ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____
ПРЕПОДАВАТЕЛЬ

Старший преподаватель		Т. А. Суетина
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

DATA SCIENCE. МОДЕЛЬ КЛАССИФИКАЦИИ СЛУЧАЙНЫЙ ЛЕС

по курсу: ИНТЕГРИРОВАННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

РАБОТУ ВЫПОЛНИЛИ

В. А. Воробьев

СТУДЕНТЫ ГР. № 4128		
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

1 Цель работы и задание

1.1 Цель работы:

Получить теоретические знания по построению модели классификации случайный леи и реализовать на практике построенную модель, доказав ее работоспособность.

1.2 Задание

1. Выполнить классификацию роз **по количеству лепестков и высоте.** Здесь все просто.

- Простой – менее 8 лепестков.
- Полумахровый – 8-20 лепестков
- Умеренно махровый – 21-29 лепестков
- Среднемахровый – 30-39 лепестков
- Густомахровый – более 40 лепестков

2. Выполнить классификацию роз по **Форме цветка и высоте,** также бывает разная:

3. Выполнить классификацию роз по группам https://www.rosebook.ru/blogs/ewqenia12/klassifikacija-roz-po-gruppam-i-ih-osobennosti/
4. Студенты могут предложить свои варианты.

2 Ход работы

2.1 Классификация по количеству лепестков и высоте

Для выполнения задачи классификации с помощью скрипта на языке программирования Python был сгенерирован датасет roses_dataset.csv, представляющий собой набор из 10 тысяч записей в формате: “количество лепестков,высота (в см),тип).

Затем с помощью другого скрипта была создана модель случайного леса, обученная на ранее сгенерированном наборе данных, который был предварительно разделён 80/20, где 80% составляет обучающий набор, а остальные 20% – тестовый.

Для обучения модели была использована сторонняя библиотека scikit-learn.

Метрики классификации представлены на рис. 1-2, а диаграмма значимости признаков – на рис. 3:

	precision	recall	f1-score	support
Густомахровый	0.99	0.97	0.98	204
Полумахровый	0.89	0.89	0.89	628
Простой	0.88	0.91	0.90	383
Среднемахровый	0.94	0.91	0.92	294
Умеренно махровый	0.90	0.91	0.91	491
accuracy			0.91	2000
macro avg	0.92	0.92	0.92	2000
weighted avg	0.91	0.91	0.91	2000

Рисунок 1 - Метрики классификации

```
Матрица сопряжённости:
[[198   0   0   6   0]
 [  0 558  46   0  24]
 [  0  34 349   0   0]
 [  3   0   0 268  23]
 [  0  33   0  12 446]]
```

Рисунок 2 - матрица сопряжённости

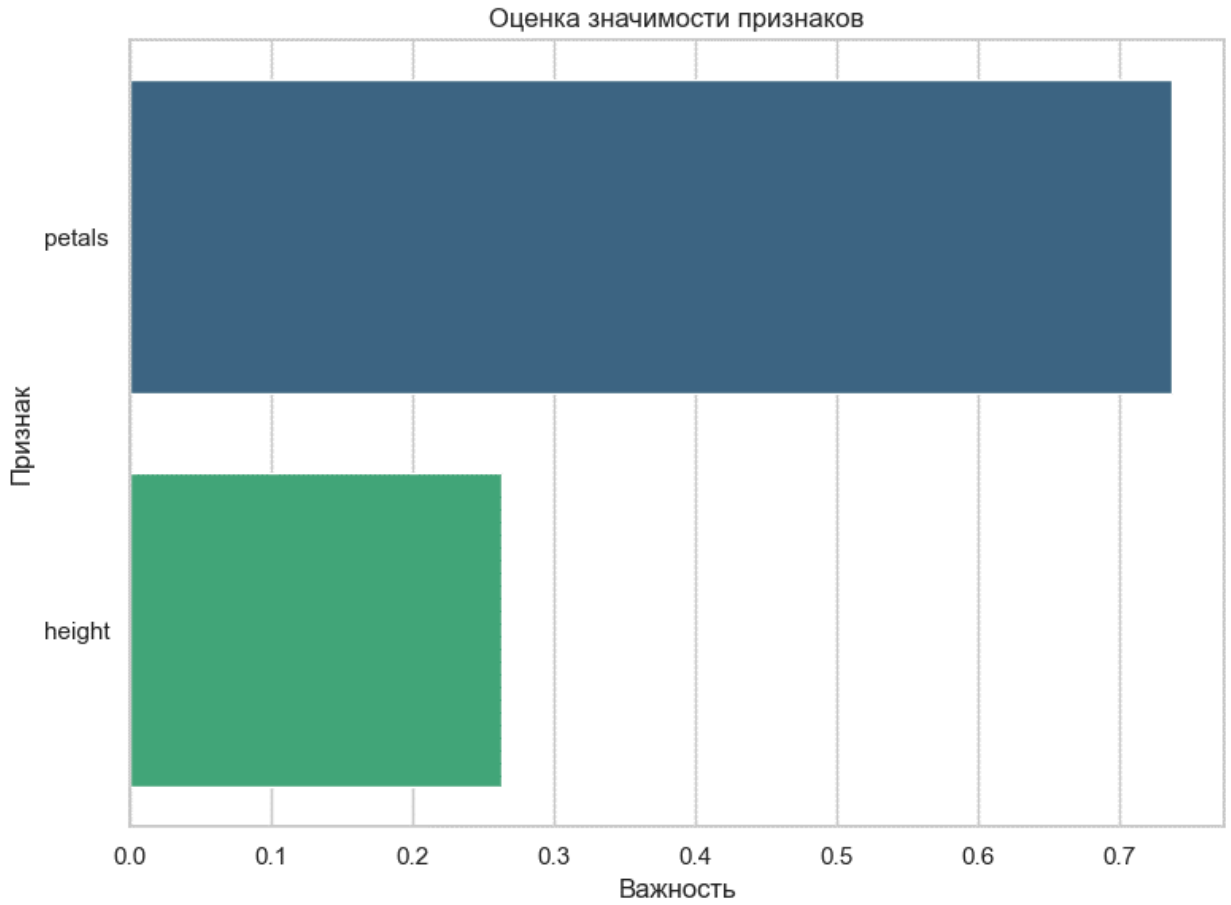

Рисунок 3 - Диаграмма значимости признаков

Исходя из рис. 4-5 можно сделать следующие выводы:

1. Модель имеет высокую точность (0.91), что говорит о том, что она работоспособна. Небольшие отклонения объяснимы смешиваемостью классов, находящихся на границе, что логично;
2. Наиболее важным признаком оказалось количество лепестков, в три раза превышающим другой признак: 0.75/0.25, где 0.25 – высота;
3. Высокая точность (Precision) у "Густомахрового" класса (0.99) – это говорит о том, что этот класс наиболее четко определяется моделью;
4. Самая сложная классификация у "Полумахровых" (Precision = 0.89, Recall = 0.89) – вероятно, из-за шума в количестве лепестков и высоте, этот класс пересекается с соседними;

5. Остальные классы ("Простой", "Среднемахровый", "Умеренно махровый") показывают точность в районе 88-94%, что является хорошим результатом;
6. Минимальные ошибки у "Густомахрового" (6 ошибок) – модель практически не путает этот класс с другими;
7. "Полумахровый" и "Умеренно махровый" иногда классифицируются неправильно (например, 46 "Полумахровых" цветов модель приняла за "Простой", а 34 "Умеренно махровых" – за "Полумахровый");

2.2 Классификация по форме

Для данной задачи был создан новый датасет – roses_dataset_v2.csv. Он имеет следующую структуру: форма цветка, высота куста, количество лепестков, длина лепестков, диаметр бутона, ширина лепестка. Задача заключается в определении формы цветка 8 классов: плоская, кувшинчатая, округлая, помпонная, чашеобразная, розеточная, коническая или бокаловидная, крестовидно-розеточная.

	precision	recall	f1-score	support
коническая	1.00	1.00	1.00	243
крестовидно-розеточная	0.83	0.86	0.85	243
кувшинчатая	0.96	0.98	0.97	252
округлая	0.95	0.96	0.96	244
плоская	0.88	0.90	0.89	244
помпонная	1.00	1.00	1.00	263
розеточная	0.99	0.96	0.97	268
чашеобразная	0.86	0.80	0.83	243
accuracy			0.94	2000
macro avg	0.93	0.93	0.93	2000
weighted avg	0.94	0.94	0.94	2000

Рисунок 4 - Метрики классификации

```
Матрица сопряжённости:
[[243   0   0   0   0   0   0   0]
 [  0 210   0   1  16   0   0  16]
 [  1   0 248   0   0   0   3   0]
 [  0   4   0 235   0   0   0   5]
 [  0  14   0   0 220   0   0  10]
 [  0   0   0   0   0 263   0   0]
 [  0   0   9   2   0   0 257   0]
 [  0  25   0   9  14   0   0 195]]
```

Рисунок 5 – матрица сопряженности

Вывод из отчёта на рис. 4-5:

1. Модель достигла точности 93%, что свидетельствует о высоком качестве классификации;
2. Коническая и помпонная формы классифицируются без ошибок (precision, recall и f1-score равны 1.00);
3. Кувшинчатая, округлая и розеточная демонстрируют очень хорошие показатели (f1-score около 0.96–0.97), что указывает на стабильное распознавание этих классов;
4. Плоская форма имеет немного более низкие показатели (f1-score 0.89), но результаты остаются удовлетворительными;
5. Наименее удачная классификация наблюдается для классов крестовидно-розеточная (f1-score 0.85) и чашеобразная (f1-score 0.83), что может свидетельствовать о большем перекрытии их признаков с другими классами или о неидеальной отделимости данных классов в датасете;
6. Основные ошибки наблюдаются для классов, где небольшое число образцов перепутаны с соседними классами (например, класс крестовидно-розеточная имеет 16 ошибок, отнесённых к другим классам);
7. Ошибки распределены не хаотично, а преимущественно происходят между теми классами, признаки которых могут быть схожи.

