

Федеральное государственное автономное образовательное учреждение высшего
образования

«Пермский государственный национальный исследовательский университет»

Институт Компьютерных Наук и Технологий

Лабораторная работа № 7

по дисциплине

«Введение в анализ данных»

Отчет

Студент Панькова Светлана

Группа ИТ-13-2023

Пермь 2025

Задание №1

1. Загрузите данные из файла "база.csv".

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score, precision_score,
recall_score
from sklearn.svm import LinearSVC

# 1. Загрузите данные из файла "база.csv".
data = pd.read_csv("База.csv",
sep=";",
encoding='cp1251')
print('\nЗагруженные данные из файла')
print(data.head())
```

Загруженные данные из файла

| | УИД_Брони | ... | Статус лида (из CRM) |
|---|--------------------------------------|-----|----------------------|
| 0 | d192173f-fc14-11eb-9512-000c29ad50ac | ... | S |
| 1 | 43574a1f-fe8b-11eb-9512-000c29ad50ac | ... | F |
| 2 | 0e7b7a81-fe97-11eb-9512-000c29ad50ac | ... | S |
| 3 | c7041428-f90b-11eb-9512-000c29ad50ac | ... | S |
| 4 | 60090518-fe8b-11eb-9512-000c29ad50ac | ... | P |

2. Предварительная фильтрация.

- Поскольку нас интересуют только сделки с жилой недвижимостью, отфильтруйте данные, оставив только те, для которых «ВидПомещения» = «жилые помещения». В дальнейшем этот столбец использоваться не будет, его можно удалить (или удалите его из датасета вообще, или просто нигде далее не рассматривайте).
- Также для нас бесполезны данные, по которым статус не определен. Отфильтруйте данные по признаку «СледующийСтатус». В оставшихся строчках замените значение «Продана» на 1, «Свободна» – на 0.

- с. Не забывайте, что столбец «УИД_Брони» для нас также не представляет интереса – удалите его из датасета вообще, или просто нигде далее не рассматривайте.

```
# 2. Предварительная фильтрация

# а. Отфильтруйте данные, оставив только те, для которых
# «ВидПомещения» = «жилые помещения».
# В дальнейшем этот столбец использоваться не будет, его
# можно удалить.
data = data[data['ВидПомещения'].str.strip() == 'жилые
помещения'].copy()
# inplace=True параметр указывает, что изменения должны
# быть внесены в исходный DataFrame.
data.drop(columns=['ВидПомещения'], inplace=True)

# б. Отфильтруйте данные по признаку «СледующийСтатус».
data = data[data['СледующийСтатус'].notnull() &
(data['СледующийСтатус'] != '') &
(data['СледующийСтатус'] != 'В резерве')].copy()
# Заменяем значения в столбце «СледующийСтатус»
data['СледующийСтатус'] =
data['СледующийСтатус'].map({'Продана': 1, 'Свободна':
0})

# с. «УИД_Брони» для нас также не представляет интереса –
# удалите его из датасета вообще
data.drop(columns=['УИД_Брони', 'ВремяБрони',
'ДатаБрони', 'ВариантОплатыДоп'], inplace=True)
print('\nОтфильтрованные данные')
print(data.head())

# Сохраняем обработанные данные в новый CSV файл
data.to_csv('Обработанная_база_2.csv', index=False,
sep=';', encoding='cp1251')
```

Отфильтрованные данные

| | ИсточникБрони | ВременнаяБронь | ... | Привилегия | Статус лода (из CRM) |
|---|---------------|----------------|-----|------------|----------------------|
| 0 | ручная | Да | ... | Нет | S |
| 1 | ручная | Да | ... | Нет | F |
| 2 | ручная | Да | ... | Нет | S |
| 3 | ручная | Да | ... | Нет | S |
| 4 | ручная | Да | ... | Нет | P |

3. Проверьте тип данных и преобразуйте все данные к числовому типу.

- Для тех полей, которые по смыслу являются числовыми (например, «ПродаваемаяПлощадь») – просто проверьте правильность типа.
- Для бинарных признаков (например, «ИсточникБрони») выполните кодирование (один вариант закодируйте 0, другой 1).
- Для категориальных не бинарных признаков (например, «Город») выполните one-hot кодирование.
- Обратите внимание на поле «Тип». По смыслу оно числовое (количество комнат), но напрямую конвертировать его в числовой тип мешает буковка «к» в конце. Напишите вручную преобразование, которое удаляет букву «к» в конце и конвертирует то, что осталось, в число. Если это невозможно (среди данных вам встретится еще вариант, когда в этом поле записано просто «с») – просто пока оставьте поле пустым (NaN).

