|  |  |
| --- | --- |
| C:\Users\DENACT~1\AppData\Local\Temp\lu135925on38x.tmp\lu135925on3bu_tmp_3360867a00ce4d37.jpg | **Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет имени Н.Э. Баумана**  **(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления и искусственный интеллект

КАФЕДРА                  Системы обработки информации и управления

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ***

***НА ТЕМУ:***

Студент: 27.05.2025 Ильин Константин Юрьевич

Группа: ИУ5-61Б (Подпись, дата) (И.О.Фамилия)

Руководитель Гапанюк Юрий Евгеньевич

(Подпись, дата) (И.О.Фамилия)

Консультант \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись, дата) (И.О.Фамилия)

*2025 г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_\_\_\_\_\_\_\_

(Индекс)

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

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

**ЗАДАНИЕ**

**на выполнение научно-исследовательской работы**

по теме: разработка робота-гида\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Студент группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ильин Константин Юрьевич \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Направленность НИР: учебная

Источник тематики: НИР

График выполнения НИР:     25% к \_\_\_ нед., 50% к \_\_\_ нед., 75% к \_\_ нед., 100% к \_\_\_ нед.

***Техническое задание:***

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ разработка робота-гида\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ с сетевым управлением\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ и функцией распознавания голосовых команд\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

***Оформление научно-исследовательской работы:***

Расчетно-пояснительная записка на \_\_\_ листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

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

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

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

Дата выдачи задания « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

**Руководитель НИР**  \_\_\_ . \_\_\_ .2025  Гапанюк Ю.Е.

(Подпись, дата)                          (И.О.Фамилия)

**Студент** \_\_\_. \_\_\_ .2025 Ильин К.Ю.

(Подпись, дата)                           (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

# Введение

В современном мире электронные платежи и банковские операции становятся все более популярными, что приводит к росту числа мошеннических транзакций. Обнаружение мошенничества с использованием методов машинного обучения позволяет повысить безопасность финансовых операций и минимизировать финансовые потери.

**Цель исследования** — построение и анализ моделей машинного обучения для выявления мошеннических транзакций по данным кредитных карт. В рамках работы проводится разведочный анализ данных, отбор признаков, обучение и сравнение различных моделей, а также оценка их эффективности с использованием релевантных метрик.

**Актуальность исследования:** Проблема мошенничества в финансовой сфере крайне актуальна для банков и платежных систем. Своевременное и точное обнаружение мошеннических операций позволяет снизить убытки и повысить доверие клиентов. Использование современных алгоритмов машинного обучения обеспечивает более высокую точность по сравнению с традиционными правилами и ручными проверками.

# Описание датасета

В исследовании используется открытый датасет транзакций по кредитным картам, содержащий 284 807 операций, из которых только 492 являются мошенническими (0.172%). Датасет включает 31 признак:

- `Time` — время с момента первой транзакции (в секундах)

- `V1`–`V28` — анонимизированные признаки, полученные методом главных компонент (PCA)

- `Amount` — сумма транзакции

- `Class` — целевая переменная (0 — обычная транзакция, 1 — мошенничество)

Особенности датасета:

- Сильный дисбаланс классов (мошенничество — редкое событие)

- Все признаки числовые, пропущенных значений нет

- Признаки V1–V28 уже нормализованы

# План Исследования

* Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
* Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
* Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
* Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
* Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.
* Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.
* Формирование обучающей и тестовой выборок на основе исходного набора данных.
* Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
* Подбор гиперпараметров для выбранных моделей. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
* Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
* Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

# Ход работы

*# Разведочный анализ данных*

*## 1. Импорт необходимых библиотек*

*import pandas as pd*

*import numpy as np*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*# Настройки для отображения графиков*

*plt.style.use('ggplot')*

*%matplotlib inline*

*sns.set(style="whitegrid")*

*## 2. Загрузка и первичный осмотр данных*

*# Загрузка данных*

*df = pd.read\_csv('creditcard.csv')*

*# Базовая информация о датасете*

*print("\nОбщая информация о датасете:")*

*df.info()*

*print("\nПервые 5 строк датасета:")*

*display(df.head())*

*print("\nСтатистическое описание числовых признаков:")*

*display(df.describe())*

*# Анализ распределения классов*

*class\_distribution = df['Class'].value\_counts(normalize=True)*

*print('Распределение классов:')*

*print(class\_distribution)*

*# Визуализация распределения классов*

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

*sns.countplot(data=df, x='Class')*

*plt.title('Распределение транзакций по классам')*

*plt.xlabel('Класс (0: обычная транзакция, 1: мошенничество)')*

*plt.ylabel('Количество транзакций')*

*plt.show()*

*# Точные значения*

*print('\nКоличество транзакций в каждом классе:')*

*print(df['Class'].value\_counts())*

*# Анализ распределения классов*

*class\_distribution = df['Class'].value\_counts(normalize=True)*

*print('Распределение классов (в процентах):')*

*print(class\_distribution.multiply(100).round(2))*

*print('\nКоличество транзакций в каждом классе:')*

*print(df['Class'].value\_counts())*

*print('\nОсновные выводы:')*

*print(f'- Всего транзакций: {len(df)}')*

*print(f'- Нормальных транзакций: {len(df[df["Class"]==0])} ({(class\_distribution[0]\*100):.2f}%)')*

*print(f'- Мошеннических транзакций: {len(df[df["Class"]==1])} ({(class\_distribution[1]\*100):.2f}%)')*

*print('- Набор данных сильно несбалансирован')*

*# Визуализация распределения классов*

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

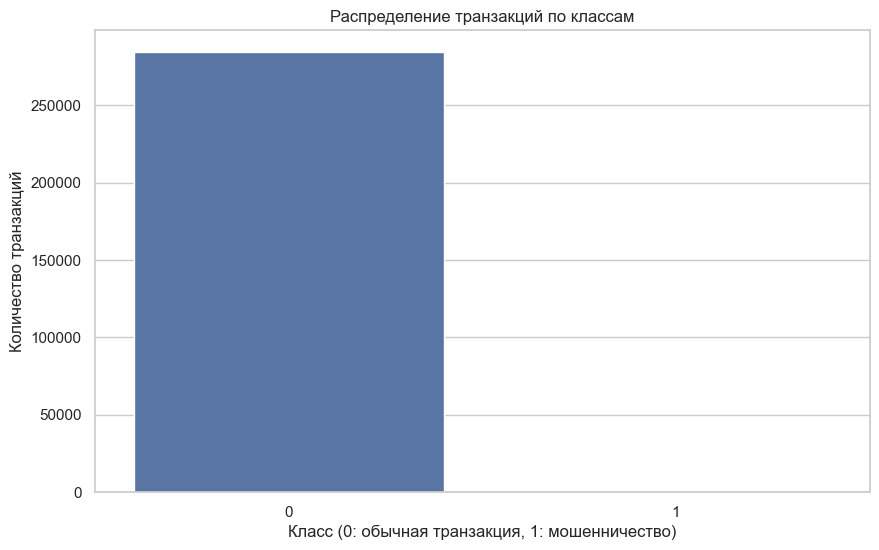
*sns.countplot(data=df, x='Class')*

*plt.title('Распределение транзакций по классам')*

*plt.xlabel('Класс (0: обычная транзакция, 1: мошенничество)')*

*plt.ylabel('Количество транзакций')*

*plt.show()*

**

*# Функция для создания графиков распределения*

*def plot\_distribution(df, columns, n\_rows, n\_cols):*

*plt.figure(figsize=(15, 5 \* n\_rows))*

*for i, column in enumerate(columns, 1):*

*# Добавляем статистику по признаку*

*print(f'\nСтатистика для признака {column}:')*

*print(df.groupby('Class')[column].describe())*

*plt.subplot(n\_rows, n\_cols, i)*

*sns.histplot(data=df, x=column, hue='Class', multiple="stack", bins=50)*

*plt.title(f'Распределение {column}')*

*plt.tight\_layout()*

*plt.show()*

*# Анализ первых 6 признаков*

*print('Анализ распределения первых 6 признаков:')*

*plot\_distribution(df, ['V1', 'V2', 'V3', 'V4', 'V5', 'V6'], 2, 3)*

*# Анализ признака Amount*

*print('\nАнализ сумм транзакций:')*

*print(df.groupby('Class')['Amount'].describe())*

*print('\nОсновные выводы по суммам транзакций:')*

*print(f'- Средняя сумма обычной транзакции: ${df[df["Class"]==0]["Amount"].mean():.2f}')*

*print(f'- Средняя сумма мошеннической транзакции: ${df[df["Class"]==1]["Amount"].mean():.2f}')*

*print(f'- Максимальная сумма обычной транзакции: ${df[df["Class"]==0]["Amount"].max():.2f}')*

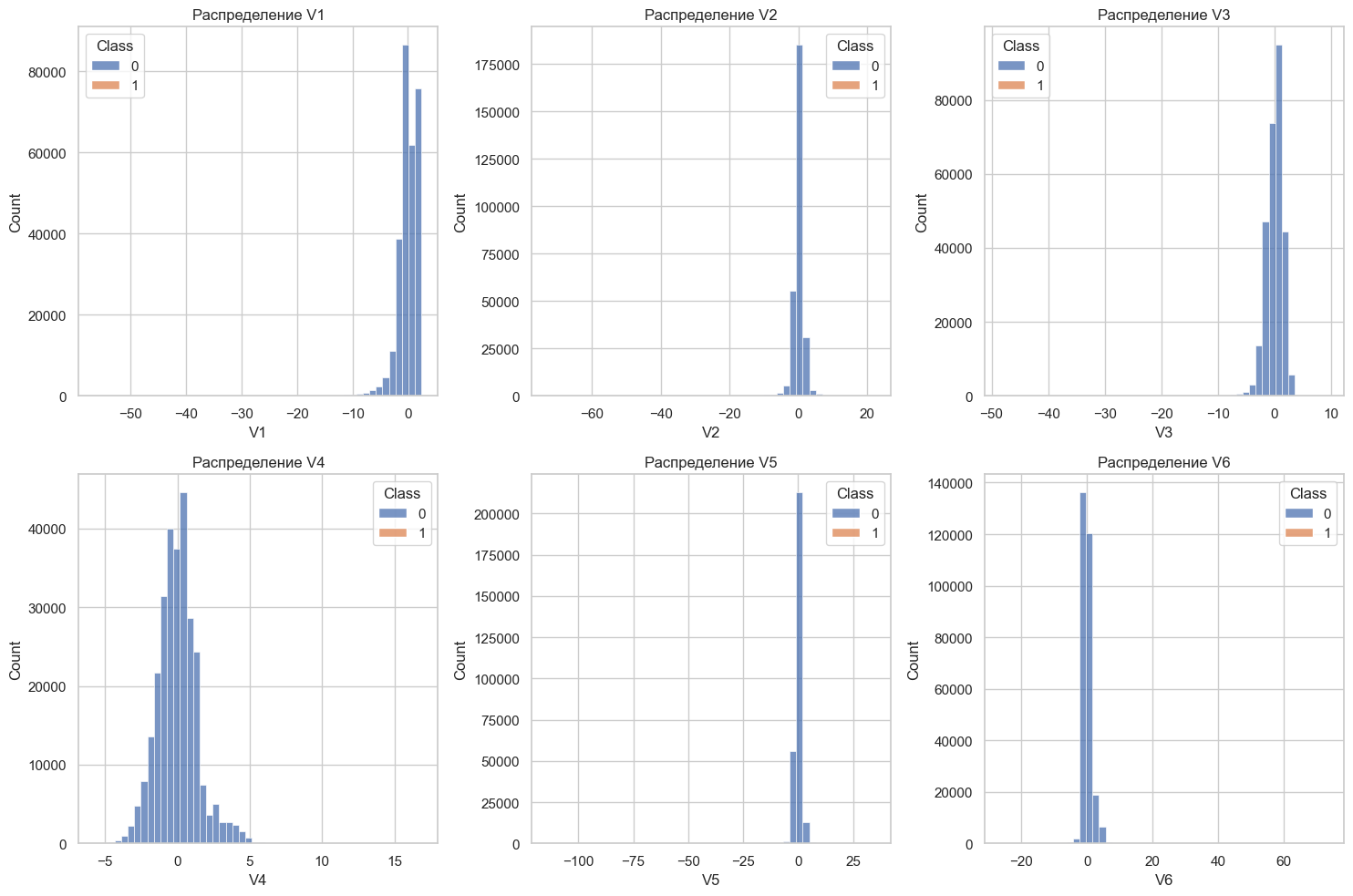
*print(f'- Максимальная сумма мошеннической транзакции: ${df[df["Class"]==1]["Amount"].max():.2f}')*

