

**Содержание**

Оглавление

[Введение 3](#_Toc137399571)

[выбор платформы и данных 4](#_Toc137399572)

[1.1 Выбор платформы для разработки и необходимых библиотек 4](#_Toc137399573)

[1.2 Получение данных 5](#_Toc137399574)

[1.2.1 Загрузка данных 5](#_Toc137399575)

[1.2.2 Предварительный анализ и обработка данных 6](#_Toc137399576)

[1.3 Выводы по разделу 13](#_Toc137399577)

[2 анализ данных 15](#_Toc137399578)

[2.1 Статистический анализ данных 15](#_Toc137399579)

[2.2 Работа с гипотезами 19](#_Toc137399580)

[2.3 Машинное обучение 30](#_Toc137399581)

[2.4 Настройка гиперпараметров выбранной модели 35](#_Toc137399582)

[2.6 Выводы по разделу 40](#_Toc137399583)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 42](#_Toc137399584)

**Введение**

Проектно-технологическая практика проходила в Московском университете им. С.Ю. Витте на кафедре информационных систем. Данная практика направлена на углублённую работу с языком Python для анализа и обработки данных и связанных с областью науки о данных библиотек. В качестве набора данных было решено взять открытый перечень аэропортов с веб-сайта <https://ourairports.com>.

Данные взяты отсюда: <https://www.kaggle.com/datasets/danishjmeo/world-airports-data>.

Официальная страница датасета: <https://ourairports.com/help/data-dictionary.html>.

Набор содержит подробную информацию о более 75000 аэропортов по всему миру, поэтому с ним интересно работать, а также есть возможность отобразить точки по координатам на карте.

**Цель практики**

Закрепление теоретических знаний библиотек Python для анализа данных, проведение полного анализа выбранного набора данных, включая предварительную обработку, очистку от цифрового мусора, анализ, поиск зависимостей, визуализацию и предсказание целевой переменной с помощью моделей машинного обучения.

**Задачи практики**

* Поработать в среде разработки Jupyter Notebook;
* Найти в интернете подходящий для обработки набор данных;
* Научиться восстанавливать недостающие данные в наборе, в том числе с помощью сторонних источников информации;
* Понять, какие признаки в наборе лучше не использовать, а удалить;
* Отобразить некоторую информацию из данных на карте. Например, с помощью Google Earth или Google Earth Engine;
* Преобразовать максимально возможное число текстовых признаков в числовые для того, чтобы можно было предсказать какой-либо признак;
* Попрактиковаться в статистической обработке данных, выделении групп записей и визуализации их с помощью различных графиков и диаграмм;
* Научиться ставить гипотезы и проверять их;
* Попрактиковаться в машинном обучении, а именно в предсказании выбранной целевой переменной из данных на основе других признаков; оптимизировать гиперпараметры одной из выбранных моделей.

В качестве материалов для изучения выступали открытые интернет-источники и документация по библиотекам Python.

**ОСНОВНАЯ ЧАСТЬ**

# выбор платформы и данных

## Выбор платформы для разработки и необходимых библиотек

В качестве среды разработки был выбран Jupyter Notebook, так как является распространённым среди специалистов обработки данных. Jupyter позволяет разделять части кода на ячейки и выводить полезную информацию прямо в окне браузера.

Для анализа данных были выбраны классические для этого библиотеки Pandas (работа с табличными данными), NumPy (для работы с массивами и матрицами), Matplotlib и Seaborn (для визуализации данных).

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction import DictVectorizer

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

import xgboost

from sklearn.model\_selection import StratifiedKFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import roc\_auc\_score

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import StratifiedKFold

%matplotlib inline

Для машинного обучения – Scikit-learn, так как является относительно простой, интуитивной и в то же время распространённой библиотекой; и xgboost, так как он представляет собой один из алгоритмов.

## Получение данных

### Загрузка данных

Источником данных стал открытый набор данных из ''Open data @ourAirports'' об аэропортах мира и прогнозах о задержках рейсов и маршрутах движения.

Официальная страница датасета: https://ourairports.com/help/data-dictionary.html

Там же находится подробное описание всех колонок.

Краткое описание колонок:

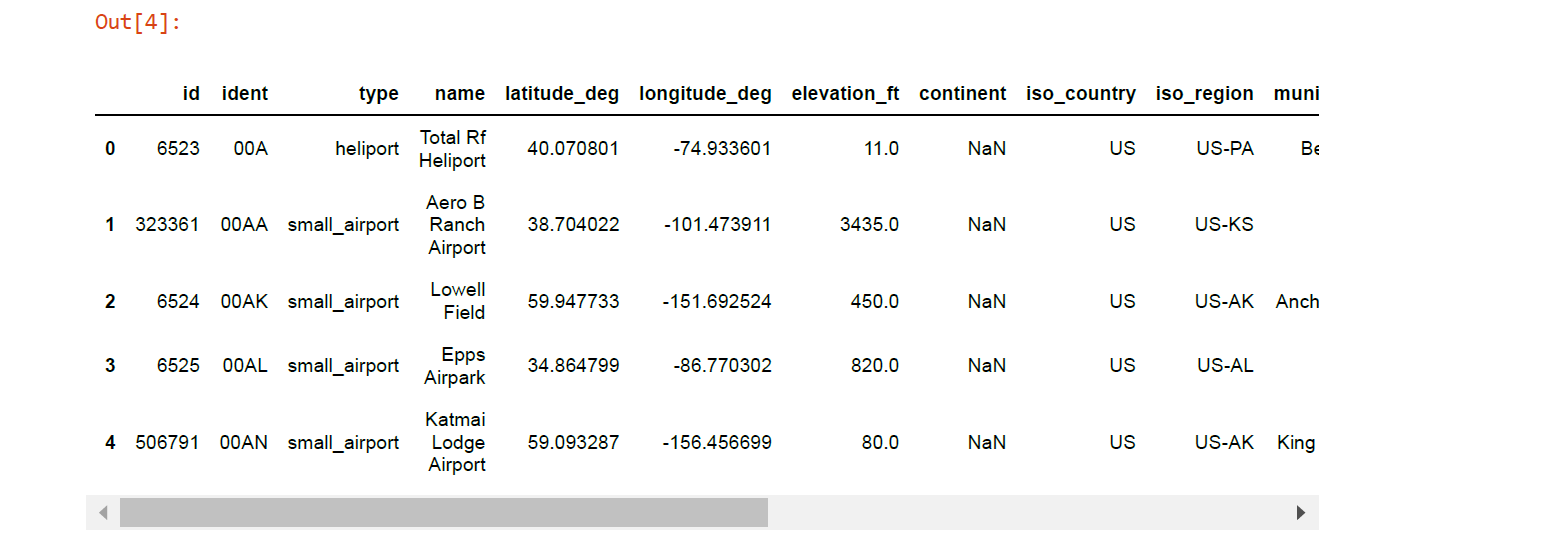
* id - идентификационный номер;
* ident - второй идентификационный номер. Название зависит от наличия ICAO и местного кода;
* type – тип;
* name – название;
* latitude\_deg - широта в десятичных градусах;
* longitude\_deg - долгота в десятичных градусах;
* elevation\_ft - высота над уровнем моря в футах;
* continent - континент (аббревиатура);
* iso\_country - страна (аббревиатура);
* iso\_region – регион;
* municipality – город;
* scheduled\_service - осуществляется регулярное авиасообщение;
* gps\_code - код GPS;
* iata\_code - Трёхбуквенный код iata;
* local\_code - местный код;
* home\_link – сайт;
* wikipedia\_link - страница на википедии;
* keywords - ключевые слова.

data = pd.read\_csv("airports.csv")

### Предварительный анализ и обработка данных

Для начала был были выведены первые значения таблицы для ознакомления с колонками и значениями:

data.head()



После количество значений и характеристик:

data.shape

(75605, 18)

Для удобства был создан отдельный датасет с кратким описанием колонок:

column\_descriptions = ["id", "идентификационный номер. То же самое, что и id, но в изменённом формате", "тип",

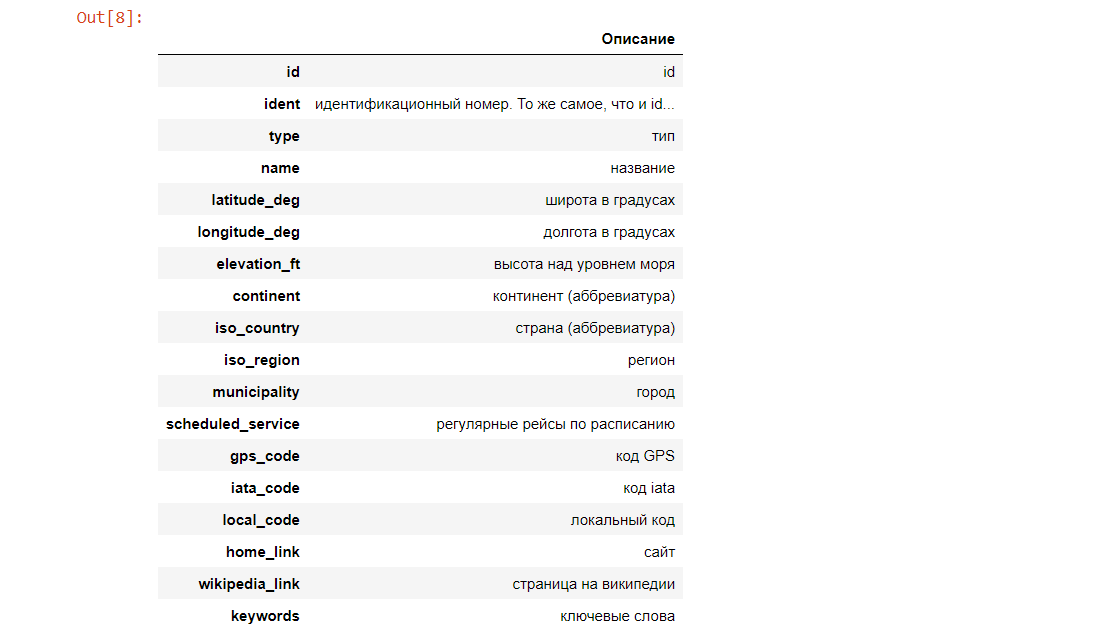
                       "название", "широта в градусах", "долгота в градусах", "высота над уровнем моря",

                       "континент (аббревиатура)", "страна (аббревиатура)", "регион", "город",

                       "регулярные рейсы по расписанию", "код GPS", "код iata", "локальный код", "сайт",

                       "страница на википедии", "ключевые слова"]

pd.DataFrame(columns=["Описание"], index = data.columns, data=column\_descriptions)



В столбце “continent” оказались значения NaN (отсутствие информации), хотя их там быть не должно. Путём двух команд я проверил, не обработался ли материк Северная Америка, подписанный “NA”, как пустое значение:

data[data["continent"] == "NA"]



data.continent.unique()



Оказалось, что материка “NA” здесь вообще нет.

