Федеральное государственное автономное

образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет» (ПГНИУ)

Региональный институт непрерывного образования (РИНО ПГНИУ)

Цифровая кафедра

Выпускная аттестационная (квалификационная) работа

по курсу профессиональной переподготовки

по курсу «Технологии машинного обучения и нейросети для решения прикладных задач»

**ОПРЕДЕЛЕНИЕ ДЕПРЕССИИ У СТУДЕНТОВ**

Разработчики проекта:

Агишев Дмитрий,

Колтырина Елена

Пермь, 2025

ОГЛАВЛЕНИЕ

[ПАСПОРТ ПРОЕКТА 3](#_Toc185847573)

[СОДЕРЖАНИЕ ПРОЕКТА 4](#_Toc185847574)

[Анализ проблемы исследования 4](#_Toc185847575)

[Реализация проекта 9](#_Toc185847576)

[Этап 1. Подготовка данных к анализу 9](#_Toc185847577)

[Этап 2. Корреляционный анализ 11](#_Toc185847578)

[Этап 3. Моделирование и прогнозирование 15](#_Toc185847579)

[Построение модели прогнозирования для станка 1 15](#_Toc185847580)

[Построение модели прогнозирования для станка 2 16](#_Toc185847581)

[Заключение 19](#_Toc185847582)

[Список использованных источников и литературы 20](#_Toc185847583)

[Приложения 22](#_Toc185847584)

# ПАСПОРТ ПРОЕКТА

**Название проекта:**определение депрессии у студентов

**Сведения об авторах:**Агишев Дмитрий, Колтырина Елена

**Цель:**создать нейросетевую систему на основе анализа социальных, академических, профессиональных и поведенческих факторов студентов с целью раннего обнаружения признаков депрессии и оказания своевременной поддержки.

**Задачи:**

1. Выполнить анализ проблемы, обосновать ее актуальность.
2. Осуществить загрузку данных и подготовку их к анализу.
3. Исследовать собранные данные для выявления закономерностей и зависимостей между факторами и наличием депрессии.
4. Выбрать и создать релевантные признаки для модели классификации, которые наиболее сильно провоцируют депрессию.
5. Разработать и обучить модель классификации, предварительно выбрав наилучшую.
6. Выполнить интерпретацию полученных результатов и сделать выводы о достижении цели.

**Краткое описание проекта:**

Требуется проанализировать данные о признаках депрессии у студентов и определить, существует ли зависимость образования депрессии от имеющихся в наборе данных факторных переменных.

Протестировать разные алгоритмы для нахождения наиболее эффективного, а также применить техники для предотвращения переобучения. Дать интерпретацию полученным результатам. Сделать выводы.

**Конкретные ожидаемые результаты:**

Построенная модель классификации и рекомендации по ее использованию.

# СОДЕРЖАНИЕ ПРОЕКТА

## Анализ проблемы исследования

В наши дни психическое состояние студентов становится все более уязвимым. Высокие требования к академической успеваемости, необходимость совмещения учебы и работы, финансовые трудности и социальное давление создают значительные психоэмоциональные нагрузки. Недостаток сна, несбалансированное питание и отсутствие поддержки со стороны окружающих усугубляют ситуацию, что может привести к депрессивным расстройствам, снижению качества жизни и даже к суицидальным мыслям.

Сложность проблемы заключается в том, что многие студенты либо не осознают своего состояния, либо не хотят обращаться за профессиональной помощью из-за страха осуждения или недоверия к врачам. Традиционные методы диагностики депрессии требуют очного участия специалистов, что не всегда возможно в условиях плотного учебного и рабочего графика.

Исследования показывают, что женщины чаще сталкиваются с депрессивными расстройствами, чем мужчины. Это связано как с биологическими особенностями (гормоны), так и с социальными факторами (эмоциональная вовлеченность и восприимчивость к стрессу). Возраст также играет значительную роль, так как молодые люди, особенно студенты (18–25 лет), подвержены повышенному уровню стресса из-за неопределенности в будущем и не до конца сформированной личности.

В мегаполисах, как правило, выше уровень конкуренции, финансовые расходы и социальная изоляция, что может усугублять уровень тревожности и депрессии. В небольших городах, напротив, могут возникать другие факторы стресса, такие как недостаток возможностей для самореализации или замкнутость.

Одним из ключевых факторов, влияющих на психическое здоровье студентов, является академическая нагрузка. Современная система образования предъявляет высокие требования к успеваемости, а постоянные дедлайны, экзамены и конкуренция среди одногруппников становятся причиной хронического стресса. Если студент испытывает чрезмерное давление в учебном процессе, у него могут развиться тревожные расстройства и симптомы депрессии.

Низкие оценки могут снижать самооценку и вызывать чувство неуверенности, тогда как стремление к высоким результатам нередко приводит к перфекционизму, выгоранию и тревожности. Важно учитывать и удовлетворенность учебным процессом – если студент ощущает комфорт и мотивацию к обучению, уровень стресса снижается, что благоприятно сказывается на его эмоциональном состоянии.

Не менее значимым фактором является совмещение учебы и работы. Рабочая нагрузка оказывает серьезное влияние на психическое здоровье, поскольку вынужденное совмещение учебы и работы сокращает время на отдых, ухудшает концентрацию и повышает уровень стресса. Вдобавок к этому, уровень стресса увеличивается, если человек испытывает дискомфорт и неудовлетворенность работой.

Физическое состояние играет не менее важную роль в развитии депрессии. Например, продолжительность сна напрямую связана с эмоциональным благополучием. Недостаток сна приводит к раздражительности, повышенной утомляемости и эмоциональной нестабильности. Неправильное питание может способствовать снижению уровня серотонина и дофамина – нейромедиаторов, отвечающих за настроение.

Одним из важнейших факторов риска является семейная предрасположенность к психическим расстройствам. Если у родственников студента были диагностированы депрессивные расстройства или другие психические заболевания, вероятность появления подобных состояний возрастает.

Также важно уточнить наличие суицидальных мыслей у студента. Люди, у которых возникают подобные мысли, чаще всего испытывают тяжелые формы депрессии и нуждаются в немедленной психологической помощи.

Использование искусственного интеллекта и машинного обучения в психологии помогает выявлять признаки депрессии на ранней стадии. Анализ таких факторов, как уровень учебной и рабочей нагрузки, качество сна, удовлетворенность учебой и работой, финансовый стресс и семейная предрасположенность, позволяет создать модели, прогнозирующие риски развития депрессивных состояний.

Цель исследования - разработка точной модели, которая позволяет определять наличие депрессии у студентов. Данная модель поможет выявить депрессивные признаки на ранней стадии и оказать своевременную психологическую помощь.

Университеты и колледжи смогут использовать данную систему в качестве инструмента для мониторинга психического состояния студентов. Это поможет администрациям своевременно реагировать на проблемы, разрабатывать стратегии по улучшению образовательной среды и снижению уровня стресса среди учащихся. Внедрение нейросетевых технологий в сферу психологии расширяет возможности цифровой медицины. Разработанная система может быть адаптирована для использования в других группах населения и интегрирована в приложения для психического здоровья.

Исходные данные

В работе анализируется список факторов, которые могут оказывать влияние на образование депрессивного состояния у студентов.

Список колонок анализируемого набора данных:

1. **Gender –** пол студента
2. **Age –** возраст студента
3. **City –** город проживания
4. **Profession –** профессия
5. **Academic Pressure –** учебная нагрузка
6. **Work Pressure –** рабочая нагрузка
7. **CGPA –** средний балл успеваемости
8. **Study Satisfaction –** удовлетворенность учебой
9. **Job Satisfaction –** удовлетворенность работой
10. **Sleep Duration –** продолжительность сна
11. **Dietary Habits –** качество питания
12. **Degree –** ученая степень
13. **Have you ever had suicidal thoughts ? –** присутствие суицидальных мыслей
14. **Work/Study Hours –** часы работы/учебы
15. **Financial Stress –** финансовая нагрузка
16. **Family History of Mental Illness –** генетическая предрасположенность

**Гипотеза:** предполагается, что если создать нейросетевую модель определения депрессии, это поспособствует раннему выявлению депрессивных состояний у студентов и снижению их негативных последствий.

## Реализация проекта

### Этап 1. Подготовка данных к анализу

Подключаем необходимые библиотеки.

# Стандартные библиотеки

import os

import re

import time

import warnings

from datetime import datetime

# Для данных

import kagglehub

import matplotlib

import matplotlib.pyplot as plt

import numpy as np

import opendatasets as od

import pandas as pd

import seaborn as sns

from tqdm import tqdm

import pexpect

# Машинное обучение и обработка данных

from sklearn.decomposition import PCA

from sklearn.ensemble import (AdaBoostClassifier, BaggingClassifier,

GradientBoostingClassifier,

HistGradientBoostingClassifier,

RandomForestClassifier)

from sklearn.feature\_selection import SelectKBest, f\_classif

from sklearn.impute import SimpleImputer

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import (accuracy\_score, auc, classification\_report,

confusion\_matrix, f1\_score, precision\_score,

recall\_score, roc\_auc\_score, roc\_curve)

from sklearn.model\_selection import KFold, cross\_val\_score, learning\_curve, train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from lightgbm import LGBMClassifier

from xgboost import XGBClassifier

# Визуализация и интерфейсы

import io

import joblib

from contextlib import redirect\_stdout

import tkinter as tk

from matplotlib.animation import FuncAnimation

from tkinter import ttk, messagebox

from IPython.display import HTML, clear\_output, display

from ipywidgets import Dropdown, FloatSlider, IntSlider, RadioButtons, interact, interactive, widgets

Загрузка датасета

# Запуск процесса загрузки

child = pexpect.spawn('python -c "import opendatasets as od; od.download(\'https://www.kaggle.com/datasets/hopesb/student-depression-dataset\', force=True )"')

# Ожидаем первый запрос на логин и автоматически вводим

child.expect('Your Kaggle username:')

child.sendline('dmytrt')

# Ожидаем запрос пароля и вводим ключ

child.expect('Your Kaggle Key:')

child.sendline('17165469')

# Ждем завершения процесса

child.expect(pexpect.EOF)

# Загрузка данных и вывод первых 5 строк набора данных

dep\_df = pd.read\_csv("student-depression-dataset/Student Depression Dataset.csv")

dep\_df.head()

На рисунке 1 представлены данныео датасета.



Рисунок 1 –Данные датасета

Размер датасета составил 27901 строку

# Распределение случаев депрессии

plt.figure(figsize=(10, 6))

sns.countplot(

x='Depression',

hue='Depression',

data=dep\_df,

palette='viridis',

legend=True

)

plt.title('Распределение случаев депрессии')

plt.show()

На рисунке 2 представлено распределение случаев депрессии по датасету.