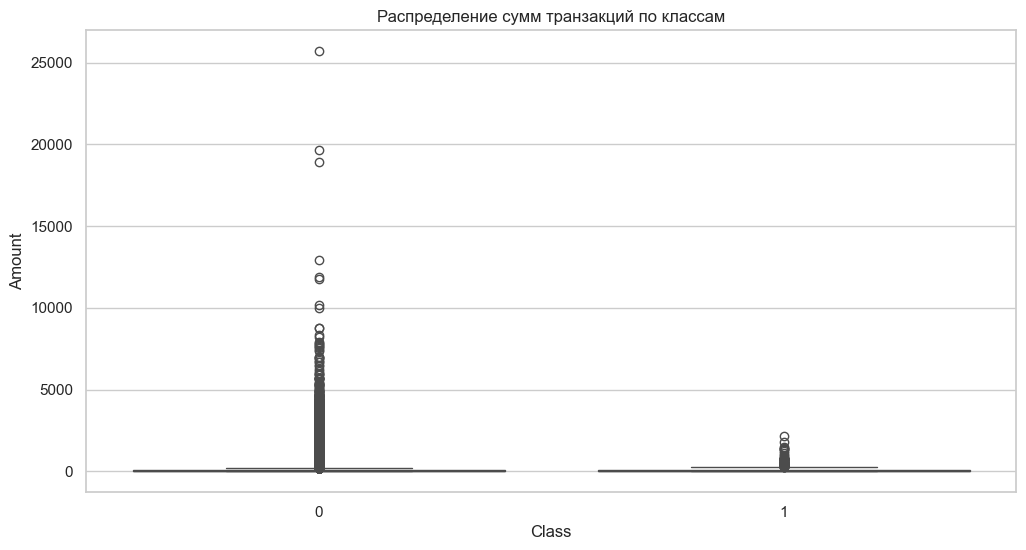
*plt.figure(figsize=(12, 6))*

*sns.boxplot(data=df, x='Class', y='Amount')*

*plt.title('Распределение сумм транзакций по классам')*

*plt.show()*

**

**

*# Расчет корреляционной матрицы*

*correlation\_matrix = df.corr()*

*# Визуализация корреляционной матрицы*

*plt.figure(figsize=(20, 16))*

*sns.heatmap(correlation\_matrix, cmap='coolwarm', center=0, annot=False)*

*plt.title('Корреляционная матрица признаков')*

*plt.show()*

*# Топ-10 признаков, наиболее коррелирующих с целевой переменной*

*correlations\_with\_target = correlation\_matrix['Class'].abs().sort\_values(ascending=False)*

*print('\nТоп-10 признаков по корреляции с целевой переменной:')*

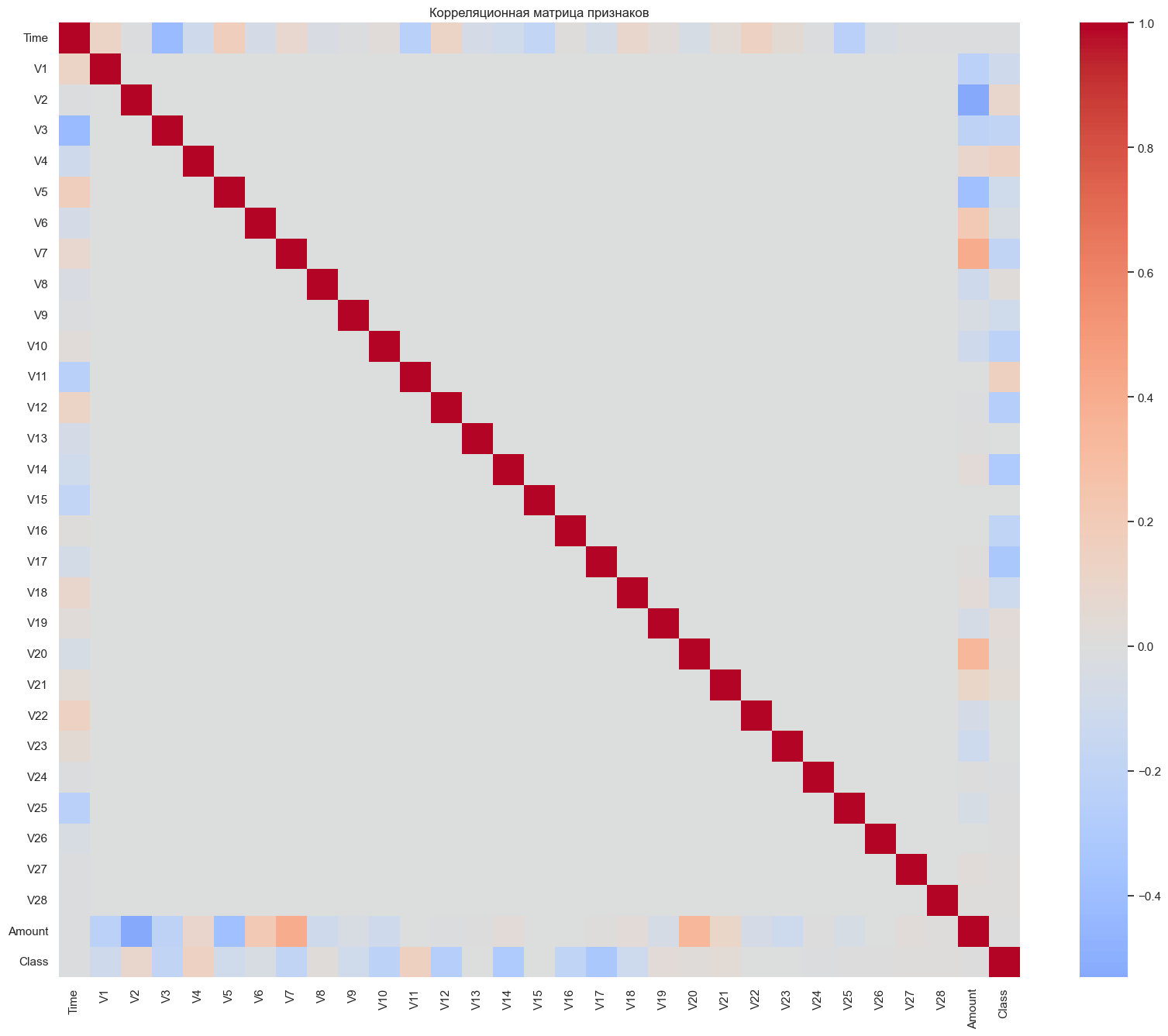
*print(correlations\_with\_target[1:11])*

*print('\nОсновные выводы:')*

*print('- Наиболее сильно коррелируют с целевой переменной следующие признаки:')*

*for feature, corr in correlations\_with\_target[1:6].items():*

*print(f' \* {feature}: {corr:.3f}')*

**

*# Функция для определения выбросов методом межквартильного размаха*

*def detect\_outliers(df, column):*

*Q1 = df[column].quantile(0.25)*

*Q3 = df[column].quantile(0.75)*

*IQR = Q3 - Q1*

*lower\_bound = Q1 - 1.5 \* IQR*

*upper\_bound = Q3 + 1.5 \* IQR*

*outliers = df[(df[column] < lower\_bound) | (df[column] > upper\_bound)]*

*return len(outliers), lower\_bound, upper\_bound*

*# Анализ выбросов в числовых признаках*

*numeric\_columns = df.select\_dtypes(include=['float64']).columns*

*outliers\_info = {col: detect\_outliers(df, col) for col in numeric\_columns}*

*print('Анализ выбросов:')*

*for col, (count, lower, upper) in outliers\_info.items():*

*print(f'\n{col}:')*

*print(f'- Количество выбросов: {count}')*

*print(f'- Нижняя граница: {lower:.2f}')*

*print(f'- Верхняя граница: {upper:.2f}')*

*print(f'- Процент выбросов: {(count/len(df)\*100):.2f}%')*

*# Визуализация количества выбросов*

*plt.figure(figsize=(15, 6))*

*plt.bar(outliers\_info.keys(), [count for count, \_, \_ in outliers\_info.values()])*

*plt.xticks(rotation=90)*

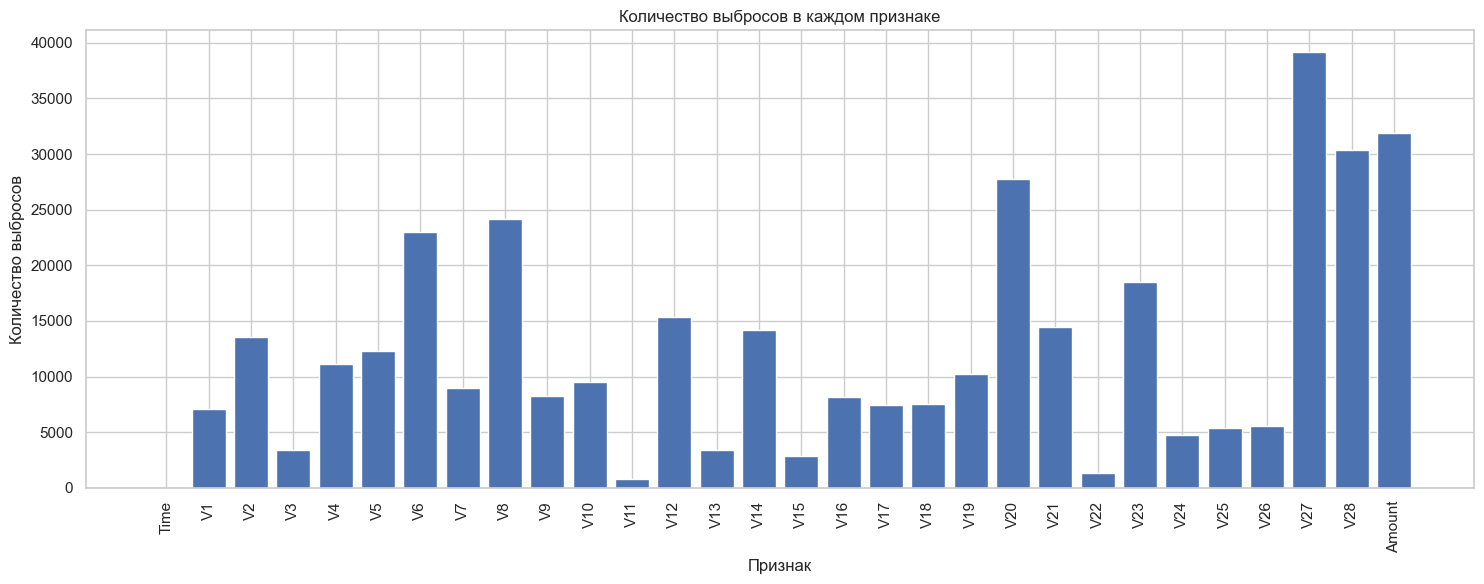
*plt.title('Количество выбросов в каждом признаке')*

*plt.xlabel('Признак')*

*plt.ylabel('Количество выбросов')*

*plt.tight\_layout()*

*plt.show()*

**

*# Анализ распределения транзакций во времени*

*print('Анализ временного распределения транзакций:')*

*print('\nСтатистика по времени для всех транзакций:')*

*print(df.groupby('Class')['Time'].describe())*

*# Конвертация времени в часы для лучшего понимания*

*df['Time\_Hour'] = df['Time'] / 3600*

*print('\nРаспределение по часам:')*

*print(f'- Минимальное время: {df["Time\_Hour"].min():.2f} часов')*

*print(f'- Максимальное время: {df["Time\_Hour"].max():.2f} часов')*

*print(f'- Общая продолжительность: {df["Time\_Hour"].max() - df["Time\_Hour"].min():.2f} часов')*

*plt.figure(figsize=(15, 6))*

*sns.histplot(data=df, x='Time', hue='Class', bins=50)*

*plt.title('Распределение транзакций во времени')*

*plt.xlabel('Время (в секундах от начала)')*

*plt.ylabel('Количество транзакций')*

*plt.show()*

*# Анализ мошеннических транзакций во времени*

*fraud\_df = df[df['Class'] == 1]*

*print('\nАнализ мошеннических транзакций:')*

*print(f'- Количество мошеннических транзакций: {len(fraud\_df)}')*

*print(f'- Среднее время между мошенническими транзакциями: {(df["Time"].max() - df["Time"].min())/len(fraud\_df)/3600:.2f} часов')*

*plt.figure(figsize=(15, 6))*

*sns.histplot(data=fraud\_df, x='Time', bins=50)*

*plt.title('Распределение мошеннических транзакций во времени')*

*plt.xlabel('Время (в секундах от начала)')*

*plt.ylabel('Количество мошеннических транзакций')*

*plt.show()*

**

**

*# Статистика по суммам транзакций для каждого класса*

*print('Статистика сумм транзакций по классам:')*

*print(df.groupby('Class')['Amount'].describe())*

*print('\nДополнительная статистика по суммам:')*

*print('Нормальные транзакции:')*

*normal\_amounts = df[df['Class'] == 0]['Amount']*

*print(f'- Медианная сумма: ${normal\_amounts.median():.2f}')*

*print(f'- 95-й перцентиль: ${normal\_amounts.quantile(0.95):.2f}')*

*print(f'- Стандартное отклонение: ${normal\_amounts.std():.2f}')*

*print('\nМошеннические транзакции:')*

*fraud\_amounts = df[df['Class'] == 1]['Amount']*

*print(f'- Медианная сумма: ${fraud\_amounts.median():.2f}')*

*print(f'- 95-й перцентиль: ${fraud\_amounts.quantile(0.95):.2f}')*

*print(f'- Стандартное отклонение: ${fraud\_amounts.std():.2f}')*

*# Визуализация распределения сумм*

*plt.figure(figsize=(15, 6))*

*sns.histplot(data=df, x='Amount', hue='Class', bins=50)*

*plt.title('Распределение сумм транзакций')*

*plt.xlabel('Сумма транзакции')*

*plt.ylabel('Количество транзакций')*

*plt.show()*

*# Логарифмическая шкала для лучшей визуализации*

*plt.figure(figsize=(15, 6))*

*sns.histplot(data=df[df['Amount'] > 0], x='Amount', hue='Class', bins=50, log\_scale=True)*

