Додаток А

Файл - Analizator.py

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import pymorphy2
import nltk

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report,
ConfusionMatrixDisplay, RocCurveDisplay
from collections import Counter
from tqdm import tqdm
from nltk.corpus import stopwords

nltk.download('stopwords')

toxic_comments = pd.read_csv('labeled.csv')
toxic_comments.head()
toxic_comments.info()
toxic_comments['toxic'].value_counts().plot(kind='pie',
title='Class distribution  "toxic"', autopct='%1.1f%%')
plt.axis('off')
plt.show()

TOKEN_RE = re.compile(r'[а-яё]+')
russian_stopwords = stopwords.words("russian")
lemmatizer = pymorphy2.MorphAnalyzer()


def tokenize_text(txt, min_length_token=2):
    txt = txt.lower()
    all_tokens = TOKEN_RE.findall(txt)
    return [token for token in all_tokens if len(token) >=
min_length_token]


def remove_stopwords(tokens):
    return list(filter(lambda token: token not in
russian_stopwords, tokens))


def lemmatizing(tokens):
    return [lemmatizer.parse(token)[0].normal_form for token in
tokens]
```

```
def text_cleaning(txt):
    tokens = tokenize_text(txt)
    tokens = lemmatizing(tokens)
    tokens = remove_stopwords(tokens)
    return ' '.join(tokens)


tqdm.pandas()

df_token = toxic_comments.copy()
df_token['comment'] =
df_token['comment'].progress_apply(text_cleaning)
df_token.head()

df = df_token.copy()
empty = df[df['comment'] == '']
print('Number of empty texts: ', len(empty))
df = df.drop(empty.index)

print('Number of duplicates:', df.duplicated().sum())

df = df.drop_duplicates()

print('Number of duplicate comments: ',
df['comment'].duplicated().sum())

comment_duplicated = df[df['comment'].duplicated('last')]

df = df.drop_duplicates(subset='comment')

zero_labels = [1084, 1198, 1250, 1394, 1456, 1586, 1631, 1637,
1659, 1693, 1703, 1739, 1781, 1814, 1820, 1877, 4256, 6044]
for row in comment_duplicated.iterrows():
    comment = row[1]['comment']
    idx = df[df['comment'] == comment].index
    if idx in zero_labels:
        label = 0.
    else:
        label = 1.
    df.loc[idx, 'toxic'] = label

print('Number of duplicates:', df.duplicated('comment').sum())

clean_text = df.copy()
clean_text.head()

# Counting the occurrence of each word in the corpus
corpus = clean_text['comment'].values

text = ' '.join(corpus)
counter = Counter(text.split())
sorted_counter = counter.most_common()

df_train, df_test = train_test_split(clean_text,
```

```
                                        random_state=311,
                                        test_size=0.33,
                                        stratify=clean_text['toxic']
                                        )

train_corpus = df_train['comment'].values
test_corpus = df_test['comment'].values

y_train = df_train['toxic']
y_test = df_test['toxic']
vectorizer = TfidfVectorizer(ngram_range=(2, 4),
analyzer='char_wb', max_df=0.95, min_df=10)
X_train = vectorizer.fit_transform(train_corpus)
X_test = vectorizer.transform(test_corpus)

print('Total features: ', len(vectorizer.get_feature_names_out()))

Logregres = LogisticRegression(max_iter=10000, C=3,
solver='liblinear')
Logregres.fit(X_train, y_train)
y_pred = Logregres.predict(X_test)
print(classification_report(y_test, y_pred))

ConfusionMatrixDisplay.from_estimator(Logregres, X_test, y_test,
cmap='binary')
plt.show()

RocCurveDisplay.from_estimator(Logregres, X_test, y_test)
plt.show()

# message = "Тестовый текст"
# print(message)
# clean_message = text_cleaning(message)
# X_example = vectorizer.transform([clean_message])
# toxic_propability = Logregres.predict_proba(X_example)[0, 1]
# print(f'Вероятность токсичности: {toxic_propability:.2f}')

# message_input = "Тестовый текст"
# clean_message = text_cleaning(message_input)
# X_example = vectorizer.transform([clean_message])
# toxic_propability = Logregres.predict_proba(X_example)[0, 1]
# print(f'Вероятность токсичности: {toxic_propability:.2f}')
```

Файл - Database.py

```
import sqlite3 as sq


def sql_start():
    global base, cur
    base = sq.connect('db.db')
    cur = base.cursor()
```

```
    if base:
        print('Data base connected OK')
    base.execute(
        'CREATE TABLE IF NOT EXISTS Users ( id_user  BIGINT
PRIMARY KEY, username CHAR);')
    base.execute(
        'CREATE TABLE IF NOT EXISTS [Groups] ( id_group BIGINT
PRIMARY KEY, State_bot BOOLEAN, Toxicity_level DOUBLE '
        ' DEFAULT (0.7) );')
    base.execute(
        'CREATE TABLE IF NOT EXISTS G_U ( id_user  BIGINT
REFERENCES Users (id_user),id_group BIGINT REFERENCES ['
        'Groups] (id_group), PRIMARY KEY ( id_user,
id_group ) );')
    base.commit()
    print("Good")


def sql_group_update(data):
    global base, cur
    base = sq.connect('db.db')
    cur = base.cursor()
    if base:
        print('Data base connected OK')
    sql = '''INSERT or REPLACE INTO Groups (id_group, State_bot)
VALUES  (?,?)'''
    cur = base.cursor()
    cur.execute(sql, data)
    base.commit()


def sql_select_group_run():
    global base, cur
    base = sq.connect('db.db')
    cur = base.cursor()
    cur.execute("Select * From Groups WHERE State_bot = 1")
    rows = cur.fetchall()

    for row in rows:
        print(row)
    base.commit()


def sql_select_group(id_group):
    global base, cur
    base = sq.connect('db.db')
    cur = base.cursor()
    cur.execute("Select state_bot From Groups where id_group = ?",
(id_group,))
    rows = cur.fetchall()
    base.commit()
    if rows == [(1,)]:
        state = True
        return state
```

```
    else:
        state = False
        return state


def set_toxicity_level(id_group, toxic_level):
    global base, cur
    base = sq.connect('db.db')
    cur = base.cursor()
    cur.execute("UPDATE Groups SET Toxicity_level = ? WHERE
id_group = ?", (toxic_level, id_group))
    base.commit()


def get_toxicity_level(id_group):
    global base, cur
    base = sq.connect('db.db')
    cur = base.cursor()
    cur.execute("Select Toxicity_level From Groups WHERE id_group
= ?", (id_group,))
    base.commit()
    toxic_level = cur.fetchall()
    return toxic_level


def adding_a_user_to_statistics():
    return
```

Файл - main.py

```
import telebot
from auth_data import token
import database
import analizator


def telegram_bot(token):
    bot = telebot.TeleBot(token)

    # check that the user is the admin of the group
    def check_is_admin(client, user_id: int, chat_id: int) ->
bool:
        for admin in client.get_chat_administrators(chat_id):
            if admin.user.id == user_id:
                return True
        return False

    # check the message is not from the group
    def check_the_message_is_not_from_the_group(message):
        if message.chat.type != 'group' and message.chat.type !=
'supergroup':
            return True
```

```
        return False

    @bot.message_handler(commands=["start"])
    def start_message(message):
        bot.send_message(message.chat.id, "Toxic_bot к вашим
услугам!\n"
                                          "Добавте бота в группу и
не забудьте выдать ему права администратора\n"
                                          "Он поможет вам
избавиться от токсичных сообщений\n"
                                          "Бот с условной
вероятностью от 0 до 1 определяет токсично ли сообщение
пользователя\n")
        bot.send_message(message.chat.id, "Команды для управления
ботом:\n"
                                         "/toxic_on - включить
анализ сообщений пользователей на токсичность в группе\n"
                                         "/toxic_setup -
установить уровень токсичности при котором сообщения будут
удаляться\n"
                                         "/toxic_off - отключить
анализ сообщений")


    @bot.message_handler(commands=["toxic_on"])
    def toxic_on(message):
        if check_the_message_is_not_from_the_group(message):
            bot.send_message(message.chat.id, "Команда работает
только в группах")
        else:
            if not check_is_admin(bot, message.from_user.id,
message.chat.id):
                bot.send_message(message.chat.id,
f'@{message.from_user.username} вы не админ!')
                return
            id_group = message.chat.id
            status_group = True
            data = (id_group, status_group)
            database.sql_group_update(data)
            bot.send_message(message.chat.id, "Бот запущен")
            database.sql_select_group_run()

    @bot.message_handler(commands=["toxic_off"])
    def toxic_off(message):
        if check_the_message_is_not_from_the_group(message):
            bot.send_message(message.chat.id, "Команда работает
только в группах")
        else:
            if not check_is_admin(bot, message.from_user.id,
message.chat.id):
                bot.send_message(message.chat.id,
f'@{message.from_user.username} вы не админ!')
                return
            id_group = message.chat.id
```

```
            status_group = False
            data = (id_group, status_group)
            database.sql_group_update(data)
            bot.send_message(message.chat.id, "Бот отключен")

    @bot.message_handler(commands=["toxic_setup"])
    def toxic_setup_message(message):
        if check_the_message_is_not_from_the_group(message):
            bot.send_message(message.chat.id, "Команда работает
только в группах")
        else:
            if not check_is_admin(bot, message.from_user.id,
message.chat.id):
                bot.send_message(message.chat.id,
f'@{message.from_user.username} вы не админ!')
                return
            id_group = message.chat.id
            bot.send_message(message.chat.id,
                             "Введите значение уровня при котором
будет реагировать бот на сообщения от 0.01 до 1.00, "
                             "по умолчанию 0.7")
            bot.register_next_step_handler(message, plus,
id_group)

    def plus(message, id_group):
        toxic_level = message.text
        try:
            if 0.0 <= float(toxic_level) <= 1.0:
                database.set_toxicity_level(id_group,
float(toxic_level))
                bot.send_message(message.chat.id, "Новый уровень
установлен")
            else:
                bot.send_message(message.chat.id, "Неверный
диапазон")
        except:
            bot.send_message(message.chat.id, "Неверный ввод")

    @bot.message_handler(content_types=["text"])
    def send_text(message):
        #
bot.send_message(message.chat.id,f"{datetime.now().strftime('%Y-
%m-%d %H:%M')}\n Сообщение получено\n")

        id_group = message.chat.id
        if database.sql_select_group(id_group):
            toxic_level = database.get_toxicity_level(id_group)
            message_input = message.text
            clean_message =
analizator.text_cleaning(message_input)
            X_example =
analizator.vectorizer.transform([clean_message])
            toxic_propability =
analizator.Logregres.predict_proba(X_example)[0, 1]
```

```
            print(f'Вероятность токсичности:
{toxic_propability:.2f}')
            if toxic_level <= toxic_propability:
                bot.delete_message(message.chat.id,
message.message_id)
                # bot.send_message(message.chat.id, f'Вероятность
токсичности: {toxic_propability:.2f}')
                bot.send_message(message.chat.id, f'Сообщение
пользователя {message.from_user.id} удалено так как вероятно
токсично на {toxic_propability:.2f} %')

    bot.polling()


if __name__ == '__main__':
    database.sql_start()
    telegram_bot(token)
```