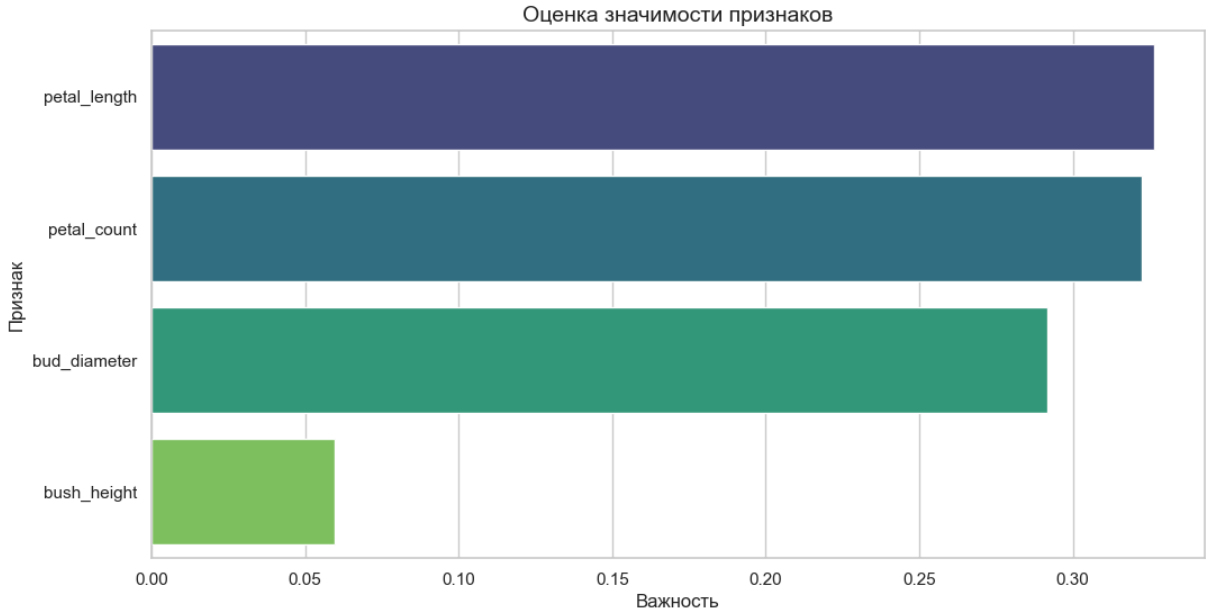

Рисунок 6 - Диаграмма значимости признаков

Это указывает, что petal_lenght является самым важным признаком для классификации формы цветка, за которым следуют petal_count и bud_diameter. Bush_height оказывает наименьшее влияние, но всё же вносит полезную информацию.

2.3 Классификация роз по группам

В новом датасете используются следующие признаки:

- stem_thickness – толщина стебля (мм)
- leaf_length – длина листа (см)
- leaf_width – ширина листа (см)
- fragrance_intensity – интенсивность аромата (шкала от 1 до 10)
- bloom_diameter – диаметр цветка (см)
- thorn_density – плотность шипов (количество шипов на 10 см стебля)

Классы (группы роз) выбраны по типичной классификации, например:

1 Гибридные чайные
2 Флорибунда
3 Грандифлора

4 Плетистые

5 Кустовые

6 Миниатюрные

Из изменений: было добавлено определение гиперпараметров:

- n_estimators – количество деревьев в лесу. Больше деревьев = лучшее качество, но в данном случае ещё и значительно увеличилось время обучения;
- max_depth – максимальная глубина деревьев. Глубокие деревья запоминают данные (переобучение), а мелкие могут плохо учиться;
- min_samples_split – минимальное число объектов в узле перед разбиением. Регулирует баланс между переобучением и обобщающей способностью;
- min_samples_leaf – минимальное число объектов в листе. Чем больше – тем плавнее разделение.

Касаемо формирования отчёта ничего не поменялось. Метрики классификации представлены на рис. 7-8, а диаграмма значимости признаков – на рис. 9:

	precision	recall	f1-score	support
Гибридные чайные	0.67	0.69	0.68	319
Грандифлора	0.85	0.80	0.82	346
Кустовые	0.95	0.96	0.95	327
Миниатюрные	1.00	1.00	1.00	345
Плетистые	0.90	0.94	0.92	337
Флорибунда	0.90	0.87	0.89	326
accuracy			0.88	2000
macro avg	0.88	0.88	0.88	2000
weighted avg	0.88	0.88	0.88	2000

Рисунок 7 - Метрики классификации,

```
Матрица ошибок:
[[221  43   8   0  17  30]
 [ 55 276   7   0   8   0]
 [  1   3 315   0   8   0]
 [  0   0   0 345   0   0]
 [ 14   4   3   0 316   0]
 [ 37   0   0   1   4 284]]
```

