NN – Домашнее задание 1

Дедлайн: 23:59, 24 июля

Вам необходимо:

Принять участие в соревновании на Kaggle:

В рамках этого соревнования перед вами стоит две задачи:

1. Написать и обучить самописную архитектуру модели на **PyTorch**.
2. Получить высокий **score** на финальном leaderboard.

Требования к `ipynb`:

- В нем должен быть сохранен **output** ячеек (не очищен).
- Внутри ноутбука указать:
    a. Ваш ник на Kaggle и скриншот с итоговой позицией на leaderboard *(важно — после завершения соревнования)*.
    b. Ваше ФИО.

Оценивание соревнования:

ДЗ 1 — макс. 10 баллов

- Корректная реализация и обучение самописной архитектуры: **5 баллов**.

Дополнительно баллы за ноутбук:

1. Логичность и корректность кода и комментариев.
2. Корректная валидация модели на отложенной выборке.
3. Применение дополнительных способов обработки данных.
4. Использование открытых моделей.

Баллы за итоговый score:

- Позиция **[1; 5]** на итоговом leaderboard: **+5 баллов**.
- Позиция **[6; 10]** на итоговом leaderboard: **+3 балла**.
- Позиция **[11; 15]** на итоговом leaderboard: **+1 балл**.

Итог:

- Зачет на 4: **7 баллов**.
- Зачет на 5: **8–10 баллов**.

Дополнительно (челлендж «Битва магов»):

- Топ-5 на итоговом leaderboard в одном из соревнований: **+5 баллов**.

- Побили baseline метрику — можно написать отчет в общий чат по шаблону: **+2 балла**.

Шаблон для +2 баллов:

1. Итоговая позиция в соревновании.
2. Intro.
3. Анализ того, что пробовали.
4. Анализ причин неудач (или потенциальных проблем).
5. Анализ применимости решения в бизнес-процессах.
6. Outro.
7. Тегнуть Арину, чтобы начислила баллы.

Как отправить результат:

1. Ссылка на файл с ноутбуком в облачном хранилище (Google Диск и т.д.).
2. Ссылка на файл с ноутбуком в GitHub.

Важно: ссылки должны быть рабочие и с открытым доступом для скачивания.

Желаем удачи!

Информация об участнике соревнования:

1) Ник на Kaggle - **Vladimir Smirnovve** 2) Скриншот с итоговой позицией на leaderboard (находится по ссылке)

- **10 место**
- **Private Score: 0.739556**
- **Public score: 0.724669** 4) ФИО - **Смирнов Владимир Евгеньевич**

Аннотация

В итоговом рейтинге соревнования я занял (если бы сдавал задание до окончания дедлайна; о причинах поздней сдачи написал в сообщении; для читающих ноутбук в двух словах - брал перерыв от учебы для поступления в магистратуру) **10 место** (по обоим лидербордам) с результатами:

- **Private Score: 0.739556**
- **Public score: 0.724669**
- **Использована модель TF-IDF + Ridge.**

Что я пробовал

Базовый пайплайн: TF-IDF + Ridge Regression.

- Собрал текстовые признаки через TF-IDF (1–2 граммы);
- Добавил engineered features (длина описания, количество скиллов, индикаторы senior/junior/middle);
- Закодировал категориальные признаки (компания, локация).

Что пошло не так / могло пойти не так

- **TF-IDF**: хорошо работает на локальных данных, но не учитывает контекст. Есть риск переобучения на редкие n-граммы.
- **BERT**: оказалось тяжёлым для обучения на доступных ресурсах.

Применимость в бизнес-процессах

Моё решение можно встроить в HR-аналитику:

- **Автоподсказка уровня зарплаты** при размещении вакансии (работодатель вводит текст, модель прогнозирует вилку);
- **Аналитика рынка труда**: агрегировать прогнозы по компаниям и регионам => выявлять недооценённые вакансии или дефицитные профессии;

Таким образом, модель может помочь и работодателям (адекватно формировать офферы), и соискателям (избегать заниженных предложений).

Самописная архитектура находится в самом низу ноутбука!

