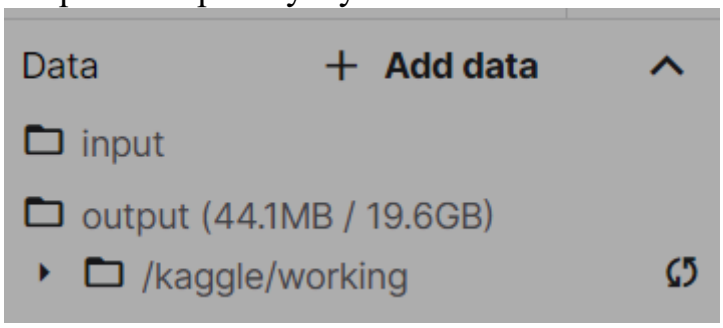
Прогнозирование задержки рейсов авиакомпании

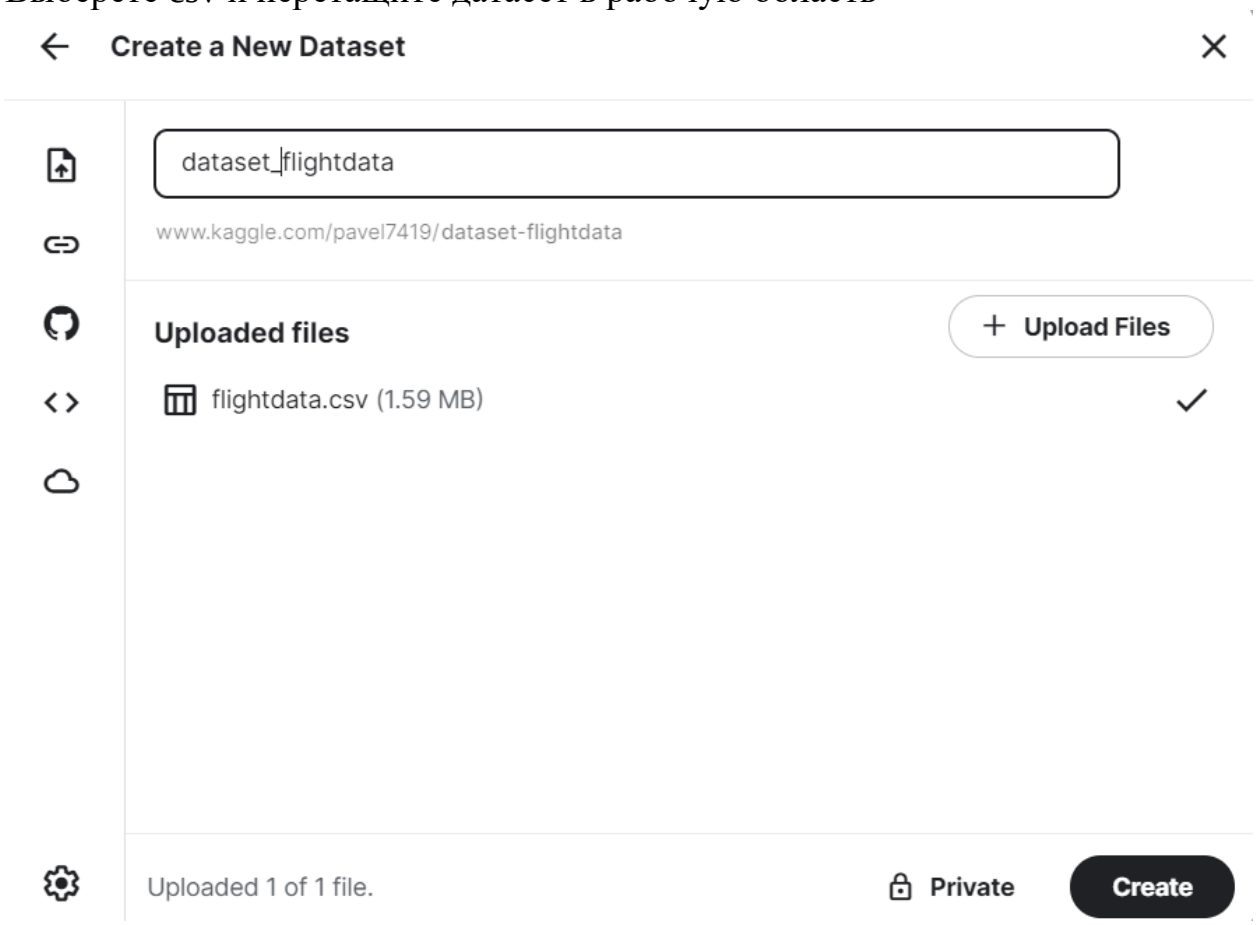
В этом уроке вы создадите записную книжку (блокнот), импортируете набор данных, содержащий сведения о своевременном прибытии рейсов крупной авиакомпании США, и загрузите набор данных в записную книжку. Затем вы очистите набор данных с помощью Pandas, создадите модель машинного обучения с помощью scikit-learn и визуализируете ее выходные данные с помощью Matplotlib.