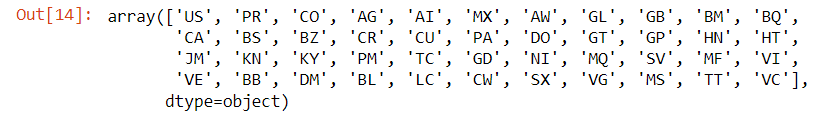
data["continent"].isna().sum()

Out[13]:

36606

NaN слишком много, поэтому я решил восстановить их по колонке страны, вместо удаления или заполнения популярным значением.

data[data["continent"].isna() == True]["iso\_country"].unique()



Выше приведены все значения стран. Так как вручную их проверять слишком долго, я воспользовался интеллектуальным чат-ботом ChatGPT. Были получены следующие значения:

* US - Соединенные Штаты Америки (NA - North America)
* PR - Пуэрто-Рико (NA - North America)
* CO - Колумбия (SA - South America)
* AG - Антигуа и Барбуда (NA - North America)
* AI - Ангилья (NA - North America)
* MX - Мексика (NA - North America)
* AW - Аруба (SA - South America)
* GL - Гренландия (NA - North America)
* GB - Великобритания (EU - Europe)
* BM - Бермудские острова (NA - North America)
* BQ - Бонайре, Синт-Эстатиус и Саба (SA - South America)
* CA - Канада (NA - North America)
* BS - Багамы (NA - North America)
* BZ - Белиз (NA - North America)
* CR - Коста-Рика (NA - North America)
* CU - Куба (NA - North America)
* PA - Панама (NA - North America)
* DO - Доминиканская Республика (NA - North America)
* GT - Гватемала (NA - North America)
* GP - Гваделупа (NA - North America)
* HN - Гондурас (NA - North America)
* HT - Гаити (NA - North America)
* JM - Ямайка (NA - North America)
* KN - Сент-Китс и Невис (NA - North America)
* KY - Каймановы острова (NA - North America)
* PM - Сен-Пьер и Микелон (NA - North America)
* TC - Теркс и Кайкос (NA - North America)
* GD - Гренада (NA - North America)
* NI - Никарагуа (NA - North America)
* MQ - Мартиника (NA - North America)
* SV - Сальвадор (NA - North America)
* MF - Сен-Мартен (NA - North America)
* VI - Виргинские острова США (NA - North America)
* VE - Венесуэла (SA - South America)
* BB - Барбадос (NA - North America)
* DM - Доминика (NA - North America)
* BL - Сен-Бартелеми (NA - North America)
* LC - Сент-Люсия (NA - North America)
* CW - Кюрасао (SA - South America)
* SX - Синт-Мартен (NA - North America)
* VG - Британские Виргинские острова (NA - North America)

Таким же способом была получена хеш-таблица со странами и материками, к которым они принадлежат:

country\_continent = {

    'US': 'NA',

    'PR': 'NA',

    'CO': 'SA',

    'AG': 'NA',

    'AI': 'NA',

    'MX': 'NA',

    'AW': 'SA',

    'GL': 'NA',

    'GB': 'EU',

    'BM': 'NA',

    'BQ': 'SA',

    'CA': 'NA',

    'BS': 'NA',

    'BZ': 'NA',

    'CR': 'NA',

    'CU': 'NA',

    'PA': 'NA',

    'DO': 'NA',

    'GT': 'NA',

    'GP': 'NA',

    'HN': 'NA',

    'HT': 'NA',

    'JM': 'NA',

    'KN': 'NA',

    'KY': 'NA',

    'PM': 'NA',

    'TC': 'NA',

    'GD': 'NA',

    'NI': 'NA',

    'MQ': 'NA',

    'SV': 'NA',

    'MF': 'NA',

    'VI': 'NA',

    'VE': 'SA',

    'BB': 'NA',

    'DM': 'NA',

    'BL': 'NA',

    'LC': 'NA',

    'CW': 'SA',

    'SX': 'NA',

    'VG': 'NA',

    'MS': 'NA',

    'TT': 'NA',

    'VC': 'NA'

}

Соответствующими значениями были заполнены недостающие строки:

data['continent'].fillna(data['iso\_country'].map(country\_continent))

data.continent.isnull().sum()

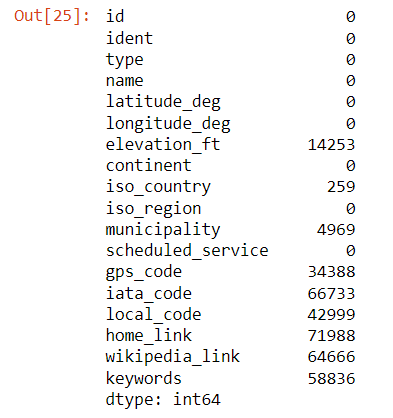
Out[23]:

0

Таким образом исчезли все пустые значения в одном из столбцов.

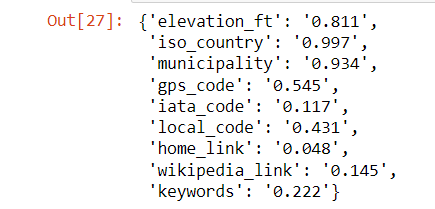
Другой командой я проверил, в каких столбцах сколько содержится пустых данных:

data.isnull().sum()



Далее я посчитал процент ненулевых данных по столбцам:

{k: "%.3f" %(v / data.shape[0]) for k, v in data.notnull().sum().to\_dict().items() if v < 75605}

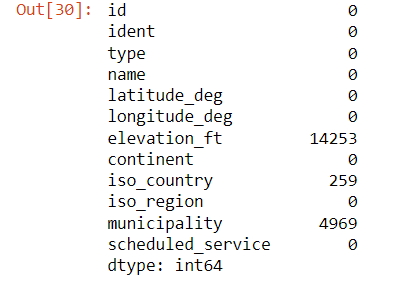


Минимальным процентом для оставления данных я взял 60%. Остальные было решено удалить из-за малой полезности. Такими оказались: gps\_code, iata\_code, local\_code, home\_link, wikipedia\_link, keywords.

useless\_columns = ["gps\_code", "iata\_code", "local\_code", "home\_link", "wikipedia\_link", "keywords"]

data = data.drop(useless\_columns, axis=1)

data.isnull().sum()



Так нулевых значений стало заметно меньше.

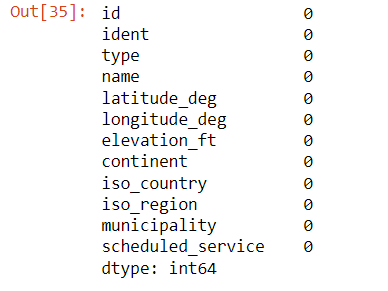
Оставшиеся пропуски были заполнены средним значением в случае количественной переменной elevation\_ft и модой в остальных случаях:

data["elevation\_ft"] = data["elevation\_ft"].fillna(data["elevation\_ft"].mean())

data["iso\_country"] = data["iso\_country"].fillna(data["iso\_country"].mode()[0])

data["municipality"] = data["municipality"].fillna(data["municipality"].mode()[0])

data.isnull().sum()



Теперь отсутствующие данные пропали.

Также было решено избавиться от признака "id". Из второго идентификационного признака "ident" можно извлечь хотя бы какую-то информацию, так как он содержит определённые паттерны, когда "id" полностью состоит из цифр без закономерностей. К тому же, в структуре таблицы Pandas уже есть числовая идентификация.

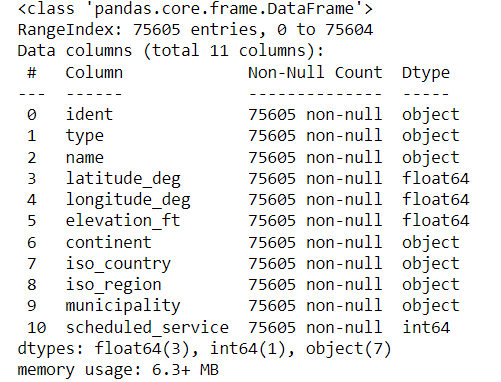
data = data.drop("id", axis=1)

Также потребовалось изменить значения признака "scheduled\_service" с "yes" и "no" на 0 и 1, чтобы качественных (то есть, нечисловых) данных стало меньше. Ведь в последующем нужно будет обучать модели для предсказания переменной, а они могут работать только с числовым видом данных.

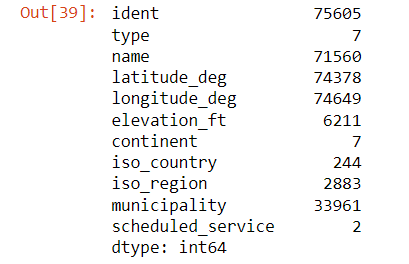
data["scheduled\_service"] = data["scheduled\_service"].replace({"no": 0, "yes": 1})

С помощью следующих команд я посмотрел, какие характеристики к какой группе отнести по типу данных:

data.info()



data.nunique()



Получилось следующее:

* качественные признаки: ident, name, municipality;
* категориальные признаки: type, continent, iso\_country, iso\_region, scheduled\_service;
* количественные признаки: latitude\_deg, longitude\_deg, elevation\_ft.

One Hot Encoding для преобразования категориальных признаков в числовые лучше сделать после анализа для удобства восприятия.

## 1.3 Выводы по разделу

По итогам первого раздела была выбрана платформа для дальнейшей работы; проведены предварительный анализ набора данных, его обработка и очистка от цифрового мусора; а также были заполнены недостающие значения записей и разбиты характеристики по группам.