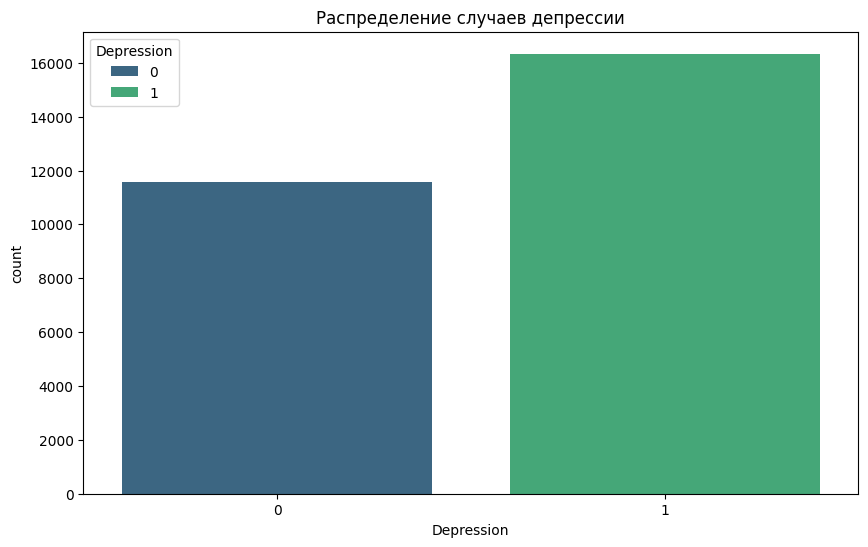


Рисунок 2 –Распространение случаев депрессии

Можем наблюдать, что данные в датасете практически сбалансированы.

Предобработка и визуализация данных

print("✏️ Создание гистограмм для сравнения распределения категориальных признаков по значениям целевой переменной Depression\n")

# Настройки

sns.set\_style('darkgrid')

plt.rcParams.update({

'font.size': 12,

'figure.figsize': (12, 8),

'figure.facecolor': 'white'

})

# Предобработка данных

dep\_df = dep\_df.drop(['City'], axis=1)

def convert\_sleep\_duration(sleep\_duration):

try:

cleaned = re.sub(r'[^\d-]', '', str(sleep\_duration))

matches = re.findall(r'\d+', cleaned)

if len(matches) == 2:

return (int(matches[0]) + int(matches[1])) / 2

return int(matches[0]) if matches else np.nan

except:

return np.nan

dep\_df['Sleep Duration'] = dep\_df['Sleep Duration'].apply(convert\_sleep\_duration)

dep\_df['Sleep Duration'] = dep\_df['Sleep Duration'].fillna(dep\_df['Sleep Duration'].median())

#numeric\_cols = ['Age', 'Academic Pressure', 'CGPA', 'Financial Stress', 'Sleep Duration']

numeric\_cols =['Age', 'Academic Pressure', 'CGPA', 'Financial Stress', 'Sleep Duration']

dep\_df[numeric\_cols] = SimpleImputer(strategy='median').fit\_transform(dep\_df[numeric\_cols])

categorical\_cols = ['Gender', 'Profession', 'Dietary Habits', 'Degree',

'Have you ever had suicidal thoughts ?', 'Family History of Mental Illness']

# Создание гистограмм для сравнения распределения категориальных признаков по значениям целевой переменной Depression

for column in categorical\_cols:

plt.figure(figsize=(8, 6))

ax = sns.countplot(x=dep\_df[column], hue='Depression', data=dep\_df)

plt.title(f'{column} vs Depression')

plt.xlabel(column)

plt.ylabel('Count')

plt.xticks(rotation=90, ha='right')

plt.tight\_layout()

plt.show()

dep\_df = pd.get\_dummies(dep\_df, columns=categorical\_cols, drop\_first=True)

dep\_df = dep\_df.drop(['id'], axis=1)

# Разделение данных

X = dep\_df.drop(['Depression'], axis=1)

y = dep\_df['Depression']

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.15, random\_state=42, stratify=y)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.15, random\_state=42, stratify=y\_temp)

scaler = StandardScaler()

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

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

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

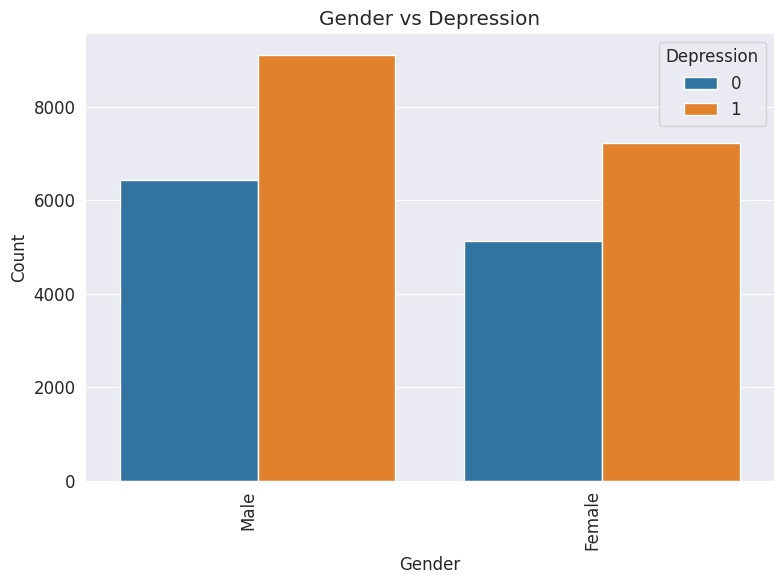
На рисунке 3 представлены данные о распределении депрессии среди мужчин и женщин в датасете. 

Рисунок 3 –Распределение депрессии по полу

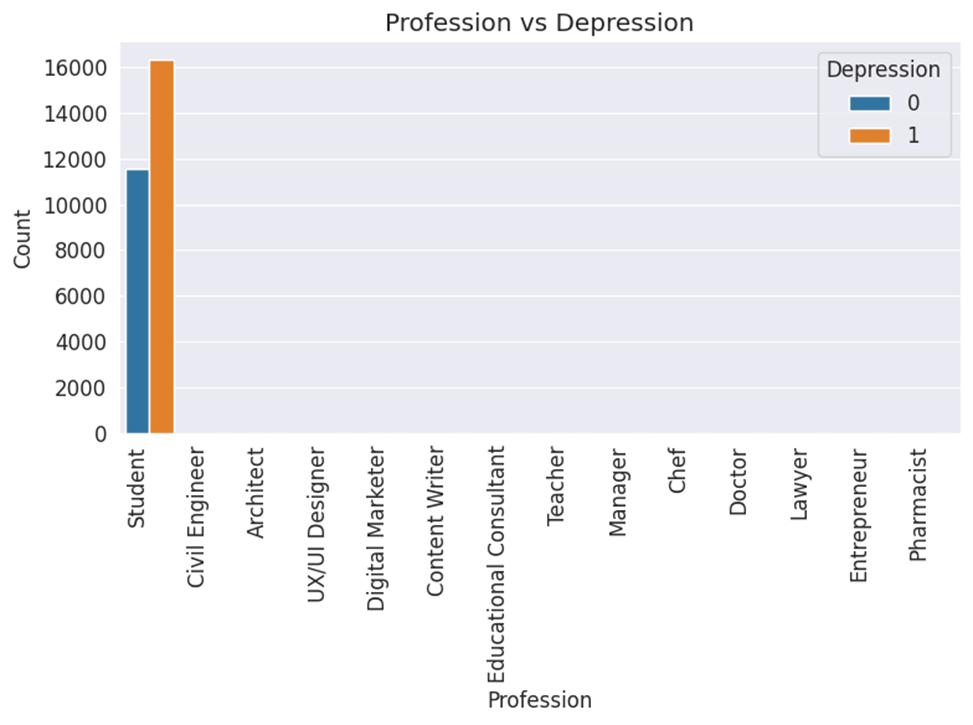
На рисунке 4 представлены данные о распределении депрессии по профессии в датасете. 

Рисунок 4 –Распределение депрессии по профессии

# Создание матрицы диаграмм рассеивания для анализа взаимосвязи между числовыми признаками и целевой переменной Depression

print("✏️ Создание матрицы диаграмм рассеивания для анализа взаимосвязи между числовыми признаками и целевой переменной Depression\n")

g = sns.pairplot(dep\_df[['Age', 'Academic Pressure', 'CGPA', 'Financial Stress', 'Sleep Duration', 'Depression']], hue='Depression', palette='coolwarm')

g.fig.suptitle("Матрица рассеивания между числовыми признаками и Депрессией", y=1.02)

plt.show()

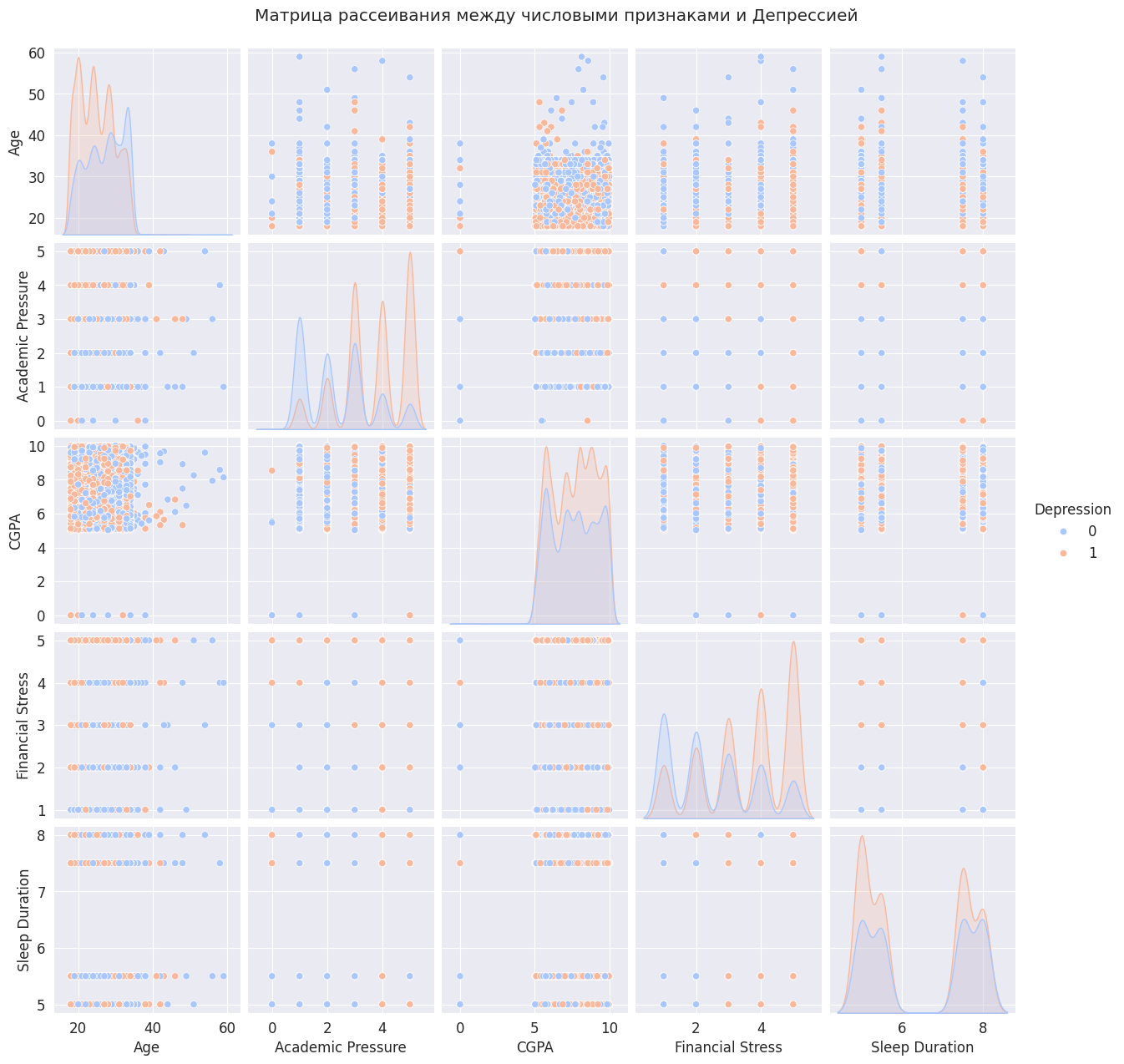
На рисунке 5 представлена диаграмма рассеивания между числовыми признаками по целевой переменной