Рисунок 8 - матрица сопряжённости

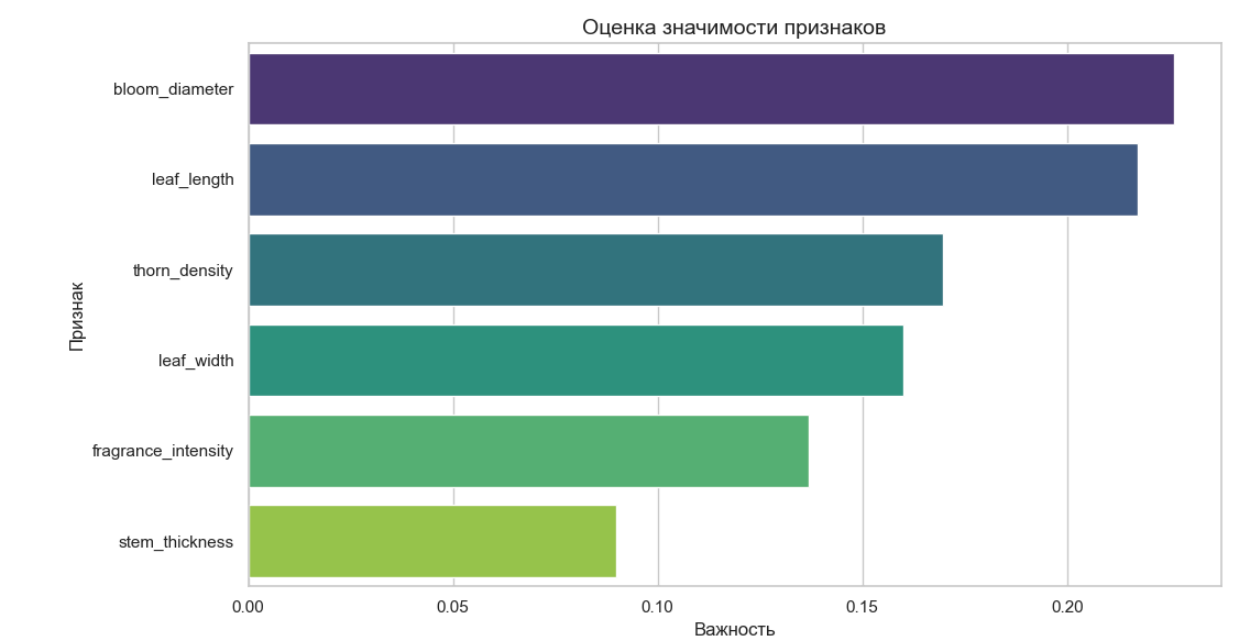

Рисунок 9 - Диаграмма значимости признаков

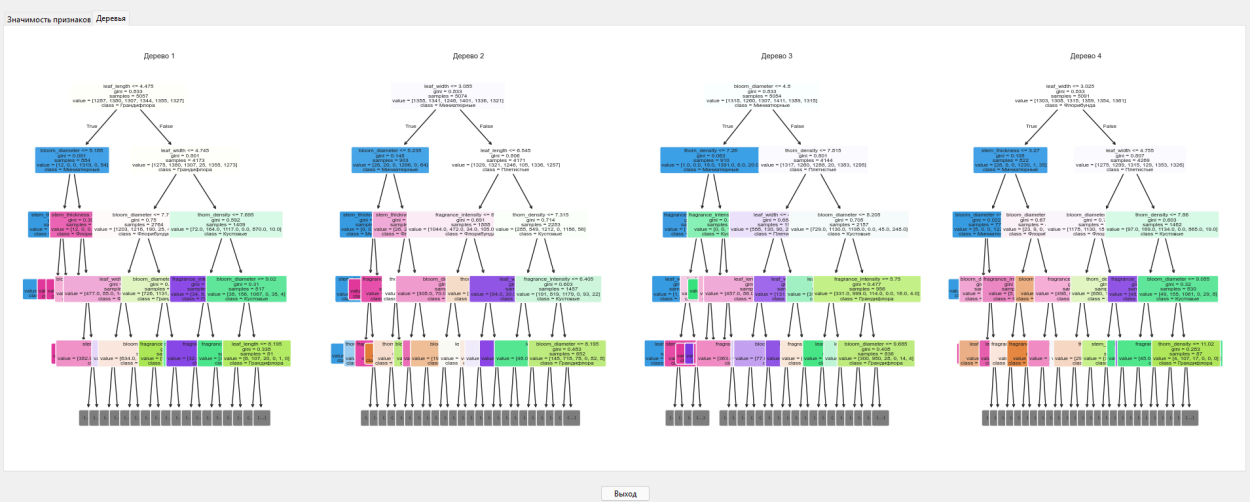

Рисунок 10 - Деревья

Пояснение к рис. 8-9:

1 Модель правильно классифицировала 88% всех образцов, что является достаточно хорошим результатом, хотя всё ещё остаётся пространство для улучшения. Далее выводы по классам:

1.1 Гибридные чайные:

Precision: 0.67 – из всех роз, предсказанных как гибридные чайные, 69% действительно таковы.

Recall: 0.69 – модель обнаружила 69% от всех гибридных чайных роз в тестовой выборке.

F1-score: 0.68 – общий показатель классификации для этого класса.

Эти значения показывают, что модель испытывает трудности с различением гибридных чайных роз.

1.2 Грандифлора:

Показатели (precision, recall, f1-score) около 0.85 свидетельствуют о хорошей способности модели различать этот класс.

1.3 Кустовые:

Очень высокие показатели (precision 0.95, recall 0.96, f1-score 0.95) – почти все кустовые розы правильно классифицированы.

1.4 Миниатюрные:

Идеальные показатели (1.00 по всем метрикам) – модель безошибочно распознаёт миниатюрные розы.

1.5 Плетистые:

Показатели около 0.90 свидетельствуют о надёжном распознавании.

1.6 Флорибунда:

Значения (precision 0.90, recall 0.87, f1-score 0.89) показывают хорошие результаты, но с небольшими ошибками.

2 Основные ошибки наблюдаются для класса Гибридные чайные, где модель ошибается, перепутав некоторые образцы с другими группами;

3 Для остальных классов ошибок меньше – особенно для миниатюрных роз, где ошибок нет, а также для кустовых роз, где модель практически безошибочно их определяет.

В целом, модель можно считать работоспособной.

ВЫВОД

В результате выполнения лабораторной работы была изучена и реализована модель классификации с использованием алгоритма случайного леса для различных характеристик роз. В ходе работы были решены три задачи: классификация роз по количеству лепестков и высоте, по форме цветка и высоте, а также по группам.

В результате выполнения работы было подтверждено, что случайный лес является эффективным методом классификации. Наилучшие результаты были достигнуты при классификации миниатюрных и кустовых роз, тогда как ошибки возникали при классификации близких по параметрам групп. Для дальнейшего улучшения модели возможно использование дополнительных признаков или оптимизация гиперпараметров алгоритма. Работа выполнена успешно, поставленные задачи решены.

ПРИЛОЖЕНИЕ А
CLASSIFICATION_RF_1.PY

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Загружаем датасет
df = pd.read_csv('roses_dataset.csv')

# Определяем признаки и целевую переменную
X = df[['petals', 'height']]
y = df['class']

# Разбиваем данные на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Создаем и обучаем модель случайного леса
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X_train, y_train)

# Предсказания на тестовой выборке
y_pred = rf_clf.predict(X_test)

# Выводим метрики классификации
print("**Точность:**", accuracy_score(y_test, y_pred))
print("\n**Отчёт о классификации:**\n", classification_report(y_test, y_pred))
print("\n**Матрица сопряжённости:**\n", confusion_matrix(y_test, y_pred))

# Получаем оценки значимости признаков
importances = rf_clf.feature_importances_

# Формируем DataFrame для визуализации
features_df = pd.DataFrame({
    'feature': X.columns,
    'importance': importances
}).sort_values(by='importance', ascending=False)

# Визуализация с помощью seaborn
sns.set(style="whitegrid")
plt.figure(figsize=(8, 6))
sns.barplot(x='importance', y='feature', data=features_df, palette='viridis')
plt.title("Оценка значимости признаков")
plt.xlabel("Важность")
plt.ylabel("Признак")
plt.tight_layout()
plt.show()
```

ПРИЛОЖЕНИЕ Б
CLASSIFICATION_RF_2.PY