*plt.title('Распределение сумм транзакций (логарифмическая шкала)')*

*plt.xlabel('Сумма транзакции (log scale)')*

*plt.ylabel('Количество транзакций')*

*plt.show()*

*# 📊 Разведочный анализ данных (EDA)*

*## 📁 Структура данных*

*- \*\*Датасет\*\* содержит \*\*284,807 транзакций\*\**

*- \*\*31 признак\*\*:*

*- `Time`, `V1`–`V28`, `Amount`, `Class`*

*- Все признаки числовые:*

*- `30` — `float64`*

*- `1` — `int64`*

*- \*\*Пропущенных значений нет\*\**

*## ⚠️ Особенности данных*

*- \*\*Сильный дисбаланс классов\*\* (`Class`)*

*- Признаки `V1`–`V28` уже \*\*нормализованы\*\* (среднее близко к 0)*

*- Признак `Amount` содержит \*\*выбросы\*\**

*- `Time` представлен в \*\*секундах\*\* с начала сбора данных*

*## 🧪 Предварительный анализ*

*- Проведен \*\*базовый статистический анализ\*\**

*- Подготовлена \*\*визуализация распределений\*\**

*- Начат \*\*анализ временного ряда\*\**

*> 📌 \*\*Примечание:\*\**

*> Данный файл представляет собой \*\*начальный этап разведочного анализа данных (EDA)\*\*.*

*> Этапы, заявленные в изначальных выводах (например, \*\*предобработка, применение SMOTE, рекомендации по моделированию\*\*), \*\*в нём отсутствуют\*\*.*

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

*from sklearn.preprocessing import StandardScaler*

*# Разделение признаков и целевой переменной*

*X = df.drop('Class', axis=1)*

*y = df['Class']*

*# Анализ важности признаков*

*selector = SelectKBest(score\_func=f\_classif, k='all')*

*selector.fit(X, y)*

*# Создание DataFrame с результатами*

*feature\_scores = pd.DataFrame({*

*'Feature': X.columns,*

*'Score': selector.scores\_*

*})*

*# Сортировка признаков по важности*

*feature\_scores = feature\_scores.sort\_values('Score', ascending=False)*

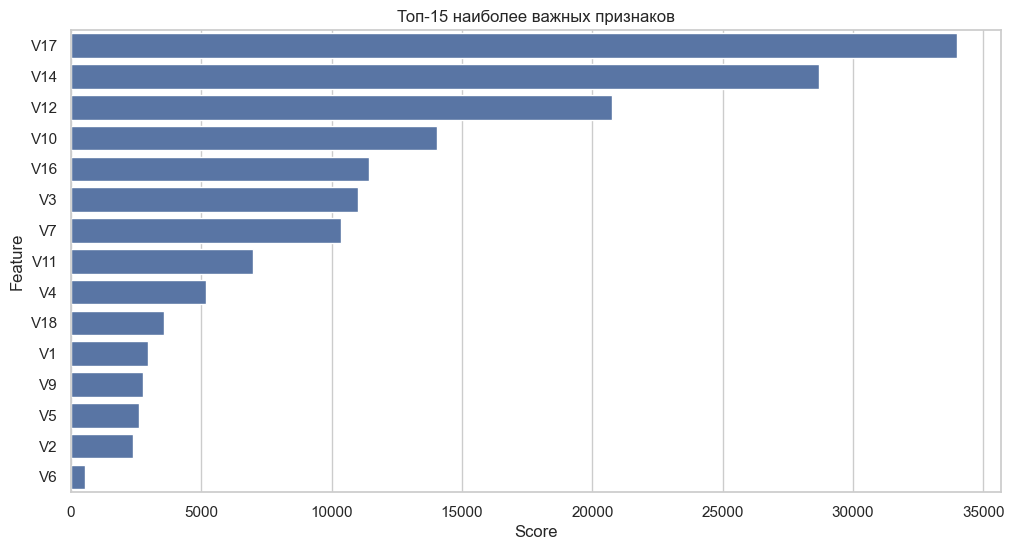
*# Визуализация важности признаков*

*plt.figure(figsize=(12, 6))*

*sns.barplot(data=feature\_scores.head(15), x='Score', y='Feature')*

*plt.title('Топ-15 наиболее важных признаков')*

*plt.show()*

**

*# Преобразование временной метки в часы и дни недели*

*df['Hour'] = df['Time'].apply(lambda x: (x / 3600) % 24)*

*df['Day'] = df['Time'].apply(lambda x: (x / (3600 \* 24)) % 7)*

*# Создание признаков на основе Amount*

*df['Amount\_Log'] = np.log1p(df['Amount'])*

*df['Amount\_Bin'] = pd.qcut(df['Amount'], q=10, labels=False)*

*# Создание взаимодействий между важными признаками*

*important\_features = feature\_scores['Feature'].head(5).tolist()*

*for i in range(len(important\_features)):*

*for j in range(i+1, len(important\_features)):*

*feat1, feat2 = important\_features[i], important\_features[j]*

*df[f'{feat1}\_{feat2}\_mult'] = df[feat1] \* df[feat2]*

*df[f'{feat1}\_{feat2}\_sum'] = df[feat1] + df[feat2]*

*# Обновление набора признаков*

*X = df.drop(['Class', 'Time'], axis=1) # удаляем исходную временную метку*

*# Масштабирование всех признаков*

*scaler = StandardScaler()*

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

*X\_scaled = pd.DataFrame(X\_scaled, columns=X.columns)*

*# Проверка результатов масштабирования*

*print('Статистика масштабированных данных:')*

*print(X\_scaled.describe().round(2))*

*# Отбор наиболее важных признаков*

*top\_features = feature\_scores['Feature'].head(15).tolist()*

*important\_new\_features = ['Hour', 'Amount\_Log'] # добавляем созданные признаки*

*final\_features = top\_features + important\_new\_features*

*# Формирование итогового набора данных*

*X\_final = X\_scaled[final\_features]*

*# Сохранение подготовленных данных*

*prepared\_data = pd.concat([X\_final, y], axis=1)*

*prepared\_data.to\_csv('prepared\_data.csv', index=False)*

*# Расчет корреляционной матрицы*

*correlation\_matrix = df.corr()*

*# Визуализация полной корреляционной матрицы*

*plt.figure(figsize=(20, 16))*

*sns.heatmap(correlation\_matrix, cmap='coolwarm', center=0, annot=False)*

*plt.title('Корреляционная матрица всех признаков')*

*plt.show()*

*# Анализ корреляций с целевой переменной*

*correlations\_with\_target = correlation\_matrix['Class'].abs().sort\_values(ascending=False)*

*print('\nТоп-10 признаков по корреляции с целевой переменной (Class):')*

*print(correlations\_with\_target[1:11])*

*# Визуализация топ-10 корреляций с целевой переменной*

*plt.figure(figsize=(12, 6))*

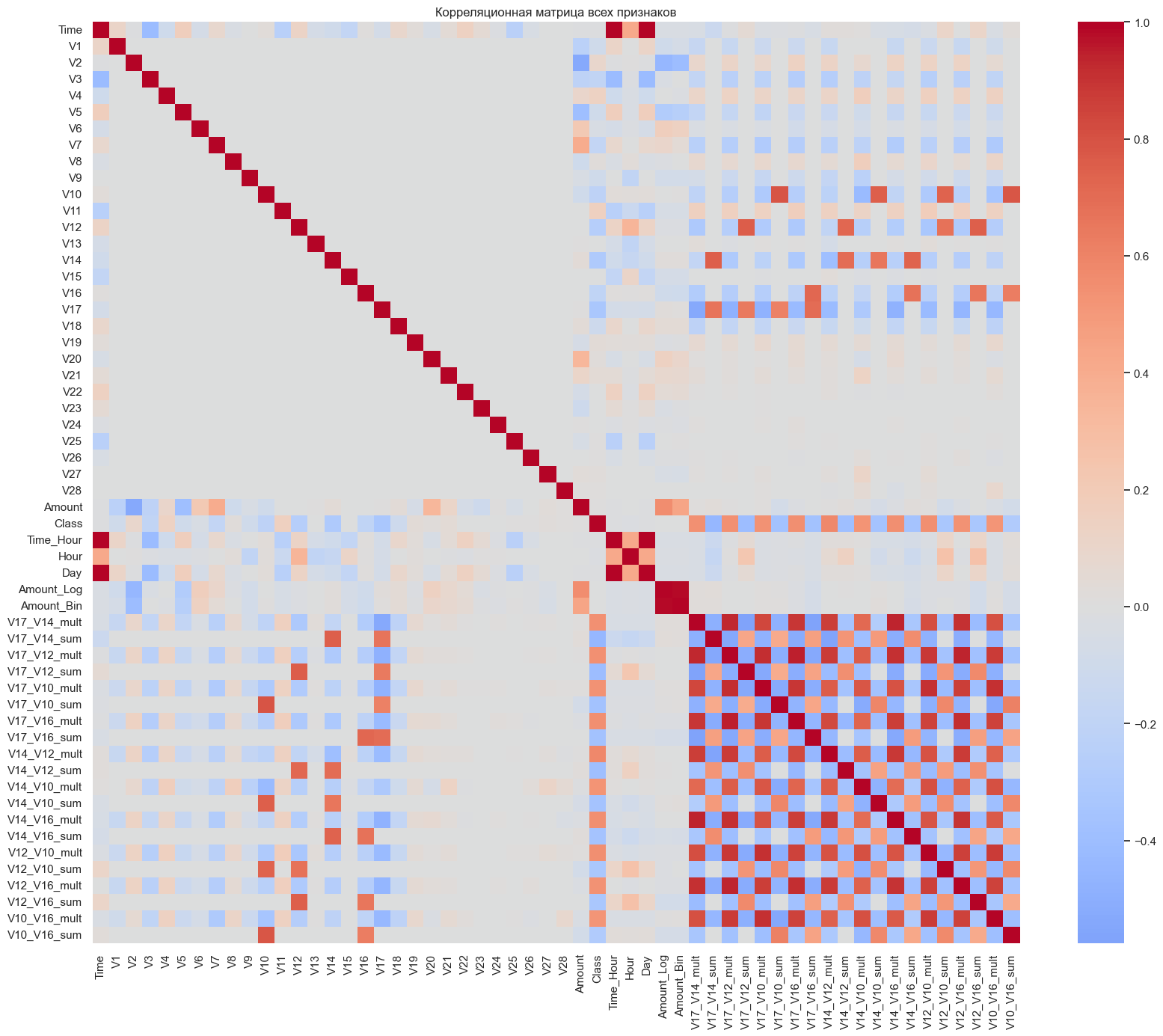
*sns.barplot(x=correlations\_with\_target[1:11].values,*

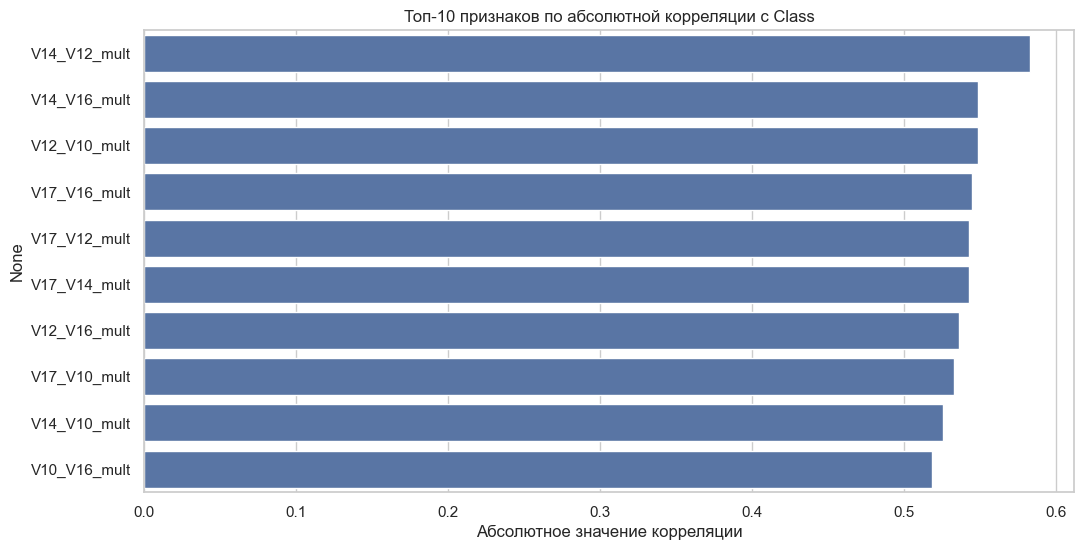
*y=correlations\_with\_target[1:11].index)*