Рисунок 5 –Диаграмма рассеивания

# Матрица корреляции

print("✏️ Создание матрицы корреляции\n")

plt.figure(figsize=(14, 12))

corr = dep\_df.corr()

sns.heatmap(corr, annot=False, cmap='coolwarm')

plt.title('Матрица корреляции')

plt.show()

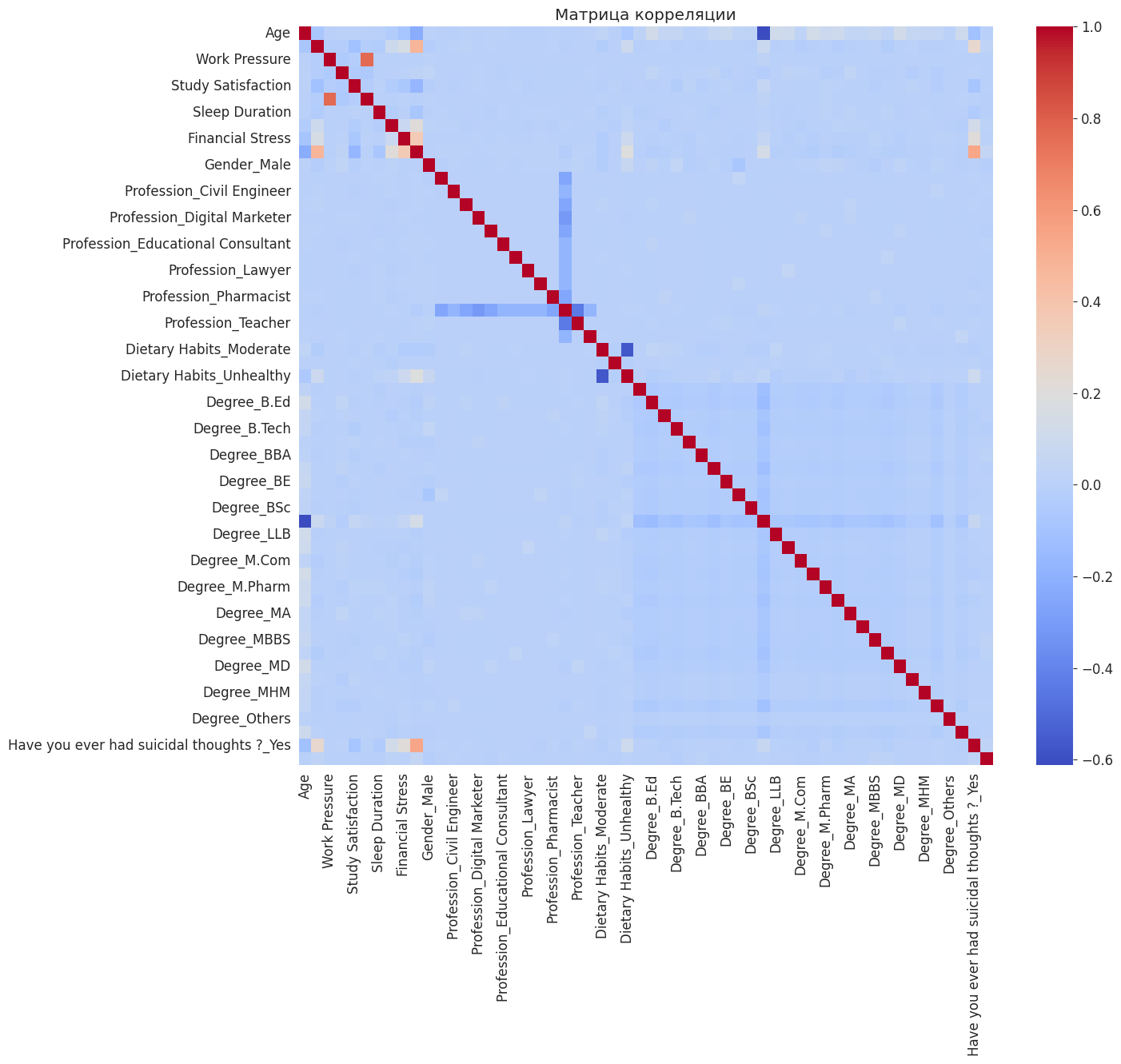
На рисунке 6 представлена матрица корреляции. 

Рисунок 6 –Матрица корреляции

# Анализ важности признаков

print("✏️ Создание даграммы важности признаков для определения наличия депрессии\n")

selector = SelectKBest(f\_classif, k=15)

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

selected\_features = X.columns[selector.get\_support()]

plt.figure(figsize=(12, 8))

pd.Series(selector.scores\_, index=X.columns).nlargest(15).plot(kind='barh', color='skyblue')

plt.title('Влияние признаков на наличие депрессии')

plt.show()

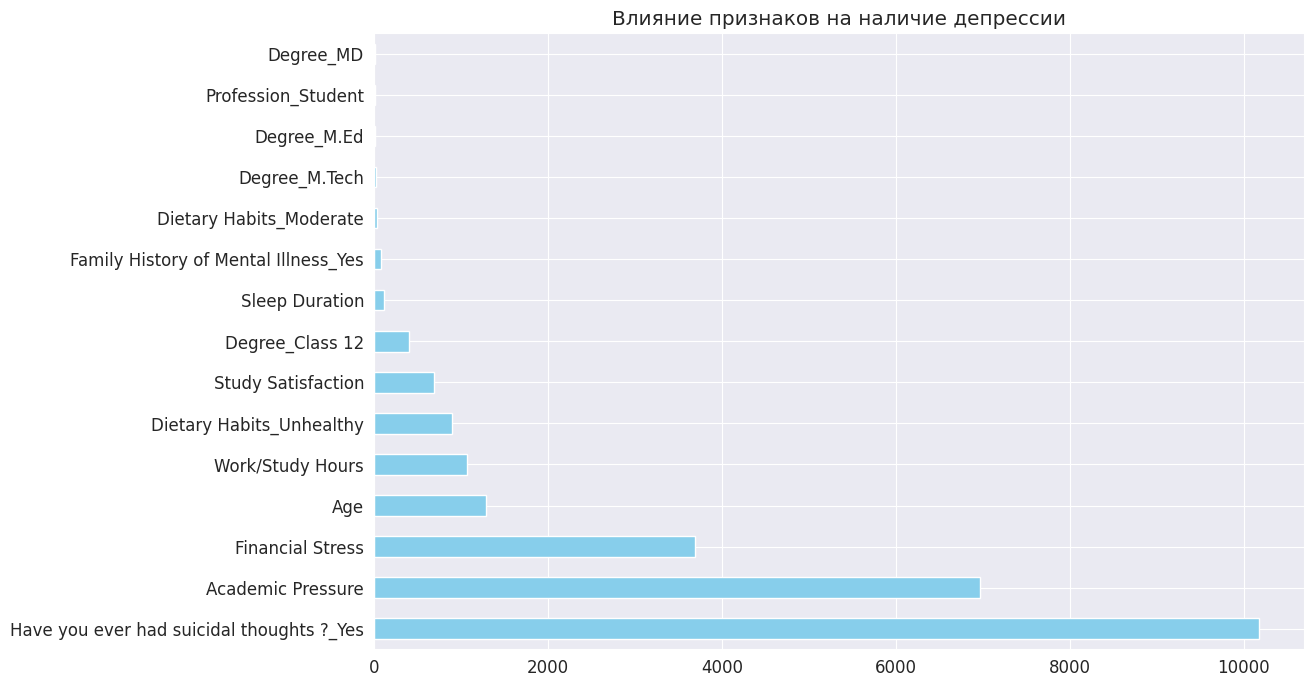
На рисунке 7 представлено влияние признаков на наличие депрессии. 

Рисунок 7 –Влияние признаков на наличие депрессии

### Этап 2. Сравнение модели

# Класс для прогресса

class ProgressAnimator:

def \_\_init\_\_(self, model\_name, total=100):

self.model\_name = model\_name

self.total = total

self.start\_time = datetime.now()

self.last\_progress = 0

def get\_color(self, progress):

r = int(255 \* (1 - progress/100))

g = int(255 \* (progress/100))

return f"\033[38;2;{r};{g};0m"

def update(self, current, total\_epochs):

progress = min(100, int(100 \* (current + 1) / self.total))

if progress == self.last\_progress:

return

self.last\_progress = progress

color = self.get\_color(progress)

elapsed = datetime.now() - self.start\_time

bar = '█' \* int(progress / 2) + ' ' \* (50 - int(progress / 2))

epoch\_str = f'Epochs: {current+1}/{total\_epochs}'

time\_str = f' | Time: {elapsed.seconds}s'

print(f'\r{color}➤ {self.model\_name:25} [{bar}] {progress}% | {epoch\_str}{time\_str}', end='')

# Обучение моделей

models = {

"Logistic Regression": LogisticRegression(max\_iter=1000, random\_state=42),

"Random Forest": RandomForestClassifier(n\_estimators=200, max\_depth=10, random\_state=42),

"SVM": SVC(probability=True, random\_state=42, class\_weight='balanced'),

"KNN": KNeighborsClassifier(n\_neighbors=7),

"XGBoost": XGBClassifier(n\_estimators=150, eval\_metric='logloss', random\_state=42),

"LightGBM": LGBMClassifier(num\_leaves=31, random\_state=42, verbose=-1)

}

trained\_models = {}

results = []

confusion\_matrices = []

print("\n👾 Начало обучения моделей:")

for name, model in models.items():

animator = ProgressAnimator(name, total=500)

start\_time = time.time()

epochs = 500

try:

if hasattr(model, 'partial\_fit'):

for epoch in range(epochs):

model.partial\_fit(X\_train\_scaled, y\_train, classes=np.unique(y\_train))

animator.update(epoch, epochs)

time.sleep(0.01)

else:

for epoch in range(epochs):

if epoch == 0:

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

animator.update(epoch, epochs)

time.sleep(0.01)

except Exception as e:

print(f"\n⚠ Ошибка при обучении {name}: {str(e)}")

train\_time = time.time() - start\_time

print(f'\033[0m\n✅ Модель "{name}" обучена за {train\_time:.1f} сек.')

y\_pred = model.predict(X\_val\_scaled)

y\_proba = model.predict\_proba(X\_val\_scaled)[:,1] if hasattr(model, 'predict\_proba') else [0]\*len(y\_val)

cm = confusion\_matrix(y\_val, y\_pred)

confusion\_matrices.append((name, cm))

metrics = {

'Model': name,

'Accuracy': accuracy\_score(y\_val, y\_pred),

'Precision': precision\_score(y\_val, y\_pred),

'Recall': recall\_score(y\_val, y\_pred),

'F1': f1\_score(y\_val, y\_pred),

'AUC': roc\_auc\_score(y\_val, y\_proba) if hasattr(model, 'predict\_proba') else 0.5,

'Time': f'{train\_time:.1f}s'

}

results.append(metrics)

trained\_models[name] = model

На рисунке 8 представлена процесс обучения моделей для сравнения.

