Регрессионные модели

Модели обучения с учителем (или контролируемого обучения) бывают двух видов: *регрессионные* и *классификационные*. Регрессионные модели предсказывают числовые результаты, например цену, за которую будет продан дом, или количество посетителей на сайте. Его можно использовать для прогнозирования спроса, для проверки кредитных заявок, учитывая кредитные баллы, соотношение долга и дохода, соотношение стоимости кредита и займа. В любом случае, когда требуется получить числовые прогнозы, регрессионное моделирование - подходящий инструмент.

При построении регрессионной модели первым и наиболее важным решением является выбор алгоритма обучения. На примере построения классификационной модели использовали алгоритм k ближайших соседей для определения вида ириса по размерам чашелистиков и лепестков цветка.

К ближайших соседей можно использовать и для регрессии, но это один из многих алгоритмов, которые можно выбрать для построения численных прогнозов. Существуют и другие алгоритмы обучения.

Сегодня рассмотрим один из распространенных алгоритмов регрессии, многие из которых могут использоваться и для классификации. Рассмотрим процесс построения регрессионной модели, предсказывающей стоимость проезда в такси на основе данных из Kaggle taxi-fares.csv. Также ознакомимся различными способами оценки точности регрессионной модели и создадим перекрестную проверку.

Линейная регрессия

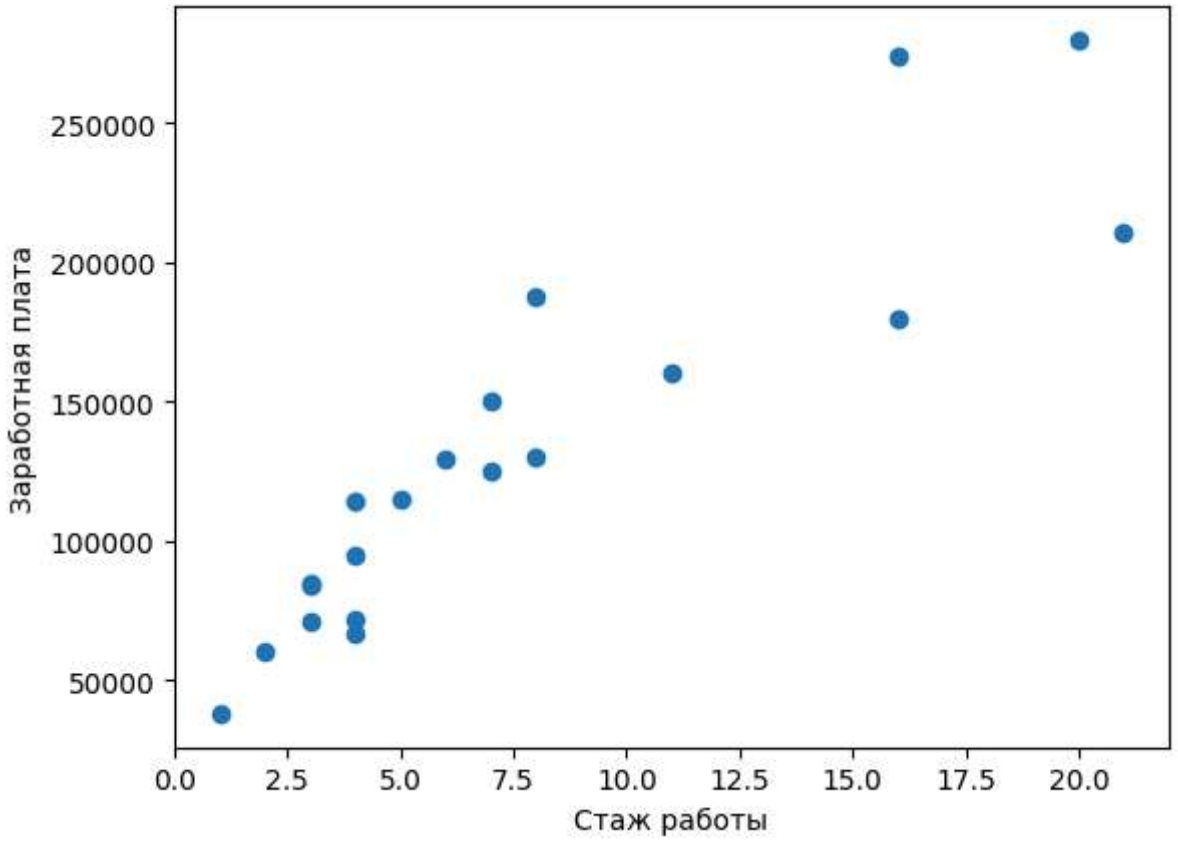
Линейная регрессия - самый простой алгоритм обучения. Он лучше всего работает с данными, которые являются относительно линейными, т. е. с наборами данных, расположенными примерно на одной прямой. Уравнение прямой в двух измерениях имеет следующий вид:

