаточно воспользоваться методом `.values` к необходимой части датафрейма.

Сортировка DataFrame

Для сортировки данных в датафрейме можно воспользоваться методом `.sort_values`. Вам также нужно указать столбец, по которому вы хотите отсортировать.

Например строка `DF.sort_values(by = 'Price(euro)')` отсортирует строки датафрейма по значениям столбца `'Price(euro)'`. По умолчанию метод `.sort_values` сортирует значения по возрастанию. Кроме того, вы можете указать «направление» сортировки, используя переменную `ascending`. Например, приведенный ниже код сортирует по убыванию столбца `'Year'`

```
DF.sort_values(by = 'Year', ascending= False)
```

Визуализация данных

Для визуализации данных можно использовать «стандартную» для Python библиотеку визуализации matplotlib [12; 13]. Однако при использовании датафреймов Pandas целесообразно пользоваться библиотекой Seaborn [14; 15]. Seaborn – это библиотека для создания разнообразной графики на Python, которая тесно интегрирована со структурами данных pandas.

```
import seaborn as sns
```

Один из лучших графиков для ознакомления с данными – это парный график (`pairplot`). Этот график отображает все возможные попарные комбинации числовых признаков наборах данных в виде скаттерограмм, а по диагонали строятся гистограммы распределений. Но будьте осторожны, если у вас много признаков, это может занять много времени, и может быть лучше разделить датафрейме на несколько частей. Категориальные признаки можно визуализировать используя, например, цвет данных. При этом seaborn

позволяет реализовать визуализацию достаточно просто, буквально в одну строчку:

```
sns.pairplot(data = DF, hue ='Transmission')
```

Для того что построить график на достаточно указать датафрейм из которого будут взяты численные признаки в переменную `data`, дополнительно можно указать в качестве переменной `hue` (цвет) один из столбцов с категориальными данными. На Рис. 2-3 представлен `pairplot` для набора данных Cars.

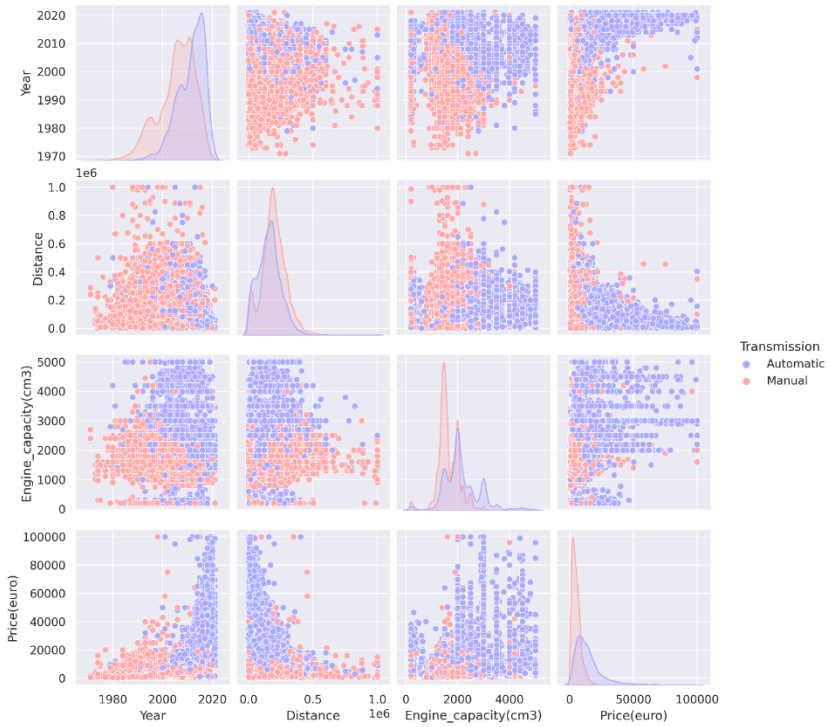

Рис. 2-3 Визуализация pairplot для набора данных Cars

Возможности библиотеки Seaborn для визуализации данных ограничены только вашим воображением. Ознакомиться с другими визуализациями Seaborn по следующей ссылке [16].