Библиотеки

```
!pip install --upgrade scikit-learn
```

```
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.11/dist-packages (1.7.1)
Requirement already satisfied: numpy>=1.22.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.8.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: mkl_fft in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.22.0->scikit-
learn) (1.3.8)
Requirement already satisfied: mkl_random in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.22.0->scikit-
learn) (1.2.4)
Requirement already satisfied: mkl_umath in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.22.0->scikit-
learn) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dist-
packages (from numpy>=1.22.0->scikit-learn) (2025.2.0)
Requirement already satisfied: tbb4py in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.22.0->scikit-
learn) (2022.2.0)
Requirement already satisfied: mkl-service in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.22.0->scikit-
learn) (2.4.1)
```

```
Requirement already satisfied: intel-openmp<2026,>=2024 in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.22.0-
>scikit-learn) (2024.2.0)
Requirement already satisfied: tbb==2022.* in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.22.0-
>scikit-learn) (2022.2.0)
Requirement already satisfied: tcmlib==1.* in
/usr/local/lib/python3.11/dist-packages (from tbb==2022.*->mkl-
>numpy>=1.22.0->scikit-learn) (1.4.0)
Requirement already satisfied: intel-cmplr-lib-rt in
/usr/local/lib/python3.11/dist-packages (from mkl_umath-
>numpy>=1.22.0->scikit-learn) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in
/usr/local/lib/python3.11/dist-packages (from intel-
openmp<2026,>=2024->mkl->numpy>=1.22.0->scikit-learn) (2024.2.0)

!pip install nltk razdel pymorphy3 wordcloud

Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-
packages (3.9.1)
Requirement already satisfied: razdel in
/usr/local/lib/python3.11/dist-packages (0.5.0)
Requirement already satisfied: pymorphy3 in
/usr/local/lib/python3.11/dist-packages (2.0.4)
Requirement already satisfied: wordcloud in
/usr/local/lib/python3.11/dist-packages (1.9.4)
Requirement already satisfied: click in
/usr/local/lib/python3.11/dist-packages (from nltk) (8.2.1)
Requirement already satisfied: joblib in
/usr/local/lib/python3.11/dist-packages (from nltk) (1.5.1)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-
packages (from nltk) (4.67.1)
Requirement already satisfied: dawg2-python>=0.8.0 in
/usr/local/lib/python3.11/dist-packages (from pymorphy3) (0.9.0)
Requirement already satisfied: pymorphy3-dicts-ru in
/usr/local/lib/python3.11/dist-packages (from pymorphy3)
(2.4.417150.4580142)
Requirement already satisfied: numpy>=1.6.1 in
/usr/local/lib/python3.11/dist-packages (from wordcloud) (1.26.4)
Requirement already satisfied: pillow in
/usr/local/lib/python3.11/dist-packages (from wordcloud) (11.2.1)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.11/dist-packages (from wordcloud) (3.7.2)
Requirement already satisfied: mkl_fft in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.6.1->wordcloud)
(1.3.8)
Requirement already satisfied: mkl_random in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.6.1->wordcloud)
```

```
(1.2.4)
Requirement already satisfied: mkl_umath in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.6.1->wordcloud)
(0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dist-
packages (from numpy>=1.6.1->wordcloud) (2025.2.0)
Requirement already satisfied: tbb4py in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.6.1->wordcloud)
(2022.2.0)
Requirement already satisfied: mkl-service in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.6.1->wordcloud)
(2.4.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(1.3.2)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(4.58.4)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(1.4.8)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(25.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(2.9.0.post0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7-
>matplotlib->wordcloud) (1.17.0)
Requirement already satisfied: intel-openmp<2026,>=2024 in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.6.1-
>wordcloud) (2024.2.0)
Requirement already satisfied: tbb==2022.* in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.6.1-
>wordcloud) (2022.2.0)
Requirement already satisfied: tcmlib==1.* in
/usr/local/lib/python3.11/dist-packages (from tbb==2022.*->mkl-
>numpy>=1.6.1->wordcloud) (1.4.0)
Requirement already satisfied: intel-cmplr-lib-rt in
/usr/local/lib/python3.11/dist-packages (from mkl_umath->numpy>=1.6.1-
>wordcloud) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in
```

```
/usr/local/lib/python3.11/dist-packages (from intel-
openmp<2026,>=2024->mkl->numpy>=1.6.1->wordcloud) (2024.2.0)

# --- Системные и общие ---
import os
import math
import random
import re
import warnings
from collections import Counter
from functools import lru_cache

import numpy as np
import pandas as pd
from scipy.sparse import hstack, issparse
from tqdm.auto import tqdm
from tqdm.notebook import tqdm as tqdm_notebook
from bs4 import BeautifulSoup
from IPython.display import display

warnings.filterwarnings('ignore')

# --- Визуализация ---
import matplotlib.pyplot as plt
import seaborn as sns
from graphviz import Digraph
from wordcloud import WordCloud

# --- NLP и лингвистика ---
import nltk
from nltk.corpus import stopwords
from razdel import tokenize as razdel_tokenize
import pymorphy3

# --- PyTorch ---
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, TensorDataset
from torch.utils.tensorboard import SummaryWriter

# --- Torchvision ---
import torchvision
import torchvision.transforms as transforms
from torchvision.models import resnet18, ResNet18_Weights

# --- Hugging Face / Transformers ---
from transformers import (
    AutoTokenizer,
```

```
    AutoModel,
    AutoModelForSequenceClassification,
    Trainer,
    TrainingArguments
)
from datasets import Dataset as HFDataset

# --- Scikit-learn ---
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer,
CountVectorizer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score, confusion_matrix,
classification_report

# --- Константы и настройка среды ---
RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)
torch.manual_seed(RANDOM_STATE)

# --- Инициализация лидерборда ---
leaderboard = pd.DataFrame(columns=['Метод', 'R2_Score'])
```

```
2025-08-20 17:17:35.449493: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
E0000 00:00:1755710255.471513     187 cuda_dnn.cc:8310] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1755710255.478255     187 cuda_blas.cc:1418] Unable to
register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
```

```
# Определяем устройство
if torch.backends.mps.is_available():
    device = torch.device("mps")
    torch.mps.manual_seed(RANDOM_STATE)
elif torch.cuda.is_available():
    device = torch.device("cuda")
    torch.cuda.manual_seed_all(RANDOM_STATE)
else:
    device = torch.device("cpu")

print(f"Используемое устройство: {device}")
```

```
Используемое устройство: cuda
```

Загружаем данные

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

# Загружаем данные из файлов соревнования
try:
    full_train_df =
pd.read_csv('/kaggle/input/teta-nn-1-2025/train.csv')
    test_df = pd.read_csv('/kaggle/input/teta-nn-1-2025/test.csv')
except FileNotFoundError:
    print("Ошибка: Убедитесь, что файлы train.csv и test.csv находятся
в папке 'teta-nn-1-2025/'")
    # Создадим заглушку, чтобы ноутбук мог работать дальше
    full_train_df = pd.DataFrame({
        'title': ['Python Developer'], 'location': ['Москва'],
'company': ['Yandex'],
        'skills': ['Python, SQL'], 'description': ['Developing cool
services'],
        'experience_from': [3], 'salary_from': [200],
'log_salary_from': [5.3]
    })
    test_df = full_train_df.copy()