Загрузка набора данных

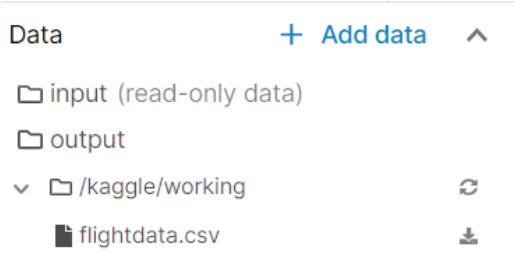
Скачать датасет:
https://disk.yandex.ru/d/qfxx5ynUf7YeKA
В правом вернем углу нажмите Add data:



Выберете csv и перетащите датасет в рабочую область



Обратите внимание: в правом верхнем углу:




```
import pandas as pd
df=pd.read_csv('/kaggle/working/flightdata.csv')
df.head()
```

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	TAIL_NUM	FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	...
0	2016	1	1	1	5	DL	N836DN	1399	10397	ATL	...
1	2016	1	1	1	5	DL	N964DN	1476	11433	DTW	...
2	2016	1	1	1	5	DL	N813DN	1597	10397	ATL	...
3	2016	1	1	1	5	DL	N587NW	1768	14747	SEA	...
4	2016	1	1	1	5	DL	N836DN	1823	14747	SEA	...

5 rows × 26 columns

Созданный **DataFrame** содержит сведения о своевременном прибытии рейсов крупной авиакомпании США. В нем более 11 000 строк и 26 столбцов. (В выходных данных сказано "5 строк", так как функция кадра данных head возвращает только первые пять строк.) Каждая строка представляет один рейс и содержит такие сведения, как аэропорт отправления и прибытия, запланированное время отправления и указание на своевременность прибытия. Мы рассмотрим данные подробнее чуть позже в этом модуле.

Очистка и подготовка данных

Прежде чем подготовить набор данных, необходимо понять их содержимое и структуру.

DataFrame — это двумерная структура данных с метками. Столбцы в кадре данных могут относиться к разным типам, так же, как и столбцы в электронной таблице или таблице в базе данных. Это наиболее часто используемый объект в Pandas. В этом уроке вы подробно исследуете DataFrame и данные внутри них.

Код, который вы добавили в записную книжку, создает DataFrame из файла **flightdata.csv** и вызывает DataFrame.head, чтобы отобразить первые пять строк. Как правило, первым делом нужно узнать, сколько строк содержит набор данных. Чтобы узнать число, введите следующую инструкцию в пустую ячейку в конце записной книжки и запустите ее:

```
df.shape
```

```
(11231, 26)
```

Теперь изучите 26 столбцов в наборе данных. Они содержат важную информацию, например дату рейса (YEAR, MONTH и DAY_OF_MONTH), место вылета и назначения (ORIGIN и DEST), запланированное время вылета и прибытия (CRS_DEP_TIME и CRS_ARR_TIME), разницу между запланированным и фактическим временем прибытия в минутах (ARR_DELAY) и указание, если рейс опоздал не менее чем на 15 минут (ARR_DEL15).
Ниже приведен полный список столбцов в наборе данных. Значения времени выражаются в 24-часовом военном формате. Например, 1130 означает 11:30, а 1500 - 15:00.

Столбец	**Описание**
YEAR	Год, когда был совершен рейс
QUARTER	Квартал, когда был совершен рейс (1–4)
MONTH	Месяц, когда был совершен рейс (1–12)
DAY_OF_MONTH	День месяца, когда был совершен рейс (1–31)
DAY_OF_WEEK	День недели, когда был совершен рейс (1 — понедельник, 2 — вторник, и т. д.)
UNIQUE_CARRIER	Код авиакомпании (например, DL)
TAIL_NUM	Хвостовой номер самолета
FL_NUM	Номер рейса
ORIGIN_AIRPORT_ID	Идентификатор аэропорта вылета
ORIGIN	Код аэропорта вылета (ATL, DFW, SEA и т. д.)
DEST_AIRPORT_ID	Идентификатор аэропорта назначения
DEST	Код аэропорта назначения (ATL, DFW, SEA и т. д.)
CRS_DEP_TIME	Запланированное время отправления
DEP_TIME	Фактическое время отправления
DEP_DELAY	Число минут задержки отправления
DEP_DEL15	0 = отправление задержано меньше, чем на 15 минут, 1 = отправление задержано на 15 минут или более
CRS_ARR_TIME	Запланированное время прибытия
ARR_TIME	Фактическое время прибытия
ARR_DELAY	Число минут задержки прибытия
ARR_DEL15	0 = рейс прибыл не более, чем на 15 минут позже, 1 = рейс прибыл позже на 15 минут и более

Столбец	Описание
CANCELLED	0 = рейс не отменен, 1 = рейс отменен
DIVERTED	0 = рейс не перенаправлен, 1 = рейс перенаправлен
CRS_ELAPSED_TIME	Запланированное время рейса в минутах
ACTUAL_ELAPSED_TIME	Фактическое время рейса в минутах
DISTANCE	Расстояние рейса в милях

Набор данных включает примерно равномерное распределение дат на протяжении года, что важно, поскольку рейс из Миннеаполиса с большей вероятностью задержат из-за снегопада в январе, чем в июле. Но этот набор данных не очищен и совершенно не готов к использованию. Напишем код Pandas для очистки.

Одним из самых важных аспектов подготовки набора данных для использования в машинном обучении является выбор столбцов признаков, имеющих отношение к результату, который вы пытаетесь спрогнозировать, и отсеивание столбцов, которые не влияют на результат, могут исказить его или привести к мультиколлинеарности. Еще одна важная задача — исключить отсутствующие значения, удалив строки или столбцы, содержащие их, или заменив их действующими значениями. В этом упражнении вы удалите лишние столбцы и замените недостающие значения в остальных столбцах.