Предварительная обработка данных

После ознакомления с данными необходимо выполнить предварительную обработку данных. Эта обработка зависит от типа данных.

Для того, чтобы определить типы данных можно воспользоваться следующим кодом

```
cat_columns = []
num_columns = []

for column_name in df.columns:
    if (df[column_name].dtypes == object):
        cat_columns +=[column_name]
    else:
        num_columns +=[column_name]
```

В переменную `cat_columns` попадут названия все колонок с категориальными признаками, а в `num_columns` – с числовыми признаками. Обсудим методы предварительной обработки характерные для разных типов данных.

Предварительная обработка числовых данных

Начать анализ числовых данных стоит с оценки статистических показателей. В библиотеке Pandas это можно сделать в одну строчку с использованием метода `.describe()`

```
DF[num_columns].describe()
```

На Рис. 2-4 представлен результат применения метода `.describe()` к набору данных Cars

	Year	Distance	Engine_capacity(cm3)	Price(euro)
count	37264.000000	3.726400e+04	37264.000000	3.726400e+04
mean	2007.709264	4.758488e+05	1858.932535	9.569387e+03
std	8.295806	4.591520e+06	707.662731	5.283315e+04
min	1900.000000	0.000000e+00	0.000000	1.000000e+00
25%	2004.000000	9.000000e+04	1499.000000	3.300000e+03
50%	2009.000000	1.700000e+05	1800.000000	6.490000e+03
75%	2014.000000	2.300000e+05	2000.000000	1.179900e+04
max	2021.000000	1.000000e+08	9999.000000	1.000000e+07

Рис. 2-4 Результат применения метода `.describe()` к числовым признакам набора данных Cars

В результате демонстрируются следующие оценки: количество данных в столбце (**count**), среднее (**mean**), стандартное отклонение (**std**), минимальное значение (**min**), 25, 50 и 75 перцентили (**25%, 50%, 75%**), максимальное значение (**max**).

Сопоставление среднего значения, минимальных и максимальных значений, а также перцентилей позволяет оценить, а существуют ли в наших данных аномалии – редко встречающиеся значения. Также для этого пригодится визуализация гистограмм распределения. Иногда, как например для колонки `'Price(euro)'` числовые признаки могут изменяться в большом диапазоне: от 1 до 10^7 , хотя среднее значение, 25 и 75 перцентили сконцентрированы в диапазоне 10^3...10^4. Тогда для визуализации гистограммы распределения рекомендуется использовать логарифмический масштаб. Это говорит, как о аномально высоких и аномально низких значениях, которые встречаются крайне редко. А если какие-то признаки встречаются крайне редко, то на таких значениях сложно обучить модель с высокой степенью обобщения. Поэтому как правило от редких высоких и низких значений признаков избавляются.

При этом аномалии могут быть вызваны ошибками при заполнении данных. Так ошибочными выглядят значения столбца стоимость `'Price(euro)'` меньше 100, или старые автомобили с общим пробегом меньше, чем 1000 км.

В датафреймах Pandas это можно реализовать с использованием метода `.drop` и логической индексации. Ниже представлены рекомендуемые условия по удалению аномальных значений

```
# Старые автомобили с низким пробегом
question_dist_year = DF[(DF.Year <2021) & (DF.Distance < 1100)]
DF = DF.drop(question_dist_year.index)
# Аномально большой пробег
question_dist = DF[(DF.Distance > 0.5e6)]
DF = DF.drop(question_dist.index)
# Слишком малые значения объема двигателя
question_engine = DF [DF["Engine_capacity(cm3)"] < 200]
DF = DF.drop(question_engine.index)
# Слишком большие значения объема двигателя
question_engine = DF[DF["Engine_capacity(cm3)"] > 5000]
```

```
DF = DF.drop(question_engine.index)
# Аномально низкие цены
question_price = DF[(DF["Price(euro)"] < 101]
DF = DF.drop(question_price.index)
# Слишком дорогие автомобили, которых мало
question_price = DF[DF["Price(euro)"] > 1e5]
DF = DF.drop(question_price.index)
#Слишком старые автомобили, которых мало
question_year = DF[DF.Year < 1971]
DF= DF.drop(question_year.index)
```

С другой стороны видно что разные столбцы имеют разный разброс данных: год и объем двигателя измеряется в тысячах, стоимость измеряется в десятках тысяч, а пробег в сотнях тысяч. Интуитивно проще сопоставлять данные, если они измеряются в одних диапазонах. Для приведения численных данных к единой шкале существуют различные подходы: стандартизация, нормализация, степенное преобразование.