# # Загружаем данные из файлов соревнования
# try:
#     full_train_df =
pd.read_csv('/content/drive/MyDrive/MTS_School_of_Data_Analysis/mts_bi
g_data/mts_neural_networks/homework_smirnov_ve_neural_networks1/
teta_nn_1_2025/train.csv')
#     test_df =
pd.read_csv('/content/drive/MyDrive/MTS_School_of_Data_Analysis/mts_bi
g_data/mts_neural_networks/homework_smirnov_ve_neural_networks1/
teta_nn_1_2025/test.csv')
# except FileNotFoundError:
#     print("Ошибка: Убедитесь, что файлы train.csv и test.csv
находятся в папке 'teta-nn-1-2025/'")
#     # Создадим заглушку, чтобы ноутбук мог работать дальше
#     full_train_df = pd.DataFrame({
#         'title': ['Python Developer'], 'location': ['Москва'],
'company': ['Yandex'],
#         'skills': ['Python, SQL'], 'description': ['Developing cool
services'],
#         'experience_from': [3], 'salary_from': [200],
'log_salary_from': [5.3]
#     })
#     test_df = full_train_df.copy()
```

```
full_train_df
```

```
                                                   title  \
0                          Специалист технической поддержки
1                          Product Owner в ИТ-стартап (B2C)
2      Системный администратор ( Клинический Госпитал...
3                             Системный Администратор Linux
4                 Начальник участка общестроительных работ
...                                                     ...
16662  Младший системный администратор (технический с...
16663                          PHP-разработчик (1С-Битрикс)
16664                                       Разработчик 1С
16665                                Менеджер IT-проектов
16666                          Инженер технической поддержки

                               location                        company  \
0                               Воронеж                           ВКТМ
1                                Москва                  Radium Finance
2                                Москва                         Медскан
3                                Москва                         Selecty
4      Пушкино (Московская область)                          NK GROUP
...                                ...                             ...
16662                            Москва                       АйПиМатика
16663                           Барнаул                          Киберия
16664                            Москва                         Lerteco
16665                             Курск  Пластилин-арт (Осипов Ю.В.)
16666                            Липецк                        IT-Лидер

                                                   skills  \
0                                                     NaN
1      CustDev, Стратегический менеджмент, Управление...
2                                                     NaN
3                                       Linux, PostgreSQL
4                                                     NaN
...                                                   ...
16662                                                 NaN
16663  PHP, MySQL, PostgreSQL, Docker, Git, 1С-Битрик...
16664  1С программирование, 1С: Бухгалтерия, 1С: Пред...
16665  CRM, Битрикс24, Управление интернет-проектами,...
16666  Администрирование сетевого оборудования, Ремон...

                                              description
experience_from  \
0      Обязанности:     Обеспечение бесперебойной раб...
1.0
1      Эта вакансия про тебя, если симбиоз управленче...
6.0
2      АО «Медскан» – динамично развивающаяся группа ...
1.0
3      Задачи:     Администрирование инфраструктуры н...
```

```
3.0
4      NK Group  – ведущий девелопер индустриальных о...
3.0
...                                                  ...
...
16662  Мы - активно растущий Value added дистрибьютор...
1.0
16663  Привет! Мы, Киберия – активно развивающаяся ко...
1.0
16664  В крупном холдинге-лидере цифровых решений во ...
3.0
16665  Пластилин-арт  специализируется на разработке ...
1.0
16666  Ищем трудолюбивого и ответственного сотрудника...
3.0

       salary_from  log_salary_from
0             60.0         4.094345
1            200.0         5.298317
2            130.0         4.867534
3            170.0         5.135798
4            200.0         5.298317
...            ...              ...
16662         90.0         4.499810
16663         60.0         4.094345
16664        340.0         5.828946
16665         45.0         3.806662
16666         40.0         3.688879

[16667 rows x 8 columns]

test_df

                                                  title  \
0           Ведущий программист 1С (г. Санкт-Петербург)
1             Ресерчер (поиск товаров на маркетплейсах)
2                                Системный администратор
3       Инженер по интеграции систем защиты информации
4     Ведущий менеджер по работе с маркетплейсом Wil...
...                                                 ...
5551                Специалист по внутреннему контролю
5552                                           Дизайнер
5553            Инженер по информационной безопасности
5554             Инженер-программист группы разработки
5555     Графический дизайнер / креатор (бренд одежды)

                          location           company  \
0                 Санкт-Петербург  Коннект персонал
1                          Москва      Right Choice
2                 Нижний Новгород          Меридиан
```

```
3                       Новосибирск          СофтМолл
4                            Москва    ДЖЕЙКЕТ РАБОТА
...                              ...                ...
5551  Ростов (Ярославская область)             Атрус
5552                         Москва          СИНЕРГИЯ
5553                         Москва       СДК СИСТЕМС
5554                         Рязань           Альфа-М
5555                 Санкт-Петербург         Бат Нортон

                                                    skills  \
0     1С программирование, MS SQL Server, 1С: ERP, О...
1     Конкурентная аналитика, Аналитические исследов...
2     Администрирование сетевого оборудования, Админ...
3     Информационная безопасность, Аналитическое мыш...
4                                                   NaN
...                                                 ...
5551                                                NaN
5552                                                NaN
5553  Информационная безопасность, PKI, Linux, АПКШ ...
5554                                                NaN
5555                                                NaN

                                               description
experience_from
0     Крупнейший производственный комплекс легкой пр...
3.0
1     Мы молодая команда селлеров, состоящая из 12 ч...
1.0
2     О компании:   Уже более 30 лет мы успешно прои...
1.0
3     SoftMall  – это аккредитованная IT-компания, к...
1.0
4     Вакансия компании:  Brosco Компания Brosco зан...
1.0
...                                                 ...                .
..
5551  Обязанности:     Обеспечивает экономическую бе...
0.0
5552  Обязанности на занимаемой позиции:     Создани...
0.0
5553  Обязанности:       Проектирование и реализация...
3.0
5554  Научно-производственному комплексу срочно треб...
1.0
5555  Стань частью крутой команды Bat Norton как Гра...
1.0

[5556 rows x 6 columns]
```

```python
# Разбиваем исходный train на обучающую и валидационную выборки
train_df, val_df = train_test_split(
    full_train_df,
    test_size=0.25,
    random_state=RANDOM_STATE
)