```
import tkinter as tk
from tkinter import ttk
from tkinter.scrolledtext import ScrolledText
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.tree import plot_tree

def load_and_split_data(file_path):
    """
    Загружает датасет из файла, выделяет признаки и целевую переменную,
    а затем разбивает данные на обучающую и тестовую выборки.
    Используем признаки: bush_height, petal_count, petal_length, bud_diameter.
    Целевая переменная: flower_shape.
    """
    df = pd.read_csv(file_path)
    X = df[['bush_height', 'petal_count', 'petal_length', 'bud_diameter']]
    y = df['flower_shape']
    return train_test_split(X, y, test_size=0.2, random_state=42)

def train_model(X_train, y_train, n_estimators=100, random_state=42):
    """
    Создает и обучает модель случайного леса.
    """
    model = RandomForestClassifier(n_estimators=n_estimators, random_state=random_state)
    model.fit(X_train, y_train)
    return model

def evaluate_model(model, X_test, y_test):
    """
    Вычисляет точность, отчёт о классификации и матрицу сопряжённости.
    """
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    return acc, report, conf_matrix

def create_feature_importance_figure(model, feature_names):
    """
    Создает график значимости признаков (barplot) и возвращает объект Figure.
    Используется параметр hue для устранения предупреждения, после чего легенда
```

удаляется.
    """
    importances = model.feature_importances_
    imp_df = pd.DataFrame({
        'Feature': feature_names,
        'Importance': importances
    }).sort_values(by='Importance', ascending=False)

    sns.set(style="whitegrid")
    fig, ax = plt.subplots(figsize=(8, 6))
    sns.barplot(x='Importance', y='Feature', data=imp_df,
                hue='Feature', dodge=False, palette='viridis', ax=ax)
    legend = ax.get_legend()
    if legend is not None:
        legend.remove()
    ax.set_title("Оценка значимости признаков", fontsize=14)
    ax.set_xlabel("Важность", fontsize=12)
    ax.set_ylabel("Признак", fontsize=12)
    plt.tight_layout()
    return fig

def create_four_trees_figure(model, feature_names, class_names, max_depth=4):
    """
    Создает и возвращает объект Figure с 4 компактными деревьями, расположенными горизонтально.
    Каждое дерево визуализируется с помощью plot_tree с обрезкой до max_depth и уменьшенным шрифтом.
    """
    fig, axs = plt.subplots(1, 4, figsize=(16, 4))
    for i in range(4):
        estimator = model.estimators_[i]
        plot_tree(estimator,
                  feature_names=feature_names,
                  class_names=class_names,
                  filled=True,
                  rounded=True,
                  ax=axs[i],
                  fontsize=6,
                  max_depth=max_depth)
        axs[i].set_title(f"Дерево {i+1}", fontsize=8)
        axs[i].tick_params(axis='both', which='both', length=0)
    fig.tight_layout()
    return fig

def build_gui():
    # Загрузка данных, обучение модели и оценка
    X_train, X_test, y_train, y_test = load_and_split_data('roses_dataset_v2.csv')
    rf_model = train_model(X_train, y_train)

```
    acc, report, conf_matrix = evaluate_model(rf_model, X_test, y_test)

    # Формирование строкового вывода метрик
    metrics_text = (
        f"Точность модели: {acc:.4f}\n\n"
        f"Отчёт о классификации:\n{report}\n\n"
        f"Матрица сопряжённости:\n{conf_matrix}"
    )

    # Определяем имена классов (сортировка для согласованности)
    class_names = sorted(list(set(y_train) | set(y_test)))

    # Создаем фигуру для графика значимости признаков
    fig_feat = create_feature_importance_figure(rf_model, X_train.columns)
    # Создаем фигуру для 4 деревьев
    fig_trees = create_four_trees_figure(rf_model, X_train.columns, class_names, max_depth=4)

    # Создаем главное окно
    root = tk.Tk()
    root.title("Оценка модели случайного леса для классификации роз")
    root.geometry("1200x900")

    # Фрейм для текстового вывода результатов
    text_frame = ttk.Frame(root)
    text_frame.pack(side=tk.TOP, fill=tk.BOTH, expand=False, padx=10, pady=10)

    title_label = ttk.Label(text_frame, text="Результаты оценки модели", font=("Helvetica", 14,
"bold"))
    title_label.pack(pady=5)

    text_widget = ScrolledText(text_frame, wrap=tk.WORD, font=("Consolas", 10), height=15)
    text_widget.pack(fill=tk.BOTH, expand=True)
    text_widget.insert(tk.END, metrics_text)
    text_widget.configure(state='disabled')

    # Создаем Notebook для графиков
    notebook = ttk.Notebook(root)
    notebook.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

    # Вкладка для значимости признаков
    tab_feat = ttk.Frame(notebook)
    notebook.add(tab_feat, text="Значимость признаков")
    canvas_feat = FigureCanvasTkAgg(fig_feat, master=tab_feat)
    canvas_feat.draw()
    canvas_feat.get_tk_widget().pack(fill=tk.BOTH, expand=True)

    # Вкладка для деревьев
    tab_trees = ttk.Frame(notebook)
```

```
    notebook.add(tab_trees, text="Деревья")
    canvas_trees = FigureCanvasTkAgg(fig_trees, master=tab_trees)
    canvas_trees.draw()
    canvas_trees.get_tk_widget().pack(fill=tk.BOTH, expand=True)

    # Кнопка для выхода
    exit_button = ttk.Button(root, text="Выход", command=root.destroy)
    exit_button.pack(pady=10)

    root.mainloop()

if __name__ == '__main__':
    build_gui()
```

ПРИЛОЖЕНИЕ В
CLASSIFICATION_RF_3.PY