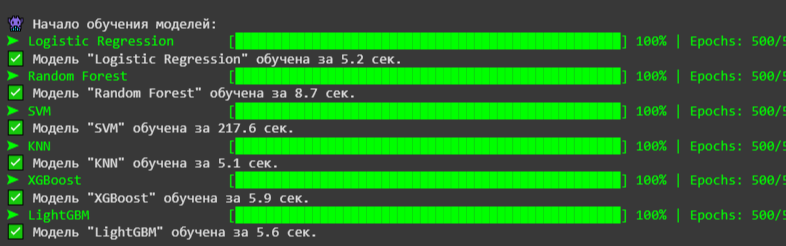


Рисунок 8 – Обучение моделей для сравнения

# Матрицы ошибок

plt.figure(figsize=(18, 12))

for i, (name, cm) in enumerate(confusion\_matrices, 1):

plt.subplot(2, 3, i)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

xticklabels=['Без депрессии', 'Депрессия'],

yticklabels=['Без депрессии', 'Депрессия'])

plt.title(f'{name}')

plt.ylabel('Фактические значения')

plt.xlabel('Предсказанные значения')

plt.tight\_layout()

plt.show()

На рисунке 9 представлены матрицы ошибок.

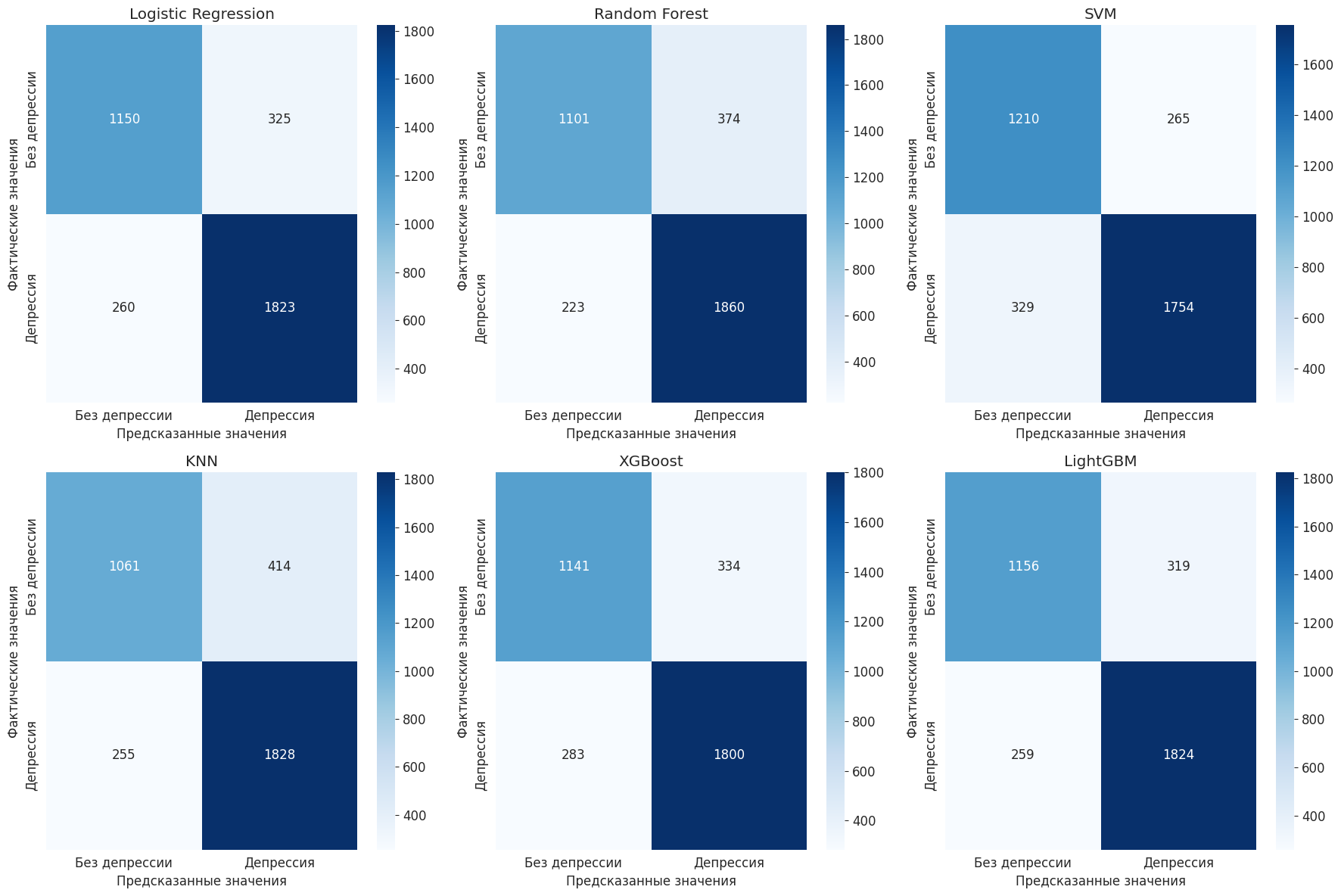


Рисунок 9 – Матрицы ошибок

# Таблица результатов обученных моделей

results\_df = pd.DataFrame(results).sort\_values('F1', ascending=False)

print("\n📈 Результаты сравнения моделей:")

display(results\_df.style.background\_gradient(cmap='Blues', subset=['Accuracy', 'Precision', 'Recall', 'F1', 'AUC']))

best\_model\_name = results\_df.iloc[0]['Model']

best\_model = trained\_models[best\_model\_name]

print(f"\n🚀 Лучшая модель: {best\_model\_name} (F1-score: {results\_df.iloc[0]['F1']:.3f})")

На рисунке 10 представлена таблица результатов обученных моделей.

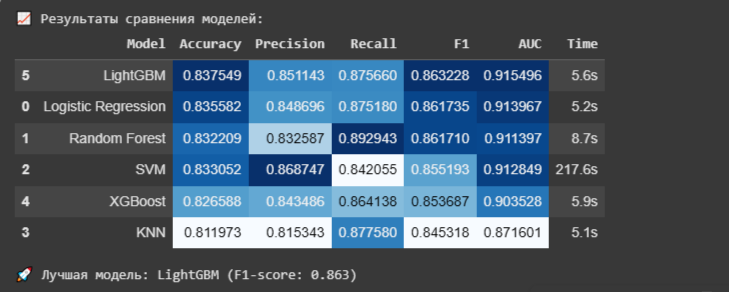
****

Рисунок 10 –Таблица результатов

Видим, что наилучший результат принадлежит модели LigthGBM

Поэтому переходим к этапу финального обучения

### Этап 3. Финальное обучение и сохранение лучшей модели

## Финальное обучение

# Финальное обучение

print(f"\n🤖 Финальное обучение {best\_model\_name} (1000 эпох):")

animator = ProgressAnimator(best\_model\_name, total=1000)

learning\_progress = []

X\_combined = np.vstack([X\_train\_scaled, X\_val\_scaled])

y\_combined = np.concatenate([y\_train, y\_val])

start\_time = time.time()

for epoch in range(1000):

if hasattr(best\_model, 'partial\_fit'):

best\_model.partial\_fit(X\_combined, y\_combined, classes=np.unique(y\_combined))

else:

if epoch == 0:

best\_model.fit(X\_combined, y\_combined)

animator.update(epoch, 1000)

time.sleep(0.02)

print(f'\033[0m\n✅ Финальное обучение завершено за {time.time()-start\_time:.1f} сек. (1000 эпох)')

t = time.time()-start\_time

# Сохранение модели

joblib.dump(best\_model, 'best\_model.pkl')

joblib.dump(scaler, 'scaler.pkl')

print("\n💾 Модель успешно сохранена!")

# Результаты финального обучения на тестовых данных

final\_metrics = {

'Model': f'{best\_model\_name} (FINAL)',

'Accuracy': accuracy\_score(y\_test, best\_model.predict(X\_test\_scaled)),

'Precision': precision\_score(y\_test, best\_model.predict(X\_test\_scaled)),

'Recall': recall\_score(y\_test, best\_model.predict(X\_test\_scaled)),

'F1': f1\_score(y\_test, best\_model.predict(X\_test\_scaled)),

'AUC': roc\_auc\_score(y\_test, best\_model.predict\_proba(X\_test\_scaled)[:,1]),

'Time': f'{t:.1f}s'

}

# Отдельный DataFrame для финального результата

final\_results\_df = pd.DataFrame([final\_metrics])

# Вывод таблицы

print("\n🏆 Результаты финальной модели после обучения:")

display(final\_results\_df.style.background\_gradient(

cmap='Greens',

subset=['Accuracy', 'Precision', 'Recall', 'F1', 'AUC']

))

На рисунке 11 представлены результаты после финального обучения лучшей модели.

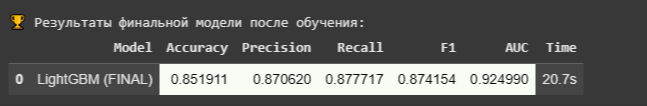


Рисунок 11 – Результаты после финального обучения

# ROC-кривая

y\_proba = best\_model.predict\_proba(X\_test\_scaled)[:,1]

fpr, tpr, \_ = roc\_curve(y\_test, y\_proba)

roc\_auc = roc\_auc\_score(y\_test, y\_proba)

plt.figure(figsize=(8,6))

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC = {roc\_auc:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('Частота ложноположительных результатов')

plt.ylabel('Частота правдивых результатов')

plt.title('ROC-кривая лучшей модели')

plt.legend(loc="lower right")

plt.show()

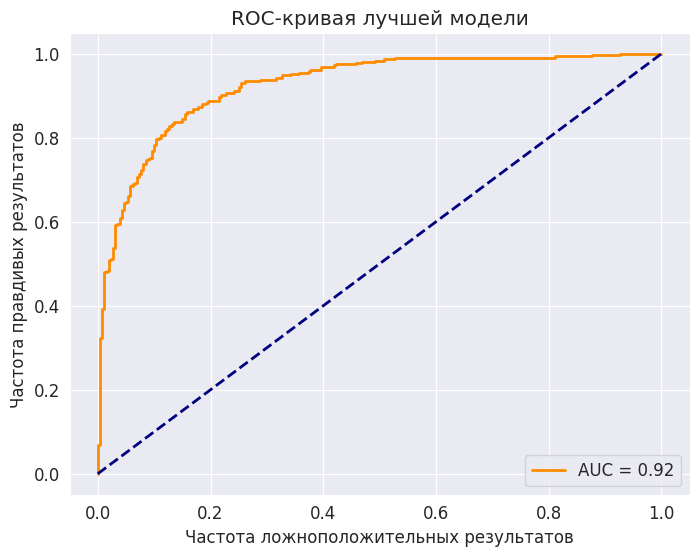
На рисунке 12 представлена ROC-кривая лучшей модели

Рисунок 12 – ROC-кривая лучшей модели

Так же для использования обученной модели был реализован интуитивно понятный пользовательский интерфейс.