# Сбрасываем индексы для удобства
train_df = train_df.reset_index(drop=True)
val_df = val_df.reset_index(drop=True)

# Создаем единое текстовое поле для анализа
text_cols = ['title', 'location', 'company', 'skills', 'description']
for df in [train_df, val_df, test_df]:
    # Заполняем пропуски пустыми строками, чтобы избежать ошибок
    df[text_cols] = df[text_cols].fillna('')
    # Объединяем текстовые колонки через разделитель
    df['full_text'] = df[text_cols].agg(' | '.join, axis=1)

print("Пример данных из train:")
display(train_df[['full_text', 'log_salary_from']].head(3))
```

```
Пример данных из train:
```

	full_text	log_salary_from
0	QA Engineer | Москва (м. Сколково / м. Крылатс...	5.298317
1	Senior Backend QA engineer | Санкт-Петербург |...	5.010635
2	Python разработчик (Django) | Москва | Hammer ...	3.912023

Предобработка текста

Частично пайплайн обработки повторяет уже проведенные манипуляции с данными, показанные на семинарах. Тем не менее, определенным образом пайплайн был доработан. Дополнительные комментарии даны в аннотации.

1. **Токенизация**: Разбиваем текст на слова (токены).
2. **Лемматизация**: Приводим каждое слово к его начальной форме (лемме). Например, "требования" -> "требование".
3. **Удаление стоп-слов**: Выкидываем неинформативные слова типа "и", "в", "на", "мы".

Полный пайплайн предобработки

Пайплайн обернут в функции для упрощения восприятия, а также для более удобного применения к различным частям набора данных. Изначально он находился в разрозненном состоянии, обертка делалась уже для готового процесса предобработки. Выборочно даны комментарии по различным частям кода.

```
# Настройка NLTK и стоп-слов
nltk.download('stopwords', quiet=True)
stop_words = set(stopwords.words('russian'))
stop_words.update(['наш', 'компания', 'команда', 'работа', 'вакансия',
'искать', 'требуется', 'условие', 'обязанность'])

template_phrases = ['дружный коллектив', 'официальное
трудоустройство', 'социальный пакет']
important_terms = {'python', 'sql', 'java', 'c++', 'linux', 'docker'}

morph = pymorphy3.MorphAnalyzer()
tqdm.pandas()

@lru_cache(maxsize=100000)
def lemmatize_token(token: str) -> str:
    return morph.parse(token)[0].normal_form

def clean_text(text: str) -> str:
    text = BeautifulSoup(text, "lxml").get_text(" ")
    text = re.sub(r'[^a-яA-Яa-zA-Z0-9+#]', ' ', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

def preprocess_text_razdel(text: str) -> str:
    text = clean_text(text)
    tokens = [t.text.lower() for t in razdel_tokenize(text) if
t.text.isalpha() or t.text in important_terms]
    lemmas = [lemmatize_token(tok) if tok not in important_terms else
tok for tok in tokens]
    lemmas = [lemma for lemma in lemmas if lemma not in stop_words and
len(lemma) > 2]
    processed = " ".join(lemmas)
    # удаление шаблонных фраз (немного доработано)
    for phrase in template_phrases:
        processed = re.sub(r'\b' + re.escape(phrase) + r'\b', '',
processed)
    return processed.strip()


# --- класс для предобработки данных ---

class FeatureProcessor(BaseEstimator, TransformerMixin):
```

```
    """
    Класс для полной предобработки данных, решающий проблему утечки
данных.
    1. Обучается на train-выборке методом fit().
    2. Применяет преобразования к любой выборке методом transform().
    """
    def __init__(
        self,
        text_cols=['title', 'location', 'company', 'skills',
'description'],
        # "razdel", "bert", "tfidf", "bow"
        mode="tfidf",
        use_structured=False,
        add_features=True,
        encode_categorical=True,
        cat_cols=['location', 'company'],
        # "ordinal" или "onehot"
        cat_encoding="ordinal",
        tfidf_params=None
    ):
        # Сохраняем
        self.text_cols = text_cols
        self.mode = mode
        self.use_structured = use_structured
        self.add_features = add_features
        self.encode_categorical = encode_categorical
        self.cat_cols = cat_cols
        self.cat_encoding = cat_encoding
        self.tfidf_params = tfidf_params or {"max_features": 10000,
"ngram_range": (1, 2)}

        # Здесь будут храниться обученные объекты
        self.vectorizer_ = None
        self.cat_encoder_ = None
        self.company_freq_map_ = None
        self.location_freq_map_ = None

    def _get_full_text(self, df: pd.DataFrame) -> pd.Series:
        """Вспомогательная функция для сборки текста."""
        df_copy = df.copy()
        df_copy[self.text_cols] = df_copy[self.text_cols].fillna('')
        if self.use_structured:
            return (
                "[TITLE] " + df_copy['title'] + " " +
                "[COMPANY] " + df_copy['company'] + " " +
                "[LOCATION] " + df_copy['location'] + " " +
                "[SKILLS] " + df_copy['skills'] + " " +
                "[DESCRIPTION] " + df_copy['description']
            )
```

```
        else:
            return df_copy[self.text_cols].agg(' | '.join, axis=1)

    def fit(self, df: pd.DataFrame, y=None):
        """
        Обучает все трансформеры (векторизатор, кодировщики) только на обучающих данных.
        """
        print("Fitting FeatureProcessor...")
        df_copy = df.copy()

        # 1. Обучаем векторизатор (TF-IDF/BoW)
        if self.mode in ["tfidf", "bow"]:
            df_copy['processed_text'] = self._get_full_text(df_copy).progress_apply(preprocess_text_razdel)
            vectorizer_cls = TfidfVectorizer if self.mode == "tfidf" else CountVectorizer
            self.vectorizer_ = vectorizer_cls(**self.tfidf_params)
            self.vectorizer_.fit(df_copy['processed_text'])