*plt.title('Топ-10 признаков по абсолютной корреляции с Class')*

*plt.xlabel('Абсолютное значение корреляции')*

*plt.show()*

**

**

*# Функция для поиска сильно коррелирующих признаков*

*def get\_high\_correlations(correlation\_matrix, threshold=0.8):*

*high\_corr = []*

*for i in range(len(correlation\_matrix.columns)):*

*for j in range(i):*

*if abs(correlation\_matrix.iloc[i, j]) > threshold:*

*high\_corr.append({*

*'feature1': correlation\_matrix.columns[i],*

*'feature2': correlation\_matrix.columns[j],*

*'correlation': correlation\_matrix.iloc[i, j]*

*})*

*return pd.DataFrame(high\_corr)*

*# Поиск сильно коррелирующих признаков*

*high\_correlations = get\_high\_correlations(correlation\_matrix)*

*if not high\_correlations.empty:*

*print('Пары признаков с высокой корреляцией (>0.8):')*

*print(high\_correlations)*

*else:*

*print('Нет пар признаков с корреляцией выше 0.8')*

*# Выбор топ-5 признаков для анализа нелинейных зависимостей*

*top\_features = correlations\_with\_target[1:6].index*

*# Создание графиков рассеяния для топ признаков*

*plt.figure(figsize=(15, 10))*

*for i, feature in enumerate(top\_features, 1):*

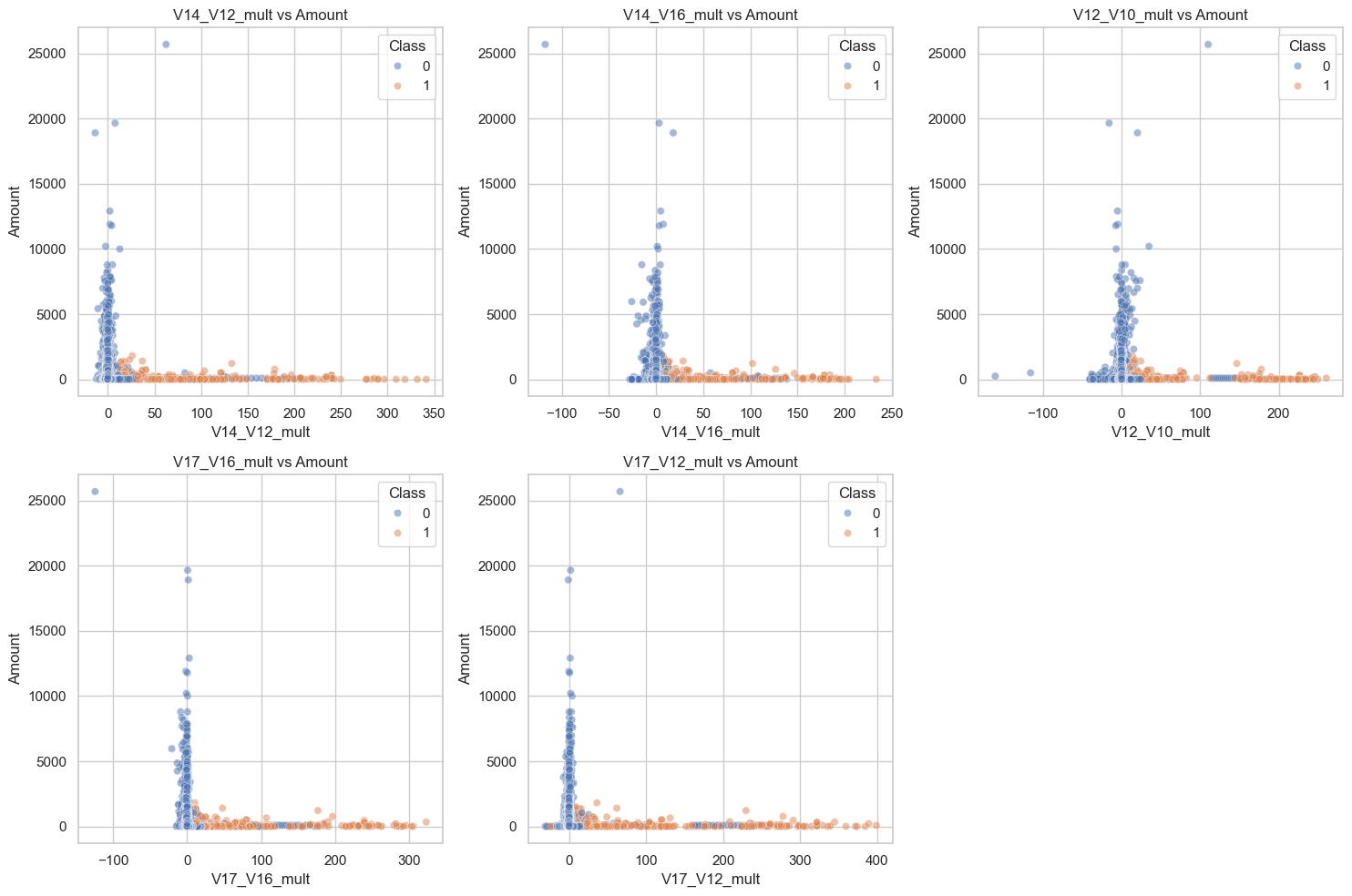
*plt.subplot(2, 3, i)*

*sns.scatterplot(data=df, x=feature, y='Amount', hue='Class', alpha=0.5)*

*plt.title(f'{feature} vs Amount')*

*plt.tight\_layout()*

*plt.show()*

**

*# ✅ Промежуточные выводы о возможности построения моделей машинного обучения*

*---*

*## 📊 1. Характеристики набора данных*

*### ➕ Положительные аспекты:*

*- 📈 \*\*Большой объем данных\*\* — 284,807 транзакций*

*- ❌ \*\*Отсутствие пропущенных значений\*\**

*- ⚙️ \*\*Предобработанные признаки\*\*: `V1`–`V28` получены через \*\*PCA\*\**

*- 🔢 \*\*Все признаки числовые\*\* и \*\*нормализованы\*\**

*### ⚠️ Проблемные аспекты:*

*- 📉 \*\*Сильный дисбаланс классов\*\* (мошеннические транзакции — меньшинство)*

*- 🚨 \*\*Выбросы\*\* в признаке `Amount`*

*- ⏱️ \*\*Временной признак\*\* требует дополнительной обработки*

*---*

*## 🤖 2. Возможности для построения моделей*

*### 🧠 Подходящие типы моделей:*

*#### 🌲 Ансамблевые методы:*

*- Random Forest*

*- XGBoost / LightGBM*

*- AdaBoost*

*#### 📉 Линейные модели:*

*- Логистическая регрессия с регуляризацией*

*- SVM с различными ядрами*

*#### 🔍 Нейронные сети:*

*- Многослойный перцептрон (MLP)*

*- Автоэнкодеры для \*\*обнаружения аномалий\*\**

*---*

*## 🛠️ 3. Необходимые шаги предобработки*

*### ⚖️ Работа с дисбалансом классов:*

*- SMOTE / ADASYN*

*- Взвешивание классов*

*- Стратификация при разделении выборок*

*### 🧼 Обработка признаков:*

*- Масштабирование `Amount`*

*- Преобразование `Time`*

*- Создание новых признаков*

*### 🔍 Отбор признаков:*

*- Оценка важности*

*- Анализ корреляций*

*- Методы снижения размерности (PCA, t-SNE и др.)*

*---*

*## 🧪 4. Рекомендации по оценке моделей*

*### 📏 Метрики:*

*- Precision-Recall AUC*

*- F1-score*

*- ROC AUC*

*- Матрица ошибок (Confusion Matrix)*

*### 🔄 Валидация:*

*- Стратифицированная кросс-валидация*

*- Отдельный тестовый набор*

*- Анализ переобучения*

*---*

*## ⚙️ 5. Ожидаемые сложности*

*### 🧱 Технические вызовы:*

*- Баланс между точностью и полнотой*

*- Работа с выбросами*

*- Тюнинг гиперпараметров*

*### 💼 Бизнес-аспекты:*

*- Минимизация ложных срабатываний*

*- Быстрое время отклика*

*- Интерпретируемость моделей*

*---*

*## 🧾 6. Заключение*

*✅ \*\*Данные пригодны\*\* для построения моделей машинного обучения.*

*### 📌 Основные преимущества:*

*- 📊 Большой объем данных*

*- 🧹 Качественная предварительная обработка*

*- ❌ Отсутствие пропусков*

*### 🔧 Необходимо:*

*1. Тщательно решить проблему дисбаланса классов*

*2. Обработать временные признаки и суммы транзакций*

*3. Использовать ансамбли и кросс-валидацию*

*4. Применять метрики, учитывающие дисбаланс*

*🎯 \*\*При правильном подходе можно достичь высоких результатов в выявлении мошеннических транзакций.\*\**

*# 📏 Выбор и обоснование метрик оценки качества моделей*

*---*

*## 🎯 1. Особенности задачи, влияющие на выбор метрик*

*### ⚠️ Сильный дисбаланс классов*

*- Мошеннические транзакции составляют \*\*0.172%\*\* от всех*

*- Стандартная метрика \*\*Accuracy\*\* неприменима (может давать >99% даже без обнаружения мошенничества)*

*### 💰 Высокая цена ошибок*

*- 🔴 \*\*FP (ложноположительные)\*\* — неудобства для клиентов (блокировки, проверки)*

*- 🔵 \*\*FN (ложноотрицательные)\*\* — \*\*финансовые потери\*\* для компании*

*### 📊 Необходимость ранжирования предсказаний*

*- Требуется \*\*оценка вероятности\*\*, чтобы:*

*- Установить пороги чувствительности*

*- Приоритизировать ручные проверки*

*---*

*## 🧪 2. Выбранные метрики*

*### 📈 2.1 PR-AUC (Precision-Recall AUC)*

*\*\*Почему выбрана:\*\**

*- Устойчивее к дисбалансу, чем ROC-AUC*

*- Ориентирована на \*\*обнаружение положительного класса\*\**

*- Независима от доли TN (нормальных транзакций)*

*\*\*Формулы:\*\**

*- `Precision = TP / (TP + FP)`*

*- `Recall = TP / (TP + FN)`*

*- `PR-AUC` = площадь под кривой Precision-Recall*

*---*

*### ⚖️ 2.2 F1-score*

*\*\*Почему выбрана:\*\**

*- Сбалансированная метрика, сочетающая precision и recall*

*- Подходит для \*\*неравных классов\*\**

*- Даёт \*\*одну обобщающую цифру\*\**

*\*\*Формула:\*\**

*- `F1 = 2 \* (Precision \* Recall) / (Precision + Recall)`*

*---*

*### 🧮 2.3 Матрица ошибок (Confusion Matrix)*

*\*\*Почему выбрана:\*\**

*- Позволяет \*\*разделить\*\* и проанализировать \*\*все типы ошибок\*\**

*- Удобна для визуализации и анализа после настройки порога*

*\*\*Ключевые производные метрики:\*\**

*- `FPR = FP / (FP + TN)` — уровень ложных тревог*

*- `FNR = FN / (TP + FN)` — упущенные мошенничества*

*- `TPR = Recall = TP / (TP + FN)` — чувствительность*

*---*

*## 🔍 Дополнительные соображения*

*### 🧭 Порог принятия решения*

*- Настройка на основе \*\*PR-кривой\*\**

*- Баланс между precision и recall в зависимости от бизнес-задачи*

*### 📊 Бизнес-метрики*

*- Финансовый ущерб от FN*

*- Стоимость обработки FP*

*### 🔄 Валидация*

*- \*\*Кросс-валидация\*\* (stratified k-fold)*

*- Сохранение пропорций классов в выборках*

*---*

*## ✅ Вывод*

*Выбранные метрики обеспечивают \*\*всестороннюю и надежную оценку моделей\*\*:*

*- 📌 \*\*PR-AUC\*\* и \*\*F1-score\*\* — для оценки качества предсказаний в условиях дисбаланса*

*- 🧮 \*\*Confusion Matrix\*\* — для анализа ошибок и настройки порога*

*- 📊 Подход учитывает как \*\*технические\*\*, так и \*\*бизнес-аспекты\*\* задачи*

*> Эти метрики обеспечат надежную оценку эффективности моделей в задаче обнаружения мошеннических транзакций.*

*from sklearn.metrics import precision\_recall\_curve, average\_precision\_score*

*from sklearn.metrics import f1\_score, confusion\_matrix*

*import numpy as np*

*def evaluate\_model(y\_true, y\_pred, y\_pred\_proba=None):*

*"""Функция для расчета всех выбранных метрик"""*

*# Confusion Matrix и производные метрики*

*cm = confusion\_matrix(y\_true, y\_pred)*

*tn, fp, fn, tp = cm.ravel()*

*# Базовые метрики*

*precision = tp / (tp + fp)*

*recall = tp / (tp + fn)*

*f1 = f1\_score(y\_true, y\_pred)*

*# Дополнительные метрики*

*fpr = fp / (fp + tn)*

*fnr = fn / (fn + tp)*

*# PR-AUC (если доступны вероятности)*

*pr\_auc = None*

*if y\_pred\_proba is not None:*

*pr\_auc = average\_precision\_score(y\_true, y\_pred\_proba)*

*# Вывод результатов*

*print('Метрики качества модели:')*

*print(f'PR-AUC: {pr\_auc:.4f}' if pr\_auc is not None else 'PR-AUC: Недоступно')*