Таблица 1: Выводы по разделу 1

|  |  |  |
| --- | --- | --- |
| Выводы | Сформированные компетенции | |
| Код | Содержание компетенции |
| Нашёл и проанализировал полезную для работы информацию | УК-1 | Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач. |
| Выделил задачи; выбрал нужные библиотеки Python и среду разработки | УК-2 | Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений |
| Посоветовался со знакомыми по возникшим проблемам и способам их решения в ходе анализа, разбил всю работу на этапы | УК-3 | Способен осуществлять социальное взаимодействие и реализовывать свою роль в команде |
| Составил отчётную документацию на русском языке, прочитал документацию и статьи на английском, перевёл необходимую информацию в ходе работы для более удобной работы | УК-4 | Способен осуществлять деловую коммуникацию в устной и письменной формах на государственном языке Российской Федерации и иностранном(ых) языке(ах) |
| Прочитал про различные подходы к анализу данных и соответствующий инструментарий | УК-5 | Способен воспринимать межкультурное разнообразие общества в социально-историческом, этическом и философском контекстах |
| Составил график выполнения задач, проанализировал большой объём необходимой информации, использовал свои предыдущие наработки и конспекты по теме | УК-6 | Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни |
| Соблюдал цифровой баланс, занимался спортом между тем, как выполнял работу, проветривал помещение 3 раза в день, выходил на улицу | УК-7 | Способен поддерживать должный уровень физической подготовленности для обеспечения полноценной социальной и профессиональной деятельности |
| Использовал для работы энергоэффективный ноутбук, не прибегал к инструментам, значительно загрязняющим окружающую среду | УК-8 | Способен создавать и поддерживать в повседневной жизни и в профессиональной деятельности безопасные условия жизнедеятельности для сохранения природной среды, обеспечения устойчивого развития общества, в том числе при угрозе и возникновении чрезвычайных ситуаций и военных конфликтов |
| Использовал открытые ресурсы в ходе выполнения практики | УК-9 | Способен принимать обоснованные экономические решения в различных областях жизнедеятельности |
| Всю работу делал сам, используя открытую информацию и свои конспекты по пройденным курсам | УК-10 | Способен формировать нетерпимое отношение к коррупционному поведению |
| Использовал полученные при обучении знания по теории вероятности, анализу и обработке данных | ОПК-1 | Способен применять естественнонаучные и общеинженерные знания, методы математического анализа и моделирования, теоретического и экспериментального исследования в профессиональной деятельности; |
| Самостоятельно использовал современные инструменты для анализа данных | ОПК-2 | Способен понимать принципы работы современных информационных технологий и программных средств, в том числе отечественного производства, и использовать их при решении задач профессиональной деятельности; |
| Произвёл поиск нужной информации в документации по инструментам и на обучающих веб-сайтах | ОПК-3 | Способен решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности; |
| Написал данный отчёт и блокнот Jupyter Notebook с кодом и поясняющим текстом | ОПК-4 | Способен участвовать в разработке стандартов, норм и правил, а также технической документации, связанной с профессиональной деятельностью; |
| Установил подходящее для данной работы программное обеспечение, импортировал библиотеки Python | ОПК-5 | Способен инсталлировать программное и аппаратное обеспечение для информационных и автоматизированных систем; |
| Организовал комфортную работу для выполнения практики, включая подходящее программное обеспечение | ОПК-6 | Способен анализировать и разрабатывать организационно-технические и экономические процессы с применением методов системного анализа и математического моделирования; |
| Произвёл грамотный процесс анализа данных | ОПК-7 | Способен разрабатывать алгоритмы и программы, пригодные для практического применения; |
| Подготовил выбранный набор данных для дальнейшей работы с ним в рамках жизненного цикла проекта | ОПК-8 | Способен принимать участие в управлении проектами создания информационных систем на стадиях жизненного цикла; |

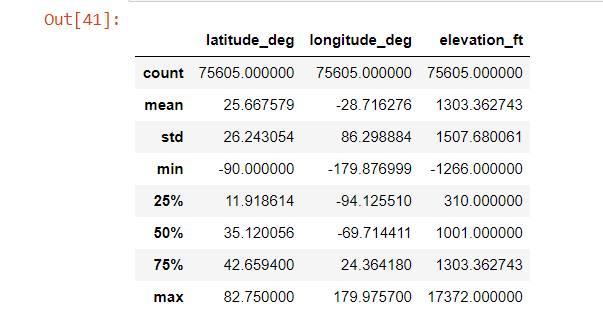
# 

# анализ данных

## 2.1 Статистический анализ данных

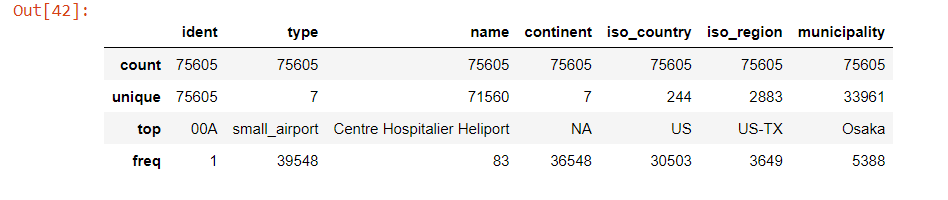
По количественным данным вывел распространённые статистические характеристики:

data[["latitude\_deg", "longitude\_deg", "elevation\_ft"]].describe()



По текстовым признакам несколько другие показатели:

data.describe(include="object")



По классифицирующим признакам вывел гистограммы:

fig = plt.figure(figsize=(15, 6))

plt.hist(data.type, rwidth=0.8, alpha=0.75)

plt.title("Типы аэропортов")

plt.plot()

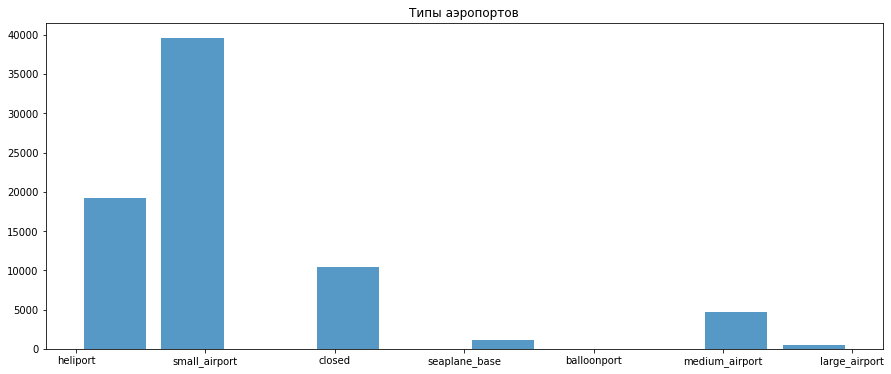


fig = plt.figure(figsize=(15, 6))

plt.hist(data.continent, rwidth=0.8, color="green", alpha=0.75)

plt.title("Континенты")

plt.plot()

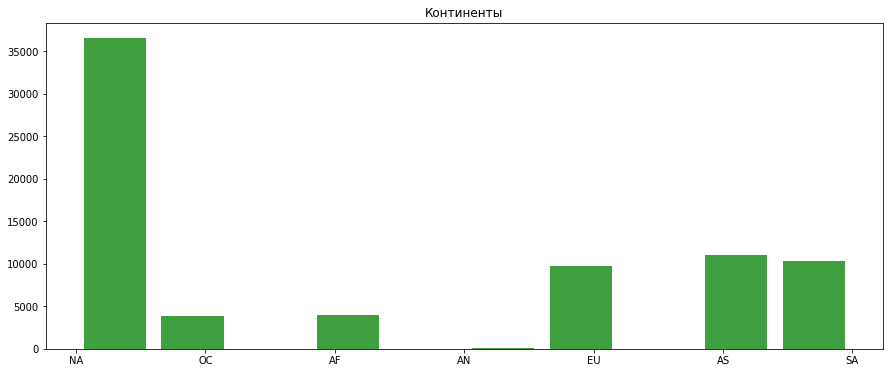
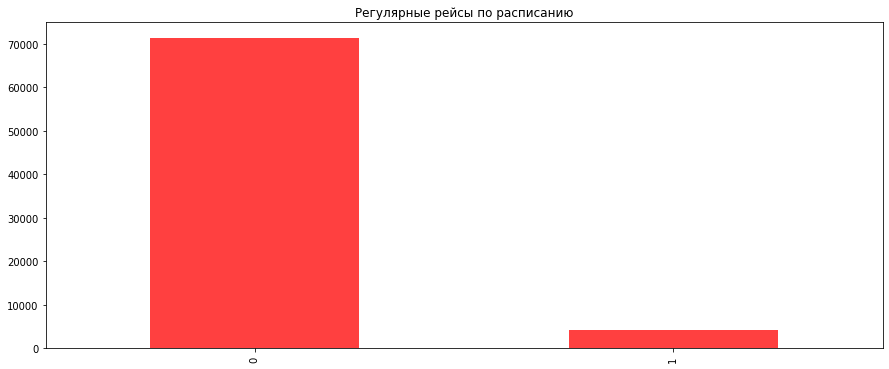


fig = plt.figure(figsize=(15, 6))

data.scheduled\_service.value\_counts().plot.bar(color="red", alpha=0.75)

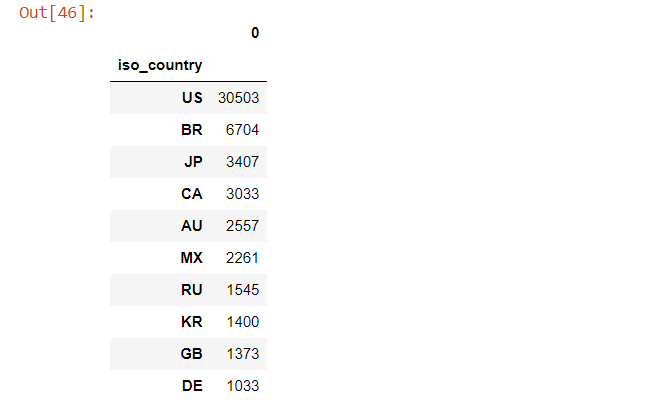
plt.title("Регулярные рейсы по расписанию")

plt.plot()

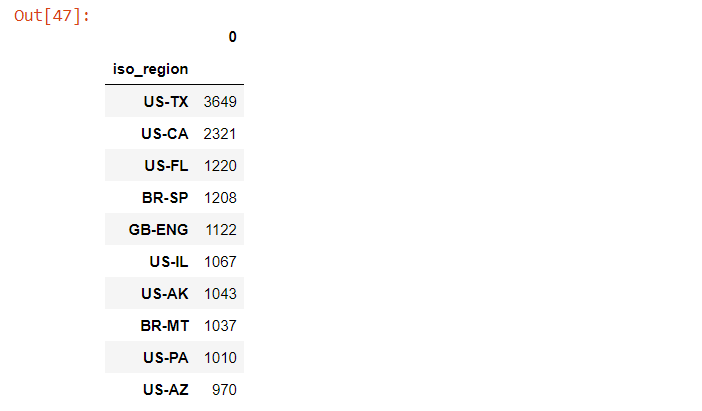


По признакам iso\_country и iso\_region вывел топ-10 популярных значений, так как в них слишком много уникальных записей:

data[["iso\_country"]].value\_counts().to\_frame().head(10)



data[["iso\_region"]].value\_counts().to\_frame().head(10)

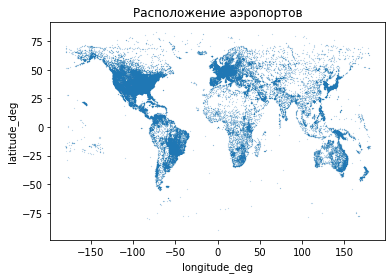


По широте и долготе отобразил преобладающие области расположения аэропортов:

data.plot.scatter(x="longitude\_deg", y="latitude\_deg", s=0.01)