        # 2. Обучаем кодировщик категорий
        if self.encode_categorical and self.cat_cols:
            if self.cat_encoding == "ordinal":
                self.cat_encoder_ = OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
            else:
                self.cat_encoder_ = OneHotEncoder(handle_unknown="ignore")
            self.cat_encoder_.fit(df_copy[self.cat_cols].fillna(''))

        # 3. Сохраняем частоты признаков
        if self.add_features:
            self.company_freq_map_ = df_copy['company'].value_counts()
            self.location_freq_map_ = df_copy['location'].value_counts()

        print("Fit completed.")
        return self

    def transform(self, df: pd.DataFrame, return_df=False):
        """
        Применяет обученные преобразования к новым данным (train, val или test).
        """
        print(f"Transforming {df.shape[0]} samples...")
        df_transformed = df.copy()

        # --- Текстовые признаки ---
        full_text = self._get_full_text(df_transformed)
        X_text = None
```

```
        if self.mode in ["tfidf", "bow"]:
            processed_text =
full_text.progress_apply(preprocess_text_razdel)
            if self.vectorizer_ is None:
                raise RuntimeError("Vectorizer has not been fitted.
Call fit() first.")
            X_text = self.vectorizer_.transform(processed_text)
        elif self.mode == "razdel":
             df_transformed['processed_text'] =
full_text.progress_apply(preprocess_text_razdel)

        # --- Дополнительно заинжиниренные признаки ---
        X_num = None
        if self.add_features:
            df_transformed['desc_len'] =
df_transformed['description'].fillna('').apply(lambda x:
len(x.split()))
            df_transformed['num_skills'] =
df_transformed['skills'].fillna('').apply(lambda x: len(x.split(','))
if x else 0)
            df_transformed['has_junior'] =
df_transformed['title'].str.contains("junior|младший",
case=False).astype(int)
            df_transformed['has_middle'] =
df_transformed['title'].str.contains("middle", case=False).astype(int)
            df_transformed['has_senior'] =
df_transformed['title'].str.contains("senior|ведущий",
case=False).astype(int)

            if self.company_freq_map_ is None or
self.location_freq_map_ is None:
                 raise RuntimeError("Frequency maps have not been
created. Call fit() first.")
            df_transformed['company_freq'] =
df_transformed['company'].map(self.company_freq_map_).fillna(0)
            df_transformed['location_freq'] =
df_transformed['location'].map(self.location_freq_map_).fillna(0)

            num_feature_cols = [
                'desc_len', 'num_skills', 'has_junior', 'has_middle',
                'has_senior', 'company_freq', 'location_freq'
            ]

            if 'experience_from' in df_transformed.columns:
                num_feature_cols.append('experience_from')

            X_num = df_transformed[num_feature_cols].values

        # --- Категориальные признаки ---
        X_cat = None
```

```
        if self.encode_categorical and self.cat_cols:
            if self.cat_encoder_ is None:
                raise RuntimeError("Categorical encoder has not been
fitted. Call fit() first.")
            X_cat =
self.cat_encoder_.transform(df_transformed[self.cat_cols].fillna(''))

        # --- Сборка итоговой матрицы ---
        matrices = [m for m in [X_text, X_num, X_cat] if m is not
None]

        if any("sparse" in str(type(m)) for m in matrices):
            X_matrix = hstack(matrices).tocsr()
        else:
            X_matrix = np.hstack(matrices)

        print("Transform completed.")

        if return_df:
            return df_transformed, X_matrix
        return X_matrix
```

Применяем пайплайн выше

```
# 1. Инициализируем наш процессор с нужными параметрами
processor = FeatureProcessor(
    mode="tfidf",
    use_structured=True,
    add_features=True,
    encode_categorical=True,
    cat_cols=['location', 'company'],
    cat_encoding="ordinal",
    tfidf_params={"max_features": 15000, "ngram_range": (1, 2),
"min_df": 5}
)

# 2. обучаем процессор
processor.fit(train_df)

# 3. применяем обученный процессор
X_train = processor.transform(train_df)
X_val = processor.transform(val_df)
X_test = processor.transform(test_df)

# Проверяем размерности
print(f"Train matrix shape: {X_train.shape}")
print(f"Validation matrix shape: {X_val.shape}")
print(f"Test matrix shape: {X_test.shape}")
```

```
Fitting FeatureProcessor...
```

```
{"model_id":"032373b1f77e41319d04e53bde00c9ad","version_major":2,"version_minor":0}

Fit completed.
Transforming 12500 samples...

{"model_id":"104d123a2af8462eb5cada2093074be8","version_major":2,"version_minor":0}

Transform completed.
Transforming 4167 samples...

{"model_id":"abd2cdd3a930463ea31f5e9d7cecfa76","version_major":2,"version_minor":0}

Transform completed.
Transforming 5556 samples...

{"model_id":"7fa5d0d4223a476990631941e0fbe640","version_major":2,"version_minor":0}

Transform completed.
Train matrix shape: (12500, 15010)
Validation matrix shape: (4167, 15010)
Test matrix shape: (5556, 15010)
```

Сделаю обертку для модели

```
def train_and_evaluate_ridge(X_train, y_train, X_val, y_val, random_state):
    """
    Обучает модель гребневой регрессии (Ridge regression) и оценивает её качество на валидационном наборе данных
    """

    model = Ridge(random_state=random_state)
    model.fit(X_train, y_train)

    preds = model.predict(X_val)

    r2 = r2_score(y_val, preds)
    print(f"R^2 score for Ridge regression: {r2:.4f}")

    return r2

y_train = train_df['log_salary_from']
y_val = val_df['log_salary_from']

r2_score_ridge = train_and_evaluate_ridge(X_train, y_train, X_val, y_val, RANDOM_STATE)
```