*print(f'F1-score: {f1:.4f}')*

*print('\nConfusion Matrix:')*

*print(f'True Negatives: {tn}')*

*print(f'False Positives: {fp}')*

*print(f'False Negatives: {fn}')*

*print(f'True Positives: {tp}')*

*print('\nДополнительные метрики:')*

*print(f'Precision: {precision:.4f}')*

*print(f'Recall: {recall:.4f}')*

*print(f'False Positive Rate: {fpr:.4f}')*

*print(f'False Negative Rate: {fnr:.4f}')*

*return {*

*'pr\_auc': pr\_auc,*

*'f1': f1,*

*'precision': precision,*

*'recall': recall,*

*'fpr': fpr,*

*'fnr': fnr,*

*'confusion\_matrix': cm*

*}*

*1. \*\*Логистическая регрессия (Logistic Regression)\*\**

*- Базовая модель для бинарной классификации*

*- Преимущества:*

*- Хорошая интерпретируемость результатов*

*- Быстрое обучение и предсказание*

*- Может работать с большими наборами данных*

*- Недостатки:*

*- Может не уловить нелинейные зависимости*

*- Чувствительна к масштабу признаков*

*2. \*\*Дерево решений (Decision Tree)\*\**

*- Преимущества:*

*- Хорошая интерпретируемость*

*- Может работать с нелинейными зависимостями*

*- Не требует масштабирования признаков*

*- Недостатки:*

*- Склонность к переобучению*

*- Может быть нестабильным*

*3. \*\*Support Vector Machine (SVM)\*\**

*- Преимущества:*

*- Эффективен в пространствах высокой размерности*

*- Хорошо работает с нелинейными зависимостями (при использовании kernel trick)*

*- Устойчив к выбросам*

*- Недостатки:*

*- Требует масштабирования признаков*

*- Может быть медленным на больших наборах данных*

*Ансамблевые модели:*

*4. \*\*Random Forest\*\**

*- Преимущества:*

*- Устойчивость к переобучению*

*- Автоматический отбор важных признаков*

*- Хорошая работа с несбалансированными данными*

*- Параллельное обучение*

*- Недостатки:*

*- Менее интерпретируемый, чем одиночное дерево*

*- Требует больше вычислительных ресурсов*

*5. \*\*XGBoost (eXtreme Gradient Boosting)\*\**

*- Преимущества:*

*- Высокая производительность*

*- Встроенная регуляризация*

*- Эффективная обработка пропущенных значений*

*- Возможность работы с несбалансированными данными*

*- Недостатки:*

*- Сложнее в настройке гиперпараметров*

*- Менее интерпретируемый*

*- Может быть склонен к переобучению при неправильной настройке*

*Особенности реализованных моделей для задачи:*

*1. \*\*Логистическая регрессия\*\* служит базовой моделью и отправной точкой для сравнения.*

*2. \*\*Дерево решений\*\* поможет понять структуру данных и важность признаков.*

*3. \*\*SVM\*\* с RBF ядром может обнаружить сложные нелинейные паттерны мошенничества.*

*4. \*\*Random Forest\*\* как ансамблевый метод обеспечит стабильность и устойчивость к шуму.*

*5. \*\*XGBoost\*\* как продвинутый бустинг-алгоритм часто показывает лучшие результаты на практике.*

*Рекомендации по дальнейшей оптимизации:*

*1. Провести поиск по сетке (GridSearchCV) для оптимизации гиперпараметров*

*2. Добавить кросс-валидацию для более надежной оценки*

*3. Реализовать стекинг или блендинг этих моделей*

*4. Добавить feature importance analysis для Random Forest и XGBoost*

*import numpy as np*

*import pandas as pd*

*from sklearn.model\_selection import train\_test\_split, cross\_val\_score*

*from sklearn.preprocessing import StandardScaler*

*from sklearn.linear\_model import LogisticRegression*

*from sklearn.tree import DecisionTreeClassifier*

*from sklearn.svm import SVC*

*from sklearn.ensemble import RandomForestClassifier*

*from xgboost import XGBClassifier*

*from sklearn.metrics import precision\_recall\_curve, auc, f1\_score, confusion\_matrix*

*from sklearn.metrics import classification\_report*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*class FraudDetectionModels:*

*def \_\_init\_\_(self):*

*self.models = {*

*'Logistic Regression': LogisticRegression(class\_weight='balanced', random\_state=42),*

*'Decision Tree': DecisionTreeClassifier(class\_weight='balanced', random\_state=42),*

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

*'Random Forest': RandomForestClassifier(class\_weight='balanced', n\_estimators=100, random\_state=42),*

*'XGBoost': XGBClassifier(scale\_pos\_weight=30, n\_estimators=100, random\_state=42)*

*}*

*self.scaler = StandardScaler()*

*def prepare\_data(self, data):*

*X = data.iloc[:, :-1]*

*y = data.iloc[:, -1]*

*# Масштабирование признаков*

*X\_scaled = self.scaler.fit\_transform(X)*

*# Разделение на обучающую и тестовую выборки*

*return train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42, stratify=y)*

*def train\_and\_evaluate(self, X\_train, X\_test, y\_train, y\_test):*

*results = {}*

*for name, model in self.models.items():*

*print(f"\nTraining {name}...")*

*# Обучение модели*

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

*# Предсказания*

*y\_pred = model.predict(X\_test)*

*# Оценка производительности*

*results[name] = {*

*'classification\_report': classification\_report(y\_test, y\_pred),*

*'confusion\_matrix': confusion\_matrix(y\_test, y\_pred),*

*'f1': f1\_score(y\_test, y\_pred)*

*}*

*# Для PR-AUC нужны вероятности*

*if hasattr(model, "predict\_proba"):*

*y\_scores = model.predict\_proba(X\_test)[:, 1]*

*precision, recall, \_ = precision\_recall\_curve(y\_test, y\_scores)*

*results[name]['pr\_auc'] = auc(recall, precision)*

*print(f"\nResults for {name}:")*

*print(results[name]['classification\_report'])*

*return results*

*def plot\_confusion\_matrices(self, results):*

*fig, axes = plt.subplots(2, 3, figsize=(15, 10))*

*axes = axes.ravel()*

*for idx, (name, result) in enumerate(results.items()):*

*if idx < len(axes):*

*sns.heatmap(result['confusion\_matrix'], annot=True, fmt='d', ax=axes[idx])*

*axes[idx].set\_title(f'{name} Confusion Matrix')*

*plt.tight\_layout()*

*plt.show()*

*def compare\_models(self, results):*

*# Сравнение F1-scores*

*f1\_scores = {name: result['f1'] for name, result in results.items()}*

*plt.figure(figsize=(10, 5))*

*plt.bar(f1\_scores.keys(), f1\_scores.values())*

*plt.title('F1 Score Comparison')*

*plt.xticks(rotation=45)*

*plt.tight\_layout()*

*plt.show()*

*# Пример использования:*

*if \_\_name\_\_ == "\_\_main\_\_":*

*data = pd.read\_csv('creditcard.csv')*

*# Инициализация класса с моделями*

*fraud\_detection = FraudDetectionModels()*

*# Подготовка данных*

*X\_train, X\_test, y\_train, y\_test = fraud\_detection.prepare\_data(data)*

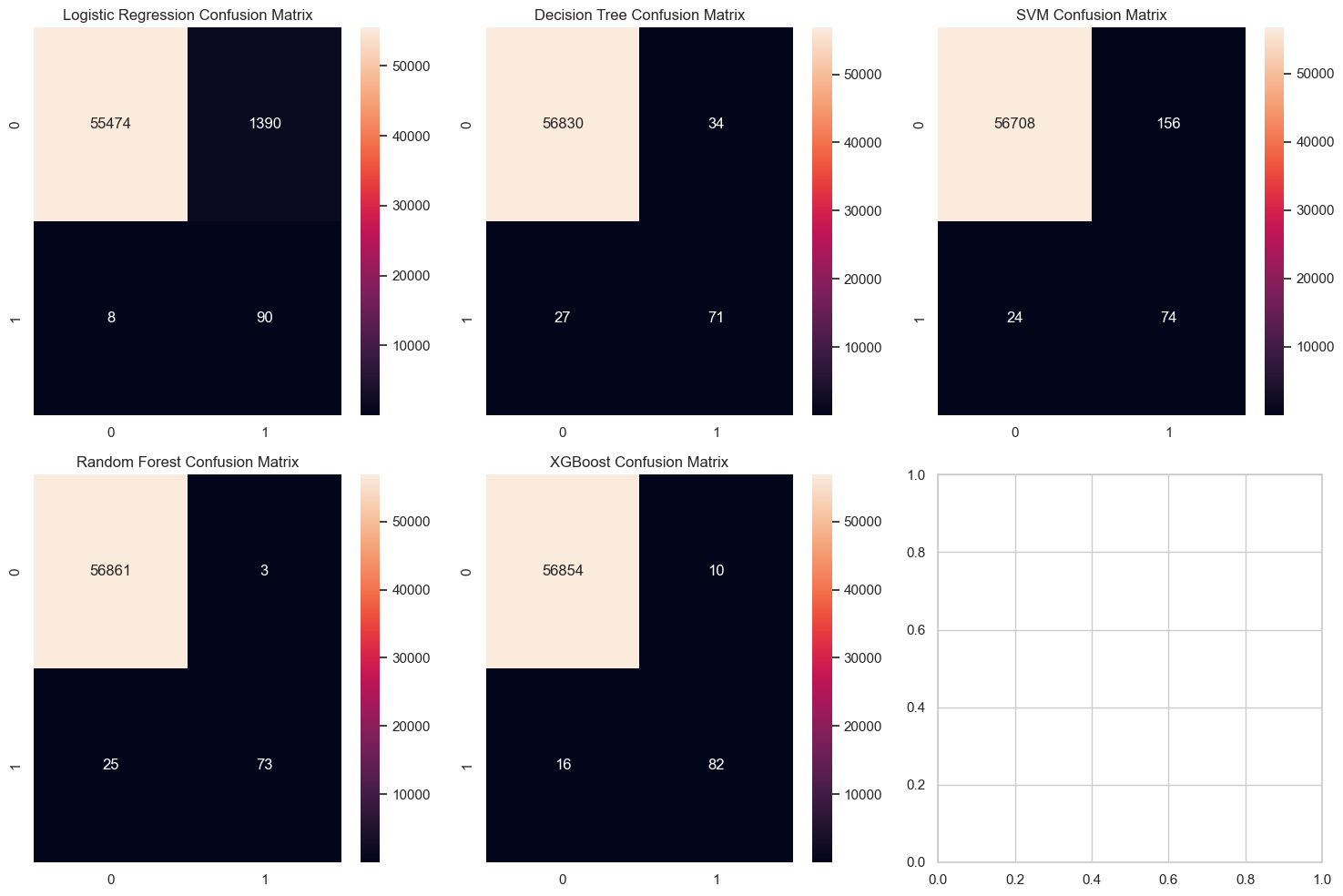
*# Обучение и оценка моделей*

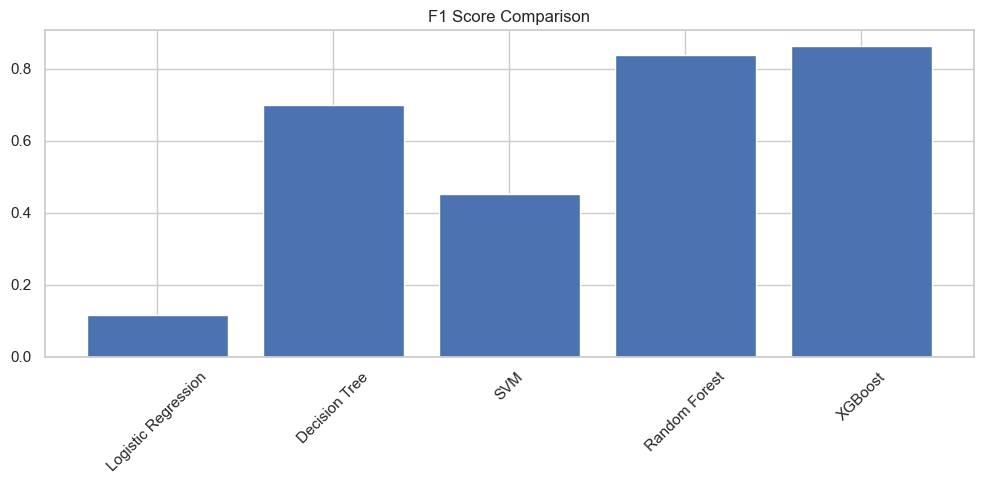
*results = fraud\_detection.train\_and\_evaluate(X\_train, X\_test, y\_train, y\_test)*