# Загрузка обученной модели

try:

model = joblib.load('best\_model.pkl')

scaler = joblib.load('scaler.pkl')

print("✅ Модель успешно загружена")

except Exception as e:

print(f"⚠ Ошибка загрузки модели: {e}")

raise

# Стили

styles = """

<style>

/\* Анимация кнопок \*/

.jupyter-button {

transition: all 0.3s ease !important;

position: relative;

overflow: hidden;

border-radius: 8px !important;

}

.jupyter-button:before {

content: '';

position: absolute;

top: 0;

left: 0;

width: 100%;

height: 100%;

background: rgba(255,255,255,0.2);

opacity: 0;

transition: opacity 0.3s ease;

}

.jupyter-button:hover:before {

opacity: 1;

}

/\* Стили для ползунков \*/

.widget-slider .ui-slider-handle {

transition: transform 0.2s ease, box-shadow 0.2s ease;

}

.widget-slider .ui-slider-handle:hover {

transform: scale(1.2);

box-shadow: 0 0 10px rgba(66,165,245,0.5);

}

</style>

"""

# Шапка приложения

header = widgets.HTML(

"<div style='text-align:center; padding:20px; margin-bottom:20px;'>"

"<h1 style='color:#FFFFFF;'>📊 Тест на риск депрессии</h1>"

"<p style='color:#666; max-width:600px; margin:0 auto;'>"

"Анонимный тест для оценки психологического состояния</p>"

"</div>"

)

# Сообщение благодарности

thank\_you\_message = widgets.HTML(

"<div style='text-align:center; padding:40px; color:#2c3e50;'>"

"<h2>🍃 Спасибо за прохождение теста!</h2>"

"<p style='color:#666'>Берегите свое психическое здоровье и не стесняйтесь обращаться за помощью.</p>"

"</div>",

layout={'display': 'none'} # Изначально скрыт

)

# Футер

footer = widgets.HTML(

"<div style='text-align:center; margin-top:20px; color:#666;'>"

"<p>Круглосуточная психологическая помощь: 8 800 2000 122</p>"

"</div>"

)

# Виджеты (слайдер и радиобатон)

def create\_styled\_slider(description, \*\*kwargs):

return FloatSlider(

\*\*kwargs,

description=description,

style={'description\_width': 'initial'},

layout={'width': '80%', 'margin': '10px auto'},

continuous\_update=False

)

def create\_styled\_radio(description, options):

return RadioButtons(

options=options,

description=description,

style={'description\_width': 'initial'},

layout={'width': '80%', 'margin': '10px auto'}

)

# Создание виджетов

widgets\_dict = {

'Age': create\_styled\_slider(

'👶 Возраст:',

value=17, min=15, max=60, step=1

),

'Academic Pressure': create\_styled\_slider(

'📚 Учебная нагрузка (0-10 где 0 - нет нагрузки, 10 - экстремальная):',

value=5, min=0, max=10, step=0.05

),

'CGPA': create\_styled\_slider(

'🎓 Средний балл (CGPA по шкале 0-5):',

value=5, min=0, max=5, step=0.01

),

'Financial Stress': create\_styled\_slider(

'💸 Финансовый стресс (0-10 где 0 - нет проблем, 10 - критическое положение):',

value=5, min=0, max=10, step=0.1

),

'Work Pressure': create\_styled\_slider(

'💼 Рабочая нагрузка (0-10 где 0 - нет нагрузки, 10 - непосильная):',

value=5, min=0, max=10, step=0.1

),

'Study Satisfaction': create\_styled\_slider(

'😊 Удовлетворенность учебой (0-10 где 0 - полная неудовлетворенность):',

value=5, min=0, max=10, step=0.1

),

'Sleep Duration': create\_styled\_slider(

'😴 Продолжительность сна (часов в сутки):',

value=7, min=0, max=24, step=0.5

),

'Gender\_Male': create\_styled\_radio(

'👤 Пол:',

[('Мужской', 1), ('Женский', 0)]

),

'Profession\_Working': create\_styled\_radio(

'👔 Основной род деятельности:',

[('Работающий', 1), ('Студент', 0)]

),

'Dietary Habits\_Vegetarian': create\_styled\_radio(

'🥦 Диетические привычки:',

[('Здорове питание', 1), ('Нездоровое питание', 0)]

),

'Degree': Dropdown(

options=[('Бакалавр', (0,0)), ('Магистр', (1,0)), ('Аспирант', (0,1))],

description='🎓 Образовательная степень:',

style={'description\_width': 'initial'},

layout={'width': '80%', 'margin': '10px auto'}

),

'Have you ever had suicidal thoughts ?\_Yes': create\_styled\_radio(

'💔 Были ли у вас суицидальные мысли?:',

[('Да', 1), ('Нет', 0)]

),

'Family History of Mental Illness\_Yes': create\_styled\_radio(

'🧬 Есть ли в семье история психических заболеваний?:',

[('Да', 1), ('Нет', 0)]

)

}

# Функция отображения результатов

def show\_results(prediction, proba):

risk\_class = "risk-high" if prediction == 1 else "risk-low"

result\_html = f"""

<div class="{risk\_class} recommendation-box">

<h3 style="color:{"#ff4444" if prediction == 1 else "#44aa44"}; margin-top:0;">

{'⚠️ Высокий риск депрессии' if prediction == 1 else '✅ Низкий риск'}

</h3>

<div style="font-size: 24px; margin: 15px 0; color: {"#ff4444" if prediction == 1 else "#44aa44"}">

Вероятность: {proba:.1f}%

</div>

<h4 style="margin-bottom: 10px;">📋 Рекомендации:</h4>

<ul style="padding-left: 20px; margin: 0;">

"""

recommendations = []

if prediction == 1:

recommendations.append("🚨 Срочно обратитесь к психологу или психотерапевту")

recommendations.append("📞 Позвоните на горячую линию психологической помощи: 8 800 2000 122")

recommendations.append("🧘 Практикуйте техники глубокого дыхания и релаксации")

if proba > 75:

recommendations.append("❗️ Немедленно обратитесь в ближайший медицинский центр")

recommendations.append("⚕️ Запишитесь на срочную консультацию специалиста")

else:

recommendations.append("📅 Запланируйте регулярные консультации с психологом")

recommendations.append("🏃 Увеличьте физическую активность")

else:

recommendations.append("👍 Продолжайте поддерживать здоровый баланс работы и отдыха")

recommendations.append("📅 Регулярно проверяйте свое эмоциональное состояние")

recommendations.append("🍎 Соблюдайте режим здорового питания и сна")

result\_html += "\n".join([f"<li>{r}</li>" for r in recommendations])

result\_html += "</ul></div>"

display(HTML(styles + result\_html))

# Контейнер вопросов

questions\_box = widgets.VBox(

[widgets\_dict[k] for k in widgets\_dict],

layout=widgets.Layout(

width='80%',

margin='20px auto',

padding='20px',

border='1px solid #eee',

border\_radius='8px',

display='none'

)

)

# Кнопки управления:

start\_button = widgets.Button(

description='Начать тестирование',

button\_style='success',

icon='play',

layout={'width': '300px', 'margin': '20px auto', 'justify\_content': 'center'}

)

finish\_button = widgets.Button(

description='Завершить тестирование',

button\_style='warning',

icon='flag-checkered',

layout={'width': '300px', 'margin': '20px auto', 'justify\_content': 'center','display': 'none'}

)

restart\_button = widgets.Button(

description='Пройти тест повторно',

button\_style='success',

icon='refresh',

layout={'width': '300px', 'margin': '20px auto',

'justify\_content': 'center', 'display': 'none'}

)

# Блок результатов

result\_output = widgets.Output(layout={'display': 'none', 'justify\_content': 'center'})

# Собираем интерфейс

app = widgets.VBox([

header,

start\_button,

questions\_box,

finish\_button,

result\_output,

thank\_you\_message,

restart\_button,

footer

], layout=widgets.Layout(width='100%'))

# Функции управления интерфейсом

def reset\_widgets():

for name, widget in widgets\_dict.items():

if isinstance(widget, FloatSlider):

widget.value = (widget.max + widget.min) / 2

elif isinstance(widget, RadioButtons):

widget.index = None

elif isinstance(widget, Dropdown):

widget.value = widget.options[0][1]

result\_output.clear\_output()

def start\_test(b):

reset\_widgets()

start\_button.layout.display = 'none'

questions\_box.layout.display = 'flex'

finish\_button.layout.display = 'flex'

restart\_button.layout.display = 'none'

result\_output.layout.display = 'none'

for widget in widgets\_dict.values():

widget.disabled = False

def finish\_test(b):

data = {name: widget.value for name, widget in widgets\_dict.items()}

with result\_output:

result\_output.clear\_output()

try:

predict\_depression(\*\*data)

except Exception as e:

display(HTML(f"<div style='color:red; padding:20px;'>Ошибка: {str(e)}</div>"))

questions\_box.layout.display = 'none'

finish\_button.layout.display = 'none'

result\_output.layout.display = 'flex'

restart\_button.layout.display = 'flex'

thank\_you\_message.layout.display = 'flex'

for widget in widgets\_dict.values():

widget.disabled = True

def restart\_test(b):

thank\_you\_message.layout.display = 'none'

start\_test(b)

# Обработчики нажатия на кнопки

start\_button.on\_click(start\_test)

finish\_button.on\_click(finish\_test)

restart\_button.on\_click(restart\_test)

# Функция прогнозирования

def predict\_depression(\*\*kwargs):

try:

data = {

'Age': kwargs['Age'],

'Academic Pressure': kwargs['Academic Pressure'],

'CGPA': kwargs['CGPA'] \* 2,

'Financial Stress': kwargs['Financial Stress'],

'Work Pressure': kwargs['Work Pressure'],

'Study Satisfaction': kwargs['Study Satisfaction'],

'Sleep Duration': kwargs['Sleep Duration'],

'Gender\_Male': kwargs['Gender\_Male'],

'Profession\_Working': kwargs['Profession\_Working'],

'Dietary Habits\_Vegetarian': kwargs['Dietary Habits\_Vegetarian'],

'Degree\_Master': kwargs['Degree'][0],

'Degree\_PhD': kwargs['Degree'][1],

'Have you ever had suicidal thoughts ?\_Yes': kwargs['Have you ever had suicidal thoughts ?\_Yes'],

'Family History of Mental Illness\_Yes': kwargs['Family History of Mental Illness\_Yes']

}

input\_df = pd.DataFrame([data], columns=X\_train.columns)

scaled\_input = scaler.transform(input\_df)

prediction = model.predict(scaled\_input)[0]

proba = model.predict\_proba(scaled\_input)[0][1] \* 100

show\_results(prediction, proba)

except Exception as e:

display(HTML(f"<div style='color:red; padding:20px;'>Ошибка: {str(e)}</div>"))

# Запуск приложения

display(HTML(styles))

display(app)