y=mx+ b,

где m - наклон прямой, а b- место пересечения прямой с осью y.

Набор данных "заработная плата в зависимости от стажа" хорошо поддается линейной регрессии.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([1,3,4,5,11,16,8,6,3,7,16,4,7,3,2,4,21,4,20,8])
y = 1000*np.array([38,84,114,115,160,180,130,129,85,125,274,72,150,71,60,95,211,67,280,188])
plt.scatter(x,y)
plt.xlabel('Стаж работы')
plt.ylabel('Заработная плата')
plt.show()
```



In [2]:
```
# Шаг 1. Выбор линейной модели
from sklearn.linear_model import LinearRegression
# Шаг 2. Создание модели
model = LinearRegression(fit_intercept=True) # подбор точки пересечения с осью координат делаем
# Шаг 3. Формирование из данных матриц признаков и целевого вектора
x.shape
```

Out[2]:
```
(20,)
```

In [3]:
```
X = x[:,np.newaxis]
X.shape
```

Out[3]:
```
(20, 1)
```

In [4]:
```
# Шаг 4. Обучение модели на наших данных
model.fit(X,y)
```

Out[4]:
```
LinearRegression()
```

In [5]:
```
model.coef_ # угловой коэф.
```

Out[5]:
```
array([10252.42152793])
```

In [6]:
```
model.intercept_ # точка пересечения с осью координат (верт)
```

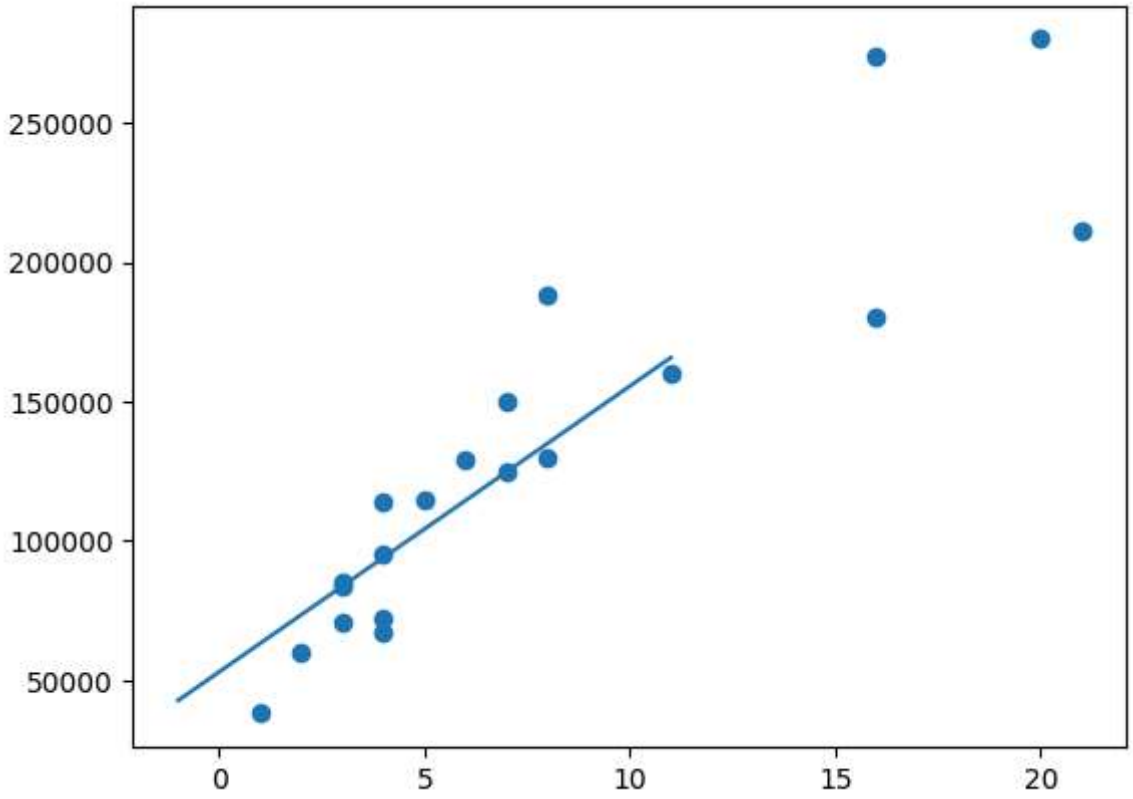
Out[6]:
```
52968.97531133931
```

In [7]:
```
# Шаг 5. Предсказание меток для новых данных
xfit = np.linspace(-1,11)
xfit.shape
```

Out[7]:
```
(50,)
```

In [8]:
```
Xfit = xfit[:,np.newaxis]
yfit = model.predict(Xfit)
```

In [9]:
```
plt.scatter(x,y)
plt.plot(Xfit, yfit);
```



In [10]:
```
y=model.coef_*10+model.intercept_
y
```

Out[10]:
```
array([155493.19059063])
```