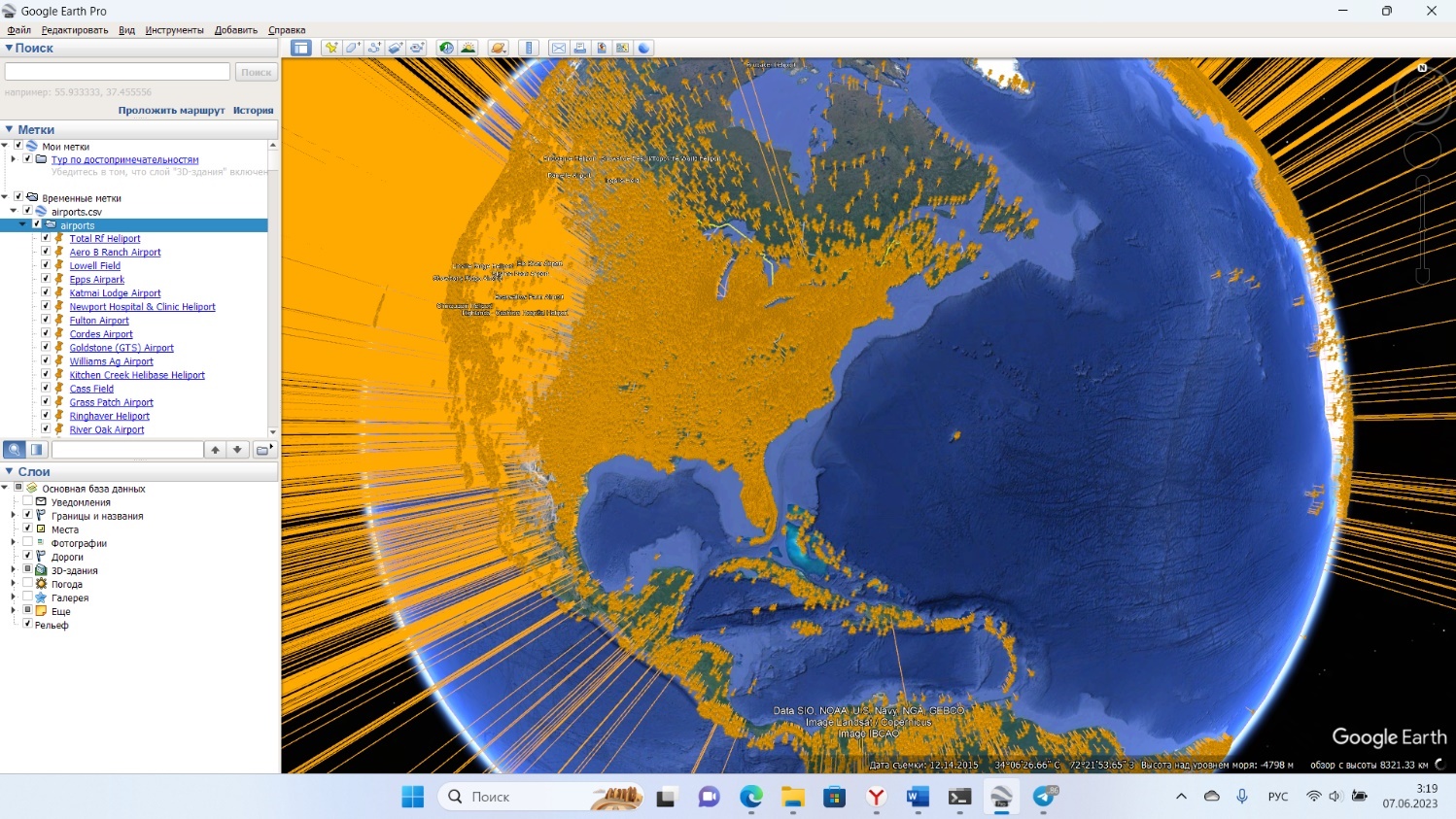
plt.title("Расположение аэропортов")

plt.plot()

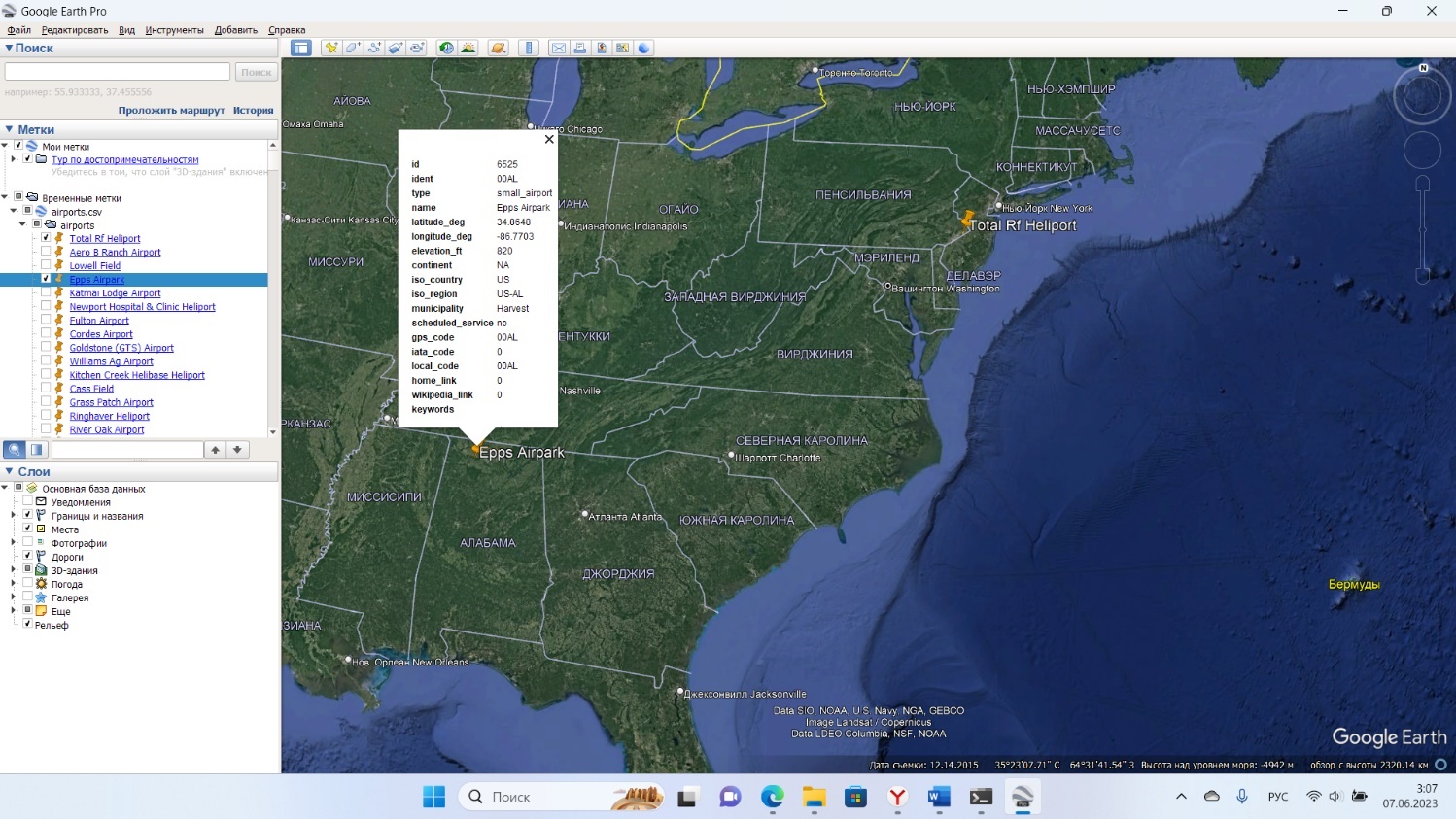


Получилась карта мира. Был сделан вывод о том, что преобладающая часть аэропортов находится в Северной и Южной Америках, а также в Европе.

Но гораздо удобнее это сделать в приложении Google Earth Pro. В силу того, что данные изменились, был использован изначальный набор данных:



Для комфортной работы с данными необходимо мощное оборудование, поэтому лучше не включать все аэропорты одновременно.



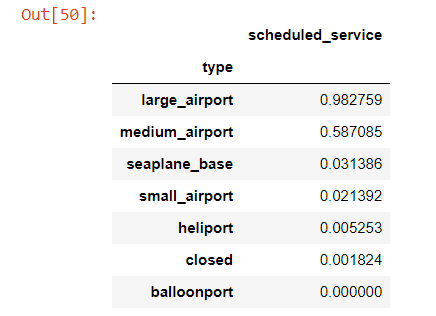
По каждой точке можно перейти к конкретному местоположению на глобусе и посмотреть подробную информацию.

## 2.2 Работа с гипотезами

Было предложено и проверено несколько гипотез.

1. Наличие регулярных рейсов по расписанию должно зависеть от типа аэропорта. Скорее всего, у больших аэропортов вероятность этого выше.

data.groupby('type').mean()["scheduled\_service"].to\_frame().sort\_values(by="scheduled\_service", ascending=False)



Действительно, у больших и средних аэропортов вероятность наличия регулярных рейсов сильно выше, чем у остальных

Отобразил типы аэропортов на графике:

fig = plt.figure(figsize=(15, 6))

data.groupby('type').mean()["scheduled\_service"].plot.bar(width=0.95, color="orange", alpha=0.75)

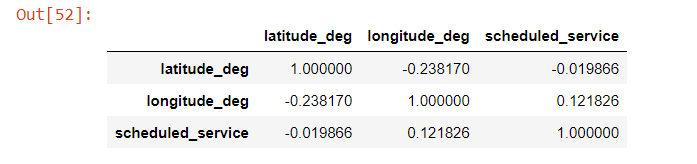
plt.title("Среднее значение наличия регулярных рейсов по типам аэропортов")

plt.show()



2) Возможно, географическое расположение (в десятичных градусах по широте и долготе) также влияет на наличие регулярных рейсов.

data[["latitude\_deg", "longitude\_deg", "scheduled\_service"]].corr()

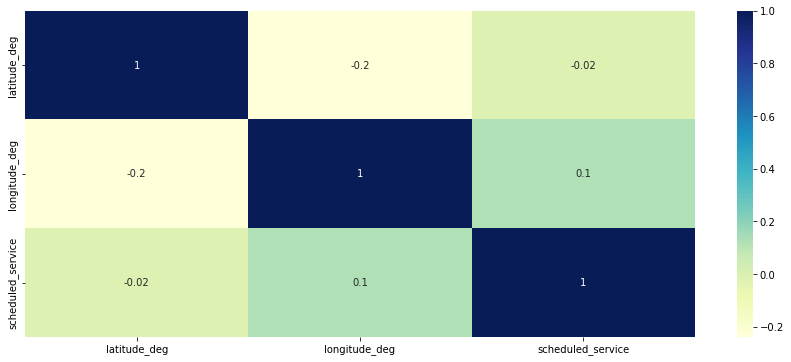


Рассчитал корреляцию между широтой, долготой и наличием регулярных рейсов:

fig = plt.figure(figsize=(15, 6))

sns.heatmap(data[["latitude\_deg", "longitude\_deg", "scheduled\_service"]].corr(), annot=True, fmt='.1g', cmap="YlGnBu")

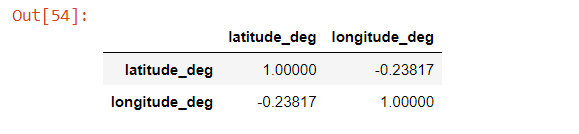
plt.plot()



Как видно из графика, на наличие регулярного авиасообщения больше всего влияет долгота аэропорта. Но зависимость не сильно большая.

3) Должна быть зависимость между широтой и долготой местоположения аэропортов. Данная информация может рассказать о географических особенностях расположения аэропортов, а также о том, в каких областях расположено больше всего аэропортов с наличием регулярных рейсов.

data[["latitude\_deg", "longitude\_deg"]].corr()



Отобразил разброс значений данных признаков:

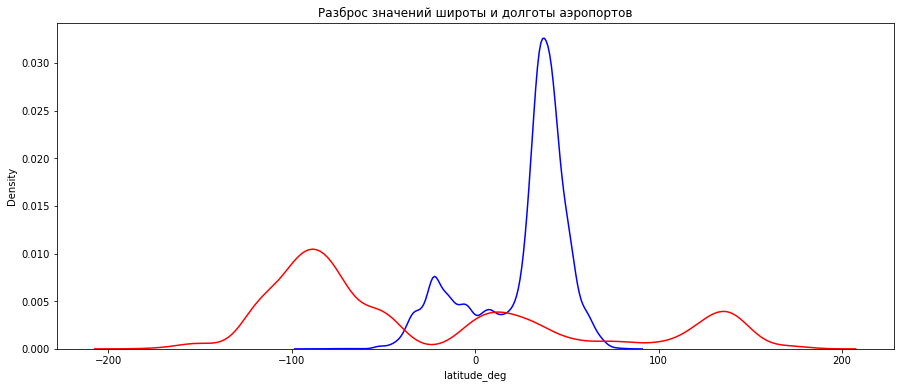
fig = plt.figure(figsize=(15, 6))

sns.kdeplot(data['latitude\_deg'], color="b")

sns.kdeplot(data['longitude\_deg'], color="r")

plt.title("Разброс значений широты и долготы аэропортов")

plt.show()



Как было показано ранее, можно выделить определённые области на карте по координатам, в которых у аэропортов больше вероятность иметь регулярные рейсы:

data[data["scheduled\_service"] == 1].plot.scatter(x="longitude\_deg", y="latitude\_deg", s=0.5)

plt.title("Расположение аэропортов")

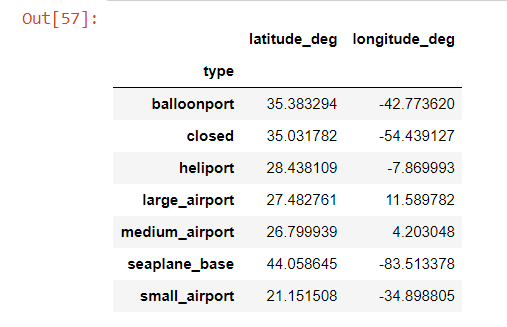
plt.plot()



Размер точек был изменён для большей наглядности. Можно заметить несколько крупных областей с преобладанием аэропортов, у которых есть регулярные рейсы.

4) На основании зависимости широты и долготы, а также зависимости наличия регулярных рейсов от типа аэропорта, была предложена гипотеза о зависимости типа аэропорта от его географического положения.

data.groupby('type').mean()[["latitude\_deg", "longitude\_deg"]]



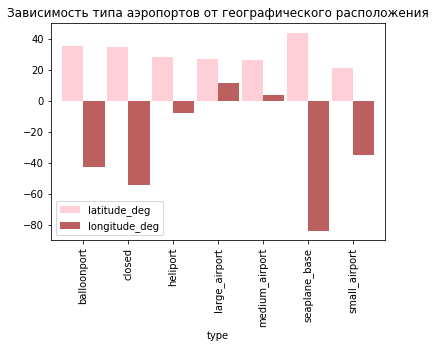
Вывел сначала график зависимости:

fig = plt.figure(figsize=(15, 6))

data.groupby('type').mean()[["latitude\_deg", "longitude\_deg"]].plot.bar(width=0.95, color=("pink", "brown"), alpha=0.75)

plt.title("Зависимость типа аэропортов от географического расположения")

plt.show()



И графики координат по типам аэропортов:

colors= iter(["brown", "green", "red", "blue", "indigo", "orange", "fuchsia"])

for t in data["type"].unique().tolist():

    fig = plt.figure(figsize=(15, 6))

    plt.scatter(data=data[data["type"] == t], x="longitude\_deg", y="latitude\_deg", color=next(colors), s=10)

    plt.title(t)

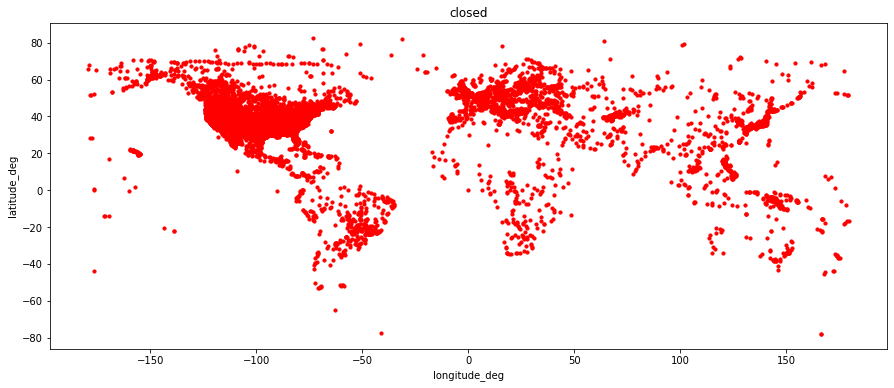
    plt.xlabel("longitude\_deg")

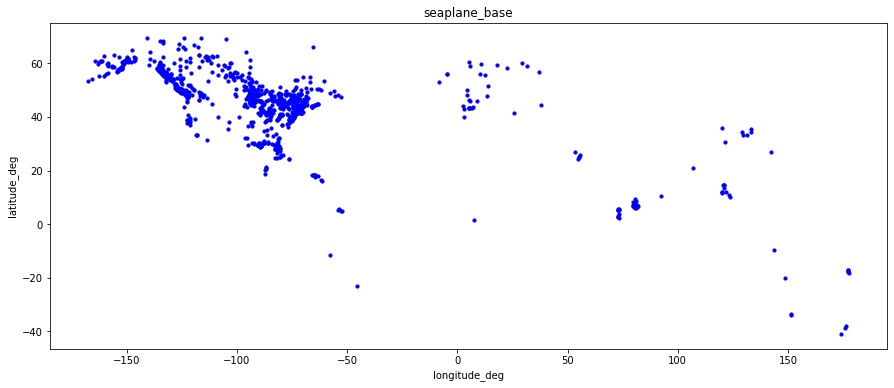
    plt.ylabel("latitude\_deg")

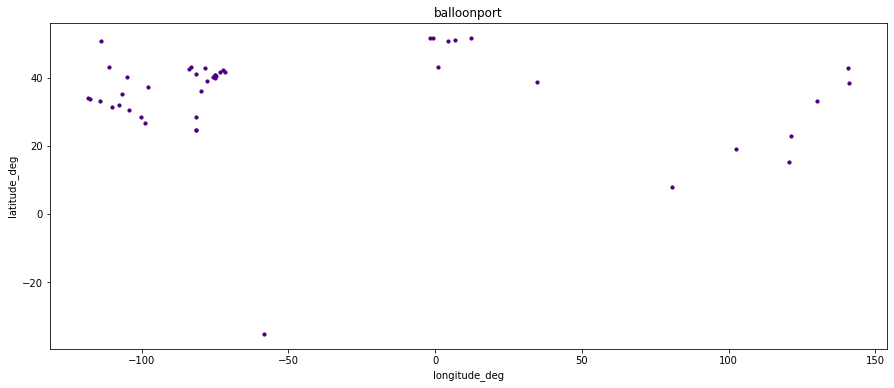
    plt.show()



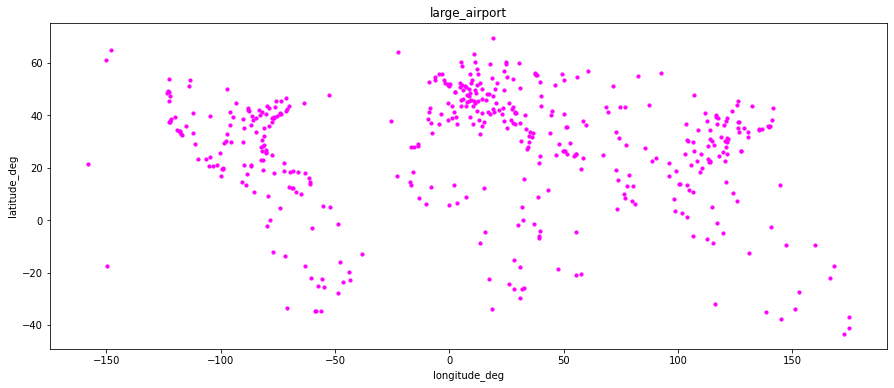












На картах можно увидеть, где каких типов аэропортов больше всего

Также, была выведена гистограмма распределения типов аэропортов:

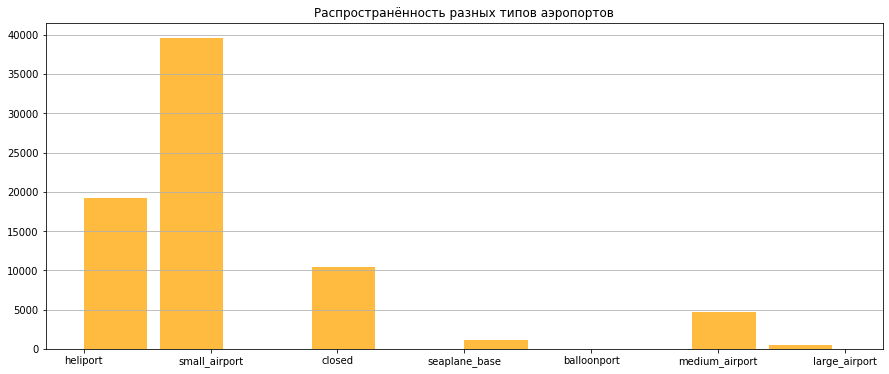
fig = plt.figure(figsize=(15, 6))

data["type"].hist(width=0.5, color="orange", alpha=0.75)

plt.title("Распространённость разных типов аэропортов")

plt.grid(False, axis="x")

plt.show()

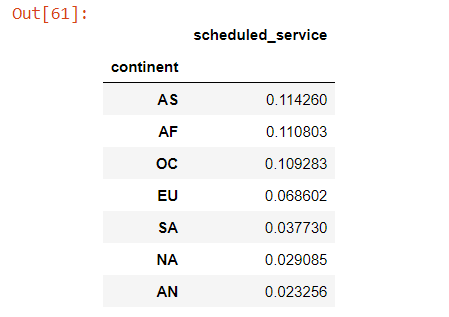


Вышло, что больше всего распространены маленькие аэропорты; меньше средние, закрытые и вертолётные площадки; больших аэропортов и баз гидросамолётов заметно меньше; а площадок запуска воздушных шаров практически нет.

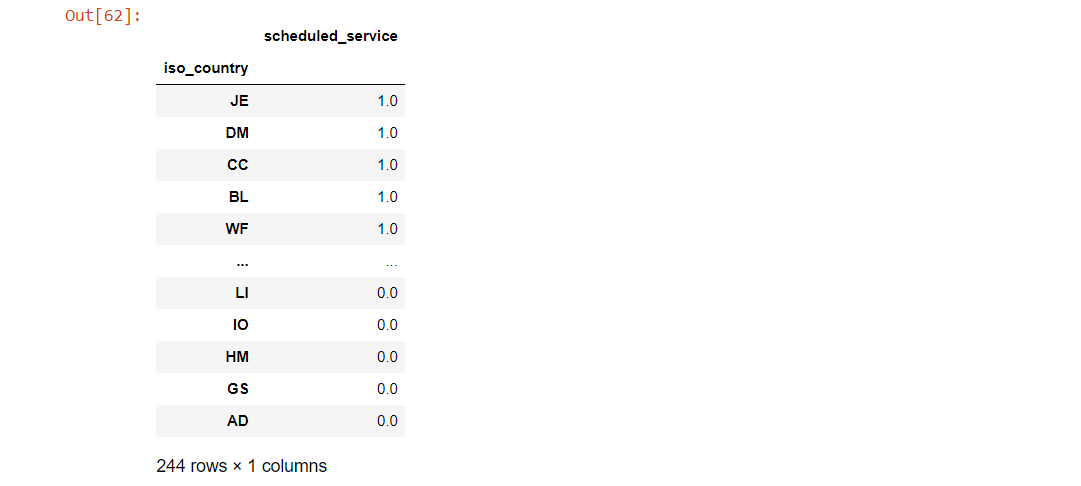
5) Ещё было предложено, что наличие регулярных рейсов зависит от материка, страны и региона.

Выделил новые наборы данных:

data.groupby('continent').mean()["scheduled\_service"].sort\_values(ascending=False).to\_frame()



data.groupby('iso\_country').mean()["scheduled\_service"].sort\_values(ascending=False).to\_frame()



data.groupby('iso\_region').mean()["scheduled\_service"].sort\_values(ascending=False).to\_frame()



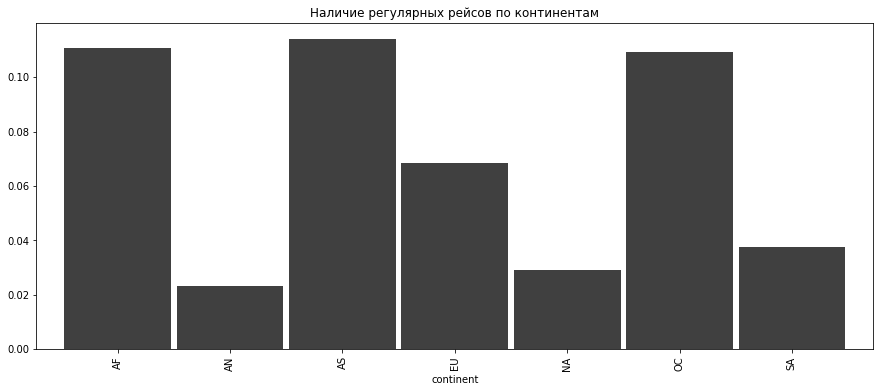
Сначала вывел диаграмму зависимости данного признака от материка, на котором расположен аэропорт:

fig = plt.figure(figsize=(15, 6))

data.groupby('continent').mean()["scheduled\_service"].plot.bar(width=0.95, color="black", alpha=0.75)

plt.title("Наличие регулярных рейсов по континентам")

plt.show()

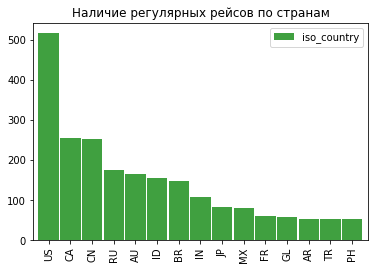


Заметно выделяются Африка, Азия, Австралия и Океания; за ними Европа. Стоит заметить, что в использованных данных Австралия и Океания относятся к признаку "Материки", хотя на самом деле являются частью Света.

data[data.scheduled\_service == True].iso\_country.value\_counts().to\_frame().head(15).plot.bar(width=0.95, color="green", alpha=0.75)

plt.title("Наличие регулярных рейсов по странам")

plt.figure()



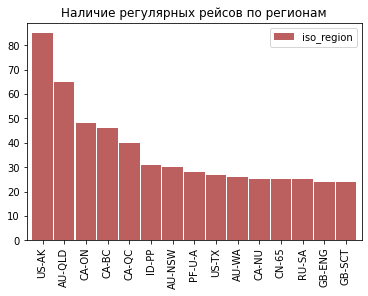
На первом месте США, после идут Канада, Китай и другие страны.

И в подобном виде отобразил регионы:

data[data.scheduled\_service == True].iso\_region.value\_counts().to\_frame().head(15).plot.bar(width=0.95, color="brown", alpha=0.75)

plt.title("Наличие регулярных рейсов по регионам")

plt.figure()



Преобладают штаты Аляска США и Квинсленд Австралии.

6) Ранее я оставил столбец “ident” для выделения новых признаков. Попробовал добавить к набору данных столбец длины “ident” и посмотреть корреляцию с наличием регулярных рейсов:

len\_ident\_list = [len(id) for id in data.ident]

data.insert(loc=len(data.columns), column='len\_ident', value=len\_ident\_list)

data.head()

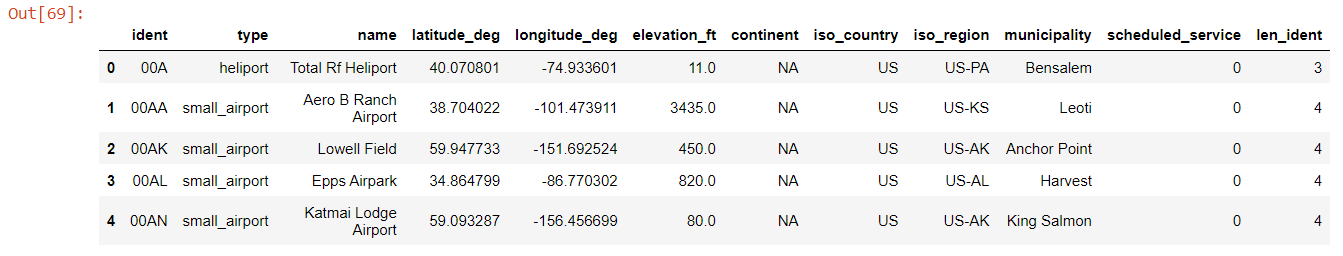
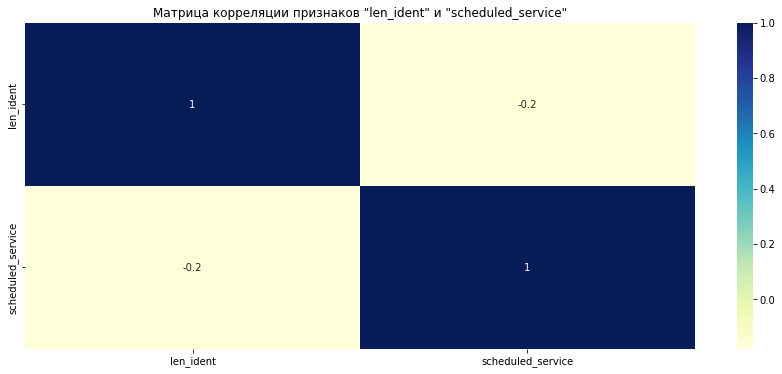


fig = plt.figure(figsize=(15, 6))

sns.heatmap(data[["len\_ident", "scheduled\_service"]].corr(), annot=True, fmt='.1g', cmap="YlGnBu")

plt.title('Матрица корреляции признаков "len\_ident" и "scheduled\_service"')

plt.plot()



Как видно на тепловой карте, существует некоторая зависимость, сравнимая с корреляцией между долготой и наличием регулярных рейсов. Новый признак было решено оставить.

## 2.3 Машинное обучение

Путём тестов было выявлено, что набор данных слишком большой для используемого компьютера. К тому же, в нём содержатся числа с плавающей точкой высокой точности. Я решил сократить объём всех записей и избавиться от характеристики "municipality".

data.shape

(75606, 12)

Это будет сделано прямо в ходе разделения данных на 2 части с функцией train\_test\_split.

Для начала датасет был разделён на тренировочные и тестовые данные. С помощью тренировочных данных будет проводиться обучение моделей, а с помощью тестовых - тестироваться их точность

train\_data, test\_data = train\_test\_split(data, train\_size=0.4, random\_state=1)

train\_data.reset\_index(inplace=True)

test\_data.reset\_index(inplace=True)

test\_data, extra\_data = train\_test\_split(test\_data, train\_size=0.2, random\_state=1)

train\_data.shape

(30242, 13)

test\_data.shape

(9072, 13)

Затем был выделен массив со значениями целевой переменной y, отвечающей за наличие регулярных рейсов по расписанию у аэропорта:

y\_train = train\_data.scheduled\_service

y\_test = test\_data.scheduled\_service

Теперь я воспользовался функцией DictVectorizer. Она использует метод One Hot Encoding, который перекодирует все качественные признаки в несколько столбцов. Если запись принадлежит к категории, ставится 1, иначе – 0. Преимущество функции DictVectorizer, по сравнению с другими функциями, имплементирующими One Hot Encoding, состоит в возможности обучить её на одном разрезе данных и применить на другом, сохраняя одинаковое количество признаков у выходящих наборов данных. То, что нужно в моём случае.

useful\_features = ['type', 'continent', 'iso\_country', 'iso\_region',

                   'latitude\_deg', 'longitude\_deg', 'elevation\_ft', 'len\_ident']

train\_dicts = train\_data[useful\_features].to\_dict(orient='records')

test\_dicts = test\_data[useful\_features].to\_dict(orient='records')

dv = DictVectorizer(sparse=False)

X\_train = dv.fit\_transform(train\_dicts)

X\_test = dv.transform(test\_dicts)

X\_train.shape

(30242, 2462)

X\_test.shape

(9072, 2462)

После я произвёл нормирование данных для ускорения обучения:

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

Я выбрал 3 разных алгоритма для проверки: логистическую регрессию, случайный лес и XGBoost. Все они на выходе дают или положительное значение или отрицательное, то есть, классифицируют объект по признакам.

Сначала обучил логистическую регрессию:

model = LogisticRegression(max\_iter=100)

model.fit(X\_train, y\_train)

Написал специальную функцию для отображения метрик:

def print\_metrics(input\_model):

    y\_train\_prediction = input\_model.predict(X\_train)

    y\_test\_prediction = input\_model.predict(X\_test)

    print('Train MSE: %.3f' % mean\_squared\_error(y\_train, y\_train\_prediction))

    print('Test MSE: %.3f' % mean\_squared\_error(y\_test, y\_test\_prediction))

    print('Train MAE: %.3f' % mean\_absolute\_error(y\_train, y\_train\_prediction))

    print('Test MAE: %.3f' % mean\_absolute\_error(y\_test, y\_test\_prediction))

    CM = confusion\_matrix(y\_test, y\_test\_prediction)

    ax= plt.subplot()

    sns.heatmap(CM, annot=True, fmt="g", cmap="YlGnBu")

    ax.set\_xlabel('Предсказанные значения')

    ax.set\_ylabel('Действительные значения')

    ax.set\_title('Матрица ошибок')

    ax.xaxis.set\_ticklabels(['False', 'True'])

    ax.yaxis.set\_ticklabels(['False', 'True'])

    plt.plot()

    TN = CM[0][0]

    FN = CM[1][0]

    TP = CM[1][1]

    FP = CM[0][1]

    accuracy = (TP + TN) / (TP + TN + FP + FN)

    precision = TP / (TP + FP)

    recall = TP / (TP + FN)

    print("accuracy: %.2f" % (accuracy))

    print("precision: %.2f" % (precision))

    print("recall: %.2f" % (recall))

    print("roc\_auc\_score: %.2f" % roc\_auc\_score(y\_train, y\_train\_prediction))

MSE - среднеквадратичное значение ошибки.

MAE - абсолютное (то есть, ризница между действительным и предсказанным значением) значение ошибки.

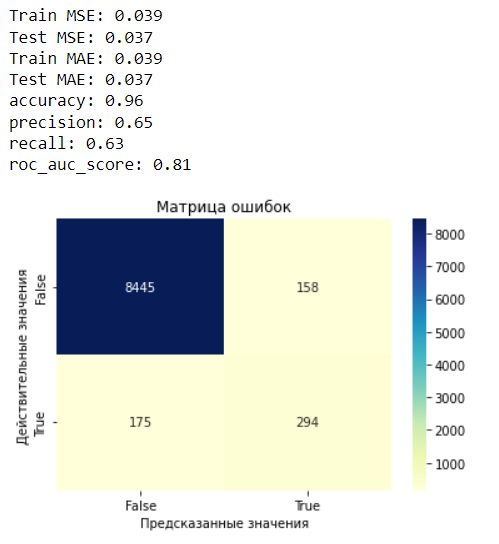
Метрика accuracy отвечает за общий процент правильных ответов.

Precison показывает точность определения положительных ответов.

Recall отвечает за то, насколько полно модель охватывает положительные ответы.

Roc\_auc\_score показывает процент площади правильных ответов, а именно то, насколько модель точно различает 2 класса. Далее, при настройке гиперпараметров я использовал именно эту метрику.

print\_metrics(model)



Даже с учётом относительно небольшого объёма данных и модели с маленьким количеством итераций результат впечатляет.

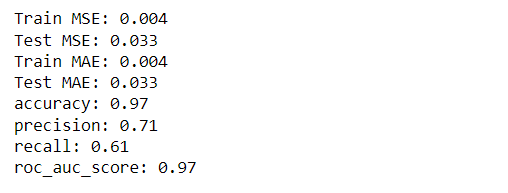
Попробовал обучить RandomForestClassifier:

model = RandomForestClassifier(n\_estimators=10, random\_state=1)

model.fit(X\_train, y\_train)

RandomForestClassifier(n\_estimators=10, random\_state=1)

print\_metrics(model)



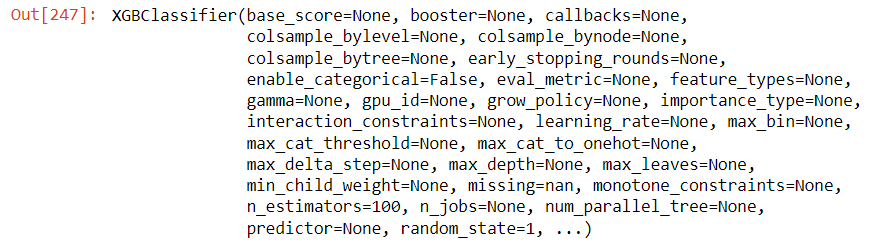


Результат значительно улучшился.

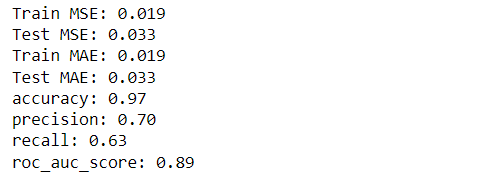
Теперь XGBoost:

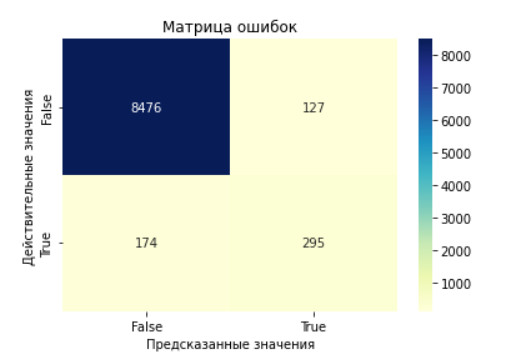
model = xgboost.XGBClassifier(n\_estimators=100, random\_state=1)

model.fit(X\_train, y\_train)



print\_metrics(model)





Алгоритм XGBoost требует слишком много ресурсов. К тому же, предыдущие модели справились лучше. Эффективнее будет подобрать параметры для RandomForestClassifier, ведь он показал наилучший результат.

## 2.4 Настройка гиперпараметров выбранной модели

Создал экземпляр объекта для кросс-валидации данных. С помощью него можно эффективно проверять качество модели без риска получить искажённый результат из-за разницы данных. Со StratifiedKFold можно разделить весь набор данных на несколько частей и проверить каждую, а потом получить усреднённый результат.

cv = StratifiedKFold(5, random\_state=1, shuffle=True)

Написал специальную функцию для настройки гиперпараметров и вывода результатов на график:

def search(X, y, model, param\_name, grid):

    parameters = {param\_name: grid}

    CV\_model = GridSearchCV(estimator=model, param\_grid=parameters, cv=cv, scoring="roc\_auc", n\_jobs=-1)

    CV\_model.fit(X, y)

    means = CV\_model.cv\_results\_["mean\_test\_score"]

    error = CV\_model.cv\_results\_["std\_test\_score"]

    plt.figure(figsize=(15, 8))

    plt.title("Выбор параметра " + param\_name)

    plt.plot(grid, means, label="mean values of score")

    plt.fill\_between(grid, means - 2 \* error, means + 2 \* error, color="green", label="Область отклонения между ошибками")

    plt.legend(loc="upper left")

    plt.xlabel("Параметр")

    plt.ylabel("roc-auc")

    plt.grid()

    plt.show()

Сначала решил подобрать оптимальное значение количества деревьев в лесу алгоритма.

Ограничил количество записей для обучения 1000, чтобы не тратить слишком много времени.

models = [RandomForestClassifier(random\_state=1, n\_jobs=-1)]

param\_names = ["n\_estimators"]

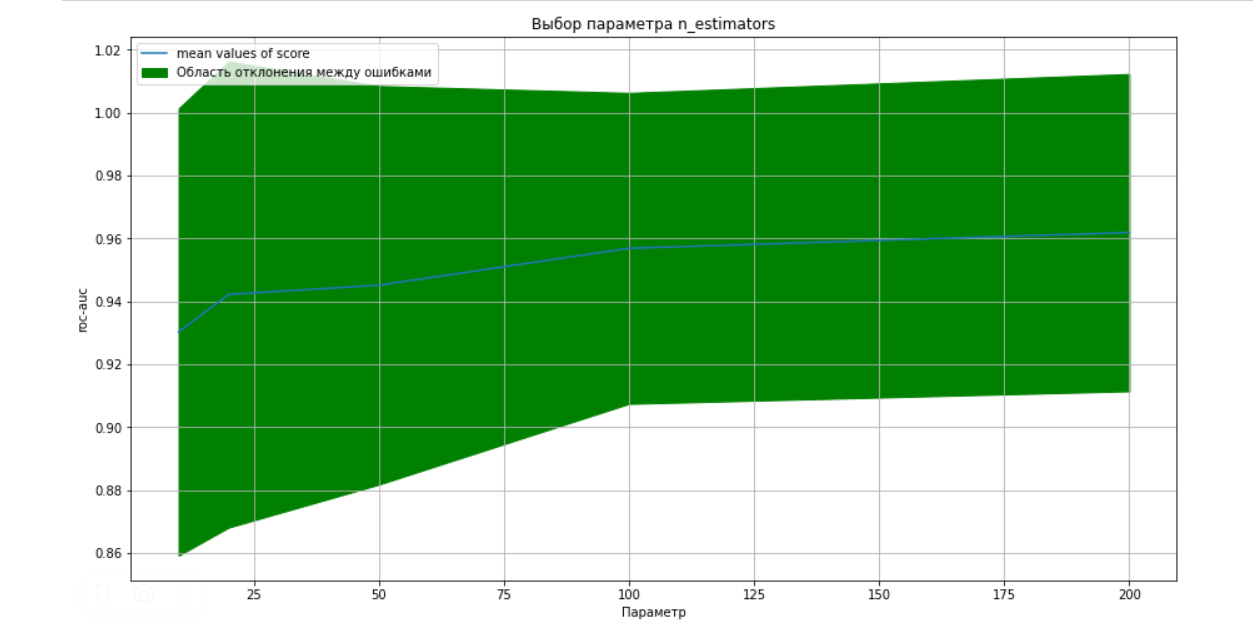
grids = [np.array([10, 20, 50, 100, 200], dtype="int")]

for model, param\_name, grid in zip(models,

                                   param\_names,

                                   grids):

    search(X\_train[:1000], y\_train[:1000], model, param\_name, grid)



Получилось, что значение 100 наиболее оптимально.

Дальше настроил таким же образом параметр максимальной глубины "max\_depth". Здесь тоже ограничил размер тренировочных данных, ятобы не ждать слишком долго. Но уже до 10000 записей.

models = [RandomForestClassifier(n\_estimators=100, random\_state=1, n\_jobs=-1)]

param\_names = ["max\_depth"]

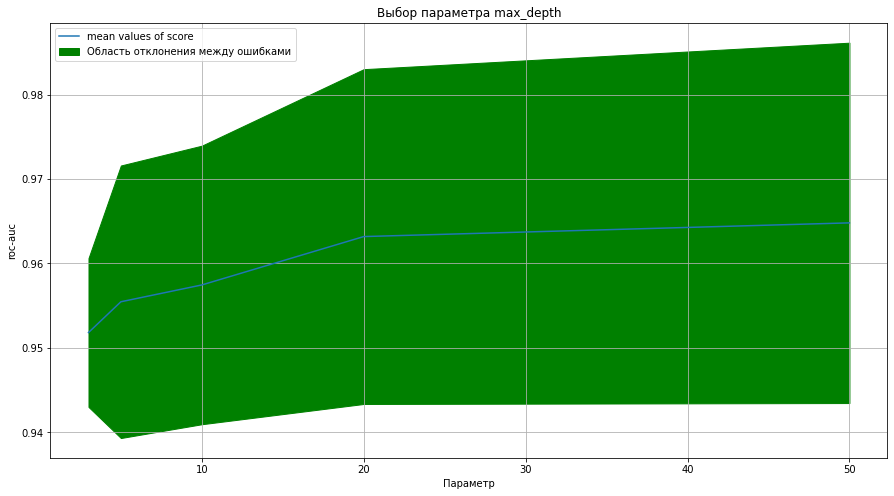
grids = [np.array([3, 5, 10, 20, 50], dtype="int")]

for model, param\_name, grid in zip(models,

                                   param\_names,

                                   grids):

    search(X\_train[:10000], y\_train[:10000], model, param\_name, grid)



Максимальной глубины 30 оказалось достаточно.

Теперь очередь параметра min\_samples\_leaf, отвечающего за минимальное число образцов в каждом из листов дерева алгоритма.

Теперь очередь параметра min\_samples\_leaf, отвечающего за минимальное число образцов в каждом из листов дерева алгоритма:

models = [RandomForestClassifier(n\_estimators=100, max\_depth=30, random\_state=1, n\_jobs=-1)]

param\_names = ["min\_samples\_leaf"]

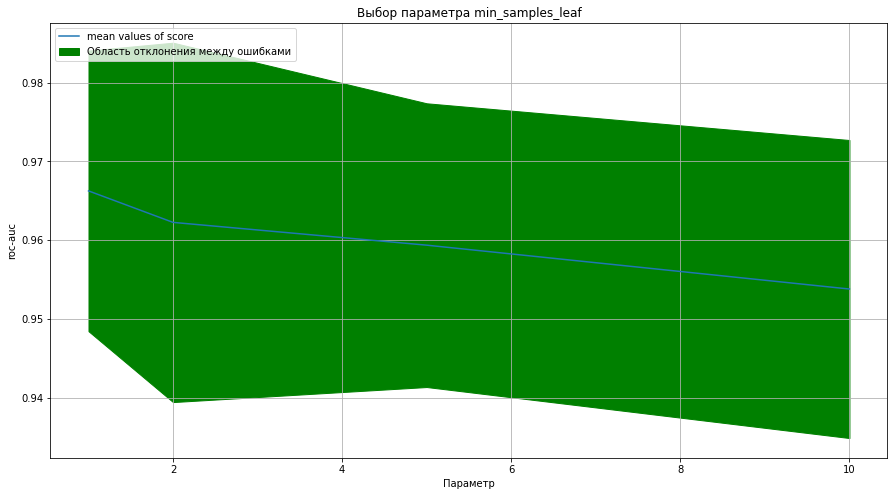
grids = [np.array([1, 2, 5, 10], dtype="int")]

for model, param\_name, grid in zip(models,

                                   param\_names,

                                   grids):

    search(X\_train[:10000], y\_train[:10000], model, param\_name, grid)



Оставил стандартное значение 1.

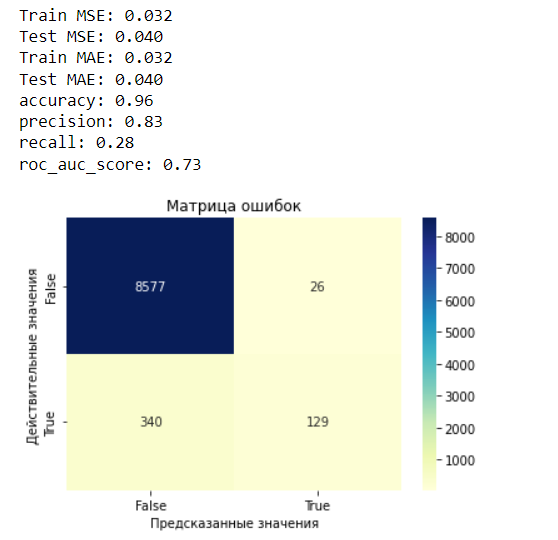
Теперь осталось обучить финальную модель и проверить её метрики:

model = RandomForestClassifier(n\_estimators=100, max\_depth=30, random\_state=1, n\_jobs=-1)

model.fit(X\_train, y\_train)

RandomForestClassifier(max\_depth=30, n\_jobs=-1, random\_state=1)

print\_metrics(model)



Сравнил метрики всех моделей:

1. Для логистической регрессии:
   * Train MSE: 0.039
   * Test MSE: 0.037
   * Train MAE: 0.039
   * Test MAE: 0.037
   * accuracy: 0.96
   * precision: 0.65
   * recall: 0.63
   * roc\_auc\_score: 0.81
2. Для случайного леса до настройки гиперпараметров:
   * Train MSE: 0.004
   * Test MSE: 0.033
   * Train MAE: 0.004
   * Test MAE: 0.033
   * accuracy: 0.97
   * precision: 0.71
   * recall: 0.61
   * roc\_auc\_score: 0.97
3. Для XGBoost:
   * Train MSE: 0.019
   * Test MSE: 0.033
   * Train MAE: 0.019
   * Test MAE: 0.033
   * accuracy: 0.97
   * precision: 0.70
   * recall: 0.63
   * roc\_auc\_score: 0.89
4. Для настроенного случайного леса:
   * Train MSE: 0.032
   * Test MSE: 0.040
   * Train MAE: 0.032
   * Test MAE: 0.040
   * accuracy: 0.96
   * precision: 0.83
   * recall: 0.28
   * roc\_auc\_score: 0.73

Получается, что настройка алгоритма случайного леса лишь незначительно увеличила значения метрики precision, в то же время остальные показатели сильно ухудшилось. Возможно, модель переобучилась. Нужно обучить модель со 100 деревьями и без указания максимальной глубины. Если метрики станут ещё хуже, значит, модель подвержена переобучению.

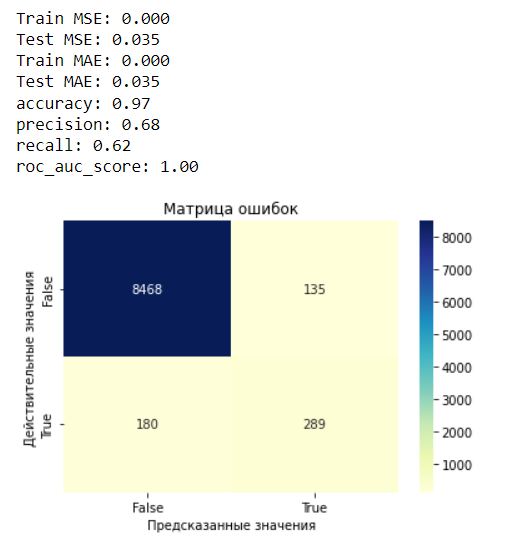
model = RandomForestClassifier(n\_estimators=100, random\_state=1, n\_jobs=-1)

model.fit(X\_train, y\_train)

RandomForestClassifier(n\_jobs=-1, random\_state=1)

y\_test\_prediction = model.predict(X\_test)

print\_metrics(model)



roc\_auc\_score(y\_test, y\_test\_prediction)

0.8002562452181727

Показатели мало отличаются от модели с количеством деревьев 10. Но теперь она действительно переобучилась, так как roc\_auc\_score на тренировочных данных стал равен единице. Решающим фактором, какую из двух моделей выбрать, станет roc\_auc\_score на тестовых данных:

model = RandomForestClassifier(n\_estimators=10, random\_state=1, n\_jobs=-1)

model.fit(X\_train, y\_train)

y\_test\_prediction = model.predict(X\_test)

roc\_auc\_score(y\_test, y\_test\_prediction)

0.7980459784073936

Всё-таки, результат улучшился, но разница слишком мала. Для более тщательной настройки моделей на таких больших данных необходимо много оперативной памяти, мощный процессор, а также такой алгоритм поиска гиперпараметров, при котором несколько параметров будут выбираться одновременно, чтобы исключить ухудшение качества метрик.

## 2.6 Выводы по разделу

В данном разделе я провёл статистический анализ данных, вывел полезные показатели, отобразил разные срезы данных на графиках, диаграммах и картах мира, поставил и проверил гипотезы. После выбрал один признак из датасета и обучил несколько моделей машинного обучения для его предсказания, выбрал лучшую модель, настроил её гиперпараметры и сделал выводы.

Таблица 2: Выводы по разделу 2

|  |  |  |
| --- | --- | --- |
| Выводы | Сформированные компетенции | |
| Код | Содержание компетенции |
| Прочитал большой объём технической литературы, включая документацию библиотек и инструментов, а также статьи. Системно провёл обширный анализ и обработку данных | УК-1 | Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач. |
| Разбил весь проект на несколько задач, подобрал наиболее подходящие инструменты для анализа, ограничил количество данных под имеющиеся ресурсы | УК-2 | Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений |
| Применил большое количество статистических методов для анализа данных, произвёл расчёт важных метрик, отрисовал на карте мира группы данных по координатам | ОПК-1 | Способен применять естественнонаучные и общеинженерные знания, методы математического анализа и моделирования, теоретического и экспериментального исследования в профессиональной деятельности; |
| Продемонстрировал умение работать с современными передовыми инструментами обработки данных | ОПК-2 | Способен понимать принципы работы современных информационных технологий и программных средств, в том числе отечественного производства, и использовать их при решении задач профессиональной деятельности; |
| Организовал и разработал сложный процесс обработки данных | ОПК-6 | Способен анализировать и разрабатывать организационно-технические и экономические процессы с применением методов системного анализа и математического моделирования; |
| Составил необходимые для анализа алгоритмы | ОПК-7 | Способен разрабатывать алгоритмы и программы, пригодные для практического применения; |
| Организовал и выполнил данный проект по анализу данных, благодаря чему продемонстрировал умение участвовать в жизненном цикле информационных систем | ОПК-8 | Способен принимать участие в управлении проектами создания информационных систем на стадиях жизненного цикла; |

**ЗАКЛЮЧЕНИЕ**

С использованием полученных в ходе обучения знаний произвёл процесс анализа и обработки данных, значительно укрепил имеющиеся навыки программирования и анализа. В работе использовал современные инструменты из области обработки данных.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. <https://github.com/DataTalksClub/mlops-zoomcamp>
2. <https://stepik.org/course/80782/promo>
3. <https://pandas.pydata.org/pandas-docs/stable/index.html>
4. <https://www.geeksforgeeks.org>
5. <https://python-lab.ru>
6. <https://datastart.ru>
7. <https://matplotlib.org>
8. <https://www.codecamp.ru>
9. <https://habr.com>
10. <https://www.python-graph-gallery.com>
11. <https://github.com/datasets/continent-codes>