На рисунке 13 предоставлен пользовательский интерфейс

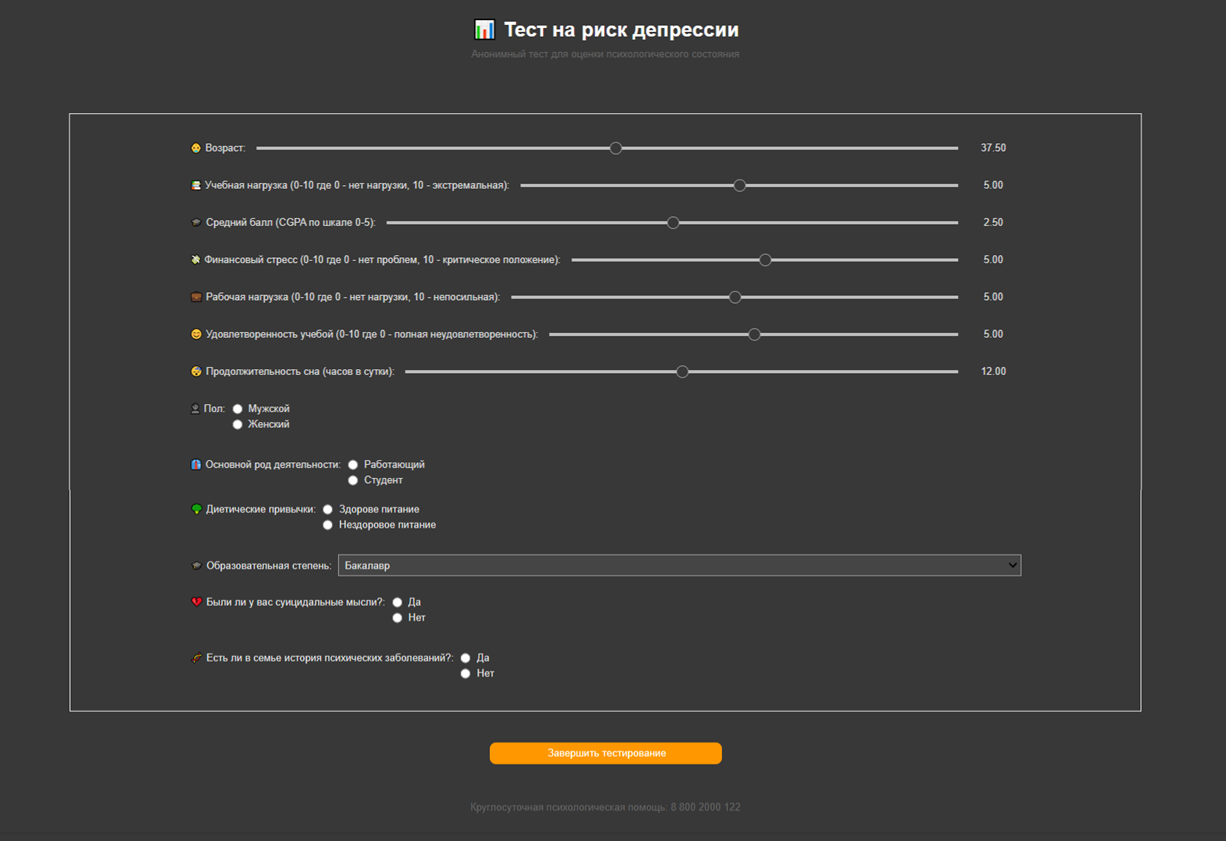


Рисунок 13 –Интерфейс

Так же после прохождения тестирования пользователю сразу выходит вероятность депрессии и необходимые рекомендации в зависимости от уровня вероятности депрессии

На рисунке 14 предоставлены результаты прохождения тестирования

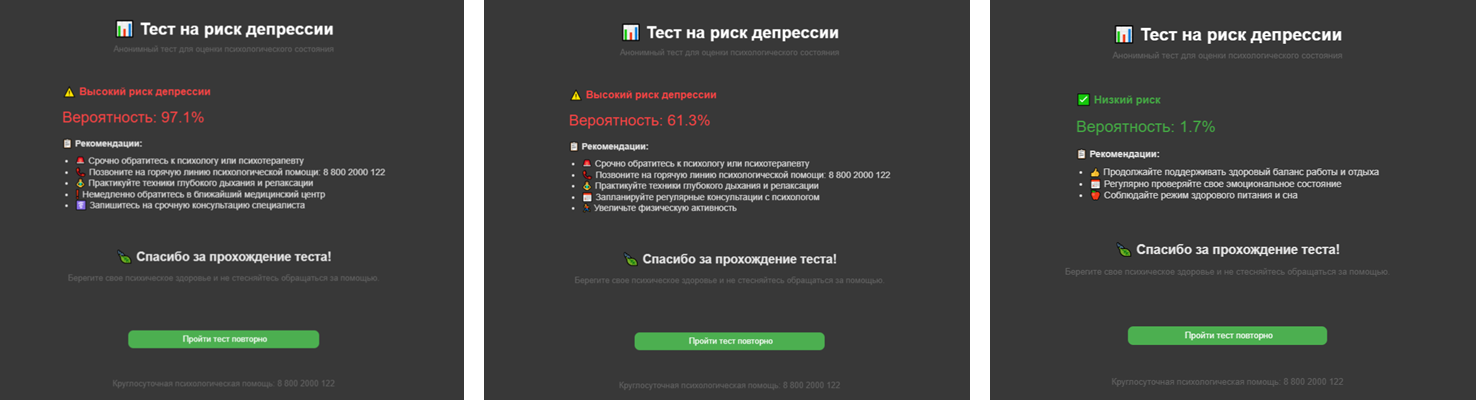


Рисунок 14 –Результаты

## Заключение

В процессе исследования была затронута очень важная тема в современном обществе, такая как ментальное здоровье студентов. Данный проект должен помочь студентам преждевременно выявлять предпосылки к психологическим недугам и помочь избежать негативных последствий в связи с внешними и внутренними факторами, которые так сильно влияют на человека.

Результаты работы уже на данный момент можно использовать для первичного тестирования и выявления предпосылок к депрессии и в дальнейшем перенаправлять студента к специалисту, который в свою очередь может выставить медицинское заключение и/или поможет разобраться с причиной возникновения такого состояния, что в дальнейшем позитивно повлияет на самочувствие студента, его продуктивность и стремление жить и развиваться.

Современные методы машинного обучения очень тесно внедряются в современный мир. Многие крупные компании используют их для понимания и анализа психологии людей для коммерческих целей. Цель же нашего проекта создать помощь для первичного обнаружения такой глобальной проблемы современного поколения как депрессии и помочь людям быстрее понять, что с ними происходит, дать рекомендации и направить при необходимости к нужному специалисту.

## Список использованных источников и литературы

Бессмертный, И. А. Интеллектуальные системы : учебник и практикум для академического бакалавриата / И. А. Бессмертный, А. Б. Нугуманова, А. В. Платонов. — Москва : Издательство Юрайт, 2019. — 243 с. — (Бакалавр. Академический курс). — ISBN 978-5-534-01042-8. — Текст : электронный // ЭБС Юрайт [сайт]. (дата обращения: 01.10.2024).

Ясницкий, Л.пН. Интеллектуальные системы : учебник / Л. Н. Ясницкий. — 2-е изд. — Москва : Лаборатория знаний, 2020. — 222 c. — ISBN 978-5-00101-897-1. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. (дата обращения: 01.10.2024).

Сузи, Р. А. Язык программирования Python : учебное пособие / Р. А. Сузи. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 350 c. — ISBN 978-5-4497-0705-5. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. (дата обращения: 01.10.2024).

Федоров, Д. Ю. Программирование на языке высокого уровня Python : учебное пособие для среднего профессионального образования / Д. Ю. Федоров. — 4-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2023. — 214 с. — (Профессиональное образование). — ISBN 978-5-534-15731-4. — Текст : электронный // Образовательная платформа Юрайт [сайт]. (дата обращения: 01.10.2024).

EDA. Поисковый анализ данных на Python. [Электронный ресурс]. URL: https://translated.turbopages.org/proxy\_u/en-ru.ru.7ef26421-67684fec-482e8cf6-74722d776562/https/www.geeksforgeeks.org/exploratory-data-analysis-in-python/(дата обращения: 01.12.2024).

Статья «Технология производства оптического волокна». URL: <https://science.fandom.com/ru/wiki/Технология_производства_оптического_волокна> (дата обращения: 02.12.2024).

Никоноров, Н.В. Материалы и технологии волоконной оптики: специальные оптические волокна. Учебное пособие, курс лекций / Н.В. Никоноров, А.И. Сидоров. – СПб.: СПбГУ ИТМО, 2009. – 130 с (дата обращения: 02.12.2024).

Nagel, S.R. An Overview of the Modified Chemical Vapor Deposition (MCVD) Process and Performance / S.R. Nagel, J.B. Macchesney, K.L. Walker // IEEE journal of quantum electronics. — 1982. — Vol. MTT-30. — No. 4. — P. 305-322 (дата обращения: 02.12.2024).

# Приложения

Приложение 1

Программный код

# Установка библиотек

import subprocess

import time

from IPython.display import HTML, display

def install\_packages\_with\_enhanced\_progress(packages):

num\_packages = len(packages)

start\_time = time.time()

block\_height = "20px"

# HTML для стилизованного индикатора прогресса на основе блоков

def create\_block\_progress\_html(percentage):

num\_blocks = 15

filled\_blocks = int(percentage / (100 / num\_blocks))

blocks = ''.join([f'<span style="color: green; padding-right: 2px; height: {block\_height}; display: inline-block; vertical-align: middle;">█</span>' for \_ in range(filled\_blocks)]) + \

''.join([f'<span style="color: lightgrey; padding-right: 2px; height: {block\_height}; display: inline-block; vertical-align: middle;">█</span>' for \_ in range(num\_blocks - filled\_blocks)])

return f"""

<div style="font-family: monospace; font-size: 16px; background-color: transparent;">

{blocks} {percentage:.1f}%

</div>

"""

progress\_display = display(HTML(create\_block\_progress\_html(0)), display\_id=True)

try:

for i, package in enumerate(packages):

process = subprocess.Popen(['pip', 'install', package],

stdout=subprocess.PIPE,

stderr=subprocess.PIPE) # Запускаем pip install как отдельный процесс

while process.poll() is None:

time.sleep(0.1)

overall\_percentage = min(100.0, ((i / num\_packages) \* 100) + ((time.time() - start\_time)/2) )

progress\_display.update(HTML(create\_block\_progress\_html(overall\_percentage))) # Обновляем индикатор прогресса

stdout, stderr = process.communicate()

if process.returncode != 0:

print(f"Ошибка при установке {package}: {stderr.decode()}")

raise Exception(f"Установка не удалась для {package}")

except Exception as e:

progress\_display.update(HTML("<div style='color: red;'>Установка не удалась! Проверьте логи ошибок.</div>"))

print(f"Общая установка не удалась: {e}")

return

progress\_display.update(HTML("<div style='color: green;'>Установка завершена!</div>"))

# Список пакетов для установки

packages = [

"tensorflow",

"kagglehub",

"pandas",

"opendatasets",

"dask[dataframe]",

"scikit-learn",

"matplotlib",

"seaborn",

"numpy",

"ipywidgets"

]

install\_packages\_with\_enhanced\_progress(packages)

#Импорт библиотек

# Стандартные библиотеки

import os

import re

import time

import warnings

from datetime import datetime

# Для данных

import kagglehub

import matplotlib

import matplotlib.pyplot as plt

import numpy as np

import opendatasets as od

import pandas as pd

import seaborn as sns

from tqdm import tqdm

import pexpect

# Машинное обучение и обработка данных

from sklearn.decomposition import PCA

from sklearn.ensemble import (AdaBoostClassifier, BaggingClassifier,

GradientBoostingClassifier,

HistGradientBoostingClassifier,

RandomForestClassifier)

from sklearn.feature\_selection import SelectKBest, f\_classif

from sklearn.impute import SimpleImputer

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import (accuracy\_score, auc, classification\_report,

confusion\_matrix, f1\_score, precision\_score,

recall\_score, roc\_auc\_score, roc\_curve)

from sklearn.model\_selection import KFold, cross\_val\_score, learning\_curve, train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from lightgbm import LGBMClassifier

from xgboost import XGBClassifier

# Визуализация и интерфейсы

import io

import joblib

from contextlib import redirect\_stdout

import tkinter as tk

from matplotlib.animation import FuncAnimation

from tkinter import ttk, messagebox

from IPython.display import HTML, clear\_output, display

from ipywidgets import Dropdown, FloatSlider, IntSlider, RadioButtons, interact, interactive, widgets

#Загрузка датасета

# Запуск процесса загрузки

child = pexpect.spawn('python -c "import opendatasets as od; od.download(\'https://www.kaggle.com/datasets/hopesb/student-depression-dataset\', force=True )"')

# Ожидаем первый запрос на логин и автоматически вводим

child.expect('Your Kaggle username:')

child.sendline('dmytrt')

# Ожидаем запрос пароля и вводим ключ

child.expect('Your Kaggle Key:')

child.sendline('17165469')

# Ждем завершения процесса

child.expect(pexpect.EOF)

# Загрузка данных и вывод первых 5 строк набора данных

dep\_df = pd.read\_csv("student-depression-dataset/Student Depression Dataset.csv")

dep\_df.head()

# Размер датасета

dep\_df.shape

# Распределение случаев депрессии

plt.figure(figsize=(10, 6))

sns.countplot(

x='Depression',

hue='Depression',

data=dep\_df,

palette='viridis',

legend=True

)

plt.title('Распределение случаев депрессии')

plt.show()

# \*\*Обучение модели\*\*

# Предобработка и визуализация данных

print("✏️ Создание гистограмм для сравнения распределения категориальных признаков по значениям целевой переменной Depression\n")

# Настройки

sns.set\_style('darkgrid')

plt.rcParams.update({

'font.size': 12,

'figure.figsize': (12, 8),

'figure.facecolor': 'white'

})

# Предобработка данных

dep\_df = dep\_df.drop(['City'], axis=1)

def convert\_sleep\_duration(sleep\_duration):

try:

cleaned = re.sub(r'[^\d-]', '', str(sleep\_duration))

matches = re.findall(r'\d+', cleaned)

if len(matches) == 2:

return (int(matches[0]) + int(matches[1])) / 2

return int(matches[0]) if matches else np.nan

except:

return np.nan

dep\_df['Sleep Duration'] = dep\_df['Sleep Duration'].apply(convert\_sleep\_duration)

dep\_df['Sleep Duration'] = dep\_df['Sleep Duration'].fillna(dep\_df['Sleep Duration'].median())

#numeric\_cols = ['Age', 'Academic Pressure', 'CGPA', 'Financial Stress', 'Sleep Duration']

numeric\_cols =['Age', 'Academic Pressure', 'CGPA', 'Financial Stress', 'Sleep Duration']

dep\_df[numeric\_cols] = SimpleImputer(strategy='median').fit\_transform(dep\_df[numeric\_cols])

categorical\_cols = ['Gender', 'Profession', 'Dietary Habits', 'Degree',

'Have you ever had suicidal thoughts ?', 'Family History of Mental Illness']

# Создание гистограмм для сравнения распределения категориальных признаков по значениям целевой переменной Depression

for column in categorical\_cols:

plt.figure(figsize=(8, 6))

ax = sns.countplot(x=dep\_df[column], hue='Depression', data=dep\_df)

plt.title(f'{column} vs Depression')

plt.xlabel(column)

plt.ylabel('Count')

plt.xticks(rotation=90, ha='right')

plt.tight\_layout()

plt.show()

dep\_df = pd.get\_dummies(dep\_df, columns=categorical\_cols, drop\_first=True)

dep\_df = dep\_df.drop(['id'], axis=1)

# Разделение данных

X = dep\_df.drop(['Depression'], axis=1)

y = dep\_df['Depression']

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.15, random\_state=42, stratify=y)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.15, random\_state=42, stratify=y\_temp)

scaler = StandardScaler()

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

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

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

# Создание матрицы диаграмм рассеивания для анализа взаимосвязи между числовыми признаками и целевой переменной Depression

print("✏️ Создание матрицы диаграмм рассеивания для анализа взаимосвязи между числовыми признаками и целевой переменной Depression\n")

g = sns.pairplot(dep\_df[['Age', 'Academic Pressure', 'CGPA', 'Financial Stress', 'Sleep Duration', 'Depression']], hue='Depression', palette='coolwarm')

g.fig.suptitle("Матрица рассеивания между числовыми признаками и Депрессией", y=1.02)

plt.show()

# Матрица корреляции

print("✏️ Создание матрицы корреляции\n")

plt.figure(figsize=(14, 12))

corr = dep\_df.corr()

sns.heatmap(corr, annot=False, cmap='coolwarm')

plt.title('Матрица корреляции')

plt.show()

# Анализ важности признаков

print("✏️ Создание даграммы важности признаков для определения наличия депрессии\n")

selector = SelectKBest(f\_classif, k=15)

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

selected\_features = X.columns[selector.get\_support()]

plt.figure(figsize=(12, 8))

pd.Series(selector.scores\_, index=X.columns).nlargest(15).plot(kind='barh', color='skyblue')

plt.title('Влияние признаков на наличие депрессии')

plt.show()

# Сравнение моделей

# Класс для прогресса

class ProgressAnimator:

def \_\_init\_\_(self, model\_name, total=100):

self.model\_name = model\_name

self.total = total

self.start\_time = datetime.now()

self.last\_progress = 0

def get\_color(self, progress):

r = int(255 \* (1 - progress/100))

g = int(255 \* (progress/100))

return f"\033[38;2;{r};{g};0m"

def update(self, current, total\_epochs):

progress = min(100, int(100 \* (current + 1) / self.total))

if progress == self.last\_progress:

return

self.last\_progress = progress

color = self.get\_color(progress)

elapsed = datetime.now() - self.start\_time

bar = '█' \* int(progress / 2) + ' ' \* (50 - int(progress / 2))

epoch\_str = f'Epochs: {current+1}/{total\_epochs}'

time\_str = f' | Time: {elapsed.seconds}s'

print(f'\r{color}➤ {self.model\_name:25} [{bar}] {progress}% | {epoch\_str}{time\_str}', end='')

# Обучение моделей

models = {

"Logistic Regression": LogisticRegression(max\_iter=1000, random\_state=42),

"Random Forest": RandomForestClassifier(n\_estimators=200, max\_depth=10, random\_state=42),

"SVM": SVC(probability=True, random\_state=42, class\_weight='balanced'),

"KNN": KNeighborsClassifier(n\_neighbors=7),

"XGBoost": XGBClassifier(n\_estimators=150, eval\_metric='logloss', random\_state=42),

"LightGBM": LGBMClassifier(num\_leaves=31, random\_state=42, verbose=-1)

}

trained\_models = {}

results = []

confusion\_matrices = []

print("\n👾 Начало обучения моделей:")

for name, model in models.items():

animator = ProgressAnimator(name, total=500)

start\_time = time.time()

epochs = 500

try:

if hasattr(model, 'partial\_fit'):

for epoch in range(epochs):

model.partial\_fit(X\_train\_scaled, y\_train, classes=np.unique(y\_train))

animator.update(epoch, epochs)

time.sleep(0.01)

else:

for epoch in range(epochs):

if epoch == 0:

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

animator.update(epoch, epochs)

time.sleep(0.01)

except Exception as e:

print(f"\n⚠ Ошибка при обучении {name}: {str(e)}")

train\_time = time.time() - start\_time

print(f'\033[0m\n✅ Модель "{name}" обучена за {train\_time:.1f} сек.')

y\_pred = model.predict(X\_val\_scaled)

y\_proba = model.predict\_proba(X\_val\_scaled)[:,1] if hasattr(model, 'predict\_proba') else [0]\*len(y\_val)

cm = confusion\_matrix(y\_val, y\_pred)

confusion\_matrices.append((name, cm))

metrics = {

'Model': name,

'Accuracy': accuracy\_score(y\_val, y\_pred),

'Precision': precision\_score(y\_val, y\_pred),

'Recall': recall\_score(y\_val, y\_pred),

'F1': f1\_score(y\_val, y\_pred),

'AUC': roc\_auc\_score(y\_val, y\_proba) if hasattr(model, 'predict\_proba') else 0.5,

'Time': f'{train\_time:.1f}s'

}

results.append(metrics)

trained\_models[name] = model

# Матрицы ошибок

plt.figure(figsize=(18, 12))

for i, (name, cm) in enumerate(confusion\_matrices, 1):

plt.subplot(2, 3, i)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

xticklabels=['Без депрессии', 'Депрессия'],

yticklabels=['Без депрессии', 'Депрессия'])

plt.title(f'{name}')

plt.ylabel('Фактические значения')

plt.xlabel('Предсказанные значения')

plt.tight\_layout()

plt.show()

# Таблица результатов обученных моделей

results\_df = pd.DataFrame(results).sort\_values('F1', ascending=False)

print("\n📈 Результаты сравнения моделей:")

display(results\_df.style.background\_gradient(cmap='Blues', subset=['Accuracy', 'Precision', 'Recall', 'F1', 'AUC']))

best\_model\_name = results\_df.iloc[0]['Model']

best\_model = trained\_models[best\_model\_name]

print(f"\n🚀 Лучшая модель: {best\_model\_name} (F1-score: {results\_df.iloc[0]['F1']:.3f})")

# Обучение и сохранение лучшей модели

# Финальное обучение

print(f"\n🤖 Финальное обучение {best\_model\_name} (1000 эпох):")

animator = ProgressAnimator(best\_model\_name, total=1000)

learning\_progress = []

X\_combined = np.vstack([X\_train\_scaled, X\_val\_scaled])

y\_combined = np.concatenate([y\_train, y\_val])

start\_time = time.time()

for epoch in range(1000):

if hasattr(best\_model, 'partial\_fit'):

best\_model.partial\_fit(X\_combined, y\_combined, classes=np.unique(y\_combined))

else:

if epoch == 0:

best\_model.fit(X\_combined, y\_combined)

animator.update(epoch, 1000)

time.sleep(0.02)

print(f'\033[0m\n✅ Финальное обучение завершено за {time.time()-start\_time:.1f} сек. (1000 эпох)')

t = time.time()-start\_time

# Сохранение модели

joblib.dump(best\_model, 'best\_model.pkl')

joblib.dump(scaler, 'scaler.pkl')

print("\n💾 Модель успешно сохранена!")