```
leaderboard.loc[len(leaderboard)] = ['TF-IDF + Ridge', r2_score_ridge]
display(leaderboard)

# R^2 score for Ridge regression: 0.7232
# Метод    R2_Score
# 0   TF-IDF + Ridge   0.72324
```

```
R^2 score for Ridge regression: 0.7232
```

```
            Метод  R2_Score
0  TF-IDF + Ridge  0.723238
```

Теперь применяем на тестовую выборку

```
model_tfidf = Ridge(random_state=RANDOM_STATE)
model_tfidf.fit(X_train, y_train)
```

```
Ridge(random_state=42)
```

```
def create_kaggle_submission(model, X_test_data,
output_filename="submission.csv"):

    print("Generating predictions on the test set...")

    preds_test = model.predict(X_test_data)

    print("Creating submission DataFrame...")

    submission = pd.DataFrame({
        "index": range(len(preds_test)),
        "prediction": preds_test
    })


    print("Submission preview:")
    display(submission.head())

    print(f"Saving submission file to '{output_filename}'...")
    submission.to_csv(output_filename, index=False)
    print("Submission file created successfully!")

create_kaggle_submission(model=model_tfidf, X_test_data=X_test)
```

```
Generating predictions on the test set...
Creating submission DataFrame...
Submission preview:
```

```
   index  prediction
0      0    5.478940
1      1    4.024928
2      2    4.288779
```

```
3      3    4.540033
4      4    4.591679

Saving submission file to 'submission.csv'...
Submission file created successfully!
```

Самописная архитектура

Архитектура **SalaryPredictor**

Поток данных через нейронную сеть выглядит следующим образом:

Вход (input_dim) -> Линейный (512) + ReLU + Dropout(0.2) -> Линейный (512) + ReLU + Dropout(0.2) -> Линейный (128) + ReLU + Dropout(0.2) -> Выход (1)

Использую оптимизитор **AdamW**.

```python
class SalaryPredictor(nn.Module):
    """
    Самописная архитектура нейронной сети для предсказания зарплаты.
    Архитектура: Вход -> 512 -> 256 -> 128 -> 1
    """
    def __init__(self, input_dim):
        super(SalaryPredictor, self).__init__()
        # Первый скрытый слой
        self.fc1 = nn.Linear(input_dim, 512)
        self.relu1 = nn.ReLU()
        self.dropout1 = nn.Dropout(0.2)

        # Второй скрытый слой
        self.fc2 = nn.Linear(512, 256)
        self.relu2 = nn.ReLU()
        self.dropout2 = nn.Dropout(0.2)

        # Третий скрытый слой
        self.fc3 = nn.Linear(256, 128)
        self.relu3 = nn.ReLU()
        self.dropout3 = nn.Dropout(0.2)

        # Выходной слой
        self.fc4 = nn.Linear(128, 1)

    def forward(self, x):
        # Проход через первый слой
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.dropout1(x)

        # Проход через второй слой
        x = self.fc2(x)
```

```
        x = self.relu2(x)
        x = self.dropout2(x)

        # Проход через третий слой
        x = self.fc3(x)
        x = self.relu3(x)
        x = self.dropout3(x)

        # Проход через выходной слой
        x = self.fc4(x)

        return x

class TorchTrainer:
    """
    Обертка для обучения и валидации модели PyTorch.
    Использует собственную архитектуру SalaryPredictor.
    """
    def __init__(self, model_class, input_dim, lr=0.0001, epochs=10,
batch_size=64): # <--- Значение lr изменено на 0.0001
        self.model = model_class(input_dim)
        self.criterion = nn.MSELoss()

        # По ходу работы оптимизатор заменен на AdamW; добавлен
weight_decay

        self.optimizer = optim.AdamW(self.model.parameters(), lr=lr,
weight_decay=1e-5)
        self.epochs = epochs
        self.batch_size = batch_size
        self.device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
        self.model.to(self.device)

    def _convert_to_tensor(self, X, y=None):
        """
        Преобразует входные матрицы (в том числе разреженные) в
тензоры PyTorch.
        """
        if issparse(X):
            X_tensor = torch.tensor(X.toarray(), dtype=torch.float32)
        else:
            X_tensor = torch.tensor(X, dtype=torch.float32)

        if y is not None:
            y_tensor = torch.tensor(y.values,
dtype=torch.float32).reshape(-1, 1)
            return TensorDataset(X_tensor, y_tensor)
        else:
            return X_tensor
```

```
    def fit(self, X_train, y_train, X_val, y_val):
        """
        Обучает модель на тренировочных данных и валидирует на
отложенной выборке.
        """
        train_dataset = self._convert_to_tensor(X_train, y_train)
        val_dataset = self._convert_to_tensor(X_val, y_val)

        train_loader = DataLoader(train_dataset,
batch_size=self.batch_size, shuffle=True)
        val_loader = DataLoader(val_dataset,
batch_size=self.batch_size, shuffle=False)

        print(f"Start training on {self.device} for {self.epochs}
epochs.")
        best_r2 = -float('inf')

        for epoch in range(self.epochs):
            self.model.train()
            for batch_X, batch_y in train_loader:
                batch_X, batch_y = batch_X.to(self.device),
batch_y.to(self.device)
                self.optimizer.zero_grad()
                outputs = self.model(batch_X)
                loss = self.criterion(outputs, batch_y)
                loss.backward()
                self.optimizer.step()

            self.model.eval()
            with torch.no_grad():
                val_preds_list = []
                val_labels_list = []
                for batch_X_val, batch_y_val in val_loader:
                    batch_X_val, batch_y_val =
batch_X_val.to(self.device), batch_y_val.to(self.device)
                    outputs = self.model(batch_X_val)
                    val_preds_list.append(outputs.cpu().numpy())
                    val_labels_list.append(batch_y_val.cpu().numpy())

                val_preds = np.concatenate(val_preds_list).flatten()
                val_labels = np.concatenate(val_labels_list).flatten()
                r2 = r2_score(val_labels, val_preds)
                print(f"Epoch {epoch+1}/{self.epochs}, Validation R^2:
{r2:.4f}")

                if r2 > best_r2:
                    best_r2 = r2
                    torch.save(self.model.state_dict(),
'best_model.pth')
```

```
                    print(f"New best R^2 found! Saving model. Best
R^2: {best_r2:.4f}")

        print("\nTraining completed. Loading best model.")
        self.model.load_state_dict(torch.load('best_model.pth'))
        return self

    def predict(self, X_test):
        """
        Делает предсказания на новых данных.
        """
        self.model.eval()
        X_test_tensor = self._convert_to_tensor(X_test)

        test_loader = DataLoader(TensorDataset(X_test_tensor),
batch_size=self.batch_size, shuffle=False)

        predictions = []
        with torch.no_grad():
            for batch_X in test_loader:
                batch_X = batch_X[0].to(self.device)
                outputs = self.model(batch_X)
                predictions.append(outputs.cpu().numpy())

        return np.concatenate(predictions).flatten()
```

Применяем

```
input_dim = X_train.shape[1]
trainer = TorchTrainer(model_class=SalaryPredictor,
input_dim=input_dim, epochs=150)

trainer.fit(X_train, y_train, X_val, y_val)

predictions = trainer.predict(X_test)
```

```
Start training on cuda for 150 epochs.
Epoch 1/150, Validation R^2: -3.8796
New best R^2 found! Saving model. Best R^2: -3.8796
Epoch 2/150, Validation R^2: -3.7280
New best R^2 found! Saving model. Best R^2: -3.7280
Epoch 3/150, Validation R^2: -2.8925
New best R^2 found! Saving model. Best R^2: -2.8925
Epoch 4/150, Validation R^2: -2.7081
New best R^2 found! Saving model. Best R^2: -2.7081
Epoch 5/150, Validation R^2: -2.8985
Epoch 6/150, Validation R^2: -2.5022
New best R^2 found! Saving model. Best R^2: -2.5022
Epoch 7/150, Validation R^2: -1.8941
New best R^2 found! Saving model. Best R^2: -1.8941
```

```
Epoch 8/150, Validation R^2: -1.5159
New best R^2 found! Saving model. Best R^2: -1.5159
Epoch 9/150, Validation R^2: -1.0421
New best R^2 found! Saving model. Best R^2: -1.0421
Epoch 10/150, Validation R^2: -1.6297
Epoch 11/150, Validation R^2: -0.5635
New best R^2 found! Saving model. Best R^2: -0.5635
Epoch 12/150, Validation R^2: -0.5788
Epoch 13/150, Validation R^2: -0.3959
New best R^2 found! Saving model. Best R^2: -0.3959
Epoch 14/150, Validation R^2: -0.0714
New best R^2 found! Saving model. Best R^2: -0.0714
Epoch 15/150, Validation R^2: 0.1668
New best R^2 found! Saving model. Best R^2: 0.1668
Epoch 16/150, Validation R^2: 0.1182
Epoch 17/150, Validation R^2: 0.2920
New best R^2 found! Saving model. Best R^2: 0.2920
Epoch 18/150, Validation R^2: 0.3670
New best R^2 found! Saving model. Best R^2: 0.3670
Epoch 19/150, Validation R^2: 0.4645
New best R^2 found! Saving model. Best R^2: 0.4645
Epoch 20/150, Validation R^2: 0.2251
Epoch 21/150, Validation R^2: 0.2048
Epoch 22/150, Validation R^2: 0.2654
Epoch 23/150, Validation R^2: 0.4300
Epoch 24/150, Validation R^2: 0.4001
Epoch 25/150, Validation R^2: 0.5251
New best R^2 found! Saving model. Best R^2: 0.5251
Epoch 26/150, Validation R^2: 0.3863
Epoch 27/150, Validation R^2: 0.3248
Epoch 28/150, Validation R^2: 0.0980
Epoch 29/150, Validation R^2: 0.4996
Epoch 30/150, Validation R^2: 0.5265
New best R^2 found! Saving model. Best R^2: 0.5265
Epoch 31/150, Validation R^2: 0.5006
Epoch 32/150, Validation R^2: 0.5293
New best R^2 found! Saving model. Best R^2: 0.5293
Epoch 33/150, Validation R^2: 0.5300
New best R^2 found! Saving model. Best R^2: 0.5300
Epoch 34/150, Validation R^2: 0.3656
Epoch 35/150, Validation R^2: 0.6334
New best R^2 found! Saving model. Best R^2: 0.6334
Epoch 36/150, Validation R^2: 0.4788
Epoch 37/150, Validation R^2: 0.5177
Epoch 38/150, Validation R^2: 0.2647
Epoch 39/150, Validation R^2: 0.2768
Epoch 40/150, Validation R^2: 0.6162
Epoch 41/150, Validation R^2: 0.5265
Epoch 42/150, Validation R^2: 0.5600
```

```
Epoch 43/150, Validation R^2: 0.6445
New best R^2 found! Saving model. Best R^2: 0.6445
Epoch 44/150, Validation R^2: 0.5685
Epoch 45/150, Validation R^2: 0.5882
Epoch 46/150, Validation R^2: 0.6093
Epoch 47/150, Validation R^2: 0.5538
Epoch 48/150, Validation R^2: 0.6218
Epoch 49/150, Validation R^2: 0.3003
Epoch 50/150, Validation R^2: 0.4923
Epoch 51/150, Validation R^2: 0.6373
Epoch 52/150, Validation R^2: 0.6323
Epoch 53/150, Validation R^2: 0.6259
Epoch 54/150, Validation R^2: 0.6044
Epoch 55/150, Validation R^2: 0.5882
Epoch 56/150, Validation R^2: 0.6368
Epoch 57/150, Validation R^2: 0.5710
Epoch 58/150, Validation R^2: 0.6134
Epoch 59/150, Validation R^2: 0.5640
Epoch 60/150, Validation R^2: 0.6535
New best R^2 found! Saving model. Best R^2: 0.6535
Epoch 61/150, Validation R^2: 0.6165
Epoch 62/150, Validation R^2: 0.6693
New best R^2 found! Saving model. Best R^2: 0.6693
Epoch 63/150, Validation R^2: 0.5810
Epoch 64/150, Validation R^2: 0.6306
Epoch 65/150, Validation R^2: 0.6486
Epoch 66/150, Validation R^2: 0.6159
Epoch 67/150, Validation R^2: 0.5212
Epoch 68/150, Validation R^2: 0.6687
Epoch 69/150, Validation R^2: 0.6023
Epoch 70/150, Validation R^2: 0.6629
Epoch 71/150, Validation R^2: 0.5122
Epoch 72/150, Validation R^2: 0.6515
Epoch 73/150, Validation R^2: 0.4831
Epoch 74/150, Validation R^2: 0.6642
Epoch 75/150, Validation R^2: 0.4949
Epoch 76/150, Validation R^2: 0.6596
Epoch 77/150, Validation R^2: 0.5945
Epoch 78/150, Validation R^2: 0.6601
Epoch 79/150, Validation R^2: 0.6502
Epoch 80/150, Validation R^2: 0.6176
Epoch 81/150, Validation R^2: 0.6433
Epoch 82/150, Validation R^2: 0.4696
Epoch 83/150, Validation R^2: 0.6359
Epoch 84/150, Validation R^2: 0.6472
Epoch 85/150, Validation R^2: 0.6485
Epoch 86/150, Validation R^2: 0.6147
Epoch 87/150, Validation R^2: 0.6585
Epoch 88/150, Validation R^2: 0.6762
```

```
New best R^2 found! Saving model. Best R^2: 0.6762
Epoch 89/150, Validation R^2: 0.6233
Epoch 90/150, Validation R^2: 0.6660
Epoch 91/150, Validation R^2: 0.6004
Epoch 92/150, Validation R^2: 0.6803
New best R^2 found! Saving model. Best R^2: 0.6803
Epoch 93/150, Validation R^2: 0.6359
Epoch 94/150, Validation R^2: 0.5618
Epoch 95/150, Validation R^2: 0.5975
Epoch 96/150, Validation R^2: 0.6348
Epoch 97/150, Validation R^2: 0.6169
Epoch 98/150, Validation R^2: 0.6421
Epoch 99/150, Validation R^2: 0.6787
Epoch 100/150, Validation R^2: 0.6565
Epoch 101/150, Validation R^2: 0.6328
Epoch 102/150, Validation R^2: 0.6661
Epoch 103/150, Validation R^2: 0.5257
Epoch 104/150, Validation R^2: 0.6850
New best R^2 found! Saving model. Best R^2: 0.6850
Epoch 105/150, Validation R^2: 0.5906
Epoch 106/150, Validation R^2: 0.6754
Epoch 107/150, Validation R^2: 0.6231
Epoch 108/150, Validation R^2: 0.6775
Epoch 109/150, Validation R^2: 0.5865
Epoch 110/150, Validation R^2: 0.6649
Epoch 111/150, Validation R^2: 0.6200
Epoch 112/150, Validation R^2: 0.6766
Epoch 113/150, Validation R^2: 0.6113
Epoch 114/150, Validation R^2: 0.6357
Epoch 115/150, Validation R^2: 0.6915
New best R^2 found! Saving model. Best R^2: 0.6915
Epoch 116/150, Validation R^2: 0.6826
Epoch 117/150, Validation R^2: 0.6213
Epoch 118/150, Validation R^2: 0.5965
Epoch 119/150, Validation R^2: 0.6636
Epoch 120/150, Validation R^2: 0.6587
Epoch 121/150, Validation R^2: 0.6906
Epoch 122/150, Validation R^2: 0.6457
Epoch 123/150, Validation R^2: 0.5232
Epoch 124/150, Validation R^2: 0.6792
Epoch 125/150, Validation R^2: 0.6854
Epoch 126/150, Validation R^2: 0.3849
Epoch 127/150, Validation R^2: 0.6845
Epoch 128/150, Validation R^2: 0.6528
Epoch 129/150, Validation R^2: 0.5739
Epoch 130/150, Validation R^2: 0.6001
Epoch 131/150, Validation R^2: 0.6056
Epoch 132/150, Validation R^2: 0.6688
Epoch 133/150, Validation R^2: 0.6996
```

```
New best R^2 found! Saving model. Best R^2: 0.6996
Epoch 134/150, Validation R^2: 0.6448
Epoch 135/150, Validation R^2: 0.6231
Epoch 136/150, Validation R^2: 0.4929
Epoch 137/150, Validation R^2: 0.6928
Epoch 138/150, Validation R^2: 0.6674
Epoch 139/150, Validation R^2: 0.6359
Epoch 140/150, Validation R^2: 0.6604
Epoch 141/150, Validation R^2: 0.6497
Epoch 142/150, Validation R^2: 0.5770
Epoch 143/150, Validation R^2: 0.6000
Epoch 144/150, Validation R^2: 0.6558
Epoch 145/150, Validation R^2: 0.6168
Epoch 146/150, Validation R^2: 0.5380
Epoch 147/150, Validation R^2: 0.6682
Epoch 148/150, Validation R^2: 0.6243
Epoch 149/150, Validation R^2: 0.6708
Epoch 150/150, Validation R^2: 0.6477

Training completed. Loading best model.
```

```
submission = pd.DataFrame({
    "index": range(len(predictions)),
    "prediction": predictions
})
submission.to_csv("submission.csv", index=False)
print("Submission file 'submission.csv' created successfully!")
```

```
Submission file 'submission.csv' created successfully!
```