```
import tkinter as tk
from tkinter import ttk
from tkinter.scrolledtext import ScrolledText
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.tree import plot_tree


def load_and_split_data(file_path):
    """
    Загружает датасет из файла, выделяет признаки и целевую переменную,
    а затем разбивает данные на обучающую и тестовую выборки.
    Используем признаки: stem_thickness, leaf_length, leaf_width,
        fragrance_intensity, bloom_diameter, thorn_density.
    Целевая переменная: rose_group.
    """
    df = pd.read_csv(file_path)
    X = df[['stem_thickness', 'leaf_length', 'leaf_width',
        'fragrance_intensity', 'bloom_diameter', 'thorn_density']]
    y = df['rose_group']
    return train_test_split(X, y, test_size=0.2, random_state=42), y


def train_model(X_train, y_train, n_estimators=100, random_state=42):
    """
    Создает и обучает модель случайного леса.
    """
    model = RandomForestClassifier(n_estimators=n_estimators, random_state=random_state)
    model.fit(X_train, y_train)
    return model


def evaluate_model(model, X_test, y_test):
    """
    Вычисляет точность, отчёт о классификации и матрицу ошибок для модели.
    """
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
```

```
    return acc, report, conf_matrix


def create_feature_importance_figure(model, feature_names):
    """
    Создает график значимости признаков (barplot) и возвращает объект Figure.
    Используется параметр hue для устранения предупреждения, после чего легенда удаляется.
    """
    importances = model.feature_importances_
    imp_df = pd.DataFrame({
        'Feature': feature_names,
        'Importance': importances
    }).sort_values(by='Importance', ascending=False)

    sns.set(style="whitegrid")
    fig, ax = plt.subplots(figsize=(8, 6))
    sns.barplot(x='Importance', y='Feature', data=imp_df,
                hue='Feature', dodge=False, palette='viridis', ax=ax)
    legend = ax.get_legend()
    if legend is not None:
        legend.remove()
    ax.set_title("Оценка значимости признаков", fontsize=14)
    ax.set_xlabel("Важность", fontsize=12)
    ax.set_ylabel("Признак", fontsize=12)
    fig.tight_layout()
    return fig


def create_four_trees_figure(model, feature_names, class_names, max_depth=4):
    """
    Создает и возвращает объект Figure с 4 компактными деревьями, расположенными горизонтально.
    Каждое дерево визуализируется с помощью plot_tree с обрезкой до max_depth и уменьшенным шрифтом.
    """
    fig, axs = plt.subplots(1, 4, figsize=(16, 4))
    for i in range(4):
        estimator = model.estimators_[i]
        plot_tree(estimator, feature_names=feature_names, class_names=class_names,
                  filled=True, rounded=True, ax=axs[i], fontsize=6, max_depth=max_depth)
        axs[i].set_title(f"Дерево {i + 1}", fontsize=8)
        axs[i].tick_params(axis='both', which='both', length=0)
    fig.tight_layout()
    return fig


def build_gui():
```

```
    # Загрузка данных и предобработка
    (X_train, X_test, y_train, y_test), y_full =
load_and_split_data("synthetic_rose_groups_dataset.csv")
    rf_model = train_model(X_train, y_train)
    acc, clf_report, conf_matrix = evaluate_model(rf_model, X_test, y_test)

    # Формирование строкового вывода метрик
    metrics_text = (
        f"{'=' * 60}\n"
        f"{'ОЦЕНКА МОДЕЛИ':^60}\n"
        f"{'=' * 60}\n"
        f"Точность модели: {acc:.4f}\n\n"
        f"Отчёт о классификации:\n{clf_report}\n"
        f"Матрица ошибок:\n{conf_matrix}\n"
        f"{'=' * 60}\n"
    )

    # Определяем список классов для отображения (используем y_full)
    class_names = sorted(y_full.unique())

    # Создаем фигуру для графика значимости признаков
    fig_feat = create_feature_importance_figure(rf_model, X_train.columns)
    # Создаем фигуру с 4 деревьями, расположенными горизонтально
    fig_trees = create_four_trees_figure(rf_model, X_train.columns, class_names, max_depth=4)

    # Создаем главное окно Tkinter
    root = tk.Tk()
    root.title("Оценка модели RandomForest для групп роз")
    root.geometry("1200x900")

    # Фрейм для текстового вывода результатов
    text_frame = ttk.Frame(root)
    text_frame.pack(side=tk.TOP, fill=tk.BOTH, expand=False, padx=10, pady=10)

    title_label = ttk.Label(text_frame, text="Результаты оценки модели", font=("Helvetica", 14,
"bold"))
    title_label.pack(pady=5)

    text_widget = ScrolledText(text_frame, wrap=tk.WORD, font=("Consolas", 10), height=12)
    text_widget.pack(fill=tk.BOTH, expand=True)
    text_widget.insert(tk.END, metrics_text)
    text_widget.configure(state='disabled')

    # Создаем Notebook для графиков
    notebook = ttk.Notebook(root)
    notebook.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

    # Вкладка для значимости признаков
```

```
    tab_feat = ttk.Frame(notebook)
    notebook.add(tab_feat, text="Значимость признаков")
    canvas_feat = FigureCanvasTkAgg(fig_feat, master=tab_feat)
    canvas_feat.draw()
    canvas_feat.get_tk_widget().pack(fill=tk.BOTH, expand=True)

    # Вкладка для деревьев
    tab_trees = ttk.Frame(notebook)
    notebook.add(tab_trees, text="Деревья")
    canvas_trees = FigureCanvasTkAgg(fig_trees, master=tab_trees)
    canvas_trees.draw()
    canvas_trees.get_tk_widget().pack(fill=tk.BOTH, expand=True)

    # Кнопка для выхода
    exit_button = ttk.Button(root, text="Выход", command=root.destroy)
    exit_button.pack(pady=10)

    root.mainloop()


if __name__ == '__main__':
    build_gui()
```

ПРИЛОЖЕНИЕ Г
GENERATE_DATASET.PY

```
import numpy as np
import pandas as pd

n_samples = 10000
np.random.seed(42)