# Результаты финального обучения на тестовых данных

final\_metrics = {

'Model': f'{best\_model\_name} (FINAL)',

'Accuracy': accuracy\_score(y\_test, best\_model.predict(X\_test\_scaled)),

'Precision': precision\_score(y\_test, best\_model.predict(X\_test\_scaled)),

'Recall': recall\_score(y\_test, best\_model.predict(X\_test\_scaled)),

'F1': f1\_score(y\_test, best\_model.predict(X\_test\_scaled)),

'AUC': roc\_auc\_score(y\_test, best\_model.predict\_proba(X\_test\_scaled)[:,1]),

'Time': f'{t:.1f}s'

}

# Отдельный DataFrame для финального результата

final\_results\_df = pd.DataFrame([final\_metrics])

# Вывод таблицы

print("\n🏆 Результаты финальной модели после обучения:")

display(final\_results\_df.style.background\_gradient(

cmap='Greens',

subset=['Accuracy', 'Precision', 'Recall', 'F1', 'AUC']

))

# ROC-кривая

y\_proba = best\_model.predict\_proba(X\_test\_scaled)[:,1]

fpr, tpr, \_ = roc\_curve(y\_test, y\_proba)

roc\_auc = roc\_auc\_score(y\_test, y\_proba)

plt.figure(figsize=(8,6))

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC = {roc\_auc:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('Частота ложноположительных результатов')

plt.ylabel('Частота правдивых результатов')

plt.title('ROC-кривая лучшей модели')

plt.legend(loc="lower right")

plt.show()

# \*\*Интерфейс для использования модели\*\*

# Загрузка модели. Интерфейс. Обработчики событий

# Загрузка обученной модели

try:

model = joblib.load('best\_model.pkl')

scaler = joblib.load('scaler.pkl')

print("✅ Модель успешно загружена")

except Exception as e:

print(f"⚠ Ошибка загрузки модели: {e}")

raise

# Стили

styles = """

<style>

/\* Анимация кнопок \*/

.jupyter-button {

transition: all 0.3s ease !important;

position: relative;

overflow: hidden;

border-radius: 8px !important;

}

.jupyter-button:before {

content: '';

position: absolute;

top: 0;

left: 0;

width: 100%;

height: 100%;

background: rgba(255,255,255,0.2);

opacity: 0;

transition: opacity 0.3s ease;

}

.jupyter-button:hover:before {

opacity: 1;

}

/\* Стили для ползунков \*/

.widget-slider .ui-slider-handle {

transition: transform 0.2s ease, box-shadow 0.2s ease;

}

.widget-slider .ui-slider-handle:hover {

transform: scale(1.2);

box-shadow: 0 0 10px rgba(66,165,245,0.5);

}

</style>

"""

# Шапка приложения

header = widgets.HTML(

"<div style='text-align:center; padding:20px; margin-bottom:20px;'>"

"<h1 style='color:#FFFFFF;'>📊 Тест на риск депрессии</h1>"

"<p style='color:#666; max-width:600px; margin:0 auto;'>"

"Анонимный тест для оценки психологического состояния</p>"

"</div>"

)

# Сообщение благодарности

thank\_you\_message = widgets.HTML(

"<div style='text-align:center; padding:40px; color:#2c3e50;'>"

"<h2>🍃 Спасибо за прохождение теста!</h2>"

"<p style='color:#666'>Берегите свое психическое здоровье и не стесняйтесь обращаться за помощью.</p>"

"</div>",

layout={'display': 'none'} # Изначально скрыт

)

# Футер

footer = widgets.HTML(

"<div style='text-align:center; margin-top:20px; color:#666;'>"

"<p>Круглосуточная психологическая помощь: 8 800 2000 122</p>"

"</div>"

)

# Виджеты (слайдер и радиобатон)

def create\_styled\_slider(description, \*\*kwargs):

return FloatSlider(

\*\*kwargs,

description=description,

style={'description\_width': 'initial'},

layout={'width': '80%', 'margin': '10px auto'},

continuous\_update=False

)

def create\_styled\_radio(description, options):

return RadioButtons(

options=options,

description=description,

style={'description\_width': 'initial'},

layout={'width': '80%', 'margin': '10px auto'}

)

# Создание виджетов

widgets\_dict = {

'Age': create\_styled\_slider(

'👶 Возраст:',

value=17, min=15, max=60, step=1

),

'Academic Pressure': create\_styled\_slider(

'📚 Учебная нагрузка (0-10 где 0 - нет нагрузки, 10 - экстремальная):',

value=5, min=0, max=10, step=0.05

),

'CGPA': create\_styled\_slider(

'🎓 Средний балл (CGPA по шкале 0-5):',

value=5, min=0, max=5, step=0.01

),

'Financial Stress': create\_styled\_slider(

'💸 Финансовый стресс (0-10 где 0 - нет проблем, 10 - критическое положение):',

value=5, min=0, max=10, step=0.1

),

'Work Pressure': create\_styled\_slider(

'💼 Рабочая нагрузка (0-10 где 0 - нет нагрузки, 10 - непосильная):',

value=5, min=0, max=10, step=0.1

),

'Study Satisfaction': create\_styled\_slider(

'😊 Удовлетворенность учебой (0-10 где 0 - полная неудовлетворенность):',

value=5, min=0, max=10, step=0.1

),

'Sleep Duration': create\_styled\_slider(

'😴 Продолжительность сна (часов в сутки):',

value=7, min=0, max=24, step=0.5

),

'Gender\_Male': create\_styled\_radio(

'👤 Пол:',

[('Мужской', 1), ('Женский', 0)]

),

'Profession\_Working': create\_styled\_radio(

'👔 Основной род деятельности:',

[('Работающий', 1), ('Студент', 0)]

),

'Dietary Habits\_Vegetarian': create\_styled\_radio(

'🥦 Диетические привычки:',

[('Здорове питание', 1), ('Нездоровое питание', 0)]

),

'Degree': Dropdown(

options=[('Бакалавр', (0,0)), ('Магистр', (1,0)), ('Аспирант', (0,1))],

description='🎓 Образовательная степень:',

style={'description\_width': 'initial'},

layout={'width': '80%', 'margin': '10px auto'}

),

'Have you ever had suicidal thoughts ?\_Yes': create\_styled\_radio(

'💔 Были ли у вас суицидальные мысли?:',

[('Да', 1), ('Нет', 0)]

),

'Family History of Mental Illness\_Yes': create\_styled\_radio(

'🧬 Есть ли в семье история психических заболеваний?:',

[('Да', 1), ('Нет', 0)]

)

}

# Функция отображения результатов

def show\_results(prediction, proba):

risk\_class = "risk-high" if prediction == 1 else "risk-low"

result\_html = f"""

<div class="{risk\_class} recommendation-box">

<h3 style="color:{"#ff4444" if prediction == 1 else "#44aa44"}; margin-top:0;">

{'⚠️ Высокий риск депрессии' if prediction == 1 else '✅ Низкий риск'}

</h3>

<div style="font-size: 24px; margin: 15px 0; color: {"#ff4444" if prediction == 1 else "#44aa44"}">

Вероятность: {proba:.1f}%

</div>

<h4 style="margin-bottom: 10px;">📋 Рекомендации:</h4>

<ul style="padding-left: 20px; margin: 0;">

"""

recommendations = []

if prediction == 1:

recommendations.append("🚨 Срочно обратитесь к психологу или психотерапевту")

recommendations.append("📞 Позвоните на горячую линию психологической помощи: 8 800 2000 122")

recommendations.append("🧘 Практикуйте техники глубокого дыхания и релаксации")

if proba > 75:

recommendations.append("❗️ Немедленно обратитесь в ближайший медицинский центр")

recommendations.append("⚕️ Запишитесь на срочную консультацию специалиста")

else:

recommendations.append("📅 Запланируйте регулярные консультации с психологом")

recommendations.append("🏃 Увеличьте физическую активность")

else:

recommendations.append("👍 Продолжайте поддерживать здоровый баланс работы и отдыха")

recommendations.append("📅 Регулярно проверяйте свое эмоциональное состояние")

recommendations.append("🍎 Соблюдайте режим здорового питания и сна")

result\_html += "\n".join([f"<li>{r}</li>" for r in recommendations])

result\_html += "</ul></div>"

display(HTML(styles + result\_html))

# Контейнер вопросов

questions\_box = widgets.VBox(

[widgets\_dict[k] for k in widgets\_dict],

layout=widgets.Layout(

width='80%',

margin='20px auto',

padding='20px',

border='1px solid #eee',

border\_radius='8px',

display='none'

)

)

# Кнопки управления:

start\_button = widgets.Button(

description='Начать тестирование',

button\_style='success',

icon='play',

layout={'width': '300px', 'margin': '20px auto', 'justify\_content': 'center'}

)

finish\_button = widgets.Button(

description='Завершить тестирование',

button\_style='warning',

icon='flag-checkered',

layout={'width': '300px', 'margin': '20px auto', 'justify\_content': 'center','display': 'none'}

)

restart\_button = widgets.Button(

description='Пройти тест повторно',

button\_style='success',

icon='refresh',

layout={'width': '300px', 'margin': '20px auto',

'justify\_content': 'center', 'display': 'none'}

)

# Блок результатов

result\_output = widgets.Output(layout={'display': 'none', 'justify\_content': 'center'})

# Собираем интерфейс

app = widgets.VBox([

header,

start\_button,

questions\_box,

finish\_button,

result\_output,

thank\_you\_message,

restart\_button,

footer

], layout=widgets.Layout(width='100%'))

# Функции управления интерфейсом

def reset\_widgets():

for name, widget in widgets\_dict.items():

if isinstance(widget, FloatSlider):

widget.value = (widget.max + widget.min) / 2

elif isinstance(widget, RadioButtons):

widget.index = None

elif isinstance(widget, Dropdown):

widget.value = widget.options[0][1]

result\_output.clear\_output()

def start\_test(b):

reset\_widgets()

start\_button.layout.display = 'none'

questions\_box.layout.display = 'flex'

finish\_button.layout.display = 'flex'

restart\_button.layout.display = 'none'

result\_output.layout.display = 'none'

for widget in widgets\_dict.values():

widget.disabled = False

def finish\_test(b):

data = {name: widget.value for name, widget in widgets\_dict.items()}

with result\_output:

result\_output.clear\_output()

try:

predict\_depression(\*\*data)

except Exception as e:

display(HTML(f"<div style='color:red; padding:20px;'>Ошибка: {str(e)}</div>"))

questions\_box.layout.display = 'none'

finish\_button.layout.display = 'none'

result\_output.layout.display = 'flex'

restart\_button.layout.display = 'flex'

thank\_you\_message.layout.display = 'flex'

for widget in widgets\_dict.values():

widget.disabled = True

def restart\_test(b):

thank\_you\_message.layout.display = 'none'

start\_test(b)

# Обработчики нажатия на кнопки

start\_button.on\_click(start\_test)

finish\_button.on\_click(finish\_test)

restart\_button.on\_click(restart\_test)

# Функция прогнозирования

def predict\_depression(\*\*kwargs):

try:

data = {

'Age': kwargs['Age'],

'Academic Pressure': kwargs['Academic Pressure'],

'CGPA': kwargs['CGPA'] \* 2,

'Financial Stress': kwargs['Financial Stress'],

'Work Pressure': kwargs['Work Pressure'],

'Study Satisfaction': kwargs['Study Satisfaction'],

'Sleep Duration': kwargs['Sleep Duration'],

'Gender\_Male': kwargs['Gender\_Male'],

'Profession\_Working': kwargs['Profession\_Working'],

'Dietary Habits\_Vegetarian': kwargs['Dietary Habits\_Vegetarian'],

'Degree\_Master': kwargs['Degree'][0],

'Degree\_PhD': kwargs['Degree'][1],

'Have you ever had suicidal thoughts ?\_Yes': kwargs['Have you ever had suicidal thoughts ?\_Yes'],

'Family History of Mental Illness\_Yes': kwargs['Family History of Mental Illness\_Yes']

}

input\_df = pd.DataFrame([data], columns=X\_train.columns)

scaled\_input = scaler.transform(input\_df)

prediction = model.predict(scaled\_input)[0]

proba = model.predict\_proba(scaled\_input)[0][1] \* 100

show\_results(prediction, proba)

except Exception as e:

display(HTML(f"<div style='color:red; padding:20px;'>Ошибка: {str(e)}</div>"))

# Запуск интерфейса

# Запуск приложения

display(HTML(styles))

display(app)