1. Сначала специалисты по обработке и анализу данных обычно ищут в наборе данных отсутствующие значения. Есть простой способ проверить наличие недостающих значений в Pandas. Выполните следующий код в ячейке в конце записной книжки:

```
df.isna().values.any()
```

```
True
```

Проверка отсутствующих значений

2. Далее необходимо выяснить, где находятся отсутствующие значения. Для этого выполните следующий код:

```
df.isna().sum()
```

Убедитесь, что в выходных данных отображается количество отсутствующих значений в каждом столбце:

```
YEAR                      0
QUARTER                   0
MONTH                     0
DAY_OF_MONTH              0
DAY_OF_WEEK               0
UNIQUE_CARRIER            0
TAIL_NUM                  0
FL_NUM                    0
ORIGIN_AIRPORT_ID         0
ORIGIN                    0
DEST_AIRPORT_ID           0
DEST                      0
CRS_DEP_TIME              0
DEP_TIME                107
DEP_DELAY               107
DEP_DEL15               107
CRS_ARR_TIME              0
ARR_TIME                115
ARR_DELAY               188
ARR_DEL15               188
CANCELLED                 0
DIVERTED                  0
CRS_ELAPSED_TIME          0
ACTUAL_ELAPSED_TIME     188
DISTANCE                  0
Unnamed: 25           11231
dtype: int64
```

Количество отсутствующих значений в каждом столбце

3. Любопытно, что 26-й столбец ("Unnamed: 25") содержит 11 231 отсутствующее значение, что равно количеству строк в наборе данных. Данный столбец создан по ошибке, так как импортированный CSV-файл содержит запятую в конце каждой строки. Чтобы исключить этот столбец, добавьте следующий код в записную книжку и выполните его:

```
df = df.drop('Unnamed: 25', axis=1)
df.isna().sum()
```

```
YEAR                    0
QUARTER                 0
MONTH                   0
DAY_OF_MONTH            0
DAY_OF_WEEK             0
UNIQUE_CARRIER          0
TAIL_NUM                0
FL_NUM                  0
ORIGIN_AIRPORT_ID       0
ORIGIN                  0
DEST_AIRPORT_ID         0
DEST                    0
CRS_DEP_TIME            0
DEP_TIME              107
DEP_DELAY             107
DEP_DEL15             107
CRS_ARR_TIME            0
ARR_TIME              115
ARR_DELAY             188
ARR_DEL15             188
CANCELLED               0
DIVERTED                0
CRS_ELAPSED_TIME        0
ACTUAL_ELAPSED_TIME   188
DISTANCE                0
dtype: int64
```

DataFrame с удаленным 26-м столбцом

(Если нужно удалить строки данных (например, 1 и 2) используйте следующий код: df.drop([1,2], axis=0).head(3))

4. DataFrame по-прежнему содержит большое количество отсутствующих значений, но некоторые из них не нужны, так как содержащие их столбцы не важны для создаваемой вами модели данных. Цель этой модели заключается в прогнозировании того, прибудет ли рейс, на который вы хотите купить билеты, вовремя. Если вы знаете, что рейс может опоздать, вы можете выбрать другой рейс.

Затем необходимо отфильтровать набор данных, чтобы исключить столбцы, которые не относятся к модели прогнозирования. Например, хвостовой номер самолета вряд ли влияет на прибытия, и на момент бронирования билетов у вас нет возможности узнать, будет ли рейс отменен, перенаправлен или отложен. А вот запланированное время отправления *значительно* влияет на своевременность прибытия. Поскольку большинство авиакомпаний используют звездообразную сеть, утренние рейсы отбывают вовремя чаще, чем дневные и вечерние. В некоторых крупных аэропортах трафик накапливается в течение дня, и вероятность задержки более поздних рейсов возрастает.

Pandas предоставляет простой способ фильтровать ненужные столбцы. Выполните следующий код в новой ячейке в конце записной книжки:

```
df = df[["MONTH", "DAY_OF_MONTH", "DAY_OF_WEEK", "ORIGIN", "DEST", "CRS_DEP_TIME", "ARR_DEL15"]]
df.isnull().sum()
```

```
MONTH             0
DAY_OF_MONTH      0
DAY_OF_WEEK       0
ORIGIN            0
DEST              0
CRS_DEP_TIME      0
ARR_DEL15       188
dtype: int64
```

Выходные данные показывают, что DataFrame теперь включает только столбцы, имеющие отношение к модели, и недостающих значений теперь гораздо меньше.

5. Единственным столбцом, который теперь содержит отсутствующие значения, является столбец ARR_DEL15, где 0 означает рейсы, прибывшие вовремя, а 1 — рейсы с задержкой. Чтобы отобразить первые пять строк с недостающими значениями, используйте следующий код:

```
df[df.isna().values.any(axis=1)].head()
```

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
177	1	9	6	MSP	SEA	701	NaN
179	1	10	7	MSP	DTW	1348	NaN
184	1	10	7	MSP	DTW	625	NaN
210	1	10	7	DTW	MSP	1200	NaN
478	1	22	5	SEA	JFK	2305	NaN

Строки с отсутствующими значениями

Pandas представляет отсутствующие значения с помощью NaN, что означает *Not a Number (не является числом)*. Выходные данные показывают, что эти строки действительно не имеют значений в столбце ARR_DEL15.