Стандартизация

Стандартизация, или z-нормировка, сводится к вычитанию из матрицы признаков $X = x_{ij} \in \mathbb{R}^{n \times p}$, вектора μ_j средних значений для каждого признака, и делению на вектор σ_j стандартного отклонений для каждого признаков

$$X' = \frac{(x_{ij} - \mu_j)}{\sigma_j},$$

Таким образом у новой матрицы признаков X' будет нулевое среднее и единичная дисперсия. При этом стандартизация – линейная операция. Т.е. распределение признаков не изменится. Изменится только масштаб в рамках которых это изменение происходит.

Чтобы реализовать стандартизацию на датафреймах Pandas достаточно применить методы `.mean()` и `.std()` для оценки среднего значения `M` и стандартного отклонения `STD` для каждого столбца.

```
M = DF[num_columns].mean()
STD = DF[num_columns].std()
```

Затем из исходного датафрейма происходит поэлементное вычитание и деление.

```
DF_scaled = (DF[num_columns]-M)/STD
```

На Рис. 2-5 представлено распределение признака `Distance` до и после операции стандартизации. Как видно из Рис. 2-5 распределение данных, остается тем же самым. Меняются границы диапазона измерений.

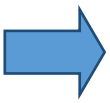

Рис. 2-5 Результат применения стандартизации к столбцу Distance набора данных Cars

Отдельно стоит отметить, что при выполнении стандартизации (как и других методов предварительной обработки) необходимо сохранять параметры преобразования, поскольку именно эти преобразования необходимо совершать на новых данных. Т. е. для тестового набора данных нужно вычитать среднее не тестового набора, а среднее тренировочного набора.

Нормализация

Нормализация – это тоже линейное преобразование численных признаков $X=x_{ij} \in \mathbb{R}^{n \times p}$. Но в отличие от стандартизации нормализация переводит распределение с интервал от 0 до 1.

$$X' = \frac{(x_{ij} - x_{j_{min}})}{(x_{j_{max}} - x_{j_{min}})}.$$

Как и в случае стандартизация нормализация также просто реализуется с использованием датафреймов Pandas. Используются методы `.min()` и `.max()` для поиска минимальных и максимальных значений по столбцам.

```
Xmin = DF[num_columns].min()
```

```
Xmax = DF[num_columns].max()
```

Затем аналогично из исходного датафрейма происходит поэлементное вычитание и деление.

```
DF_norm = (DF[num_columns]- Xmin)/( Xmax - Xmin)
```

На рисунке Рис. 2-6 представлено распределение признака `Distance` до и после операции нормализации. Аналогично, распределение данных, остается тем же самым. Меняются границы диапазона измерений.

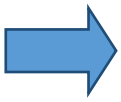

Рис. 2-6 Результат применения нормализации к столбцу Distance набора данных Cars

Степенное Преобразование

Стоит отметить, что корректное применение преобразований стандартизации и нормализации требует нормального распределения числовых данных. Однако такое происходит не всегда. Тогда рекомендуется предварительно использовать нелинейные преобразования (Power Transform), и уже потом результат нелинейного преобразования стандартизировать или нормализовать. К нелинейным преобразованиям относят логарифмирование и степенные преобразования (как правило это корни, т.е. степени меньше единицы).

Так на Рис. 2-7 представлено распределение признака `Price(euro)` до и после операции степенного преобразования. В конкретно этом примере мы сначала логарифмировали столбец, а потом применили стандартизацию.