На рисунке показана линия регрессии, соответствующая точкам данных. Прогнозирование зарплаты с 10-летним опытом работы сводится к нахождению точки на линии, где x = 10. Уравнение прямой таково: y= 10252x + 52968. Подставив 10 в это уравнение для x, получим прогнозируемый доход 155 493 долларов.

Использование регрессии для прогнозирования тарифов на такси

В службе такси многие клиенты не знают, сколько будет стоить поездка, пока она не закончится. Это связано с тем, что расстояние не единственная переменная, определяющая стоимость проезда. И Вы как программист решили создать мобильное приложение, с помощью которого клиенты, усаживаясь в такси, смогут оценить стоимость проезда. Для создания такого приложения вы должны собрать огромные массивы данных о стоимости проезда, которые компания собирала в течение многих лет. Вы будете использовать их для построения модели машинного обучения.

Давайте обучим **регрессионную модель** для предсказания суммы оплаты проезда с учетом времени суток, дня недели, а также места посадки и высадки пассажиров.

Используем наборы данных из Kaggle taxi-fares.csv. Файл содержит около 55 ООО строк. Прежде чем использовать эти данные, необходимо провести **подготовительную работу**.

In [16]:
```
import pandas as pd
df = pd.read_csv('Data/taxi-fares.csv')
df['pickup_datetime'].head()
```

Out[16]:
```
0    2014-06-15 17:11:00 UTC
1    2011-03-14 22:43:00 UTC
2    2011-02-14 15:14:00 UTC
3    2009-10-29 11:29:00 UTC
4    2011-07-02 10:38:00 UTC
Name: pickup_datetime, dtype: object
```

In [17]:
```
import pandas as pd