6. Эти строки не содержат значений ARR_DEL15, потому что соответствуют отмененным или перенаправленным рейсам. Можно вызвать dropna в кадре данных, чтобы удалить эти строки. Но поскольку рейс, который был отменен или перенаправлен в другой аэропорт, может считаться опоздавшим, давайте используем метод fillna, чтобы заменить отсутствующие значения значением 1.

Используйте следующий код, чтобы заменить недостающие значения в столбце ARR_DEL15 на 1 и отобразить строки с 177 по 184:

```
df = df.fillna({'ARR_DEL15': 1})
df.iloc[177:185]
```

Убедитесь, что NaN в строках 177, 179 и 184 были заменены значением 1, указывающим на задержку прибытия:

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
177	1	9	6	MSP	SEA	701	1.0
178	1	9	6	DTW	JFK	1527	0.0
179	1	10	7	MSP	DTW	1348	1.0
180	1	10	7	DTW	MSP	1540	0.0
181	1	10	7	JFK	ATL	1325	0.0
182	1	10	7	JFK	ATL	610	0.0
183	1	10	7	JFK	SEA	1615	0.0
184	1	10	7	MSP	DTW	625	1.0

NaN заменены на 1

Теперь набор данных очищен: все отсутствующие значения заменены, лишние столбцы удалены. Но работа еще не закончена. Нужно выполнить еще несколько задач, чтобы подготовить набор данных для использования в машинном обучении.

Столбец CRS_DEP_TIME в наборе данных представляет время запланированного отправления. Подробные числа в этом столбце — он содержит более 500 уникальных значений — могут отрицательно повлиять на точность модели машинного обучения. Чтобы исправить эту проблему, можно использовать прием под названием группирование или квантование. Что если разделить каждое число в этом столбце на 100 и округлить результат до ближайшего целого числа? 1030 превратилось бы в 10, 1925 — в 19 и т. д., и осталось бы максимум 24 отдельных значения. Интуитивно это кажется логичным, потому что неважно, отправляется рейс в 10:30 или в 10:40. Но важна разница между 10:30 и 5:30.

Кроме того, столбцы ORIGIN и DEST содержат коды аэропортов, которые представляют категориальные значения в машинном обучении. Эти столбцы должны быть преобразованы в дискретные столбцы, содержащие переменные индикатора, которые иногда называют формальными. Другими словами, столбец ORIGIN, который содержит пять кодов аэропортов, должен быть преобразован в пять столбцов, по одному на каждый аэропорт, и каждый столбец должен содержать значение 1 и 0, указывающее, отбыл ли рейс из аэропорта, который представляет этот столбец. Столбец DEST должен обрабатываться точно так же.

В этом упражнении вы сгруппируете время отправления в столбце CRS_DEP_TIME и используете метод Pandas get_dummies, чтобы создать столбцы индикатора из столбцов ORIGIN и DEST.

1. Чтобы отобразить первые пять строк из кадра данных, используйте следующую команду:

```
df.head()
```

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
0	1	1	5	ATL	SEA	1905	0.0
1	1	1	5	DTW	MSP	1345	0.0
2	1	1	5	ATL	SEA	940	0.0
3	1	1	5	SEA	MSP	819	0.0
4	1	1	5	SEA	DTW	2300	0.0

DataFrame без группирования времени отправления

2. Используйте следующие инструкции для группирования времени отправления:

```
import math

for index, row in df.iterrows():
    df.loc[index, 'CRS_DEP_TIME'] = math.floor(row['CRS_DEP_TIME'] / 100)
df.head()
```

Убедитесь, что числа в столбце CRS_DEP_TIME теперь попадают в диапазон от 0 до 23:

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
0	1	1	5	ATL	SEA	19	0.0
1	1	1	5	DTW	MSP	13	0.0
2	1	1	5	ATL	SEA	9	0.0
3	1	1	5	SEA	MSP	8	0.0
4	1	1	5	SEA	DTW	23	0.0

3. Теперь используйте следующие инструкции, чтобы создать столбцы индикатора из столбцов ORIGIN и DEST и удалить сами столбцы ORIGIN и DEST:

```
df = pd.get_dummies(df, columns=['ORIGIN', 'DEST'])
df.head()
```

Изучите итоговый DataFrame и обратите внимание, что столбцы ORIGIN и DEST заменены на столбцы, соответствующие кодам аэропортов из исходных

столбцов. Новые столбцы имеют значения 1 и 0, обозначающие аэропорт прибытия или отправления рейса.

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	CRS_DEP_TIME	ARR_DEL15	ORIGIN_ATL	ORIGIN_DTW	ORIGIN_JFK	ORIGIN
0	1	1	5	19	0.0	1	0	0	
1	1	1	5	13	0.0	0	1	0	
2	1	1	5	9	0.0	1	0	0	
3	1	1	5	8	0.0	0	0	0	
4	1	1	5	23	0.0	0	0	0	

DataFrame со столбцами индикатора

Набор данных теперь существенно отличается от исходного, но он оптимизирован для использования в машинном обучении.

Создание модели машинного обучения

Чтобы создать модель машинного обучения, вам потребуется два набора данных: один для обучения и один — для тестирования. На практике у вас бывает только один набор данных, поэтому его можно разделить на две части. В этом упражнении вы разделите подготовленный ранее DataFrame в соотношении 80 к 20, чтобы его можно было использовать для обучения модели машинного обучения. Вы также разделите DataFrame на столбцы признаков и столбцы меток. Столбцы признаков используются в качестве входных данных для модели (например, аэропорт отправления и прибытия и запланированное время отправления), а столбцы меток содержат значения, которые модель попытается предсказать, — в данном случае это столбец ARR_DEL15, который указывает, прибудет ли рейс вовремя.

```
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(df.drop('ARR_DEL15', axis=1),
df['ARR_DEL15'], test_size=0.2, random_state=42)
```

Первая инструкция импортирует вспомогательную функцию scikit-learn train_test_split. Во второй строке указана функция для разделения кадра данных на обучающий набор, содержащий 80 % исходных данных, и набор тестирования, содержащий оставшиеся 20 %. Параметр random_state заполняет генератор случайных чисел, используемый для разделения, а первый и второй параметры — это кадры данных, содержащие столбцы признаков и столбцы меток.

train_test_split возвращает четыре кадра данных. Используйте следующую команду для отображения числа строк и столбцов в кадре данных, содержащем столбцы признаков, используемые для обучения:

```
train_x.shape
```

Теперь используйте следующую команду для отображения числа строк и столбцов в кадре данных, содержащем столбцы признаков, используемые для тестирования:

```
test_x.shape
```

1. Чем отличаются два результата и почему?

Можно ли предсказать, что вы увидите, если вызовете shape в других двух кадрах данных, train_y и test_y? Если вы не знаете точно, попробуйте и узнаете.

Существует много типов моделей машинного обучения. Одним из наиболее распространенных является модель регрессии, которая использует один из алгоритмов регрессии для получения числового значения, например возраст человека или вероятность того, что операция по кредитной карте является мошеннической. Вы будете обучать модель классификации, которая пытается разрешить набор входных данных в один из наборов известных выходных данных. Классический пример модели классификации — это модель, которая проверяет сообщения электронной почты и классифицирует их как "спам" или "не спам". У вас будет модель двоичной классификации, прогнозирующая, прибудет ли рейс вовремя или с опозданием ("двоичная", поскольку здесь возможно только два результата).

Одним из преимуществ использования библиотеки scikit-learn является то, что вам не нужно создавать эти модели или реализовать используемые в них алгоритмы вручную. Библиотека scikit-learn включает различные классы для реализации распространенных моделей машинного обучения. Один из них — RandomForestClassifier, который сопоставляет множество деревьев принятия решений с данными и вычисляет средние значения, чтобы повысить общую точность и ограничить переобучение.

1. Выполните следующий код в новой ячейке, чтобы создать объект RandomForestClassifier и обучить его с помощью метода fit.

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random_state=13)
model.fit(train_x, train_y)
```

Выходные данные показывают параметры, используемые в классификаторе, включая n_estimators, который указывает число деревьев в каждом лесу деревьев принятия решений, и max_depth, который указывает максимальную глубину дерева принятия решений. Указанные значения используются по умолчанию, но их можно переопределить при создании объекта RandomForestClassifier.

```
In [16]: from sklearn.ensemble import RandomForestClassifier

         model = RandomForestClassifier(random_state=13)
         model.fit(train_x, train_y)

Out[16]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_split=1e-07, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=10, n_jobs=1, oob_score=False, random_state=13,
                     verbose=0, warm_start=False)
```

Обучение модели

2. Теперь вызовите метод predict для тестирования модели, используя значения в test_x, а затем метод score, чтобы определить среднюю точность модели:

```
predicted = model.predict(test_x)
model.score(test_x, test_y)
```

Вы увидите такой результат:

```
predicted = model.predict(test_x)
model.score(test_x, test_y)

0.86025812194036488
```

Средняя точность составляет 86 %, что, на первый взгляд, кажется хорошим результатом. Но средняя точность не всегда является надежным показателем точности модели классификации. Давайте копнем глубже и посмотрим, насколько точна модель на самом деле, то есть как хорошо она определяет своевременность прибытия рейсов.

Существует несколько способов для измерения точности модели классификации. Одним из лучших показателей модели двоичной классификации является площадь под ROC-кривой (иногда называется ROC AUC), которая, по сути, предназначена для количественной оценки того, как часто модель делает правильный прогноз независимо от результата. В этом модуле вы вычислите показатель ROC AUC для модели, созданной ранее, и узнаете, почему этот показатель меньше, чем средняя точность результатов по методу score. Вы также узнаете о других способах оценить точность модели.

1. Прежде чем вычислить ROC AUC, необходимо создать *вероятности прогноза* для тестового набора. Эти вероятности являются приблизительными для каждого из классов, или ответов, которые может предсказать модель. Например, [0.88199435, 0.11800565] означает, что существует 89 % вероятности того, что рейс прибудет вовремя (ARR_DEL15 = 0) и 12 % вероятности того, что он опоздает (ARR_DEL15 = 1). Сумма двух вероятностей составляет 100 %.

Выполните следующий код, чтобы создать набор вероятностей прогнозирования на основе тестовых данных:

```
from sklearn.metrics import roc_auc_score
probabilities = model.predict_proba(test_x)
```

2. Теперь выполните следующую инструкцию для создания оценки ROC AUC по вероятностям с помощью метода scikit-learn roc_auc_score:

```
roc_auc_score(test_y, probabilities[:, 1])
```

Убедитесь, что в выходных данных отображается оценка 67%:

```
roc_auc_score(test_y, probabilities[:, 1])
```

```
0.67438249049985388
```

Создание оценки AUC

Почему показатель AUC ниже, чем средняя точность, вычисленная в предыдущем упражнении?
В выходных данных метода score отражается, сколько элементов тестового набора модель предсказала правильно. Эта оценка искажена потому, что набор данных, на котором модель обучалась и тестировалась, содержит гораздо больше строк с прибытием вовремя, чем с опозданием. Из-за этого дисбаланса в данных у вас больше шансов, когда вы предсказываете своевременное прибытие, а не опоздание.
ROC AUC учитывает это и предоставляет более точное обозначение вероятности того, насколько прогноз прибытия вовремя *или* с опозданием будет правильным.

3. Чтобы узнать больше о поведении модели, создайте матрицу неточностей, также известную как *матрица ошибок*. Матрица неточностей считает количество раз, когда каждый ответ был классифицирован как правильный или неправильный. В частности, она предназначена для оценки количества ложных положительных, ложных отрицательных, истинных отрицательных и истинных положительных результатов. Это важно, ведь если модель двоичной классификации, обученная для различения кошек и собак, тестируется с набором данных, на 95 % состоящим из собак, она может получить оценку 95 %, просто если будет каждый раз отвечать "собака". Но если она не распознает ни одной кошки, она будет бесполезна.

Чтобы создать матрицу неточностей для модели, используйте следующий код:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(test_y, predicted)
```

Первая строка в выходных данных представляет рейсы, которые прибыли вовремя. Первый столбец в этой строке показывает, сколько рейсов правильно

предсказаны как прибывшие вовремя, а второй — сколько рейсов предсказаны как опоздавшие, но на самом деле не опоздали. По этим данным кажется, что модель справляется с прогнозированием своевременного прибытия.

```
from sklearn.metrics import confusion_matrix
confusion_matrix(test_y, predicted)
```

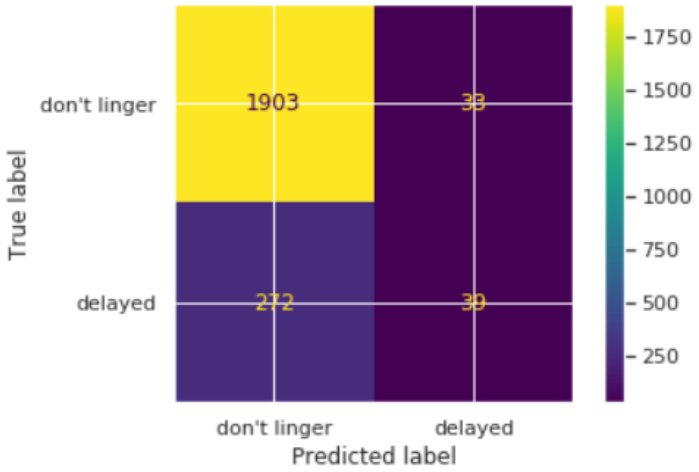
```
array([[1903,   33],
       [ 272,   39]])
```

Создание матрицы неточностей

Но посмотрите на вторую строку, где показаны опоздавшие рейсы. В первом столбце показано, сколько опоздавших рейсов были неправильно предсказаны как своевременные. Во втором столбце показано, сколько опоздавших рейсов было предсказано правильно. Очевидно, модель предсказывает задержки гораздо хуже, чем своевременное прибытие. В *идеале* в матрице неточностей должны быть большие числа в левом верхнем и правом нижнем углах и маленькие (желательно 0) в правом верхнем и левом нижнем углах.

Отличным дополнением к последней версии sklearn является новая функция *plot_confusion_matrix*. Это создает чрезвычайно интуитивно понятную и настраиваемую матрицу путаницы для вашего классификатора.

```
from sklearn import metrics
disp = metrics.plot_confusion_matrix(model, test_x,
test_y,values_format = 'n',
display_labels= ["don't linger", "delayed"])
```



4. Другие меры правильности для модели классификации — *точность* и *отзыв*. Предположим, модели предоставлено три своевременных и три задержанных рейса, и она правильно спрогнозировала два своевременных, но не угадала с двумя задержанными. В этом случае точность модели будет равна 50 % (два из

четырех рейсов были классифицированы как своевременные, и действительно прибыли вовремя), тогда как отзыв будет составлять 67 % (правильно определила два из трех своевременных рейса). Дополнительные сведения о точности и полноте можно получить здесь: https://en.wikipedia.org/wiki/Precision_and_recall.

Библиотека scikit-learn содержит удобный метод precision_score для вычисления точности. Чтобы измерить точность модели, выполните следующие инструкции:

```
from sklearn.metrics import precision_score

train_predictions = model.predict(train_x)
precision_score(train_y, train_predictions)
```

```
from sklearn.metrics import recall_score

recall_score(train_y, train_predictions)
```

```
0.9992012779552716
```

Измерение точности

5. Библиотека scikit-learn также содержит метод recall_score для вычисления полноты. Чтобы измерить полноту модели, запустите следующие инструкции:

```
from sklearn.metrics import recall_score

recall_score(train_y, train_predictions)
```

В реальном мире специалист по обработке обучающих данных будет искать способы повысить точность модели. Например, можно попробовать разные алгоритмы и *настроить* выбранный алгоритм, чтобы найти оптимальное сочетание параметров. Кроме того, можно расширить набор данных до миллионов, а не нескольких тысяч строк, и попытаться сократить дисбаланс между своевременными и опоздавшими рейсами. Но для наших целей модель подходит без изменений.

В этом модуле вы импортируете библиотеку Matplotlib в записную книжку, с которой вы работали, и настроите записную книжку для поддержки встроенных выходных данных Matplotlib.

1. В новой ячейке в конце записной книжки выполните следующие инструкции. Игнорируйте все предупреждающие сообщения, относящиеся к кэшированию шрифта:

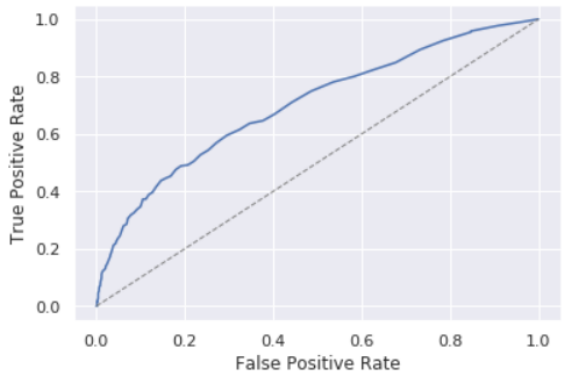
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
```

Первая инструкция является одной из нескольких магических команд, поддерживаемых ядром Python, которое вы выбрали при создании записной книжки. Она позволяет Jupyter подготовить выходные данные Matplotlib к просмотру в записной книжке без повторных вызовов команды show. И она должна находиться перед всеми ссылками на саму библиотеку Matplotlib. Последняя инструкция настраивает Seaborn для улучшения выходных данных Matplotlib.

2. Чтобы посмотреть на Matplotlib в действии, выполните следующий код в новой ячейке, чтобы построить ROC-кривую для модели машинного обучения, созданной в предыдущем задании:

```
from sklearn.metrics import roc_curve

fpr, tpr, _ = roc_curve(test_y, probabilities[:, 1])
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], color='grey', lw=1, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
metrics.auc(fpr, tpr)
```

```
0.7014819895830565
```



ROC-кривая, созданная с помощью Matplotlib

Пунктирная линия в середине диаграммы представляет вероятность получения правильного ответа 50 на 50. Синяя кривая представляет точность вашей модели. Что еще важнее, сам факт существования этой диаграммы доказывает, что вы можете использовать Matplotlib в записной книжке Jupyter.

Вы создали эту модель машинного обучения, чтобы предсказывать, прибудет рейс вовремя или опоздает. В этом упражнении вы напишете функцию Python,

которая вызывает модель машинного обучения, созданную в предыдущем задании, для вычисления вероятности того, что рейс прибудет вовремя. Затем вы используете эту функцию для анализа нескольких рейсов.

1. Введите следующее определение функции в новую ячейку, а затем выполните ячейку.

```
def predict_delay(departure_date_time, origin, destination):
    from datetime import datetime

    try:
        departure_date_time_parsed = datetime.strptime(departure_date_time,
'%d/%m/%Y %H:%M:%S')
    except ValueError as e:
        return 'Error parsing date/time - {}'.format(e)

    month = departure_date_time_parsed.month
    day = departure_date_time_parsed.day
    day_of_week = departure_date_time_parsed.isoweekday()
    hour = departure_date_time_parsed.hour

    origin = origin.upper()
    destination = destination.upper()

    input = [{'MONTH': month,
              'DAY': day,
              'DAY_OF_WEEK': day_of_week,
              'CRS_DEP_TIME': hour,
              'ORIGIN_ATL': 1 if origin == 'ATL' else 0,
              'ORIGIN_DTW': 1 if origin == 'DTW' else 0,
              'ORIGIN_JFK': 1 if origin == 'JFK' else 0,
              'ORIGIN_MSP': 1 if origin == 'MSP' else 0,
              'ORIGIN_SEA': 1 if origin == 'SEA' else 0,
              'DEST_ATL': 1 if destination == 'ATL' else 0,
              'DEST_DTW': 1 if destination == 'DTW' else 0,
              'DEST_JFK': 1 if destination == 'JFK' else 0,
              'DEST_MSP': 1 if destination == 'MSP' else 0,
              'DEST_SEA': 1 if destination == 'SEA' else 0 }]

    return model.predict_proba(pd.DataFrame(input))[0][0]
```

Эта функция принимает в качестве входных данных дату и время, коды аэропортов отправления и прибытия и возвращает значение от 0,0 до 1,0, определяющее вероятность того, что рейс прибудет вовремя. Она использует модель машинного обучения, которую вы создали в предыдущем задании, для вычисления вероятности. И для вызова модели она передает DataFrame, содержащий входные значения, в predict_proba. Структура этого кадра данных точно соответствует структуре кадра данных, который мы использовали ранее.
Входные данные даты для функции predict_delay представлены в международном формате dd/mm/year.

2. Используйте приведенный ниже код для вычисления вероятности того, что рейс из Нью-Йорка в Атланту вечером 1 октября прибудет вовремя. Год не имеет значения, так как он не используется в модели.

```
predict_delay('1/10/2018 21:45:00', 'JFK', 'ATL')
```

```
0.59999999999999998
```

3. Измените код, чтобы вычислить вероятность того, что тот же рейс, но на следующий день, прибудет вовремя:

```
predict_delay('2/10/2018 21:45:00', 'JFK', 'ATL')
```

4. Теперь измените код, чтобы вычислить вероятность того, что утренний рейс из Атланты в Сиэтл в тот же день прибудет вовремя:

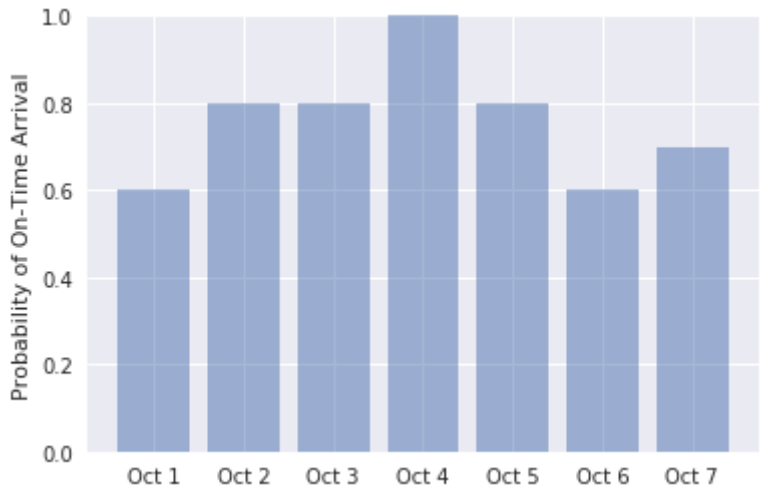
Этот рейс прибудет вовремя?

Теперь у вас есть простой способ прогнозирования своевременности прибытия с помощью одной строки кода. Вы можете поэкспериментировать с другими датами, временем, городами отправления и прибытия. Но имейте в виду, что результаты имеют смысл только для аэропортов с кодами ATL, DTW, JFK, MSP и SEA, так как модель обучалась только на этих кодах.

1. Выполните следующий код, чтобы вычислить вероятность своевременного прибытия вечернего рейса из JFK в ATL в диапазоне дней:

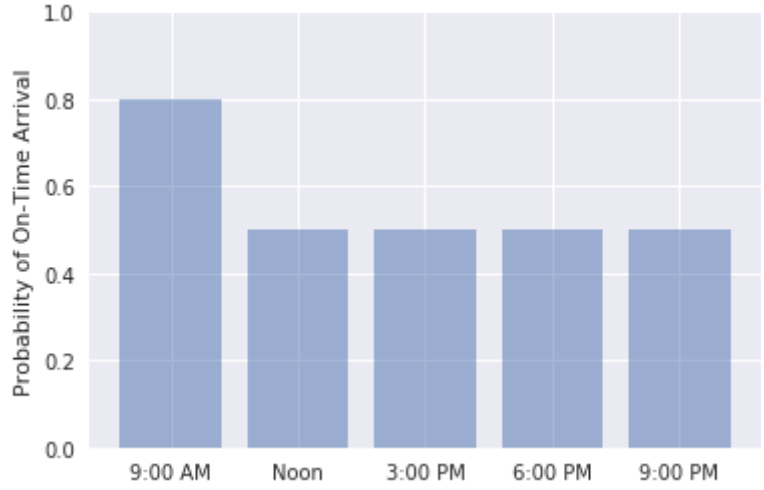
```
import numpy as np

labels = ('Oct 1', 'Oct 2', 'Oct 3', 'Oct 4', 'Oct 5', 'Oct 6', 'Oct 7')
values = (predict_delay('1/10/2018 21:45:00', 'JFK', 'ATL'),
          predict_delay('2/10/2018 21:45:00', 'JFK', 'ATL'),
          predict_delay('3/10/2018 21:45:00', 'JFK', 'ATL'),
          predict_delay('4/10/2018 21:45:00', 'JFK', 'ATL'),
          predict_delay('5/10/2018 21:45:00', 'JFK', 'ATL'),
          predict_delay('6/10/2018 21:45:00', 'JFK', 'ATL'),
          predict_delay('7/10/2018 21:45:00', 'JFK', 'ATL'))
alabels = np.arange(len(labels))

plt.bar(alabels, values, align='center', alpha=0.5)
plt.xticks(alabels, labels)
plt.ylabel('Probability of On-Time Arrival')
plt.ylim((0.0, 1.0))
```

2. Убедитесь, что выходные данные выглядят следующим образом:



Вероятность своевременного прибытия для диапазона дат

3. Измените код, чтобы создать аналогичную диаграмму для рейсов из JFK в MSP в 13:00 с 10 по 16 апреля. Как эти выходные данные соотносятся с выходными данными из предыдущего шага?
4. Самостоятельно напишите код, чтобы рассчитать вероятность того, что рейсы из SEA в ATL с отправлением в 9:00, в 12:00, в 15:00, в 18:00 и в 21:00 30 января прибудут вовремя. Убедитесь, что выходные данные выглядят следующим образом:



Вероятность своевременного прибытия для диапазона времени

Если вы не знакомы с библиотекой Matplotlib и хотите узнать о ней больше, ознакомьтесь с интересным учебником на странице https://www.labri.fr/perso/nrougier/teaching/matplotlib/. Библиотека Matplotlib умеет *гораздо* больше, чем мы здесь увидели, поэтому она так популярна в сообществе Python.

Итоги

Pandas, scikit-learn и Matplotlib являются одними из самых популярных библиотек Python в мире. С их помощью можно подготовить данные для

использования в машинном обучении, создать сложную модель машинного обучения на основе данных и вывести выходные данные.