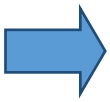

Рис. 2-7 Результат применения степенного преобразования к столбцу Price(euro) набора данных Cars

Стоит подчеркнуть, что заранее нельзя узнать какой из типов предварительной обработки данных лучше скажется на предсказаниях модели. Поэтому тип предварительной обработки данных можно рассматривать как дополнительный гиперпараметр модели.

Предварительная обработка категориальных данных

Анализ категориальных данных начинается с оценки частоты встречаемости отдельных категорий в рамках столбцов. Первым этапом необходимо воспользоваться методом `.nunique()` и оценить количество уникальных значений для каждой категории.

```
DF[cat_columns].nunique()
```

Для набора данных Cars столбцы `Make` и `Model` имеют 78 и 777 уникальных значений, соответственно.

Далее необходимо оценить частоту встречаемости уникальных значений с использованием метода `.value_counts()`

```
counts = DF.Make.value_counts()
```

Можно заметить, что для столбца `Make`, порядка 15 уникальных значений встречаются 10 и меньше раз. Для выборки из ~30000 объектов это слишком мало. Но в отличии от численных признаков, для которых мы удаляли редко встречающиеся значения, для категориальных признаков мы можем сделать замену на единый класс – редкий (`rare`). Для этого мы воспользуемся логическим индексированием и методом `.replace()`

```
rare =  counts[(counts.values < 25)]
DF['Make'] = DF['Make'].replace(rare.index.values, 'Rare')
```

Приведение категориальных признаков к числовым

Далее стоит обсудить формат хранения категориальных данных. В наборе данных Cars категориальные данные представляют собой тип данных `object`, и по сути являются строками. Для уменьшения объема хранимой информации достаточно часто эти данные переводят в числовой формат.

Если уникальных значений не много, то такой перевод удобно реализовать с помощью метода `.map()` и словаря. Так в строке ниже мы преобразуем столбец `Transmission`, в котором всего 2 уникальных значения.

```
DF['Transmission'] = DF['Transmission'].map({'Automatic': 1, 'Manual': 0})
```

Однако если уникальных значений много, то такое преобразование может быть затруднено. Поэтому рекомендуется воспользоваться изменением типа данных от `object` к `category`. А затем кодирование отдельных столбцов в числовые значения с помощью `.cat.codes`. Полный пример перехода к числовым значениям представлен ниже.

```
DF_ce = df.copy()
DF_ce[cat_columns] = DF_ce[cat_columns].astype('category')

for _, column_name in enumerate(cat_columns):
    DF_ce[column_name] =  DF_ce[column_name].cat.codes
```

Такое кодирование называют порядковым кодированием (Ordinal Encoding). Стоит отметить, что порядковое кодирование позволило сохранить объем данных примерно на треть: с 2.2 MB до 1.4 MB.

One-hot Кодирование

Существуют некоторые алгоритмы, которые способны работать с категориальными данными с порядковым кодированием. Это естественное кодирование порядковых переменных. Для категориальных переменных он налагает порядковое отношение там, где такого отношения может не быть. В общем случае использование этого кодирования и разрешение модели принимать естественное упорядочение между категориями может привести к

снижению производительности или неожиданным результатам (прогнозы на полпути между категориями).

Достаточно распространено использование One-hot кодирование (Encoding). Суть этого кодирование состоит в том, что вместо одного вектора $\boldsymbol{x}^{n\times 1}$ категориального признака создается матрица $\Xi^{n\times m}$, где m – количество уникальных категорий в векторе $\boldsymbol{x}$. При этом матрица Ξ состоит из только 0 и 1. Пример перевода категориального признака с использованием One-hot Encoding представлен на Рис. 2-8.