df = pd.read_csv('Data/taxi-fares.csv', parse_dates=['pickup_datetime'])
df.head()
```

Out[17]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2014-06-15 17:11:00.000000107	7.0	2014-06-15 17:11:00+00:00	-73.995420	40.759662	-73.987607	40.751247	1
1	2011-03-14 22:43:00.00000095	4.9	2011-03-14 22:43:00+00:00	-73.993552	40.731110	-73.998497	40.737200	5
2	2011-02-14 15:14:00.00000067	6.1	2011-02-14 15:14:00+00:00	-73.972380	40.749527	-73.990638	40.745328	1
3	2009-10-29 11:29:00.00000040	6.9	2009-10-29 11:29:00+00:00	-73.973703	40.763542	-73.984253	40.758603	5
4	2011-07-02 10:38:00.00000028	10.5	2011-07-02 10:38:00+00:00	-73.921262	40.743615	-73.967383	40.765162	1

Обратите внимание на использование параметра *parse_dates* функции *read_csv* для разбора строк в столбце pickup_datetime.

Узнаем количество записей и полей

In [18]:
```
df.shape
```

Out[18]:
```
(55368, 8)
```

Есть ли в столбцах отсутствующие значения?

In [19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55368 entries, 0 to 55367
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   key                55368 non-null  object
 1   fare_amount        55368 non-null  float64
 2   pickup_datetime    55368 non-null  datetime64[ns, UTC]
 3   pickup_longitude   55368 non-null  float64
 4   pickup_latitude    55368 non-null  float64
 5   dropoff_longitude  55368 non-null  float64
 6   dropoff_latitude   55368 non-null  float64
 7   passenger_count    55368 non-null  int64
dtypes: datetime64[ns, UTC](1), float64(5), int64(1), object(1)
memory usage: 3.4+ MB
```

Названия столбцов:

- key - ключ
- fare_amount - стоимость проезда
- pickup_datetime - время посадки
- pickup_longitude - координаты посадки (широта)
- pickup_latitude - координаты посадки (долгота)
- dropoff_longitude - координаты высадки (широта)
- dropoff_latitude - координаты высадки (долгота)
- passenger_count - количество пассажиров.

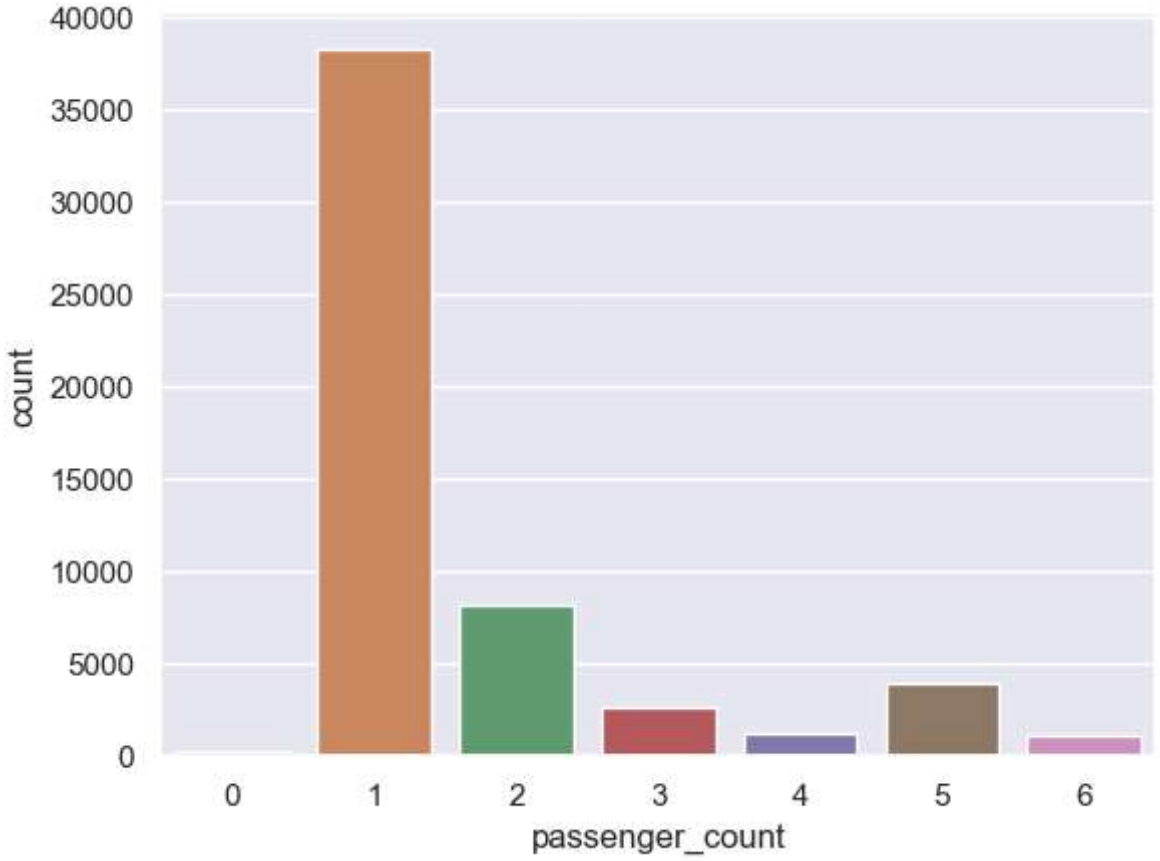
Каждая строка представляет собой поездку на такси и содержит такую информацию, как стоимость проезда, места посадки и высадки, количество пассажиров. Мы хотим предсказать именно стоимость проезда.