```
# 3. Проверьте тип данных и преобразуйте все данные к
числовому типу.

print('\nПроверка типов данных\n')
# а. Проверка числовых типов
print(data.dtypes)

# Преобразуем указанные столбцы в числовые типы, заменяя
некорректные значения на NaN
num_cols = ['ПродаваемаяПлощадь', 'Этаж',
'СтоимостьНадатуБрони', 'СкидкаНаквартиру',
'ФактическаяСтоимостьПомещения']
for col in num_cols:
    data[col] = pd.to_numeric(data[col], errors='coerce')

# б. Для бинарных признаков выполните кодирование
# (один вариант закодируйте 0, другой 1).
bin_mappings = {
    'ИсточникБрони': {'МП': 1, 'ручная': 0},
```

```

        'ВременнаяБронь': {'Да': 1, 'Нет': 0},
        'ТипСтоимости': {'Стоимость при 100% оплате': 1,
        'Стоимость в рассрочку': 0},
        'ВариантОплаты': {'Единовременная оплата': 1, 'Оплата
в рассрочку': 0},
        'СделкаАН': {'Да': 1, 'Нет': 0},
        'ИнвестиционныйПродукт': {'Да': 1, 'Нет': 0},
        'Привилегия': {'Да': 1, 'Нет': 0}
    }
for col, mapping in bin_mappings.items():
    data[col] = data[col].map(mapping)
    data[col] = pd.to_numeric(data[col], errors='coerce')

# с. Для категориальных не бинарных признаков выполните
one-hot кодирование.
# One hot кодирование (get_dummies) – это метод
преобразования категориальных данных в числовые
# для эффективного использования в моделях машинного
обучения.
# Принцип работы: для каждого категориального значения
создаётся отдельный столбец
# с бинарными значениями (1 или 0). При этом только один
столбец, соответствующий
# определённой категории, имеет значение 1, а все
остальные – 0.
# nan заменяем на N
data['Статус лида (из CRM)'] = data['Статус лида (из
CRM)'].fillna('N')
data = pd.get_dummies(data, columns=['Город', 'Статус
лида (из CRM)'], drop_first=True, dtype=int)

# d. Обработка поля «Тип»
def convert_rooms(value):
    # Проверяем, является ли значение строкой и
    оканчивается ли оно на 'к'
    if isinstance(value, str) and value.endswith('к'):
        # Преобразуем значение в число, убирая последний
        символ
        return pd.to_numeric(value[:-1].replace(',', '.'),
        errors='coerce')
    # Возвращаем NaN для некорректных значений
    return np.nan

```

```
# Применяем функцию к столбцу 'Тип'
data['Тип'] = data['Тип'].apply(convert_rooms)

print('\nОтфильтрованные данные')
print(data.head())

# Сохраняем отфильтрованные и преобразованные данные в
# новый файл
data.to_csv('Обработанная_база_3.csv', index=False,
sep=';', encoding='cp1251')
```

Проверка типов данных

```
ИсточникБрони          object
ВременнаяБронь         object
СледующийСтатус        int64
Город                  object
Тип                    object
ПродаваемаяПлощадь    object
Этаж                  float64
СтоимостьНаДатуБрони   object
ТипСтоимости           object
ВариантОплаты          object
СкидкаНаКвартиру      object
ФактическаяСтоимостьПомещения object
СделкаАН                object
ИнвестиционныйПродукт   object
Привилегия              object
Статус лида (из CRM)    object
dtype: object
```

Отфильтрованные данные

```
ИсточникБрони ... Статус лида (из CRM)_S
0          0 ... 1
1          0 ... 0
2          0 ... 1
3          0 ... 1
4          0 ... 0
```

4. Проверьте, есть ли по каким-либо признакам отсутствующие данные.
- Отсутствующие данные в поле «СкидкаНаКвартиру» замените на 0 (это значение по умолчанию – если поле не заполнено, то скидки, по всей видимости, нет).
 - Отсутствующие данные в полях «Тип» и «ПродаваемаяПлощадь» замените на медианное значение, вычисленное по всему набору данных (*признаки кажутся достаточно важными, поэтому удалять эти столбцы не хочется; пустых значений довольно много, поэтому удалять строки тоже не очень хорошо; какого-то значения «по умолчанию» для этих полей нет; поэтому заменить эти значения на медиану*).

представляется наилучшим решением).

- c. Что делать с полем «ВариантОплатыДоп» решите самостоятельно (допустимо также совсем убрать этот столбец из рассмотрения).
- d. По всем остальным полям примите решение самостоятельно. Если отсутствующих данных не много, то удалите соответствующие строки.