*# Визуализация результатов*

*fraud\_detection.plot\_confusion\_matrices(results)*

*fraud\_detection.compare\_models(results)*

**

**

*Предобработка данных:*

*Стандартизация признаков с помощью StandardScaler*

*Стратифицированное разделение на обучающую и тестовую выборки (80/20)*

*Настройка моделей:*

*Все модели настроены с учетом несбалансированности классов*

*Использованы веса классов (class\_weight='balanced') для большинства моделей*

*Для XGBoost установлен параметр scale\_pos\_weight для работы с несбалансированными данными*

*Метрики оценки:*

*F1-score*

*Confusion Matrix*

*Classification Report (precision, recall, f1-score для каждого класса)*

*PR-AUC (для моделей, поддерживающих вероятностные предсказания)*

*import pandas as pd*

*import numpy as np*

*from model\_selection import FraudDetectionModels*

*from sklearn.metrics import precision\_recall\_curve, average\_precision\_score*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*def evaluate\_baseline\_models():*

*# Загрузка данных*

*print("Loading data...")*

*data = pd.read\_csv('creditcard.csv')*

*# Вывод информации о датасете*

*print("\nDataset Info:")*

*print(f"Total samples: {len(data)}")*

*print(f"Fraud cases: {len(data[data['Class'] == 1])}")*

*print(f"Normal cases: {len(data[data['Class'] == 0])}")*

*print(f"Fraud ratio: {len(data[data['Class'] == 1]) / len(data):.4%}")*

*# Инициализация класса моделей*

*fraud\_detection = FraudDetectionModels()*

*# Подготовка данных*

*print("\nPreparing data...")*

*X\_train, X\_test, y\_train, y\_test = fraud\_detection.prepare\_data(data)*

*print(f"\nTraining set size: {X\_train.shape}")*

*print(f"Test set size: {X\_test.shape}")*

*# Обучение и оценка моделей*

*print("\nTraining and evaluating models...")*

*results = fraud\_detection.train\_and\_evaluate(X\_train, X\_test, y\_train, y\_test)*

*# Сравнительный анализ*

*print("\nModel Comparison:")*

*model\_metrics = {}*

*for name, result in results.items():*

*metrics = {*

*'F1 Score': result['f1'],*

*'PR-AUC': result.get('pr\_auc', 'N/A') # Some models might not have PR-AUC*

*}*

*model\_metrics[name] = metrics*

*print(f"\n{name}:")*

*print(f"F1 Score: {metrics['F1 Score']:.4f}")*

*print(f"PR-AUC: {metrics['PR-AUC'] if isinstance(metrics['PR-AUC'], str) else metrics['PR-AUC']:.4f}")*

*print("\nConfusion Matrix:")*

*print(result['confusion\_matrix'])*

*print("\nDetailed Classification Report:")*

*print(result['classification\_report'])*

*# Визуализация результатов*

*plot\_model\_comparison(model\_metrics)*

*# Построение матриц ошибок*

*fraud\_detection.plot\_confusion\_matrices(results)*

*return results, model\_metrics*

*def plot\_model\_comparison(model\_metrics):*

*# Создание DataFrame для визуализации*

*metrics\_df = pd.DataFrame(model\_metrics).T*

*# Plotting*

*plt.figure(figsize=(12, 6))*

*# Plot F1 Scores*

*plt.subplot(1, 2, 1)*

*metrics\_df['F1 Score'].plot(kind='bar')*

*plt.title('F1 Scores by Model')*

*plt.xlabel('Model')*

*plt.ylabel('F1 Score')*

*plt.xticks(rotation=45)*

*# Plot PR-AUC*

*plt.subplot(1, 2, 2)*

*pr\_auc\_values = metrics\_df['PR-AUC'].replace('N/A', np.nan).astype(float)*

*pr\_auc\_values.plot(kind='bar')*

*plt.title('PR-AUC by Model')*

*plt.xlabel('Model')*

*plt.ylabel('PR-AUC')*

*plt.xticks(rotation=45)*

*plt.tight\_layout()*

*plt.show()*

*if \_\_name\_\_ == "\_\_main\_\_":*

*print("Starting baseline model evaluation...")*

*results, metrics = evaluate\_baseline\_models()*

*print("\nBaseline evaluation completed!")*

*# Печать финального сравнения в табличном виде*

*print("\nSummary Table of Metrics:")*

*summary\_df = pd.DataFrame(metrics).T*

*print(summary\_df.to\_string(float\_format="%.4f"))*

*import numpy as np*

*import pandas as pd*

*from sklearn.model\_selection import RandomizedSearchCV, StratifiedKFold*

*from sklearn.preprocessing import StandardScaler*

*from sklearn.linear\_model import LogisticRegression*

*from sklearn.tree import DecisionTreeClassifier*

*from sklearn.svm import SVC*

*from sklearn.ensemble import RandomForestClassifier*

*from xgboost import XGBClassifier*

*from sklearn.metrics import make\_scorer, f1\_score, precision\_recall\_curve, auc*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*import warnings*

*import time*

*warnings.filterwarnings('ignore')*

*class ModelTuner:*

*def \_\_init\_\_(self):*

*self.param\_grids = {*

*'Logistic Regression': {*

*'model': LogisticRegression(random\_state=42),*

*'params': {*

*'C': [0.01, 0.1, 1],*

*'class\_weight': ['balanced'],*

*'solver': ['liblinear'],*

*'max\_iter': [1000]*

*}*

*},*

*'Decision Tree': {*

*'model': DecisionTreeClassifier(random\_state=42),*

*'params': {*

*'max\_depth': [5, 10],*

*'min\_samples\_split': [2, 5],*

*'min\_samples\_leaf': [1, 2],*

*'class\_weight': ['balanced']*

*}*

*},*

*'SVM': {*

*'model': SVC(random\_state=42, probability=True),*

*'params': {*

*'C': [0.1, 1],*

*'kernel': ['rbf'],*

*'gamma': ['scale', 0.1],*

*'class\_weight': ['balanced']*

*}*

*},*

*'Random Forest': {*

*'model': RandomForestClassifier(random\_state=42),*

*'params': {*

*'n\_estimators': [100],*

*'max\_depth': [10, 20],*

*'min\_samples\_split': [2],*

*'min\_samples\_leaf': [1],*

*'class\_weight': ['balanced']*

*}*

*},*

*'XGBoost': {*

*'model': XGBClassifier(random\_state=42, use\_label\_encoder=False, eval\_metric='logloss'),*

*'params': {*

*'n\_estimators': [100],*

*'max\_depth': [3, 5],*

*'learning\_rate': [0.01, 0.1],*

*'scale\_pos\_weight': [30],*

*'min\_child\_weight': [1]*

*}*

*}*

*}*

*self.best\_models = {}*

*self.cv\_results = {}*

*def prepare\_data(self, data):*

*X = data.iloc[:, :-1]*

*y = data.iloc[:, -1]*

*scaler = StandardScaler()*

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

*return X\_scaled, y*

*def tune\_models(self, X, y, cv=3, n\_iter=10):*

*scoring = {*

*'f1': make\_scorer(f1\_score),*

*'pr\_auc': make\_scorer(self.\_pr\_auc\_score)*

*}*

*skf = StratifiedKFold(n\_splits=cv, shuffle=True, random\_state=42)*

*for name, model\_info in self.param\_grids.items():*

*print(f"\nTuning {name}...")*

*search = RandomizedSearchCV(*

*estimator=model\_info['model'],*

*param\_distributions=model\_info['params'],*

*scoring=scoring,*

*cv=skf,*

*n\_iter=n\_iter,*

*n\_jobs=-1,*

*refit='f1',*

*verbose=1,*

*random\_state=42,*

*return\_train\_score=True*

*)*

*start = time.time()*

*search.fit(X, y)*

*end = time.time()*

*self.best\_models[name] = search.best\_estimator\_*

*self.cv\_results[name] = {*

*'best\_params': search.best\_params\_,*

*'best\_score': search.best\_score\_,*

*'cv\_results': search.cv\_results\_*

*}*

*print(f"\n{name} tuning complete in {end - start:.2f} seconds")*

*print(f"Best parameters for {name}: {search.best\_params\_}")*

*print(f"Best F1 score: {search.best\_score\_:.4f}")*

*def \_pr\_auc\_score(self, y\_true, y\_prob):*

*precision, recall, \_ = precision\_recall\_curve(y\_true, y\_prob)*

*return auc(recall, precision)*

*def plot\_results(self):*

*best\_scores = {name: results['best\_score']*

*for name, results in self.cv\_results.items()}*

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

*plt.bar(best\_scores.keys(), best\_scores.values(), color='skyblue')*

*plt.title('Best F1 Scores After Tuning')*

*plt.xlabel('Model')*

*plt.ylabel('F1 Score')*

*plt.xticks(rotation=45)*

*plt.tight\_layout()*

*plt.show()*

*# Boxplot*

*plt.figure(figsize=(12, 6))*

*scores\_data = []*

*model\_names = []*

*for name, results in self.cv\_results.items():*

*scores = results['cv\_results']['mean\_test\_f1']*

*scores\_data.extend(scores)*

*model\_names.extend([name] \* len(scores))*

*sns.boxplot(x=model\_names, y=scores\_data, palette='Set2')*

*plt.title('Distribution of F1 Scores During Tuning')*

*plt.xlabel('Model')*

*plt.ylabel('F1 Score')*

*plt.xticks(rotation=45)*

*plt.tight\_layout()*

*plt.show()*

*def main():*

*print("Loading data...")*

*data = pd.read\_csv('creditcard.csv')*

*data = data.sample(n=10000, random\_state=42) # подвыборка*

*tuner = ModelTuner()*

*X, y = tuner.prepare\_data(data)*

*tuner.tune\_models(X, y, cv=3, n\_iter=10) # ускоренный подбор*

*tuner.plot\_results()*

*results\_df = pd.DataFrame()*

*for name, results in tuner.cv\_results.items():*

*model\_results = {*

*'Model': name,*

*'Best F1 Score': results['best\_score'],*

*'Best Parameters': str(results['best\_params'])*

*}*

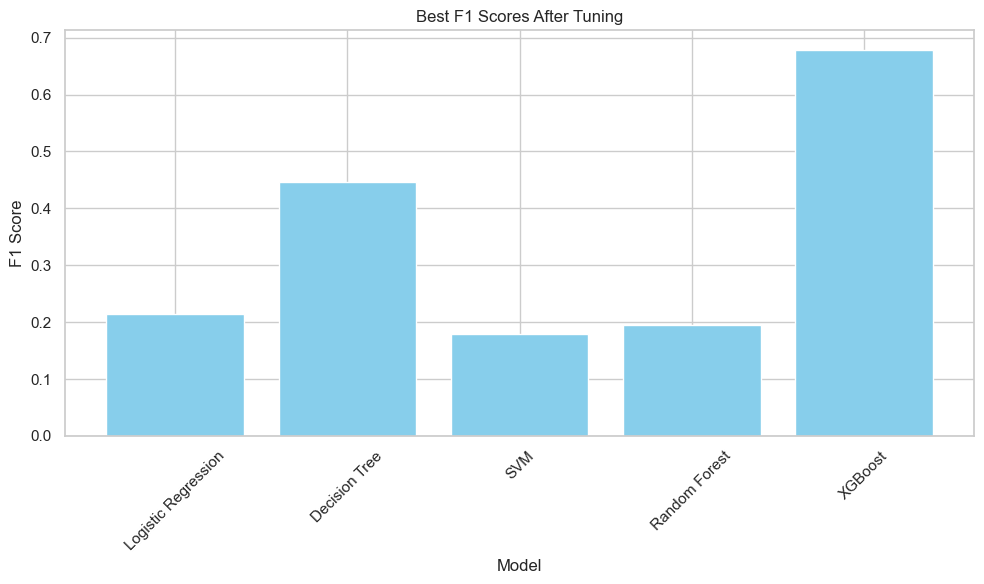
*results\_df = pd.concat([results\_df, pd.DataFrame([model\_results])], ignore\_index=True)*

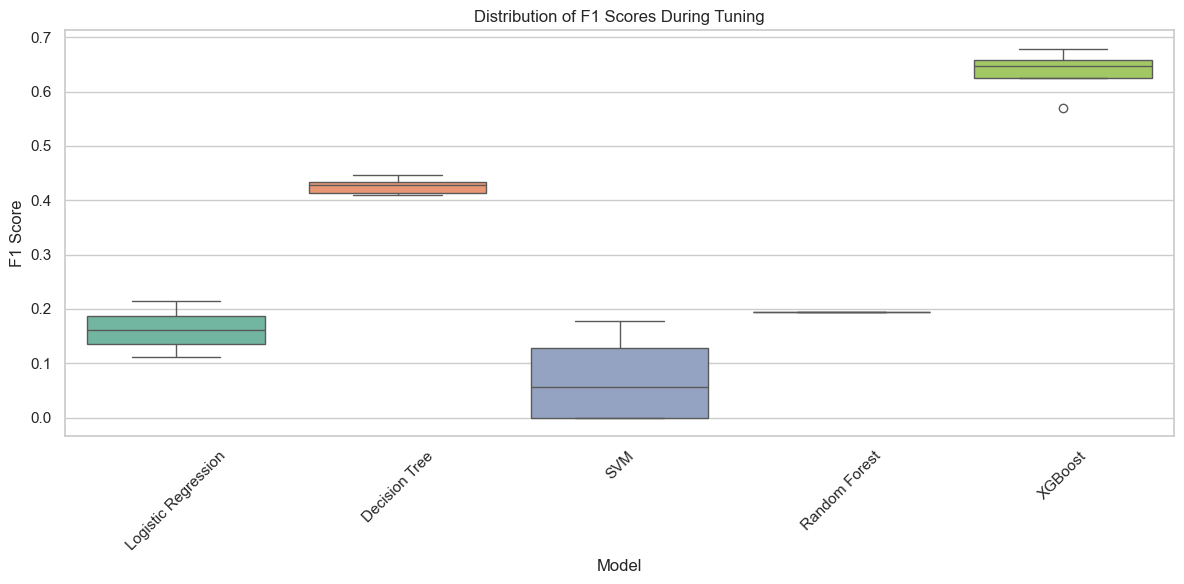
*results\_df.to\_csv('tuning\_results.csv', index=False)*