Построим гистограмму, показывающую, сколько строк содержит количество пассажиров, равное 1, сколько - 2 и т. д.

In [20]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

sns.countplot(x=df['passenger_count'])
```

Out[20]:
```
<AxesSubplot:xlabel='passenger_count', ylabel='count'>
```



Поскольку нас интересует прогнозирование стоимости проезда только для одного пассажира, удалим все строки с несколькими пассажирами и столбец key.

In [21]:
```
df = df[df['passenger_count'] == 1]
df = df.drop(['key', 'passenger_count'], axis=1)
df.head()
```

Out[21]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
0	7.0	2014-06-15 17:11:00+00:00	-73.995420	40.759662	-73.987607	40.751247
2	6.1	2011-02-14 15:14:00+00:00	-73.972380	40.749527	-73.990638	40.745328
4	10.5	2011-07-02 10:38:00+00:00	-73.921262	40.743615	-73.967383	40.765162
5	15.3	2011-12-09 20:03:00+00:00	-73.973500	40.792610	-73.936035	40.856548
8	7.7	2011-04-02 01:05:15+00:00	-73.979564	40.735405	-73.955686	40.768065

В результате в наборе данных остается 38 233 строки, в чем можно убедиться с помощью shape

In [22]:
```
df.shape
```

Out[22]:
```
(38233, 6)
```

Теперь с помощью метода corr() библиотеки Pandas выясним, насколько сильно влияют входные переменные, такие как широта и долгота, на значения в столбце fare_amount

In [23]:
```
corr_matrix = df.corr()
corr_matrix['fare_amount'].sort_values(ascending=False)
```

Out[23]:
```
fare_amount          1.000000
dropoff_longitude    0.020438
pickup_longitude     0.015742
pickup_latitude     -0.015915
dropoff_latitude    -0.021711
Name: fare_amount, dtype: float64
```

Здесь широта и долгота мало связаны со стоимостью проезда. Однако интуитивно понятно, что они должны сильно влиять на стоимость, поскольку определяют начальный и конечный пункты, а более длительные поездки требуют более высоких тарифов.

Создадим новые столбцы, которые оказывают большее влияние на результат. Их значения вычисляются из значений других столбцов. Добавим столбцы с указанием дня недели (0- понедельник (Monday), 1 - воскресенье (Sunday) и т. д.), часа дня, когда пассажира забрали (0-23), и расстояния (по воздуху, а не по улице) в милях, которое потребовала поездка. При расчете расстояний в данном коде

In [24]:
```
import datetime
from math import sqrt
# просматриваем каждую строку в наборе
for i, row in df.iterrows():
    dt = row['pickup_datetime'] # время посадки определенного клиента
    df.at[i, 'day_of_week'] = dt.weekday() # определим день недели
    '''
    Метод at используется для быстрого доступа к элементам по меткам.
    dt.weekday() возвращает день недели
    Результат будет целым числом от 0 до 6,
    где 0 соответствует понедельнику, а 6 — воскресенью.
    '''
    df.at[i, 'pickup_time'] = dt.hour # определим время
    # определим координаты посадки и высадки (широта, долгота) и подсчитаем расстояние в милях
    x = (row['dropoff_longitude'] - row['pickup_longitude']) * 54.6 # 1 degree == 54.6 miles
    y = (row['dropoff_latitude'] - row['pickup_latitude']) * 69.0   # 1 degree == 69 miles
    distance = sqrt(x**2 + y**2)
    df.at[i, 'distance'] = distance