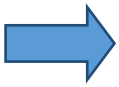

Рис. 2-8 Пример реализации One-hot кодирования для признака Цвет

Категориальный признак «цвет» состоит из 4 уникальных категорий. В результате One-hot кодирование создается 4 новых столбца, по одному на каждое уникальное значение цвета (красный, зеленый, синий, белый). При этом эти столбцы заполнятся в основном 0, а для тех индексов, которые соответствуют численному значению признака, ставится 1. По сути, One-hot кодирование преобразует вектора категориальных признаков в матрицы бинарных признаков. Это позволяет придавать хоть какое-то осмысление числовым значениям категориальных признаков. В библиотеке Pandas это кодирование реализуется с помощью метода `.get_dummies()`. При этом можно преобразовывать полный датафрейм. One-hot кодирование применится только к столбцам с категориальными данными

```
DF_ohe = pd.get_dummies(DF.copy())
```

Стоит отметить, что использование One-hot кодирования значительно увеличивает размер матрицы данных. Поэтому для признаков с большим количеством уникальных значений рекомендуется предварительно уменьшить количество категорий, объединив редкие категории в одну, как было показано ранее.

Инженерия признаков

Достаточно часто встречается, что с исходными признаками необходимо выполнить дополнительные преобразования, чтобы «помочь модели» выявить нужные закономерности. Как правило базовая инженерия признаков сводится к «умной» комбинации исходных признаков. Рассмотрим несколько примеров для набора данных Cars.

Есть столбец данных `Year` (год выпуска автомобилей). Но с точки зрения предсказания стоимости автомобиля сам год как таковой имеет не очень большое значение. Большей смысл несет «возраст» автомобиля, т.е. количество лет, которое прошло от года выпуска до настоящего времени.

```
DF['Age'] = 2022 - DF.Year
```

Другим кажущимся логичным признаком является средний пробег в год, т.е. насколько интенсивно автомобиль использовался в среднем за год. Это чуть более полезная информация, чем просто общий пробег.

```
DF['km_year'] = DF.Distance/DF.Age
```

Инженерия новых признаков – достаточно творческий процесс, и иногда требует чуть более глубокого погружения в предметную область. При этом можно проверять различные гипотезы, например, создать категориальный признак, связанный с объемом двигателя: если объем двигателя больше, допустим, 3 литров то его стоит пометить отдельной категорией. Это связано с тем, что автомобили с большим объемом двигателя меньше востребованы из-за большего расхода и повышенным налогообложением. Это тоже должно сказываться на цене.

При добавлении признаков в набор данных, на основе имеющихся, важно отслеживать, а не добавляем ли мы в модель «то же самое что уже есть».

Рекомендуется отслеживать коэффициент корреляции между признаков, и если он близок к единице, то какой-то из признаков рекомендуется удалить.

В библиотеке Pandas вычисление матрицы корреляции делается с использованием метода `.corr()`. Однако для лучшего восприятия информации добавим «раскраску» полученной результатов

```
cm = sns.color_palette("vlag", as_cmap=True)
DF.corr().style.background_gradient(cmap=cm, vmin = -1, vmax=1)
```

В результате вы должны получить раскрашенный датафрейм, похожий на Рис. 2-9

	Year	Distance	Engine_capacity(cm3)	Price(euro)	Age	km_year
Year	1.000000	-0.434034	-0.025707	0.551627	-1.000000	0.426025
Distance	-0.434034	1.000000	0.067378	-0.347236	0.434034	0.462777
Engine_capacity(cm3)	-0.025707	0.067378	1.000000	0.382831	0.025707	-0.010386
Price(euro)	0.551627	-0.347236	0.382831	1.000000	-0.551627	0.157024
Age	-1.000000	0.434034	0.025707	-0.551627	1.000000	-0.426025
km_year	0.426025	0.462777	-0.010386	0.157024	-0.426025	1.000000

Рис. 2-9 Матрица корреляция для набора данных Cars

Видно, что признаки `Age` и `Year` имеют 100 обратную корреляцию. Что не удивительно, ведь один признак – это константа минус второй признак. С другой стороны, корреляция между новым признаком `km_year` не превышает 0.5, что говорит о том, что этот признак может нести дополнительную информацию.

Практическое Задание к Главе 2

1. Скачайте набор данных Cars и используйте функции и методы библиотеки Pandas для загрузки и начальной работы с данными.
2. Выполните визуализацию данных с использованием библиотеки Pandas. Попробуйте разные виды графиков для числовых признаков – скатерограммы, гистограммы, и т. д.
   - Для скатерограмм попробуйте использовать категориальные данные для таких параметров графиков, как оттенок (`hue`), размер маркера (`size`),

тип маркера (`style`). Таким образом, вы можете объединить информацию о нескольких признаков в один двумерный график.

3. Выполните предварительную обработку данных. Сохраните результаты разных методов предварительной обработки в разные файлы, чтобы потом у вас была возможность протестировать различные гипотезы.
4. Попытайтесь добавить в модель дополнительные признаки на основе имеющихся. Проверьте корреляцию новых признаков с добавленными признаками.

Контрольные вопросы

1. Допустим у вас есть файл с данными, который называется ‘iris.csv’. Этот файл находится в папке ‘/data/’. Вы открываете его в текстовом редакторе и видите следующие первые строки

```
sepal length in cm; sepal width in cm; petal length in cm; petal width in cm; class
5.1; 3.5; 1.4; 0.2; 0
```

Какая должна выглядеть команда для считывания данных в датафрейм Pandas?

2. Набор данных Cars после удаления дубликатов. Выберете из полного датафрейма строки с индекса 69 по 322. Отсортируйте полученный датафрейм по колонке `'Distance'` по убыванию. Какое значение колонки `'Style'` у полученного датафрейма во второй строке сверху.
3. Набор данных Cars. Оцените количество строк, которые были удалены после анализа гистограмм распределения и удаления аномальных значений.
4. Набор данных Cars. Назовите самую распространенную марку автомобилей (столбец `Make`)
5. Набор данных Cars. Визуализируйте скатерограмму для двух столбцов `Distance` и `Year`, с использованием столбца `Transmission` в качестве цвета маркера (hue). К какому типу `Transmission` относится точка, которая наиболее близка к координатам (`Year` = 1980, `Distance` = 500 000)
6. Представим, что вы визуализировали некий набор данных (Рис. 2-10). Какие новые признаки, основанные на имеющихся, необходимо сконструировать чтобы иметь возможность отделить все красные точки от всех синих точек с помощью прямой линии.

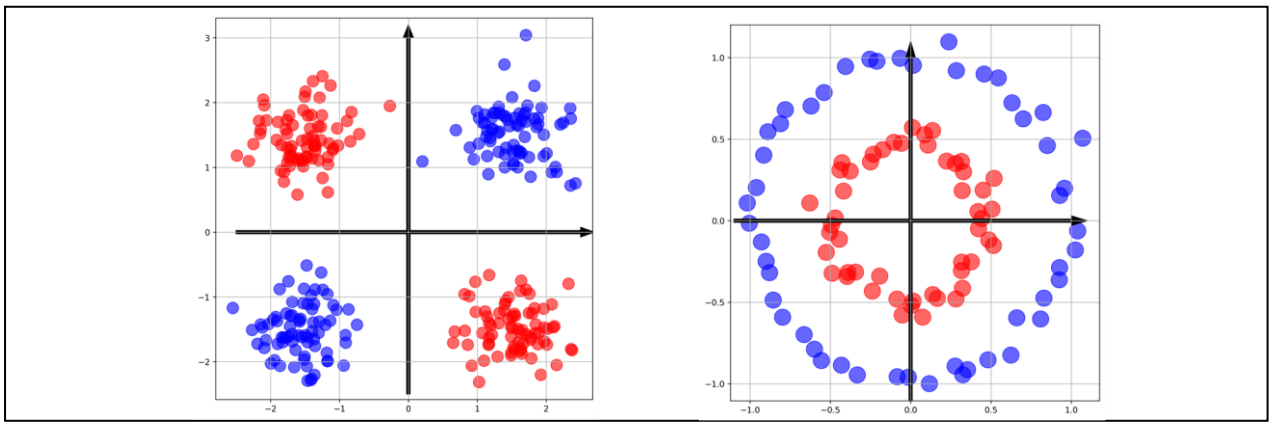

Рис. 2-10 Данные которые линейно разделяются с использованием грамотного конструирования признаков