*print("\nResults have been saved to 'tuning\_results.csv'")*

*if \_\_name\_\_ == "\_\_main\_\_":*

*main()*

**

**

*import pandas as pd*

*import numpy as np*

*import time*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.preprocessing import StandardScaler*

*from sklearn.linear\_model import LogisticRegression*

*from sklearn.tree import DecisionTreeClassifier*

*from sklearn.svm import SVC*

*from sklearn.ensemble import RandomForestClassifier*

*from xgboost import XGBClassifier*

*from sklearn.metrics import classification\_report, confusion\_matrix*

*from sklearn.metrics import precision\_recall\_curve, auc, f1\_score*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*import ast*

*import warnings*

*warnings.filterwarnings('ignore')*

*class OptimizedModelsTrainer:*

*def \_\_init\_\_(self):*

*self.models = {}*

*self.results = {}*

*self.scaler = StandardScaler()*

*def load\_optimal\_parameters(self):*

*"""Загрузка оптимальных параметров из CSV файла"""*

*tuning\_results = pd.read\_csv('tuning\_results.csv')*

*for \_, row in tuning\_results.iterrows():*

*model\_name = row['Model']*

*params = ast.literal\_eval(row['Best Parameters'])*

*if model\_name == 'Logistic Regression':*

*self.models[model\_name] = LogisticRegression(\*\*params, random\_state=42)*

*elif model\_name == 'Decision Tree':*

*self.models[model\_name] = DecisionTreeClassifier(\*\*params, random\_state=42)*

*elif model\_name == 'SVM':*

*# Установка безопасных параметров по умолчанию*

*params.setdefault('kernel', 'linear')*

*params.setdefault('probability', False)*

*params.setdefault('max\_iter', 1000)*

*self.models[model\_name] = SVC(\*\*params, random\_state=42)*

*elif model\_name == 'Random Forest':*

*self.models[model\_name] = RandomForestClassifier(\*\*params, random\_state=42)*

*elif model\_name == 'XGBoost':*

*self.models[model\_name] = XGBClassifier(\*\*params, random\_state=42)*

*def prepare\_data(self, data):*

*"""Подготовка данных для обучения"""*

*X = data.iloc[:, :-1]*

*y = data.iloc[:, -1]*

*# Масштабирование признаков*

*X\_scaled = self.scaler.fit\_transform(X)*

*# Разделение на обучающую и тестовую выборки*

*return train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42, stratify=y)*

*def train\_and\_evaluate(self, X\_train, X\_test, y\_train, y\_test):*

*"""Обучение и оценка моделей"""*

*for name, model in self.models.items():*

*print(f"\nTraining {name}...")*

*start\_time = time.time()*

*try:*

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

*except Exception as e:*

*print(f"Error training {name}: {e}")*

*continue*

*elapsed = time.time() - start\_time*

*if elapsed > 60:*

*print(f"⚠️ Warning: Training {name} took {elapsed:.2f} seconds!")*

*y\_pred = model.predict(X\_test)*

*if hasattr(model, "predict\_proba") and getattr(model, "probability", True):*

*y\_scores = model.predict\_proba(X\_test)[:, 1]*

*precision, recall, \_ = precision\_recall\_curve(y\_test, y\_scores)*

*pr\_auc = auc(recall, precision)*

*else:*

*pr\_auc = None*

*self.results[name] = {*

*'predictions': y\_pred,*

*'f1': f1\_score(y\_test, y\_pred),*

*'pr\_auc': pr\_auc,*

*'confusion\_matrix': confusion\_matrix(y\_test, y\_pred),*

*'classification\_report': classification\_report(y\_test, y\_pred)*

*}*

*# Вывод результатов*

*print(f"\nResults for {name}:")*

*print(f"F1 Score: {self.results[name]['f1']:.4f}")*

*if pr\_auc is not None:*

*print(f"PR-AUC: {pr\_auc:.4f}")*

*print("\nConfusion Matrix:")*

*print(self.results[name]['confusion\_matrix'])*

*print("\nClassification Report:")*

*print(self.results[name]['classification\_report'])*

*def plot\_results(self):*

*"""Визуализация результатов"""*

*# График сравнения F1-scores*

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

*f1\_scores = {name: results['f1'] for name, results in self.results.items()}*

*plt.bar(f1\_scores.keys(), f1\_scores.values())*

*plt.title('F1 Scores of Optimized Models')*

*plt.xlabel('Model')*

*plt.ylabel('F1 Score')*

*plt.xticks(rotation=45)*

*plt.tight\_layout()*

*plt.show()*

*# Матрицы ошибок*

*fig, axes = plt.subplots(2, 3, figsize=(15, 10))*

*axes = axes.ravel()*

*for idx, (name, results) in enumerate(self.results.items()):*

*if idx < len(axes):*

*sns.heatmap(results['confusion\_matrix'], annot=True, fmt='d', ax=axes[idx], cmap="Blues")*

*axes[idx].set\_title(f'{name} Confusion Matrix')*

*plt.tight\_layout()*

*plt.show()*

*def save\_results(self):*

*"""Сохранение результатов в CSV"""*

*results\_df = pd.DataFrame()*

*for name, results in self.results.items():*

*model\_results = {*

*'Model': name,*

*'F1 Score': results['f1'],*

*'PR-AUC': results.get('pr\_auc', 'N/A'),*

*'Classification Report': results['classification\_report']*

*}*

*results\_df = pd.concat([results\_df, pd.DataFrame([model\_results])], ignore\_index=True)*

*results\_df.to\_csv('optimized\_models\_results.csv', index=False)*

*print("\nResults have been saved to 'optimized\_models\_results.csv'")*

*def main():*

*print("Loading data...")*

*data = pd.read\_csv('creditcard.csv')*

*# Можно временно уменьшить выборку для тестов*

*# data = data.sample(frac=0.1, random\_state=42)*

*trainer = OptimizedModelsTrainer()*

*print("Loading optimal parameters...")*

*trainer.load\_optimal\_parameters()*

*print("Preparing data...")*

*X\_train, X\_test, y\_train, y\_test = trainer.prepare\_data(data)*

*print("Training and evaluating models...")*

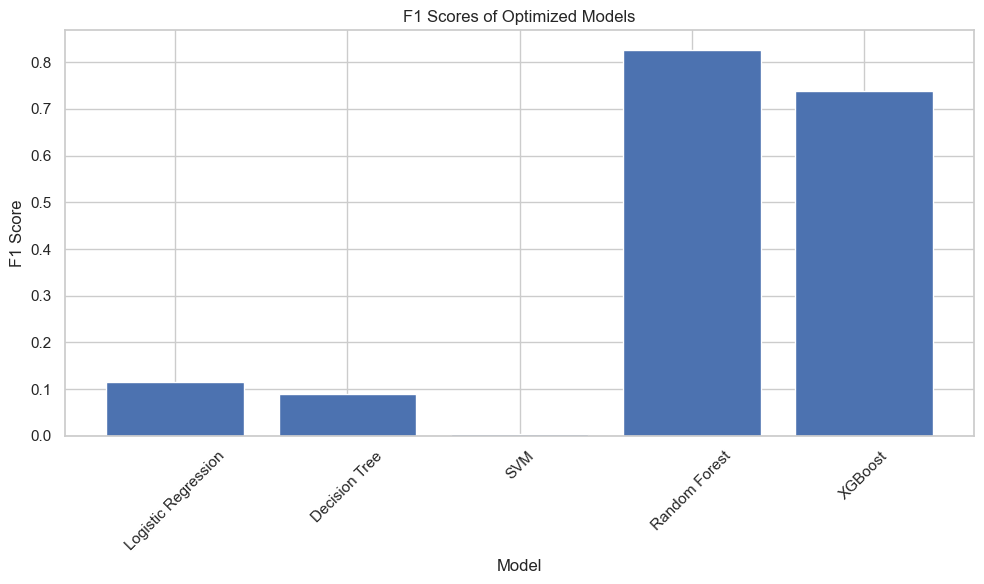
*trainer.train\_and\_evaluate(X\_train, X\_test, y\_train, y\_test)*

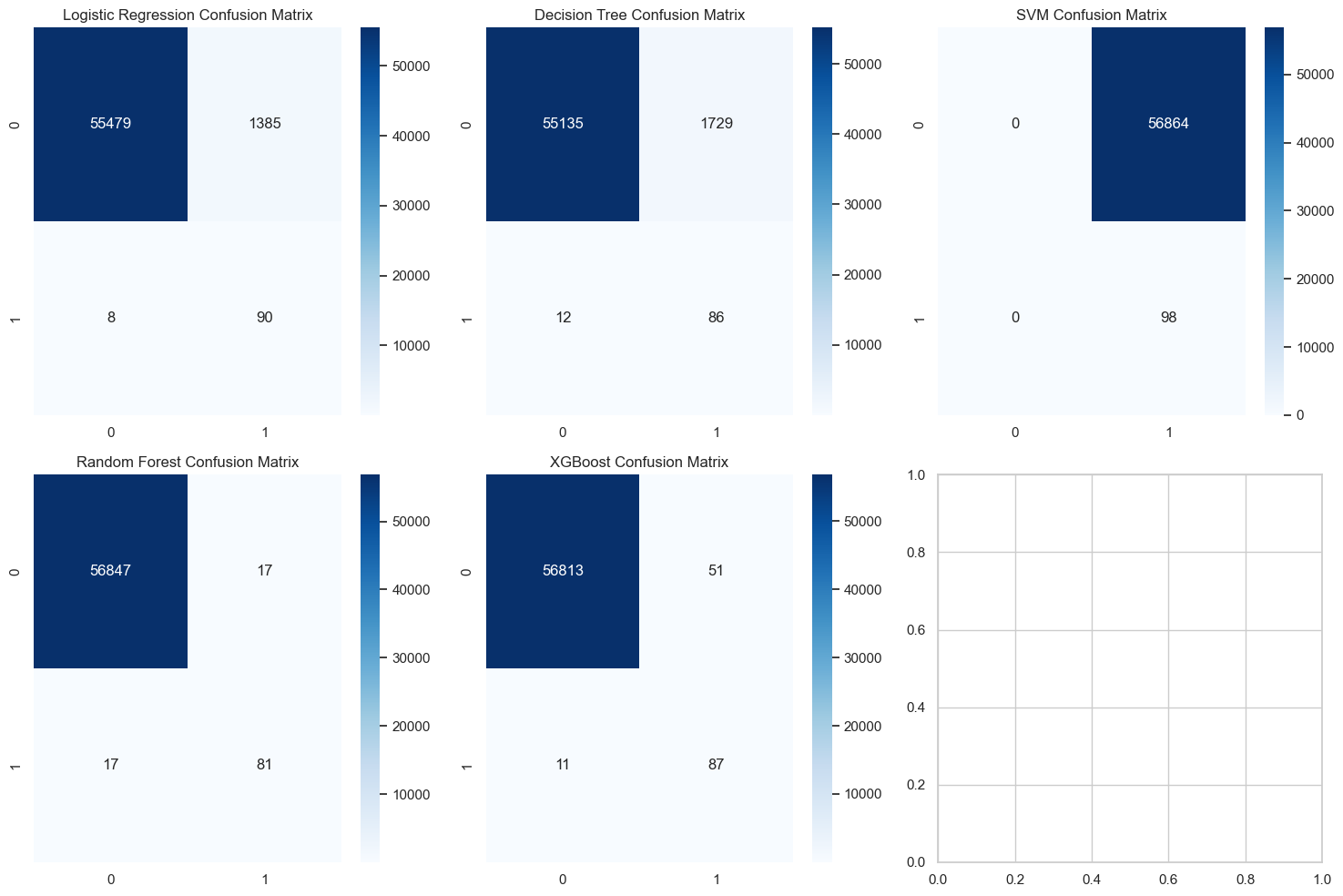
*trainer.plot\_results()*

*trainer.save\_results()*

*if \_\_name\_\_ == "\_\_main\_\_":*

*main()*

**

**

*import pandas as pd*

*def print\_optimized\_models\_results(file\_path='optimized\_models\_results.csv'):*

*try:*

*# Загрузка результатов*

*df = pd.read\_csv(file\_path)*

*# Преобразование и печать таблицы*

*pd.set\_option('display.max\_colwidth', None) # Полный вывод Classification Report*

*print("\n📊 Результаты оптимизированных моделей:\n")*

*print(df[['Model', 'F1 Score', 'PR-AUC']].to\_string(index=False)) # Основные метрики*

*print("\n🔍 Подробный отчёт классификации для каждой модели:\n")*

*for \_, row in df.iterrows():*

*print(f"=== {row['Model']} ===")*

*print(row['Classification Report'])*

*print("-" \* 60)*

*except FileNotFoundError:*

*print(f"Файл {file\_path} не найден. Убедитесь, что вы запустили обучение и сохранили результаты.")*

*except Exception as e:*

*print(f"Произошла ошибка при чтении файла: {e}")*

*# Вызов*

*print\_optimized\_models\_results()*

*import pandas as pd*

*import numpy as np*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*from sklearn.metrics import precision\_recall\_curve, roc\_curve, auc*

*import json*

*import ast*

*from scipy import stats*

*class ModelAnalyzer:*

*def \_\_init\_\_(self):*

*# Загрузка результатов*

*self.tuning\_results = pd.read\_csv('tuning\_results.csv')*

*self.optimized\_results = pd.read\_csv('optimized\_models\_results.csv')*

*# Настройка стиля графиков*

*self.colors = sns.color\_palette("husl", n\_colors=5)*

*def analyze\_model\_performance(self):*

*"""Анализ производительности моделей"""*

*print("=== Анализ производительности моделей ===\n")*

*# Сортировка моделей по F1-score*

*sorted\_models = self.optimized\_results.sort\_values('F1 Score', ascending=False)*

*print("Рейтинг моделей по F1-score:")*

*for idx, row in sorted\_models.iterrows():*

*print(f"{row['Model']}:")*

*print(f" F1-score: {row['F1 Score']:.4f}")*

*if row['PR-AUC'] != 'N/A':*

*print(f" PR-AUC: {float(row['PR-AUC']):.4f}")*

*print()*

*# Определение лучшей модели*

*best\_model = sorted\_models.iloc[0]*

*print(f"Лучшая модель: {best\_model['Model']}")*

*print(f"F1-score: {best\_model['F1 Score']:.4f}")*

*if best\_model['PR-AUC'] != 'N/A':*

*print(f"PR-AUC: {float(best\_model['PR-AUC']):.4f}")*

*return sorted\_models*

*def plot\_performance\_comparison(self):*

*"""Визуализация сравнения производительности моделей"""*

*plt.figure(figsize=(12, 6))*

*# График F1-scores*

*plt.subplot(1, 2, 1)*

*scores = self.optimized\_results['F1 Score'].values*

*models = self.optimized\_results['Model'].values*

*bars = plt.bar(models, scores, color=self.colors)*

*plt.title('Сравнение F1-scores моделей')*

*plt.xlabel('Модель')*

*plt.ylabel('F1-score')*

*plt.xticks(rotation=45)*

*# Добавление значений над столбцами*

*for bar in bars:*

*height = bar.get\_height()*

*plt.text(bar.get\_x() + bar.get\_width()/2., height,*

*f'{height:.4f}',*

*ha='center', va='bottom')*

*# График PR-AUC*

*plt.subplot(1, 2, 2)*

*pr\_auc\_values = []*

*for value in self.optimized\_results['PR-AUC'].values:*

*try:*

*pr\_auc\_values.append(float(value))*

*except:*

*pr\_auc\_values.append(0)*

*bars = plt.bar(models, pr\_auc\_values, color=self.colors)*

*plt.title('Сравнение PR-AUC моделей')*

*plt.xlabel('Модель')*

*plt.ylabel('PR-AUC')*

*plt.xticks(rotation=45)*

*# Добавление значений над столбцами*

*for bar in bars:*

*height = bar.get\_height()*

*if height > 0:*

*plt.text(bar.get\_x() + bar.get\_width()/2., height,*

*f'{height:.4f}',*

*ha='center', va='bottom')*

*plt.tight\_layout()*

*plt.show()*

*def analyze\_hyperparameters(self):*

*"""Анализ влияния гиперпараметров"""*

*print("\n=== Анализ влияния гиперпараметров ===\n")*

*for \_, row in self.tuning\_results.iterrows():*

*model\_name = row['Model']*

*params = ast.literal\_eval(row['Best Parameters'])*

*print(f"\nМодель: {model\_name}")*

*print("Оптимальные гиперпараметры:")*

*for param, value in params.items():*

*print(f" {param}: {value}")*

*def generate\_report(self):*

*"""Генерация итогового отчета"""*

*report = "=== Итоговый отчет по анализу моделей ===\n\n"*

*# Анализ производительности*

*sorted\_models = self.analyze\_model\_performance()*

*best\_model = sorted\_models.iloc[0]*

*report += "1. Общие выводы:\n"*

*report += f" - Лучшая модель: {best\_model['Model']}\n"*

*report += f" - F1-score лучшей модели: {best\_model['F1 Score']:.4f}\n"*

*if best\_model['PR-AUC'] != 'N/A':*

*report += f" - PR-AUC лучшей модели: {float(best\_model['PR-AUC']):.4f}\n"*

*report += "\n2. Сравнительный анализ моделей:\n"*

*for idx, row in sorted\_models.iterrows():*

*report += f" {row['Model']}:\n"*

*report += f" - F1-score: {row['F1 Score']:.4f}\n"*

*if row['PR-AUC'] != 'N/A':*

*report += f" - PR-AUC: {float(row['PR-AUC']):.4f}\n"*

*report += "\n3. Рекомендации:\n"*

*report += " - Для продакшена рекомендуется использовать "*

*report += f"{best\_model['Model']} как наиболее эффективную модель\n"*

*report += " - Возможно дальнейшее улучшение путем:\n"*

*report += " \* Ансамблирования лучших моделей\n"*

*report += " \* Дополнительной оптимизации гиперпараметров\n"*

*report += " \* Применения техник обработки несбалансированных данных\n"*

*# Сохранение отчета*

*with open('model\_analysis\_report.txt', 'w') as f:*

*f.write(report)*

*return report*

*def main():*

*analyzer = ModelAnalyzer()*

*# Анализ производительности моделей*

*analyzer.analyze\_model\_performance()*

*# Визуализация результатов*

*analyzer.plot\_performance\_comparison()*

*# Анализ гиперпараметров*

*analyzer.analyze\_hyperparameters()*

*# Генерация отчета*

*report = analyzer.generate\_report()*

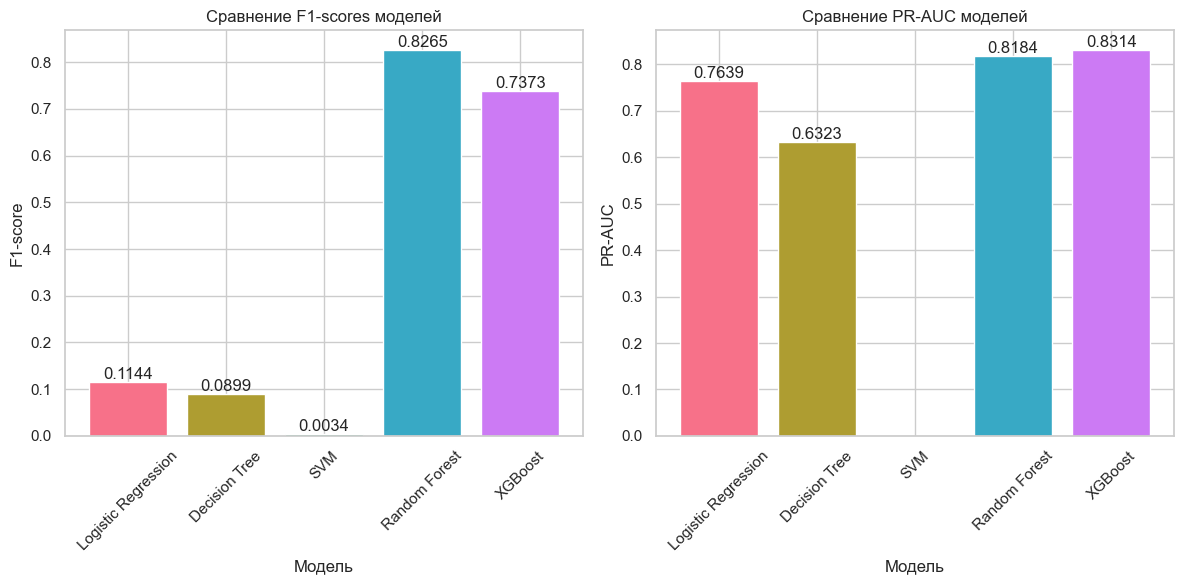
*print("\nПолный отчет сохранен в 'model\_analysis\_report.txt'")*

*print("\nКраткие выводы:")*

*print(report)*

*if \_\_name\_\_ == "\_\_main\_\_":*

*main()*

**

*import numpy as np*

*import pandas as pd*

*import ast*

*import matplotlib.pyplot as plt*

*from sklearn.linear\_model import LogisticRegression*

*from sklearn.tree import DecisionTreeClassifier*

*from sklearn.svm import SVC*

*from sklearn.ensemble import RandomForestClassifier*

*from xgboost import XGBClassifier*

*from sklearn.model\_selection import learning\_curve, validation\_curve, StratifiedKFold*

*class LearningCurvesAnalyzer:*

*def \_\_init\_\_(self):*

*self.tuning\_results = pd.read\_csv('tuning\_results.csv')*

*self.cv = StratifiedKFold(n\_splits=3, shuffle=True, random\_state=42)*

*def plot\_learning\_curves(self, X, y, model, model\_name):*

*train\_sizes = np.linspace(0.6, 1.0, 3)*

*train\_sizes, train\_scores, val\_scores = learning\_curve(*

*model, X, y,*

*train\_sizes=train\_sizes,*

*cv=self.cv,*

*scoring='accuracy', # быстрое и простое*

*n\_jobs=1,*

*shuffle=True,*

*random\_state=42*

*)*

*train\_mean = np.mean(train\_scores, axis=1)*

*val\_mean = np.mean(val\_scores, axis=1)*

*plt.figure(figsize=(8, 5))*

*plt.plot(train\_sizes, train\_mean, 'o-', color='blue', label='Training score')*

*plt.plot(train\_sizes, val\_mean, 'o-', color='red', label='Cross-validation score')*

*plt.title(f'Learning Curve - {model\_name}')*

*plt.xlabel('Training examples')*

*plt.ylabel('Accuracy')*

*plt.legend()*

*plt.grid(True)*

*plt.show()*

*def plot\_validation\_curves(self, X, y, model, param\_name, param\_range, model\_name):*

*train\_scores, val\_scores = validation\_curve(*

*model, X, y,*

*param\_name=param\_name,*

*param\_range=param\_range,*

*cv=self.cv,*

*scoring='accuracy',*

*n\_jobs=1*

*)*

*train\_mean = np.mean(train\_scores, axis=1)*

*val\_mean = np.mean(val\_scores, axis=1)*

*plt.figure(figsize=(8, 5))*

*plt.plot(param\_range, train\_mean, 'o-', color='blue', label='Training score')*

*plt.plot(param\_range, val\_mean, 'o-', color='red', label='Cross-validation score')*

*plt.title(f'Validation Curve - {model\_name} ({param\_name})')*

*plt.xlabel(param\_name)*

*plt.ylabel('Accuracy')*

*plt.legend()*

*plt.grid(True)*

*plt.show()*

*def analyze\_model\_curves(self, X, y):*

*for \_, row in self.tuning\_results.iterrows():*

*model\_name = row['Model']*

*params = ast.literal\_eval(row['Best Parameters'])*

*params['random\_state'] = 42*

*if model\_name == 'Logistic Regression':*

*model = LogisticRegression(\*\*params)*

*self.plot\_validation\_curves(X, y, model, 'C', [0.01, 0.1, 1], model\_name)*

*elif model\_name == 'Decision Tree':*

*model = DecisionTreeClassifier(\*\*params)*

*self.plot\_validation\_curves(X, y, model, 'max\_depth', [3, 5, 7], model\_name)*

*elif model\_name == 'SVM':*

*model = SVC(\*\*params, probability=True)*

*self.plot\_validation\_curves(X, y, model, 'C', [0.1, 1, 10], model\_name)*

*elif model\_name == 'Random Forest':*

*model = RandomForestClassifier(\*\*params)*

*self.plot\_validation\_curves(X, y, model, 'n\_estimators', [10, 50, 100], model\_name)*

*elif model\_name == 'XGBoost':*

*model = XGBClassifier(\*\*params)*

*self.plot\_validation\_curves(X, y, model, 'max\_depth', [3, 5, 7], model\_name)*

*self.plot\_learning\_curves(X, y, model, model\_name)*

*def main():*

*data = pd.read\_csv('creditcard.csv')*

*normal = data[data['Class'] == 0].sample(n=5000, random\_state=42)*

*fraud = data[data['Class'] == 1]*

*data\_small = pd.concat([normal, fraud])*

*X = data\_small.drop('Class', axis=1)*

*y = data\_small['Class']*

*analyzer = LearningCurvesAnalyzer()*

*analyzer.analyze\_model\_curves(X, y)*

*if \_\_name\_\_ == "\_\_main\_\_":*

*main()*