df.head()
```

Out[24]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	day_of_week	pickup_time	distance
0	7.0	2014-06-15 17:11:00+00:00	-73.995420	40.759662	-73.987607	40.751247	6.0	17.0	0.720497
2	6.1	2011-02-14 15:14:00+00:00	-73.972380	40.749527	-73.990638	40.745328	0.0	15.0	1.038136
4	10.5	2011-07-02 10:38:00+00:00	-73.921262	40.743615	-73.967383	40.765162	5.0	10.0	2.924341
5	15.3	2011-12-09 20:03:00+00:00	-73.973500	40.792610	-73.936035	40.856548	4.0	20.0	4.862893
8	7.7	2011-04-02 01:05:15+00:00	-73.979564	40.735405	-73.955686	40.768065	5.0	1.0	2.603493

Теперь нет необходимости во всех столбцах, поэтому удалите те, которые не будут использоваться

In [25]:
```
df.drop(columns=['pickup_datetime', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude'], i
df.head()
```

Out[25]:

	fare_amount	day_of_week	pickup_time	distance
0	7.0	6.0	17.0	0.720497
2	6.1	0.0	15.0	1.038136
4	10.5	5.0	10.0	2.924341
5	15.3	4.0	20.0	4.862893
8	7.7	5.0	1.0	2.603493

Проверим корреляцию еще раз

In [26]:
```
corr_matrix = df.corr()
corr_matrix['fare_amount'].sort_values(ascending=False)
```

Out[26]:
```
fare_amount    1.000000
distance       0.045873
day_of_week    0.009196
pickup_time   -0.019722
Name: fare_amount, dtype: float64
```

По-прежнему нет сильной зависимости между расстоянием поездки и размером оплаты за проезд. Возможно, это объяснит и причину.
Посмотрим стат.данные

In [27]:
```
df.describe()
```

Out[27]:

	fare_amount	day_of_week	pickup_time	distance
count	38233.000000	38233.000000	38233.000000	38233.000000
mean	11.214115	2.951534	13.387989	12.018397
std	9.703149	1.932809	6.446519	217.357022
min	-22.100000	0.000000	0.000000	0.000000
25%	6.000000	1.000000	9.000000	0.762116
50%	8.500000	3.000000	14.000000	1.331326
75%	12.500000	5.000000	19.000000	2.402226
max	256.000000	6.000000	23.000000	4923.837280

В наборе данных присутствуют выбросы, а выбросы часто искажают результаты моделей машинного обучения . Отфильтруйте данные, исключив отрицательные стоимости проезда и установив разумные ограничения на стоимость проезда и расстояние, а затем снова проведите корреляцию:

In [28]:
```
df = df[(df['distance'] > 1.0) & (df['distance'] < 10.0)]
df = df[(df['fare_amount'] > 0.0) & (df['fare_amount'] < 50.0)]