# Функция для генерации количества лепестков с шумом
def generate_petals(class_label):
    if class_label == 'Простой':
        return np.random.randint(3, 9) + np.random.choice([-1, 0, 1])
    elif class_label == 'Полумахровый':
        return np.random.randint(8, 21) + np.random.choice([-1, 0, 1])
    elif class_label == 'Умеренно махровый':
        return np.random.randint(21, 30) + np.random.choice([-1, 0, 1])
    elif class_label == 'Среднемахровый':
        return np.random.randint(30, 40) + np.random.choice([-1, 0, 1])
    elif class_label == 'Густомахровый':
        return np.random.randint(41, 51) + np.random.choice([-1, 0, 1])
    else:
        raise ValueError("Неизвестный класс")

# Определяем вероятности появления каждого класса
classes = ['Простой', 'Полумахровый', 'Умеренно махровый', 'Среднемахровый',
'Густомахровый']
class_probs = [0.2, 0.3, 0.25, 0.15, 0.1]  # сумма должна быть 1

# Генерируем случайные классы для каждого объекта
labels = np.random.choice(classes, size=n_samples, p=class_probs)

# Генерируем датасет
data = []
for label in labels:
    petals = generate_petals(label)

    # Генерируем высоту с дополнительным шумом
    base_height = 20  # базовая высота в см
    height = base_height + 0.7 * petals + np.random.normal(0, 5)  # увеличенный шум

    data.append({
        'petals': max(3, petals),  # Минимум 3 лепестка
        'height': round(height, 2),
        'class': label
    })

# Создаем DataFrame и сохраняем в CSV
df = pd.DataFrame(data)
df.to_csv('roses_dataset.csv', index=False)
```

ПРИЛОЖЕНИЕ Д
GENERATE_DATASET_2.PY

```
import numpy as np
import pandas as pd

# Фиксируем seed для воспроизводимости
np.random.seed(42)

# Количество генерируемых записей
n_samples = 10000

# Определяем классы форм цветка
flower_shapes = [
    "плоская", "кувшинчатая", "округлая", "помпонная",
    "чашеобразная", "розеточная", "коническая", "крестовидно-розеточная"
]

# Задаём параметры для каждого признака по форме цветка
# Параметры: (среднее, стандартное отклонение)

# Высота куста (см)
shape_height_params = {
    "плоская": (60, 5),
    "кувшинчатая": (80, 5),
    "округлая": (70, 5),
    "помпонная": (55, 5),
    "чашеобразная": (65, 5),
    "розеточная": (75, 5),
    "коническая": (85, 5),
    "крестовидно-розеточная": (68, 5)
}

# Количество лепестков (целое число)
shape_petal_count = {
    "плоская": (25, 1),
    "кувшинчатая": (35, 1),
    "округлая": (30, 1),
    "помпонная": (20, 1),
    "чашеобразная": (28, 1),
    "розеточная": (32, 1),
    "коническая": (38, 1),
    "крестовидно-розеточная": (26, 1)
}

# Длина лепестка (см)
shape_petal_length = {
    "плоская": (5.0, 0.2),
    "кувшинчатая": (6.5, 0.2),
    "округлая": (5.8, 0.2),
    "помпонная": (4.0, 0.2),
    "чашеобразная": (5.2, 0.2),
    "розеточная": (6.0, 0.2),
```

```
    "коническая": (7.2, 0.2),
    "крестовидно-розеточная": (5.5, 0.2)
}

# Диаметр бутона (см)
shape_bud_diameter = {
    "плоская": (3.0, 0.1),
    "кувшинчатая": (3.8, 0.1),
    "округлая": (3.2, 0.1),
    "помпонная": (2.5, 0.1),
    "чашеобразная": (3.0, 0.1),
    "розеточная": (3.6, 0.1),
    "коническая": (4.2, 0.1),
    "крестовидно-розеточная": (3.1, 0.1)
}

# Новый признак: ширина лепестка (см)
shape_petal_width = {
    "плоская": (1.2, 0.1),
    "кувшинчатая": (1.8, 0.1),
    "округлая": (1.4, 0.1),
    "помпонная": (1.0, 0.1),
    "чашеобразная": (1.3, 0.1),
    "розеточная": (1.7, 0.1),
    "коническая": (2.0, 0.1),
    "крестовидно-розеточная": (1.3, 0.1)
}

# Список для накопления сгенерированных записей
data = []

for _ in range(n_samples):
    # Случайно выбираем форму цветка
    shape = np.random.choice(flower_shapes)

    # Генерируем высоту куста
    mean_height, std_height = shape_height_params[shape]
    bush_height = np.random.normal(mean_height, std_height)

    # Генерируем количество лепестков
    mean_pc, std_pc = shape_petal_count[shape]
    petal_count = int(np.round(np.random.normal(mean_pc, std_pc)))

    # Генерируем длину лепестка
    mean_pl, std_pl = shape_petal_length[shape]
    petal_length = np.random.normal(mean_pl, std_pl)

    # Генерируем диаметр бутона
    mean_bd, std_bd = shape_bud_diameter[shape]
    bud_diameter = np.random.normal(mean_bd, std_bd)

    # Генерируем ширину лепестка
```

```
    mean_pw, std_pw = shape_petal_width[shape]
    petal_width = np.random.normal(mean_pw, std_pw)

    data.append({
        "flower_shape": shape,
        "bush_height": bush_height,
        "petal_count": petal_count,
        "petal_length": petal_length,
        "bud_diameter": bud_diameter,
        "petal_width": petal_width
    })

# Создаем DataFrame
df = pd.DataFrame(data)

# Внедряем выбросы: изменяем значения в 0.5% записей
n_outliers = int(0.005 * n_samples)
outlier_indices = np.random.choice(df.index, size=n_outliers, replace=False)
# Для выбросов умножаем bush_height и petal_length на коэффициент [1.1, 1.3]
df.loc[outlier_indices, "bush_height"] *= np.random.uniform(1.1, 1.3, size=n_outliers)
df.loc[outlier_indices, "petal_length"] *= np.random.uniform(1.1, 1.3, size=n_outliers)

# Внедряем дополнительный шум: изменяем случайным образом один признак в 2%
записей
n_noisy = int(0.02 * n_samples)
noisy_indices = np.random.choice(df.index, size=n_noisy, replace=False)

for idx in noisy_indices:
    feature = np.random.choice(["bush_height", "petal_count", "petal_length", "bud_diameter",
"petal_width"])
    # Добавляем шум с небольшим стандартным отклонением (0.5)
    df.loc[idx, feature] = df.loc[idx, feature] + np.random.normal(0, 0.5)

# Перемешиваем данные для случайного распределения записей
df = df.sample(frac=1).reset_index(drop=True)

# Сохраняем датасет в CSV файл
df.to_csv("roses_dataset_v2.csv", index=False)

print(f"Сгенерирован датасет с {n_samples} образцами и сохранён в 'roses_dataset_v2.csv'")
```

ПРИЛОЖЕНИЕ Е
GENERATE_DATASET_3.PY

```
import numpy as np
import pandas as pd

# Фиксируем seed для воспроизводимости
np.random.seed(42)

# Количество генерируемых записей
n_samples = 10000

# Определяем группы роз
rose_groups = [
    "Гибридные чайные", "Флорибунда", "Грандифлора",
    "Плетистые", "Кустовые", "Миниатюрные"
]

# Задаём параметры для каждого признака по каждой группе: (среднее, стандартное
отклонение)
# Толщина стебля (мм)
group_stem_thickness = {
    "Гибридные чайные": (4.0, 0.5),
    "Флорибунда": (3.5, 0.5),
    "Грандифлора": (4.2, 0.4),
    "Плетистые": (3.8, 0.6),
    "Кустовые": (4.5, 0.5),
    "Миниатюрные": (2.5, 0.3)
}

# Длина листа (см)
group_leaf_length = {
    "Гибридные чайные": (6.0, 0.7),
    "Флорибунда": (5.5, 0.6),
    "Грандифлора": (6.8, 0.7),
    "Плетистые": (7.5, 0.8),
    "Кустовые": (8.0, 0.8),
    "Миниатюрные": (3.5, 0.5)
}

# Ширина листа (см)
group_leaf_width = {
    "Гибридные чайные": (4.0, 0.5),
    "Флорибунда": (3.8, 0.4),
    "Грандифлора": (4.2, 0.5),
    "Плетистые": (5.0, 0.6),
    "Кустовые": (5.5, 0.7),
    "Миниатюрные": (2.5, 0.4)
}

# Интенсивность аромата (от 1 до 10)
group_fragrance = {
    "Гибридные чайные": (7.0, 1.0),
```

```
    "Флорибунда": (6.0, 1.0),
    "Грандифлора": (7.5, 1.0),
    "Плетистые": (8.0, 1.2),
    "Кустовые": (5.0, 1.0),
    "Миниатюрные": (4.0, 0.8)
}

# Диаметр цветка (см)
group_bloom_diameter = {
    "Гибридные чайные": (8.0, 1.0),
    "Флорибунда": (6.5, 0.8),
    "Грандифлора": (9.5, 1.2),
    "Плетистые": (7.5, 1.0),
    "Кустовые": (7.0, 1.0),
    "Миниатюрные": (3.5, 0.6)
}

# Плотность шипов (количество шипов на 10 см)
group_thorn_density = {
    "Гибридные чайные": (8, 1.5),
    "Флорибунда": (7, 1.0),
    "Грандифлора": (9, 1.0),
    "Плетистые": (6, 1.0),
    "Кустовые": (10, 1.5),
    "Миниатюрные": (5, 0.8)
}

data = []

for _ in range(n_samples):
    # Случайно выбираем группу роз
    group = np.random.choice(rose_groups)

    # Генерируем значения признаков согласно параметрам группы
    stem_thickness = np.random.normal(*group_stem_thickness[group])
    leaf_length = np.random.normal(*group_leaf_length[group])
    leaf_width = np.random.normal(*group_leaf_width[group])
    fragrance_intensity = np.random.normal(*group_fragrance[group])
    bloom_diameter = np.random.normal(*group_bloom_diameter[group])
    thorn_density = np.random.normal(*group_thorn_density[group])

    data.append({
        "rose_group": group,
        "stem_thickness": round(stem_thickness, 2),
        "leaf_length": round(leaf_length, 2),
        "leaf_width": round(leaf_width, 2),
        "fragrance_intensity": round(fragrance_intensity, 2),
        "bloom_diameter": round(bloom_diameter, 2),
        "thorn_density": round(thorn_density, 2)
    })

df = pd.DataFrame(data)
```

```
# Внедряем небольшие выбросы: изменяем значения в 0.5% записей
n_outliers = int(0.005 * n_samples)
outlier_indices = np.random.choice(df.index, size=n_outliers, replace=False)
# Для выбросов изменим bloom_diameter и stem_thickness на случайный коэффициент [1.1,
1.3]
df.loc[outlier_indices, "bloom_diameter"] *= np.random.uniform(1.1, 1.3, size=n_outliers)
df.loc[outlier_indices, "stem_thickness"] *= np.random.uniform(1.1, 1.3, size=n_outliers)

# Внедряем дополнительный шум: изменяем случайным образом один признак в 2%
записей
n_noisy = int(0.02 * n_samples)
noisy_indices = np.random.choice(df.index, size=n_noisy, replace=False)
for idx in noisy_indices:
    feature = np.random.choice(["stem_thickness", "leaf_length", "leaf_width",
"fragrance_intensity", "bloom_diameter", "thorn_density"])
    df.loc[idx, feature] = df.loc[idx, feature] + np.random.normal(0, 0.5)

# Перемешиваем записи
df = df.sample(frac=1).reset_index(drop=True)

# Сохраняем датасет в CSV файл
df.to_csv("synthetic_rose_groups_dataset.csv", index=False)
print(f"Сгенерирован датасет с {n_samples} образцами и сохранён в
'synthetic_rose_groups_dataset.csv'")
```