```
# 4. Проверка на отсутствие данных

# а. Заменяем отсутствующие данные в поле
«СкидкаНаКвартиру» на 0
data['СкидкаНаКвартиру'] =
data['СкидкаНаКвартиру'].fillna(0)

# б. Заменяем отсутствующие данные в полях «Тип» и
«ПродаваемаяПлощадь» на медиану
data['Тип'] = data['Тип'].fillna(data['Тип'].median())
# Заполняем пропуски в числовом поле медианой
data['ПродаваемаяПлощадь'] =
data['ПродаваемаяПлощадь'].fillna(data['ПродаваемаяПлоща
дь'].median())

# д. Удаление строк с отсутствующими данными по другим
полям
# Удаляем все строки из DataFrame 'data', которые
содержат хотя бы одно значение NaN. Изменения происходят
на месте.
data.dropna(inplace=True)

# Экспортируем обновленный DataFrame 'data' в файл
'обработанная_база_5.csv'.
data.to_csv('Обработанная_база_4.csv', index=False,
sep=';', encoding='cp1251')
```

5. Дополнение данных.

- a. Добавьте новый признак «Цена за квадратный метр». Он должен вычисляться на основе значений признаков «ФактическаяСтоимостьПомещения» и «ПродаваемаяПлощадь».
- b. Добавьте новый признак «Скидка в процентах», на основе значений «ФактическаяСтоимостьПомещения» и «СкидкаНаКвартиру».

```
# 5. Дополнение данных
print('\nДополнение данных')
```

```

# а. Добавляем новый признак «Цена за квадратный метр»
# Создаем новый столбец 'ЦенаЗАКвадратныйМетр', который
равен фактической
# стоимости помещения, деленной на продаваемую площадь.
data['ЦенаЗАКвадратныйМетр'] =
data['ФактическаяСтоимостьПомещения'] /
data['ПродаваемаяПлощадь']
print(data['ЦенаЗАКвадратныйМетр'].head())

# б. Добавляем новый признак «Скидка в процентах»
# Создаем новый столбец 'СкидкаВПроцентах', который
равен скидке на квартиру,
# деленной на фактическую стоимость помещения,
умноженной на 100.
data['СкидкаВПроцентах'] = (data['СкидкаНАКвартиру'] /
data['ФактическаяСтоимостьПомещения']) * 100
print(data['СкидкаВПроцентах'].head())

# Экспортируем обновленный DataFrame 'data' в файл
'обработанная_база_5.csv'.
data.to_csv('Обработанная_база_5.csv', index=False,
sep=';', encoding='cp1251')

```

```

Дополнение данных
0      59668.055556
1      101530.769231
2      80796.153846
3      104125.000000
4      80796.153846
Name: ЦенаЗАКвадратныйМетр, dtype: float64
0      0.0
1      0.0
2      0.0
3      0.0
4      0.0
Name: СкидкаВПроцентах, dtype: float64

```

6. Выполните нормализацию. Можете самостоятельно выбрать способ нормализации. «По умолчанию» предлагается выполнить минимаксную нормализацию и привести все значения к диапазону [0;1], кроме признака

«СкидкаНаКвартиру» - его логичнее приводить к диапазону [-0,5; 0,5].

```
# 6. Нормализация
# предлагаются выполнить минимаксную нормализацию и
# привести все значения к диапазону [0;1],
# кроме признака «СкидкаНаКвартиру» - его логичнее
# приводить к диапазону [-0,5; 0,5].

# сохраняем исходные значения скидки до масштабирования
original_discount = data['СкидкаНаКвартиру'].copy()

# Инициализируем объект MinMaxScaler для нормализации
# данных в диапазоне от 0 до 1
scaler = MinMaxScaler(feature_range=(0, 1))

# Получаем список всех числовых колонок
num_cols = data.select_dtypes(include=['float64',
                                         'int64']).columns.tolist()

# Убираем скидки, так как они будут нормализованы
# отдельно
cols_to_scale = [c for c in num_cols if c != 'СкидкаНаКвартиру']

# Масштабируем выбранные числовые признаки в диапазон
# [0,1]
data[cols_to_scale] =
scaler.fit_transform(data[cols_to_scale])

# нормализация скидки в диапазон [-0.5; 0.5]
discount_scaler = MinMaxScaler(feature_range=(-0.5,
                                              0.5))

data['СкидкаНаКвартиру'] =
discount_scaler.fit_transform(original_discount.values.reshape(-1, 1))

# Экспортируем обновленный DataFrame 'data' в файл
# 'обработанная_база_6.csv'.
data.to_csv('Обработанная_база_6.csv', index=False,
sep=';', encoding='cp1251')
```

7. Проверьте датасет на сбалансированность (количество строк со значением целевого признака 0 и со значением 1). Является ли датасет сбалансированным?

```
# 7. Проверьте датасет на сбалансированность
# по целевому признаку
print('\nПроверка на сбалансированность')
class_counts =
data['СледующийСтатус'].value_counts(normalize=True)
print(class_counts)
is_balanced = abs(class_counts[0] - class_counts[1]) <=
0.1
print(f"Датасет сбалансирован? {is_balanced}")
```

```
Проверка на сбалансированность
СледующийСтатус
0.0    0.715564
1.0    0.284436
Name: proportion, dtype: float64
Датасет сбалансирован? False
```

8. Сформируйте список факторных признаков и целевой признак.

```
# 8. Сформируйте список факторных признаков и целевой
признак.
print('\nФормируем список факторных признаков и целевой
признак')
# Получаем список всех столбцов, кроме целевого признака
'СледующийСтатус'
factor_feat =
data.columns.difference(['СледующийСтатус'])
# Определяем целевой признак
target_feat = 'СледующийСтатус'
# Выводим список факторных признаков.
print("Факторные признаки:", factor_feat.tolist())
# Выводим целевой признак.
print("Целевой признак:", target_feat)
```

```
Формируем список факторных признаков и целевой признак
Факторные признаки: ['ВариантОплаты', 'ВременнаяБронь', 'Город_Набережные Челны', 'Город_Пермь',
Целевой признак: СледующийСтатус
```

9. Выполните разбиение датасета на обучающую и тестовую выборки. При

формировании обучающей и тестовой выборок строки из исходного датафрейма должны выбираться в случайному порядке.

```
# 9. Выполните разбиение датасета на обучающую и тестовую выборки.
print('\nРазбиение датасета на обучающую и тестовую выборки')
# При формировании обучающей и тестовой выборок строки из исходного датафрейма
# должны выбираться в случайному порядке.
# Определяем обучающие признаки X как все факторные признаки.
X = data[factor_feat]
# Определяем целевой признак y как 'СледующийСтатус'.
y = data[target_feat]
# stratify следить за пропорциями классов в целевом векторе y.
X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42, stratify=y)
print("Размер обучающей выборки:", X_train.shape[0])
print("Размер тестовой выборки:", X_test.shape[0])
```

Разбиение датасета на обучающую и тестовую выборки

Размер обучающей выборки: 3130

Размер тестовой выборки: 783

10. Из библиотеки `sklearn.neighbors` возьмите алгоритм классификации KNN (`KNeighborsClassifier`). Постройте (обучите) модель. Для параметров используйте значения по умолчанию.

```
# 10. Из библиотеки sklearn.neighbors возьмите алгоритм классификации KNN
# Постройте (обучите) модель. Для параметров используйте значения по умолчанию.
# Создаем объект модели KNN (к ближайших соседей).
# Алгоритм классификации в машинном обучении. Он категоризирует точки
# данных на основе тенденций в ближайших точках данных или соседях.
knn_model = KNeighborsClassifier()
# Обучаем модель KNN на обучающих данных.
knn_model.fit(X_train, y_train)
```

11. Из библиотеки `sklearn.tree` возьмите алгоритм классификации на основе деревьев решений (`DecisionTreeClassifier`). Постройте (обучите) модель. Для параметров используйте значения по умолчанию.

```
# 11. Из библиотеки sklearn.tree возьмите алгоритм классификации
# на основе деревьев решений. Постройте (обучите) модель.
# Для параметров используйте значения по умолчанию.
# Он создаёт модель в виде структуры дерева, где каждый внутренний
# узел представляет решение на основе признака, каждая
# ветвь – результат решения, а каждый листовой узел –
# значение регрессии или метку класса.
# Создаем объект модели дерева решений (Decision Tree).
dt_model = DecisionTreeClassifier()
# Обучаем модель дерева решений на обучающих данных.
dt_model.fit(X_train, y_train)
```

12. Получите векторы прогнозных значений целевой переменной на обучающей и на тестовой выборках для каждой из моделей.

```
# 12. Получите векторы прогнозных значений целевой
переменной на обучающей
# и на тестовой выборках для каждой из моделей.
# Получаем прогнозы модели KNN на обучающих данных.
knn_train_preds = knn_model.predict(X_train)
# Получаем прогнозы модели KNN на тестовых данных.
knn_test_preds = knn_model.predict(X_test)

# Получаем прогнозы модели дерева решений на обучающих
данных.
dt_train_preds = dt_model.predict(X_train)
# Получаем прогнозы модели дерева решений на тестовых
данных.
dt_test_preds = dt_model.predict(X_test)
```

13. Посчитайте показатели качества: «F-мера», точность (`Precision`) и полнота (`Recall`) на обучающей и на тестовой выборках для каждой из моделей.

```
# 13. Посчитайте показатели качества: «F-мера»,
точность (Precision)
# и полнота (Recall) на обучающей и на тестовой
```

```
выборках для каждой из моделей.

print(' \nПоказатели качества')
# «F-мера» — это метрика, которая представляет собой гармоническое среднее между
# точностью (Precision) и полнотой (Recall). Она стремится к нулю, если точность
# или полнота стремится к нулю.
# Точность — доля объектов, действительно принадлежащих данному классу
# относительно всех объектов, которые система отнесла к этому классу.
# Полнота показывает, какую долю объектов, реально относящихся к положительному классу, предсказали верно.

# Определяем функцию для вычисления и вывода
# показателей качества модели
def evaluate_model(y_train_true, y_train_pred,
y_test_true, y_test_pred, model_name):
    print(f"Модель: {model_name}")
    print("Выборка:\tОбучающая\tТестовая")
    print(f"Precision:\t{precision_score(y_train_true, y_train_pred):.4f}\t{precision_score(y_test_true, y_test_pred):.4f}")
    print(f"Recall: \t{recall_score(y_train_true, y_train_pred):.4f}\t{recall_score(y_test_true, y_test_pred):.4f}")
    print(f"F1-score:\t{f1_score(y_train_true, y_train_pred):.4f}\t{f1_score(y_test_true, y_test_pred):.4f}\n")

# Оцениваем модели
evaluate_model(y_train, knn_train_preds, y_test,
knn_test_preds, "KNN")
evaluate_model(y_train, dt_train_preds, y_test,
dt_test_preds, "Decision Tree")
```

```
Показатели качества

Модель: KNN

Выборка: Обучающая Тестовая
Precision: 0.8630 0.8119
Recall: 0.7787 0.7354
F1-score: 0.8187 0.7718
```

```
Модель: Decision Tree

Выборка: Обучающая Тестовая
Precision: 1.0000 0.7810
Recall: 0.9809 0.7354
F1-score: 0.9904 0.7575
```

14. Сделайте вывод о том, насколько хорошо удалось решить задачу прогнозирования. Какая модель оказалась лучше? Дайте интерпретацию полученных значений Precision и Recall.

```
# При решении задачи прогнозирования KNN показала более стабильные результаты на тестовой выборке (Precision 0.812, Recall 0.735, F1-score 0.772), что указывает на хорошую способность модели обобщать новые данные. Decision Tree показала идеальные показатели на обучении, но сильное падение на teste (Precision 0.774, Recall 0.735, F1-score 0.754) # свидетельствует о переобучении. Таким образом, KNN оказалась лучше для прогнозирования: # её высокая Precision означает, что большинство предсказанных положительных случаев действительно верны, а разумный Recall показывает, что модель улавливает большую часть реальных положительных случаев, обеспечивая баланс между точностью и полнотой.
```

15. (1 балл) Постройте boxplot («ящик с усами») для всех числовых признаков. Отфильтруйте исходные данные, удалив выбросы. Пересоздайте тестовую и обучающую выборки, переобучите модели. Рассчитайте показатели качества. Как они изменились?

```
# 15. Постройте boxplot («ящик с усами») для всех числовых признаков.
```

```

# Отфильтруйте исходные данные, удалив выбросы.
# Пересоздайте тестовую и обучающую выборки, переобучите
# модели.

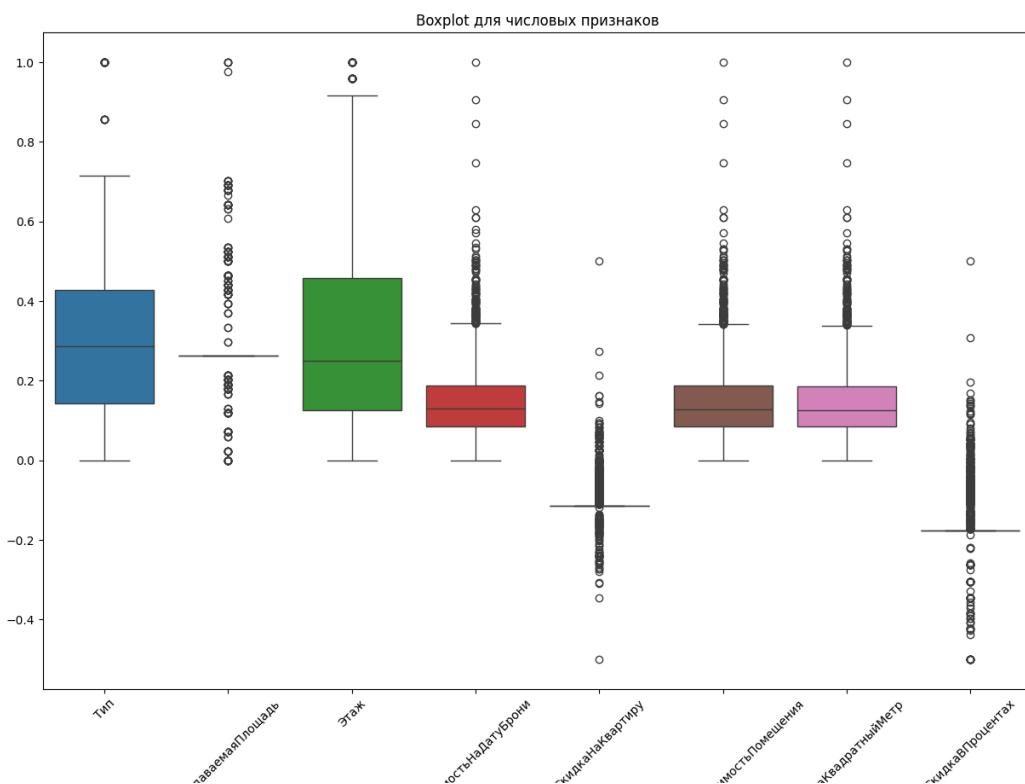
# Посчитайте показатели качества. Как они изменились?
plt.figure(figsize=(15, 10))

# Строим boxplot для числовых признаков из DataFrame
'data'.

cols_plot = ['Тип', 'ПродаваемаяПлощадь', 'Этаж',
'СтоимостьНадатуБрони', 'СкидкаНаКвартиру', 'ФактическаяСт
оимостьПомещения', 'ЦенаЗАквадратныйМетр', 'СкидкаВПроцент
ах']

sns.boxplot(data=data[cols_plot])
plt.title("Boxplot для числовых признаков")
# Поворачиваем метки на оси X на 45 градусов для лучшей
# читаемости.
plt.xticks(rotation=45)
plt.show()

```



```

# Удаление выбросов по 1.5*IQR
Q1 = data[cols_plot].quantile(0.25)
Q3 = data[cols_plot].quantile(0.75)

# Диапазон средних данных
IQR = Q3-Q1

```

```
# Определяем границы
lower_bounds = Q1 - 1.5 * IQR
upper_bounds = Q3 + 1.5 * IQR

# Создаем маску для строк, где все значения в нормальном диапазоне
all_values_normal = ((data[cols_plot] >= lower_bounds) &
                      (data[cols_plot] <= upper_bounds)).all(axis=1)

# Сохраняем только хорошие строки
filt_data = data[all_values_normal]

# Пересоздание обучающей и тестовой выборок

# Определяем обучающие признаки X из отфильтрованных данных
X_filt = filt_data[factor_feat]
# Определяем целевой признак y из отфильтрованных данных
y_filt = filt_data[target_feat]

# Разделяем отфильтрованные данные на обучающую и тестовую выборки (80% на обучение и 20% на тестирование).
X_train_filt, X_test_filt, y_train_filt, y_test_filt = train_test_split(X_filt, y_filt, test_size=0.2,
random_state=42, stratify=y_filt)

# Переобучение моделей

# Обучаем модель KNN на отфильтрованных обучающих данных.
knn_model.fit(X_train_filt, y_train_filt)
# Обучаем модель дерева решений на отфильтрованных обучающих данных.
dt_model.fit(X_train_filt, y_train_filt)

print('\nПрогнозирование на отфильтрованных данных и Показатели качества после фильтрации выбросов')
# Прогнозирование на отфильтрованных данных

# Получаем прогнозы модели KNN на отфильтрованных
```

```

обучающих данных.

knn_filt_train_pred = knn_model.predict(X_train_filt)
# Получаем прогнозы модели KNN на отфильтрованных тестовых данных.

knn_filt_test_pred = knn_model.predict(X_test_filt)

# Получаем прогнозы модели дерева решений на отфильтрованных обучающих данных.

dt_filt_train_pred = dt_model.predict(X_train_filt)
# Получаем прогнозы модели дерева решений на отфильтрованных тестовых данных.

dt_filt_test_pred = dt_model.predict(X_test_filt)

# Показатели качества после фильтрации выбросов
evaluate_model(y_train_filt, knn_filt_train_pred,
y_test_filt, knn_filt_test_pred, "KNN (фильтр)")
evaluate_model(y_train_filt, dt_filt_train_pred,
y_test_filt, dt_filt_test_pred, "Decision Tree
(фильтр)")

# Удаление выбросов ухудшило качество обеих моделей.

```

```

Прогнозирование на отфильтрованных данных и Показатели качества после фильтрации выбросов
Модель: KNN (фильтр)
Выборка: Обучающая Тестовая
Precision: 0.9124 0.7941
Recall: 0.6667 0.5745
F1-score: 0.7704 0.6667

Модель: Decision Tree (фильтр)
Выборка: Обучающая Тестовая
Precision: 1.0000 0.6628
Recall: 0.9680 0.6064
F1-score: 0.9837 0.6333

```

Все показатели ухудшились.

16.(2 балла) Выполните подбор параметров для алгоритмов KNN и деревьев решений. Для KNN попробуйте изменять параметр k – количество соседей, для деревьев решений – глубину дерева. Постройте графики зависимости показателей качества от значения параметра (от k в случае KNN и от глубины дерева в случае деревьев решений). Для параметра k рассматривайте диапазон от 1 до 40. Для глубины дерева – от 2 до 40. По графикам определите оптимальные значения параметров.

```
# 16. Выполните подбор параметров для алгоритмов KNN и
деревьев решений.

# Для KNN попробуйте изменять параметр k - количество
соседей,
# для деревьев решений - глубину дерева. Постройте
графики зависимости
# показателей качества от значения параметра (от k в
случае KNN и от
# глубины дерева в случае деревьев решений). Для
параметра k рассматривайте
# диапазон от 1 до 40. Для глубины дерева - от 2 до 40.
По графикам определите
# оптимальные значения параметров.

# Задаем диапазон значений k от 1 до 40 для KNN.
# Количество соседей
k_values = range(1, 41)
# Инициализируем список для хранения F1-меры для KNN.
f1_scores_knn = []

# Проходим по каждому значению k.
for k in k_values:
    # Создаем временную модель KNN с текущим значением k.
    knn_temp_model = KNeighborsClassifier(n_neighbors=k)
    # Обучаем временную модель на исходных обучающих
    # данных.
    knn_temp_model.fit(X_train, y_train)
    # Получаем прогнозы на тестовых данных.
    preds = knn_temp_model.predict(X_test)
    # Вычисляем F1-меру и добавляем в список.
    f1_scores_knn.append(f1_score(y_test, preds))

# Задаем диапазон значений глубины дерева от 2 до 40.
depth_values = range(2, 41)
# Инициализируем список для хранения F1-меры для дерева
решений
f1dt_depths = []

# Проходим по каждому значению глубины.
for depth in depth_values:
    # Создаем временную модель дерева решений с текущей
```

```
глубиной.

dt_temp_model =
DecisionTreeClassifier(max_depth=depth)
    # Обучаем временную модель на исходных обучающих
данных.
    dt_temp_model.fit(X_train, y_train)
    # Получаем прогнозы на тестовых данных.
    preds = dt_temp_model.predict(X_test)
    # Вычисляем F1-меру и добавляем в список.
    f1dt_depths.append(f1_score(y_test, preds))

# Графики зависимости показателей качества от значения
параметра

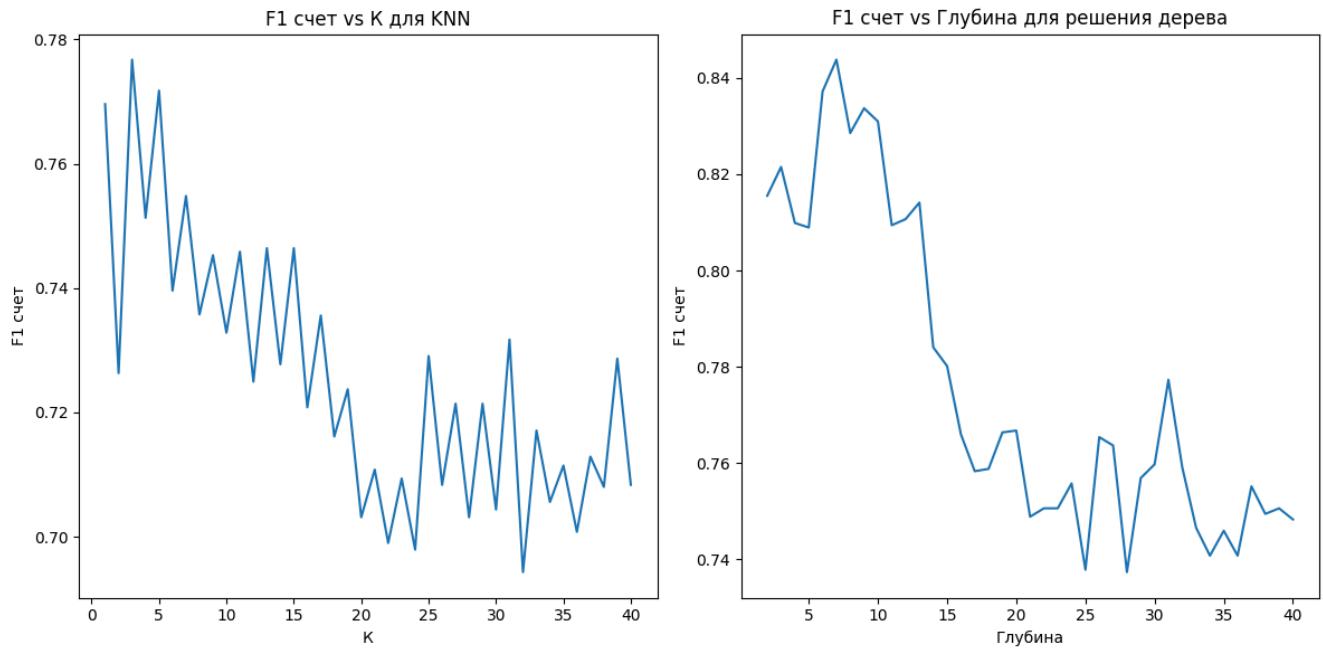
# Создаем фигуру для графиков с заданным размером.
plt.figure(figsize=(12, 6))
# Определяем подграфик для первого графика.
plt.subplot(1, 2, 1)
# Строим график зависимости F1-меры от k для KNN.
plt.plot(k_values, f1_scores_knn)
# Устанавливаем заголовок для графика KNN.
plt.title('F1 счет vs К для KNN')
# Устанавливаем подпись для оси X.
plt.xlabel('К')
# Устанавливаем подпись для оси X.
plt.ylabel('F1 счет')

# Определяем подграфик для второго графика.
plt.subplot(1, 2, 2)
# Строим график зависимости F1-меры от глубины дерева
решений.
plt.plot(depth_values, f1dt_depths)
# Устанавливаем заголовок для графика дерева решений.
plt.title('F1 счет vs Глубина для решения дерева')
# Устанавливаем подпись для оси X.
plt.xlabel('Глубина')
# Устанавливаем подпись для оси Y.
plt.ylabel('F1 счет')

# Автоматически регулируем параметры подграфиков для
лучшего отображения.
plt.tight_layout()
```

```
plt.show()
```

```
# При подборе параметров оптимальным для KNN оказалось  
k=5-7 (F1-score ≈ 0.78),  
# для Decision Tree наилучшие результаты достигаются  
# при глубине 4-6 (F1-score ≈ 0.83-0.84).
```



17. (1 балл) Из библиотеки `sklearn.linear_model` возьмите алгоритм логистической регрессии (`LogisticRegression`). Постройте (обучите) модель. Посчитайте показатели качества. Сравните результат с другими моделями.

```
# 17. Из библиотеки sklearn.linear_model возьмите  
алгоритм логистической регрессии  
# Логистическая регрессия – статистическая модель,  
используемая для предсказания  
# вероятности возникновения интересующего события с  
помощью логистической функции  
# Постройте (обучите) модель. Посчитайте показатели  
качества.  
# Сравните результат с другими моделями.  
# Создаем объект логистической регрессии с максимальным  
числом итераций равным 1000.  
log_model = LogisticRegression(max_iter=1000)  
# Обучаем модель логистической регрессии на обучающих  
данных.  
log_model.fit(X_train, y_train)  
  
# Получаем прогнозы логистической регрессии  
log_train_pred = log_model.predict(X_train)
```

```

log_test_pred = log_model.predict(X_test)

# Показатели качества
evaluate_model(y_train, log_train_pred, y_test,
log_test_pred, "Логистическая регрессия")

# Логистическая регрессия показала высокую точность
# (Precision=0.8757) на тестовых данных,
# превзойдя KNN (0.8119) и Decision Tree (0.7773), что
# означает минимальное количество
# ложных срабатываний при прогнозировании продаж. При
# этом её полнота (Recall=0.6951)
# оказалась ниже, чем у KNN (0.7354), но F1-score
# (0.7750) сравним с лучшими результатами,
# демонстрируя сбалансированность между точностью и
# охватом реальных продаж.

Модель: Логистическая регрессия
Выборка: Обучающая Тестовая
Precision: 0.8720 0.8757
Recall: 0.6584 0.6951
F1-score: 0.7503 0.7750

```

18.(1 балл) Из библиотеки sklearn.svm возьмите алгоритм SVM (машины опорных векторов) (LinearSVC). Постройте (обучите) модель. Посчитайте показатели качества. Сравните результат с другими моделями.

```

# 18. Из библиотеки sklearn.svm возьмите алгоритм SVM
# Метод опорных векторов
# Постройте (обучите) модель. Посчитайте показатели
# качества.
# Сравните результат с другими моделями.
svm_model = LinearSVC()
# Обучаем модель
svm_model.fit(X_train, y_train)

# Получаем прогнозы
svm_train_pred = svm_model.predict(X_train)
svm_test_pred = svm_model.predict(X_test)

# Показатели качества
evaluate_model(y_train, svm_train_pred, y_test,

```

```
svm_test_pred, "SVM")

# SVM продемонстрировала наилучший баланс метрик среди
всех моделей: самую высокую
# F1-score (0.7893) на тестовых данных, сочетая высокую
точность (0.8579) и хорошую
# полноту (0.7309). Это превосходит результаты KNN
(F1=0.7718), Decision Tree (F1=0.7558)
# и логистической регрессии (F1=0.7750), что делает SVM
оптимальным выбором для задачи
# прогнозирования продаж с точки зрения общего качества
классификации.
```

Модель: SVM
Выборка: Обучающая Тестовая
Precision: 0.8605 0.8579
Recall: 0.7067 0.7309
F1-score: 0.7761 0.7893