corr_matrix = df.corr()
corr_matrix['fare_amount'].sort_values(ascending=False)
```

Out[28]:
```
fare_amount    1.000000
distance       0.851913
day_of_week   -0.003570
pickup_time   -0.023085
Name: fare_amount, dtype: float64
```

Это уже лучше! Большая часть (85%) дисперсии в стоимости проезда объясняется расстоянием. Корреляция между днем недели, временем суток и стоимостью проезда по-прежнему слабая, но это и неудивительно, поскольку именно расстояние поездки является основным фактором, определяющим стоимость проезда в такси. Оставим эти столбцы, поскольку вполне логично, что в час пик дорога из пункта А в пункт Б может занять больше времени, или что движение в 17:00 пятницы может отличаться от движения в 17:00 субботы.

Обучение регрессионной модели

Будем использовать три различных алгоритма обучения, чтобы определить, какой из них дает наиболее точный результат, и используем перекрестную валидацию для оценки точности. Начнем с линейной регрессионной модели:

In [29]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

x = df.drop(['fare_amount'], axis=1)
y = df['fare_amount']

model = LinearRegression()
cross_val_score(model, x, y, cv=5).mean()
```

Out[29]:
```
0.7258845061910318
```

Попробуйте использовать `RandomForestRegressor` с тем же набором данных и сравните результаты. Напомним, что модели со случайным лесом обучают несколько деревьев решений на данных и усредняют результаты всех деревьев для получения прогноза.

Д/з. Модель со случайным лесом конспектировать

In [30]:
```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(random_state=0)
cross_val_score(model, x, y, cv=5).mean()
```

Out[30]:
```
0.706157807448991
```

Теперь воспользуемся `GradientBoostingRegressor`. Машины с градиентным бустингом работают на основе нескольких деревьев решений, каждое из которых обучено компенсировать ошибку в выходных данных предыдущего.

Д/з. Модель с градиентным бустингом конспектировать

Как использовать метод **cross_val_score** конспектировать

In [31]:
```
from sklearn.ensemble import GradientBoostingRegressor

model = GradientBoostingRegressor(random_state=0)
cross_val_score(model, x, y, cv=5).mean()
```

Out[31]:
```
0.750496262408626
```

`GradientBoostingRegressor` дал наибольший кросс-валидированный коэффициент детерминации, обучим его на всем наборе данных

In [32]:
```
model.fit(x, y)
```

Out[32]:
```
GradientBoostingRegressor(random_state=0)
```

Модель обучена и готова к прогнозированию

Использование модели для прогноза стоимости найма такси

Обученную модель используем для выполнения нескольких прогнозов. Во-первых, оцените, какой будет стоимость найма такси для поездки на расстояние 2 мили в пятницу в 17:00

In [33]:
```
model.predict(pd.DataFrame({ 'day_of_week': [4], 'pickup_time': [17], 'distance': [2.0] }))
```

Out[33]:
```
array([11.49105571])
```

Теперь спрогнозируем стоимость для поездки длиной 2 мили, совершенной в 17:00 через день (в субботу)

In [34]:
```
model.predict(pd.DataFrame({ 'day_of_week': [5], 'pickup_time': [17], 'distance': [2.0] }))
```

Out[34]:
```
array([10.95309995])
```

Как видно из результата модель предсказывает, что в пятницу днем движение в час пик, скорее всего, будет интенсивнее, чем в субботу. Поэтому стоимость небольшая.

Д/з конспектировать

Основные алгоритмы обучения, используемые для регрессии:

- Линейная регрессия
- Деревья решений
- Случайные леса
- Градиентный бустинг
- Метод опорных векторов

In [ ]: