МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Систем обработки информации и управления»

ОТЧЕТ

**Лабораторная работа №3**

по дисциплине«Методы машинного обучения»

Тема: «Обработка признаков (часть 2)»  
Вариант 9

ИСПОЛНИТЕЛЬ: \_\_\_Очеретная С.В.\_\_

ФИО

группа ИУ5-25М \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

"\_\_"\_\_\_\_\_\_\_\_\_2024 г.

ПРЕПОДАВАТЕЛЬ: \_\_\_\_\_Гапанюк Ю.Е.\_\_\_\_

ФИО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

"\_\_"\_\_\_\_\_\_\_\_\_2024 г.

Москва - 2024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Цель лабораторной работы:** изучение продвинутых способов предварительной обработки данных для дальнейшего формирования моделей.

# Задание

1. Выбрать один или несколько наборов данных (датасетов) для решения следующих задач. Каждая задача может быть решена на отдельном датасете, или несколько задач могут быть решены на одном датасете. Просьба не использовать датасет, на котором данная задача решалась в лекции.
2. Для выбранного датасета (датасетов) на основе материалов лекций решить следующие задачи:

* масштабирование признаков (не менее чем тремя способами);
* обработку выбросов для числовых признаков (по одному способу для удаления выбросов и для замены выбросов);
* обработку по крайней мере одного нестандартного признака (который не является числовым или категориальным);
* отбор признаков:
  + один метод из группы методов фильтрации (filter methods);
  + один метод из группы методов обертывания (wrapper methods);
  + один метод из группы методов вложений (embedded methods).

# Ход работы

## Импорт данных и библиотек[¶](#Xc4345e497f098f117ccf867af39bbf741d27a1f)

Импорт библиотек

import numpy as np  
import pandas as pd  
import matplotlib as mpl  
import matplotlib.pyplot as plt  
%matplotlib inline  
import seaborn as sns  
  
from sklearn.impute import SimpleImputer  
from sklearn.impute import MissingIndicator  
  
from sklearn.preprocessing import LabelEncoder  
  
from sklearn.preprocessing import StandardScaler  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.preprocessing import RobustScaler  
from sklearn.preprocessing import MaxAbsScaler  
  
import re  
from mlxtend.feature\_selection import ExhaustiveFeatureSelector as EFS  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.feature\_selection import SelectFromModel

Подключение к диску

from google.colab import drive  
drive.mount('/content/drive/')

Импорт датасетов

data = pd.read\_csv("/content/drive/My Drive/Учеба/магистратура/2 сем/ММО/Air\_Traffic\_Passenger\_Statistics.csv")

data3\_train = pd.read\_csv("/content/drive/My Drive/Учеба/магистратура/2 сем/ММО/airline\_passenger\_satisfaction\_80.csv")  
data3\_test = pd.read\_csv("/content/drive/My Drive/Учеба/магистратура/2 сем/ММО/airline\_passenger\_satisfaction\_20.csv")  
data3 = pd.concat([data3\_train, data3\_test])

data4 = pd.read\_csv("/content/drive/My Drive/Учеба/магистратура/2 сем/ММО/EaseMyTrip\_Flight \_Fare\_Travel\_Listings.csv")

## Описание датасета Airline Passenger Satisfaction[¶](#Xd90e3d236d90f9bb1cf6107aee802eb278da53b)

В данном наборе, как и в наборе из предыдущей лабы (rynair\_reviews) представлены оценки пассажиров о качестве разных характеристик рейса. Но в этом наборе, в отличие от предыдущего, есть и другие численные значения другой размерности, такие как дистанция полета, возраст пассажира, время задержки полета. Такой набор уже имеет смысл масштабировать, что и является одним из заданий данной лабораторной.

Целевой признак:

data3\_Y = data3["satisfaction"]  
data3["satisfaction"].unique()

array(['neutral or dissatisfied', 'satisfied'], dtype=object)

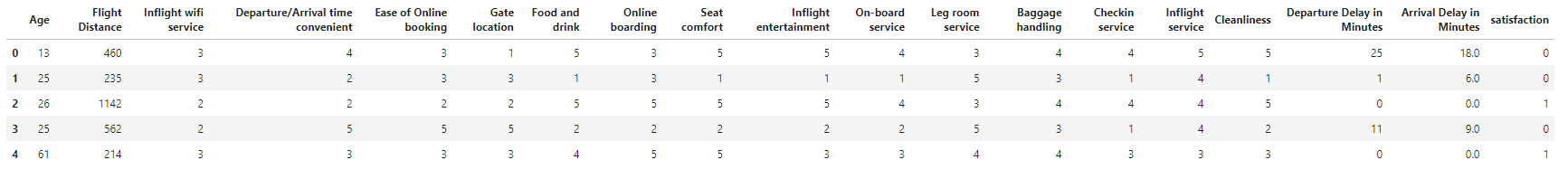
Кодирование целевого признака:

def encode\_satisfaction\_col(data):  
 le\_satisfaction = LabelEncoder()  
 rec\_le\_satisfaction = le\_satisfaction.fit\_transform(data['satisfaction'])  
 data['satisfaction'] = rec\_le\_satisfaction  
 le\_satisfaction.inverse\_transform([0, 1])  
  
encode\_satisfaction\_col(data3)  
encode\_satisfaction\_col(data3\_train)  
encode\_satisfaction\_col(data3\_test)

Категориальные признаки использовать не будем, поэтому сразу их уберем. Также уберем столбцы unnamed и id.

cols\_to\_drop = [c for c in data3.columns if (data3[c].dtype == 'O')] + ['Unnamed: 0', 'id']  
data3.drop(cols\_to\_drop, axis=1, inplace=True)  
data3\_train.drop(cols\_to\_drop, axis=1, inplace=True)  
data3\_test.drop(cols\_to\_drop, axis=1, inplace=True)

data3.head()



Набор был взят из kaggle, где он уже был разделен на 2 части: обучающую (80%) и тестовую (20%) выборки.

data3.shape, data3\_train.shape, data3\_test.shape

((129880, 19), (103904, 19), (25976, 19))

Обработка пропусков:

def impute\_column(dataset, column, strategy\_param, fill\_value\_param=None):  
 """  
 Заполнение пропусков в одном признаке  
 """  
 temp\_data = dataset[[column]].values  
 size = temp\_data.shape[0]  
  
 indicator = MissingIndicator()  
 mask\_missing\_values\_only = indicator.fit\_transform(temp\_data)  
  
 imputer = SimpleImputer(strategy=strategy\_param,  
 fill\_value=fill\_value\_param)  
 all\_data = imputer.fit\_transform(temp\_data)  
  
 missed\_data = temp\_data[mask\_missing\_values\_only]  
 filled\_data = all\_data[mask\_missing\_values\_only]  
  
 return all\_data.reshape((size,)), filled\_data, missed\_data

data3\_train["Arrival Delay in Minutes"] = impute\_column(data3\_train, "Arrival Delay in Minutes", 'median')[0]  
data3\_test["Arrival Delay in Minutes"] = impute\_column(data3\_test, "Arrival Delay in Minutes", 'median')[0]  
data3["Arrival Delay in Minutes"] = impute\_column(data3, "Arrival Delay in Minutes", 'median')[0]

Проверка, что пропусков нет:

data3\_train.isnull().sum().sum(), data3\_test.isnull().sum().sum(), data3.isnull().sum().sum()

(0, 0, 0)

Набор без целевого признака:

data3\_without\_satisfaction = data3.drop("satisfaction", axis=1)  
data3\_test\_without\_satisfaction = data3\_test.drop("satisfaction", axis=1)  
data3\_train\_without\_satisfaction = data3\_train.drop("satisfaction", axis=1)

Функция для восстановления датафрейма

def arr\_to\_df(arr\_scaled):  
 res = pd.DataFrame(arr\_scaled, columns=data3\_without\_satisfaction.columns)  
 return res

## Масштабирование признаков[¶](#X5f7f48cb7d410184e5cfdc97541371555c2ab75)

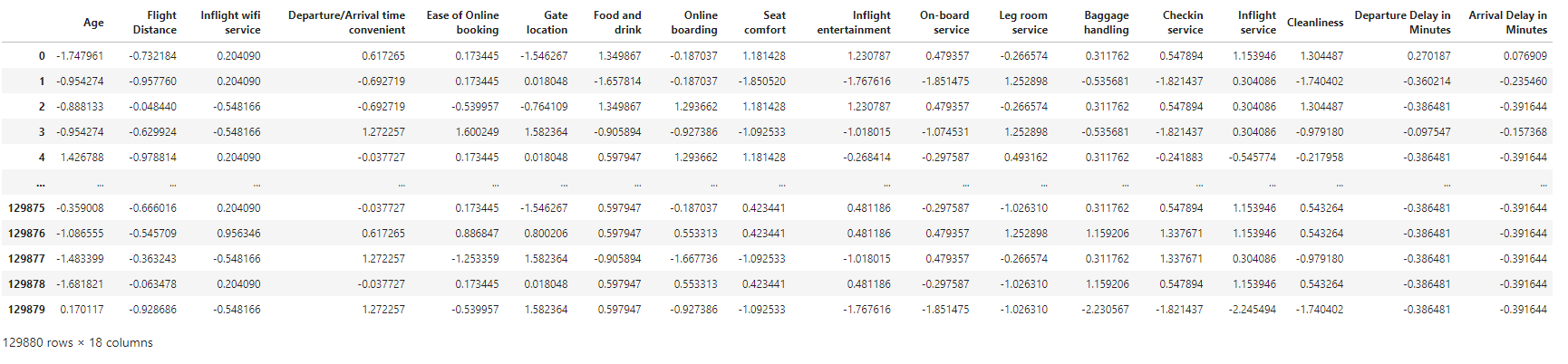
Будем использовать следующие методы:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Метод** | **Ср. значение** | **Ср. кв. отклонение** | **Форма исх. распределения** | **Макс. и мин.** | **Выбросы** |
| **Z-оценка** | привод. к 0 | привод. к 1 | сохр. | могут варьироваться | сохр. |
| **Mean Normalisation** | привод. к 0 | может варьироваться | может изменяться | в диапазоне [-1;1]. | сохр. |
| **MinMax** | может варьироваться | может варьироваться | может изменяться | в диапазоне [0;1]. | сохр. |
| **По медиане** | привод. к 0 | может варьироваться | может изменяться | могут варьироваться | устран. |
| **По макс. значению** | не центрируется | не масштабируется | может изменяться | в диапазоне [-1;1]. | - |

### Z-оценка[¶](#Z-%D0%BE%D1%86%D0%B5%D0%BD%D0%BA%D0%B0)

![image.png](data:image/png;base64;base64,)

cs11 = StandardScaler()  
data\_cs11\_scaled\_temp = cs11.fit\_transform(data3\_without\_satisfaction)  
data\_cs11\_scaled = arr\_to\_df(data\_cs11\_scaled\_temp)  
data\_cs11\_scaled



Построение плотности распределения

def draw\_kde(col\_list, df1, df2, label1, label2):  
 fig, (ax1, ax2) = plt.subplots(  
 ncols=2, figsize=(12, 5))  
 # первый график  
 ax1.set\_title(label1)  
 sns.kdeplot(data=df1[col\_list], ax=ax1)  
 # второй график  
 ax2.set\_title(label2)  
 sns.kdeplot(data=df2[col\_list], ax=ax2)  
 plt.show()

draw\_kde(['Age', 'Flight Distance', 'Departure Delay in Minutes', 'On-board service'], data3, data\_cs11\_scaled, 'до масштабирования', 'после масштабирования')

![](data:image/png;base64;base64,)

Как видим, на 1ом рисунке из-за отличий размерностей метрик их невозможно сравнить. На 2ом рисунке, после масштабирования, уже можем сравнивать признаки разных размерностей.

Теперь произведем масштабирование отдельно для тестовой и обучающей выборок. Обучать StandardScaler будем на обучающей выборке.

cs12 = StandardScaler()  
cs12.fit(data3\_train\_without\_satisfaction)  
data\_cs12\_scaled\_train\_temp = cs12.transform(data3\_train\_without\_satisfaction)  
data\_cs12\_scaled\_test\_temp = cs12.transform(data3\_test\_without\_satisfaction)  
  
data\_cs12\_scaled\_train = arr\_to\_df(data\_cs12\_scaled\_train\_temp)  
data\_cs12\_scaled\_test = arr\_to\_df(data\_cs12\_scaled\_test\_temp)

Заметим, что распределения для тестовой и обучающей выборок немного отличаются:

draw\_kde(['Age', 'Flight Distance', 'Departure Delay in Minutes', 'On-board service'], data\_cs12\_scaled\_train, data\_cs12\_scaled\_test, 'до масштабирования', 'после масштабирования')

![](data:image/png;base64;base64,)

### Масштабирование "Mean Normalisation"[¶](#Xc4c8f32a41f5012428752922204e304820428c1)

![image.png](data:image/png;base64;base64,)

class MeanNormalisation:  
 def fit(self, param\_df):  
 self.means = data3\_train\_without\_satisfaction.mean(axis=0)  
 maxs = data3\_train\_without\_satisfaction.max(axis=0)  
 mins = data3\_train\_without\_satisfaction.min(axis=0)  
 self.ranges = maxs - mins  
  
 def transform(self, param\_df):  
 param\_df\_scaled = (param\_df - self.means) / self.ranges  
 return param\_df\_scaled  
  
 def fit\_transform(self, param\_df):  
 self.fit(param\_df)  
 return self.transform(param\_df)

sc21 = MeanNormalisation()  
data\_cs21\_scaled = sc21.fit\_transform(data3\_without\_satisfaction)  
#data\_cs21\_scaled.describe()  
  
cs22 = MeanNormalisation()  
cs22.fit(data3\_train\_without\_satisfaction)  
data\_cs22\_scaled\_train = cs22.transform(data3\_train\_without\_satisfaction)  
data\_cs22\_scaled\_test = cs22.transform(data3\_test\_without\_satisfaction)

draw\_kde(['Age', 'Flight Distance', 'Departure Delay in Minutes', 'On-board service'], data3, data\_cs21\_scaled, 'до масштабирования', 'после масштабирования')

![](data:image/png;base64;base64,)

draw\_kde(['Age', 'Flight Distance', 'Departure Delay in Minutes', 'On-board service'], data\_cs22\_scaled\_train, data\_cs22\_scaled\_test, 'обучающая', 'тестовая')

![](data:image/png;base64;base64,)

Как видим, в этом методе форма распределения поменялась и значения лежат в интервале [-1, 1]. Это метод более удобен, чем предыдущий

### MinMax масштабирование[¶](#X2f91d558aca4d395e9679a3956f0c562939ffc3)

![image.png](data:image/png;base64;base64,)

cs31 = MinMaxScaler()  
data\_cs31\_scaled\_temp = cs31.fit\_transform(data3\_without\_satisfaction)  
data\_cs31\_scaled = arr\_to\_df(data\_cs31\_scaled\_temp)  
  
cs32 = MinMaxScaler()  
cs32.fit(data3\_train\_without\_satisfaction)  
data\_cs32\_scaled\_train\_temp = cs32.transform(data3\_train\_without\_satisfaction)  
data\_cs32\_scaled\_test\_temp = cs32.transform(data3\_test\_without\_satisfaction)  
  
data\_cs32\_scaled\_train = arr\_to\_df(data\_cs32\_scaled\_train\_temp)  
data\_cs32\_scaled\_test = arr\_to\_df(data\_cs32\_scaled\_test\_temp)

Графики:

draw\_kde(['Age', 'Flight Distance', 'Departure Delay in Minutes', 'On-board service'], data3, data\_cs31\_scaled, 'до масштабирования', 'после масштабирования')

![](data:image/png;base64;base64,)

draw\_kde(['Age', 'Flight Distance', 'Departure Delay in Minutes', 'On-board service'], data\_cs32\_scaled\_train, data\_cs32\_scaled\_test, 'обучающая', 'тестовая')

![](data:image/png;base64;base64,)

В этом методе также поменялась форма исходного распределения, значения получились в диапазоне [0,1]

### Масштабирование по медиане[¶](#Xe5fed055dbc7fb2c96230f12563bff871602955)

![image.png](data:image/png;base64;base64,)

cs41 = RobustScaler()  
data\_cs41\_scaled\_temp = cs41.fit\_transform(data3\_without\_satisfaction)  
data\_cs41\_scaled = arr\_to\_df(data\_cs41\_scaled\_temp)  
  
cs42 = RobustScaler()  
cs42.fit(data3\_train\_without\_satisfaction)  
data\_cs42\_scaled\_train\_temp = cs42.transform(data3\_train\_without\_satisfaction)  
data\_cs42\_scaled\_test\_temp = cs42.transform(data3\_test\_without\_satisfaction)  
  
data\_cs42\_scaled\_train = arr\_to\_df(data\_cs42\_scaled\_train\_temp)  
data\_cs42\_scaled\_test = arr\_to\_df(data\_cs42\_scaled\_test\_temp)

Графики:

draw\_kde(['Age', 'Flight Distance', 'Departure Delay in Minutes', 'On-board service'], data3, data\_cs41\_scaled, 'до масштабирования', 'после масштабирования')

![](data:image/png;base64;base64,)

draw\_kde(['Age', 'Flight Distance', 'Departure Delay in Minutes', 'On-board service'], data\_cs42\_scaled\_train, data\_cs42\_scaled\_test, 'обучающая', 'тестовая')

![](data:image/png;base64;base64,)

В этом методе также поменялась форма исходного распределения, значения остались в том же диапазоне. Отличие данного метода еще в том, что здесь устраняются выбросы

### Масштабирование по максимальному значению[¶](#X89cd6df67a05477b3d9d192cc5764db30e1fbd2)

![image.png](data:image/png;base64;base64,)

cs51 = MaxAbsScaler()  
data\_cs51\_scaled\_temp = cs51.fit\_transform(data3\_without\_satisfaction)  
data\_cs51\_scaled = arr\_to\_df(data\_cs51\_scaled\_temp)  
  
cs52\_mas = MaxAbsScaler()  
cs52\_mean = StandardScaler(with\_mean=True, with\_std=False)  
cs52\_mas.fit(data3\_train\_without\_satisfaction)  
cs52\_mean.fit(data3\_train\_without\_satisfaction)  
data\_cs52\_scaled\_train\_temp = cs52\_mas.transform(cs52\_mean.transform(data3\_train\_without\_satisfaction))  
data\_cs52\_scaled\_test\_temp = cs52\_mas.transform(cs52\_mean.transform(data3\_test\_without\_satisfaction))  
  
data\_cs52\_scaled\_train = arr\_to\_df(data\_cs52\_scaled\_train\_temp)  
data\_cs52\_scaled\_test = arr\_to\_df(data\_cs52\_scaled\_test\_temp)

Графики:

draw\_kde(['Age', 'Flight Distance', 'Departure Delay in Minutes', 'On-board service'], data3, data\_cs51\_scaled, 'до масштабирования', 'после масштабирования')

![](data:image/png;base64;base64,)

draw\_kde(['Age', 'Flight Distance', 'Departure Delay in Minutes', 'On-board service'], data\_cs52\_scaled\_train, data\_cs52\_scaled\_test, 'обучающая', 'тестовая')

![](data:image/png;base64;base64,)

В этом методе также поменялась форма исходного распределения, значения получились в диапазоне [-1,1]. При этом, здесь не центрируется ср. значение

## Обработка выбросов для числовых признаков[¶](#X5a28cef7ab813288f7394aa3f77d04472a2c1f9)

### Анализ набора[¶](#X946692e59b638cfc8e60f5d9fdf91cb8ca18305)

Перед обработкой выбросов сначала проверим распределение данных набора. На основе графиков, выводимых для масштабирования, можно сказать, что в обработке выбросов нуждаются столбцы 'Departure Delay in Minutes', Arrival Delay in Minutes - с ними и будем работать.

x\_col\_list = ['Departure Delay in Minutes', 'Arrival Delay in Minutes']

def diagnostic\_plots(df, variable, title):  
 fig, ax = plt.subplots(figsize=(10,7))  
 # гистограмма  
 plt.subplot(2, 2, 1)  
 df[variable].hist(bins=30)  
 ## Q-Q plot  
 plt.subplot(2, 2, 2)  
 stats.probplot(df[variable], dist="norm", plot=plt)  
 # ящик с усами  
 plt.subplot(2, 2, 3)  
 sns.violinplot(x=df[variable])  
 # ящик с усами  
 plt.subplot(2, 2, 4)  
 sns.boxplot(x=df[variable])  
 fig.suptitle(title)  
 plt.show()

diagnostic\_plots(data3, 'Departure Delay in Minutes', 'Departure Delay in Minutes')

![](data:image/png;base64;base64,)

diagnostic\_plots(data3, 'Arrival Delay in Minutes', 'Arrival Delay in Minutes')

![](data:image/png;base64;base64,)

По графикам видим, что распределение столбца сильно смещается влево и есть выбросы при очень больших задержках полета. Скорее всего, эти задержки соответствуют каким-то происшествиям, которые привели к сильной задержке рейса (или прибытия в пункт назначения). Но такие задержки не должны влиять на общую оценку рейса.

В нашем случае распределение ассиметричное, поэтому для устранения выбросов будем использовать метод межквартильного размаха. Значение К возьмем как 5

![image.png](data:image/png;base64;base64,)

Функция вычисления верхней и нижней границы выбросов:

def get\_outlier\_boundaries(df, col):  
 K = 5  
 IQR = df[col].quantile(0.75) - df[col].quantile(0.25)  
 lower\_boundary = df[col].quantile(0.25) - (K \* IQR)  
 upper\_boundary = df[col].quantile(0.75) + (K \* IQR)  
  
 return lower\_boundary, upper\_boundary

### Удаление выбросов[¶](#X560f7f42f2bace97a54021472aee1da326fd4d2)

for col in x\_col\_list:  
 lower\_boundary, upper\_boundary = get\_outlier\_boundaries(data3, col)  
  
 # Флаги для удаления выбросов  
 outliers\_temp = np.where(data3[col] > upper\_boundary, True,  
 np.where(data3[col] < lower\_boundary, True, False))  
  
 # Удаление данных на основе флага  
 data3\_trimmed = data3.loc[~(outliers\_temp), ]  
 title = 'Поле-{}, строк-{}'.format(col, data3\_trimmed.shape[0])  
 diagnostic\_plots(data3\_trimmed, col, title)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

В целом, значения немного выровнялись. При подборе больших значений К, распределение постепенно возвращалась к исходному, при малых - на графике Q-Q plot наблюдались большие расхождения. Поэтому оставили К как 5

### Замена выбросов[¶](#X0e75f25c41031c17e9972f635d52446ac65b93a)

data3\_changed = data3.copy(deep=True)  
  
for col in x\_col\_list:  
 lower\_boundary, upper\_boundary = get\_outlier\_boundaries(data3, col)  
  
 # Изменение данных  
 data3\_changed[col] = np.where(data3\_changed[col] > upper\_boundary, upper\_boundary,  
 np.where(data3\_changed[col] < lower\_boundary, lower\_boundary, data3\_changed[col]))  
 title = 'Поле-{}'.format(col)  
 diagnostic\_plots(data3\_changed, col, title)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

## Обработка нестандартных признаков[¶](#Xcb4dd02f8f38733bc6c53d671e120895699f6d9)

### Описание датасета EaseMyTrip Flight Fare Travel Listings[¶](#Xec671ef2bb33d0df2ad52b1fecffbaa1bf07f03)

Обрабатывать нестандартные признаки будем для набора c ценами авиабилетов. Здесь присутствуют различные нестандартные форматы дат и времени, которые необходимо обработать.

data4.head()



Размер набора:

data4.shape

(30000, 16)

Обработанный далее набор будем записывать в эту переменнную:

data4\_normal\_formats = data4.copy(deep=True)

### Обработка форматов столбцов[¶](#X7b7eb602257ec9f78ff36007100a66d46386bc8)

Сконвертируем дату и время для столбцов 'Departure Date', 'Arrival Date' в формат datetime, при этом объединив время и дату:

data4\_normal\_formats['departure\_datetime'] = data4.apply(lambda x: pd.to\_datetime(x['Departure Date'] + x['Departure Time'], format='%d%b%Y%H:%M'), axis=1)  
data4\_normal\_formats['arrival\_datetime'] = data4.apply(lambda x: pd.to\_datetime(x['Arrival Date'] + x['Arrival Time'], format='%d%b%Y%H:%M'), axis=1)

data4\_normal\_formats=data4\_normal\_formats.drop(columns=['Departure Date', 'Departure Time', 'Arrival Date', 'Arrival Time'],axis=1)

Таким образом мы получаем 2 столбца в формате datetime, с которыми можно дальше удобно работать. При этом дата и время у нас теперь в одном столбце

Например, можем вычислить общее время полета:

datetime\_diff = data4\_normal\_formats['arrival\_datetime'] - data4\_normal\_formats['departure\_datetime']

def calc\_total\_minutes(x):  
 return x.total\_seconds() / 60  
  
total\_minutes\_col = datetime\_diff.apply(calc\_total\_minutes)  
total\_minutes\_col

0 530.0  
1 730.0  
2 1675.0  
3 1030.0  
4 370.0  
 ...   
29995 525.0  
29996 785.0  
29997 1510.0  
29998 1940.0  
29999 135.0  
Length: 30000, dtype: float64

Вычисленный столбец аналогичен столбцу Total Time из нашего набора. Выходит, мы провели вычисления верно.

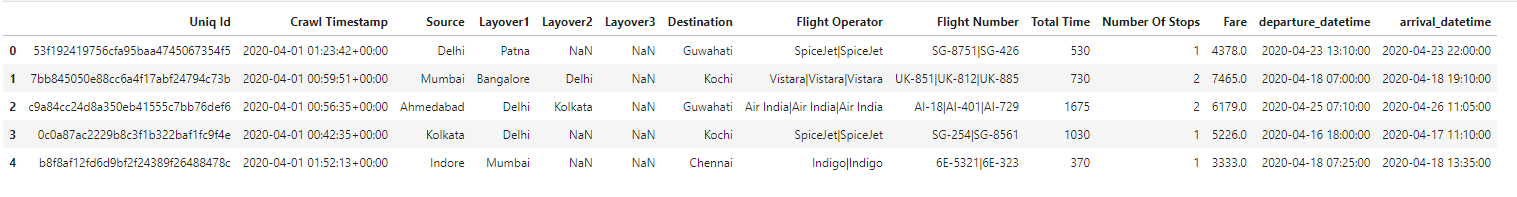
Также сконвентируем столбец 'Crawl Timestamp' в формат datetime:

data4\_normal\_formats['Crawl Timestamp'] = data4.apply(lambda x: pd.to\_datetime(x['Crawl Timestamp'], format='%Y-%m-%d %H:%M:%S %z'), axis=1)

Столбец 'Total Time' переведем в минуты:

def total\_time\_to\_minutes(val):  
 hours, minutes = re.findall(r"\d+", val)  
 return (int(hours) \* 60 + int(minutes))  
  
data4\_normal\_formats['Total Time'] = data4.apply(lambda x: total\_time\_to\_minutes(x['Total Time']), axis=1)

data4\_normal\_formats.head()



## Отбор признаков[¶](#Xdf459feeaee28b9b4ba11100072aee2b0e56bc1)

### Метод фильтрации (filter method)[¶](#Xacbd66ad396680d80886eb5e9c9a564f4a63880)

Отбор будем проводить для датасета Air\_Traffic\_Passenger\_Statistics из 1 лабораторной

#### Обработка датасета Air\_Traffic\_Passenger\_Statistics[¶](#Xbe6944c49b9ea8ae9f808dd9ac713931e75e8e5)

Нулевые значения всего в 2х колонках, но для анализа мы их использовать не будем. Вырежем колонки, которые не будем использовать:

data=data.drop(columns=['index','Operating Airline IATA Code','Published Airline IATA Code'],axis=1)

data.isnull().sum().sum()

0

data\_enc = data.copy(deep=True)  
for col in ([c for c in data.columns if data[c].dtype == 'O'] + ['Activity Period']):  
 le = LabelEncoder()  
 data\_enc[col] = le.fit\_transform(data\_enc[col])

#### Выполнение фильтрации[¶](#Xbb48909ccf0883e4a4f034efd94285be5c993dd)

Будем использовать метод, основанный на корреляции. То есть найдем группы коррелирующих признаков, на основе чего сможем понять, какие признаки можем удалить

data\_enc\_copy = data\_enc.copy(deep=True)

Формирование DataFrame с сильными корреляциями

def make\_corr\_df(df):  
 cr = data.corr(numeric\_only=True)  
 cr = cr.abs().unstack()  
 cr = cr.sort\_values(ascending=False)  
 cr = cr[cr >= 0.8]  
 cr = cr[cr < 1]  
 cr = pd.DataFrame(cr).reset\_index()  
 cr.columns = ['f1', 'f2', 'corr']  
 return cr

Обнаружение групп коррелирующих признаков

def corr\_groups(cr):  
 grouped\_feature\_list = []  
 correlated\_groups = []  
  
 for feature in cr['f1'].unique():  
 if feature not in grouped\_feature\_list:  
 # находим коррелирующие признаки  
 correlated\_block = cr[cr['f1'] == feature]  
 cur\_dups = list(correlated\_block['f2'].unique()) + [feature]  
 grouped\_feature\_list = grouped\_feature\_list + cur\_dups  
 correlated\_groups.append(cur\_dups)  
 return correlated\_groups

Группы коррелирующих признаков

corr\_groups(make\_corr\_df(data))

[['Adjusted Passenger Count', 'Passenger Count'], ['Year', 'Activity Period']]

Таким образом, можно сделать вывод о том, что столбцы 'Passenger Count' и 'Activity Period' стоит удалить.

### Метод обертывания (wrapper method)[¶](#X207c49bb51c0fdfa8efd7a31351921c2927ea56)

Сделаем полный перебор набора Airline Passenger Satisfaction с разным числом признаков и определим, какие вариации признаков лучше всего

Будем перебирать только часть набора, т.к. весь набор перебирается очень долго (много часов)

data3\_part = data3\_without\_satisfaction.copy(deep=True)[data3\_without\_satisfaction.columns[0:5]]

knn = KNeighborsClassifier(n\_neighbors=3)

efs1 = EFS(knn,  
 min\_features=3,  
 max\_features=4,  
 scoring='accuracy',  
 print\_progress=True,  
 cv=5)  
  
efs1 = efs1.fit(data3\_part, data3\_Y)  
  
print('Best accuracy score: %.2f' % efs1.best\_score\_)  
print('Best subset (indices):', efs1.best\_idx\_)  
print('Best subset (corresponding names):', efs1.best\_feature\_names\_)

Features: 15/15

Best accuracy score: 0.81  
Best subset (indices): (0, 2, 3)  
Best subset (corresponding names): ('Age', 'Inflight wifi service', 'Departure/Arrival time convenient')

Таким образом, получили лучший набор признаков при их количестве 3-4 из 5

### Метод вложений (embedded method)[¶](#Xc6402c572898bc4785e05f6a65a318449bf5622)

Будем использовать метод логистической регрессии для решения задачи классификации набора Airline Passenger Satisfaction

# Используем L1-регуляризацию  
e\_lr1 = LogisticRegression(C=1000, solver='liblinear', penalty='l1', max\_iter=500, random\_state=1)  
e\_lr1.fit(data3\_without\_satisfaction, data3\_Y)  
  
# Коэффициенты регрессии  
e\_lr1.coef\_

array([[ 5.05395242e-03, 5.46604394e-04, 3.85759506e-01,  
 -2.98822285e-01, -1.64012337e-01, 1.08797580e-01,  
 -5.68178111e-02, 7.59002231e-01, 1.12547545e-01,  
 2.59946067e-01, 2.60491007e-01, 3.23632493e-01,  
 7.19304252e-02, 2.28922985e-01, 2.68821055e-02,  
 8.42771050e-02, 2.43052324e-03, -6.25292180e-03]])

Посмотрим, какие признаки были отобраны:

sel\_e\_lr1 = SelectFromModel(e\_lr1)  
sel\_e\_lr1.fit(data3\_without\_satisfaction, data3\_Y)  
sel\_e\_lr1.get\_support()

array([ True, True, True, True, True, True, True, True, True,  
 True, True, True, True, True, True, True, True, True])

Вышло, что все признаки важны для набора