Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №5
дисциплины
«Искусственный интеллект и машинное обучение»
Вариант 1

Выполнил:
Бакулин Вадим Романович
2 курс, группа ИТС-б-о-23-1,
11.03.02 «Инфокоммуникационные технологии и системы связи»,
направленность (профиль)
«Инфокоммуникационные системы и сети», очная форма обучения

_____
(подпись)

Проверил:
Доцент департамента цифровых, робототехнических систем и электроники Воронкин Р.А.

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Тема: Введение в pandas: изучение структуры DataFrame и базовых операций

Цель: познакомить с основами работы с библиотекой pandas, в частности, со структурой данных DataFrame.

Ссылка на репозиторий: https://github.com/zepteloid/AI_ML_LR_5

Порядок выполнения работы:

1. Выполнение задания №1.
2. Создание DataFrame из словоря списков с данными о пяти сотрудниках:

```python
import pandas as pd

# Создаем DataFrame из словаря списков
data_dict = {
    "ID": [1, 2, 3, 4, 5],
    "Имя": ["Иван", "Ольга", "Алексей", "Мария", "Сергей"],
    "Возраст": [25, 30, 40, 35, 28],
    "Должность": ["Инженер", "Аналитик", "Менеджер", "Программист", "Специалист"],
    "Отдел": ["IT", "Маркетинг", "Продажи", "IT", "HR"],
    "Зарплата": [60000, 75000, 90000, 80000, 50000],
    "Стаж работы": [2, 5, 15, 7, 3]
}

df_dict = pd.DataFrame(data_dict)
display(df_dict)
```

Рисунок 1. Задание №1.1

3. Создание DataFrame из списка словарей

```python
# Создаем DataFrame из списка словарей
data_list = [
    {"ID": 1, "Имя": "Иван", "Возраст": 25, "Должность": "Инженер", "Отдел": "IT", "Зарплата": 60000, "Стаж работы": 2},
    {"ID": 2, "Имя": "Ольга", "Возраст": 30, "Должность": "Аналитик", "Отдел": "Маркетинг", "Зарплата": 75000, "Стаж работы": 5},
    {"ID": 3, "Имя": "Алексей", "Возраст": 40, "Должность": "Менеджер", "Отдел": "Продажи", "Зарплата": 90000, "Стаж работы": 15},
    {"ID": 4, "Имя": "Мария", "Возраст": 35, "Должность": "Программист", "Отдел": "IT", "Зарплата": 80000, "Стаж работы": 7},
    {"ID": 5, "Имя": "Сергей", "Возраст": 28, "Должность": "Специалист", "Отдел": "HR", "Зарплата": 50000, "Стаж работы": 3}
]

df_list = pd.DataFrame(data_list)
display(df_list)
```

Рисунок 2. Задание №1.2

4. Создание DataFrame из массива NumPy

```python
import numpy as np

# Создаем массив NumPy с случайными возрастами от 20 до 60
np.random.seed(42)
ages = np.random.randint(20, 61, size=5)

# Создаем DataFrame
df_numpy = pd.DataFrame({
    "ID": [1, 2, 3, 4, 5],
    "Имя": ["Иван", "Ольга", "Алексей", "Мария", "Сергей"],
    "Возраст": ages,
    "Должность": ["Инженер", "Аналитик", "Менеджер", "Программист", "Специалист"],
    "Отдел": ["IT", "Маркетинг", "Продажи", "IT", "HR"],
    "Зарплата": [60000, 75000, 90000, 80000, 50000],
    "Стаж работы": [2, 5, 15, 7, 3]
})

display(df_numpy)
```

Рисунок 3. Задание №1.3

5. Проверка типов данных

```python
# Проверяем типы данных в каждом DataFrame
print("Типы данных в df_dict:")
df_dict.info()

print("\nТипы данных в df_list:")
df_list.info()

print("\nТипы данных в df_numpy:")
df_numpy.info()
```

Рисунок 4. Задание №1.4

6. Выполнение задания №2.
7. Работа с CSV

```python
# Создаем DataFrame из предоставленных данных
data = {
    "ID": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
    "Имя": ["Иван", "Ольга", "Алексей", "Мария", "Сергей", "Анна", "Дмитрий", "Елена",
            "Виктор", "Алиса", "Павел", "Светлана", "Роман", "Татьяна", "Николай",
            "Валерия", "Григорий", "Юлия", "Степан", "Василиса"],
    "Возраст": [25, 30, 40, 35, 28, 32, 45, 29, 31, 27, 33, 26, 42, 37, 39, 24, 50, 45, 41, 38],
    "Должность": ["Инженер", "Аналитик", "Менеджер", "Программист", "Специалист", "Разработчик",
                  "HR", "Маркетолог", "Юрист", "Дизайнер", "Администратор", "Тестировщик",
                  "Финансист", "Редактор", "Логист", "SEO-специалист", "Бухгалтер", "Директор",
                  "Экономист", "Проект-менеджер"],
    "Отдел": ["IT", "Маркетинг", "Продажи", "IT", "HR", "IT", "HR", "Маркетинг", "Юридический",
              "Дизайн", "Администрация", "Тестирование", "Финансы", "Редакция", "Логистика", "SEO",
              "Бухгалтерия", "Финансы", "Экономика", "Продажи"],
    "Зарплата": [60000, 75000, 90000, 80000, 50000, 85000, 48000, 70000, 95000, 62000, 55000,
                 67000, 105000, 72000, 75000, 64000, 110000, 150000, 98000, 88000],
    "Стаж работы": [2, 5, 15, 7, 3, 6, 12, 4, 10, 5, 7, 2, 20, 9, 11, 3, 25, 20, 14, 8]
}

df = pd.DataFrame(data)

# 1. Сохраняем DataFrame в CSV файл
df.to_csv("employees_table1.csv", index=False, encoding='utf-8-sig')
print("Таблица успешно сохранена в employees_table1.csv")

# 2. Загружаем данные обратно в DataFrame
df_loaded = pd.read_csv("employees_table1.csv")

# 3. Выводим первые 5 строк загруженного DataFrame
print("\nПервые 5 строк загруженной таблицы:")
display(df_loaded.head())
```

Рисунок 5. Задание №2.1

8. Работа с Excel

```python
# Создаем DataFrame из предоставленных данных о клиентах
clients_data = {
    "ID": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
    "Имя": ["Иван", "Ольга", "Алексей", "Мария", "Сергей", "Анна", "Дмитрий", "Елена",
            "Виктор", "Алиса", "Павел", "Светлана", "Роман", "Татьяна", "Николай",
            "Валерия", "Григорий", "Юлия", "Степан", "Василиса"],
    "Возраст": [34, 27, 45, 38, 29, 50, 31, 40, 28, 33, 46, 37, 41, 25, 39, 42, 49, 50, 30, 35],
    "Город": ["Москва", "Санкт-Петербург", "Казань", "Новосибирск", "Екатеринбург",
              "Воронеж", "Челябинск", "Краснодар", "Ростов-на-Дону", "Уфа", "Омск",
              "Пермь", "Тюмень", "Саратов", "Самара", "Волгоград", "Барнаул", "Иркутск",
              "Хабаровск", "Томск"],
    "Баланс на счете": [120000, 80000, 150000, 200000, 95000, 300000, 140000, 175000,
                        110000, 98000, 250000, 210000, 135000, 155000, 125000, 180000,
                        275000, 320000, 105000, 90000],
    "Кредитная история": ["Хорошая", "Средняя", "Плохая", "Хорошая", "Средняя", "Отличная",
                          "Средняя", "Хорошая", "Плохая", "Средняя", "Хорошая", "Отличная",
                          "Средняя", "Хорошая", "Средняя", "Плохая", "Отличная", "Хорошая",
                          "Средняя", "Плохая"]
}

df_clients = pd.DataFrame(clients_data)

# 1. Сохраняем DataFrame в Excel файл на лист "Клиенты"
with pd.ExcelWriter('data.xlsx', engine='openpyxl') as writer:
    df_clients.to_excel(writer, sheet_name='Клиенты', index=False)
print("Таблица успешно сохранена в data.xlsx на лист 'Клиенты'")

# 2. Загружаем данные обратно в DataFrame
df_loaded = pd.read_excel('data.xlsx', sheet_name='Клиенты')

# 3. Выводим первые 5 строк загруженного DataFrame
print("\nПервые 5 строк загруженной таблицы клиентов:")
display(df_loaded.head())
```

Рисунок 6. Задание №2.2

9. Работа с JSON

```python
# Создаем DataFrame из предоставленных данных
employees_data = {
    "ID": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
    "Имя": ["Иван", "Ольга", "Алексей", "Мария", "Сергей", "Анна", "Дмитрий", "Елена",
            "Виктор", "Алиса", "Павел", "Светлана", "Роман", "Татьяна", "Николай",
            "Валерия", "Григорий", "Юлия", "Степан", "Василиса"],
    "Возраст": [25, 30, 40, 35, 28, 32, 45, 29, 31, 27, 33, 26, 42, 37, 39, 24, 50, 45, 41, 38],
    "Должность": ["Инженер", "Аналитик", "Менеджер", "Программист", "Специалист",
                  "Разработчик", "HR", "Маркетолог", "Юрист", "Дизайнер",
                  "Администратор", "Тестировщик", "Финансист", "Редактор",
                  "Логист", "SEO-специалист", "Бухгалтер", "Директор",
                  "Экономист", "Проект-менеджер"],
    "Отдел": ["IT", "Маркетинг", "Продажи", "IT", "HR", "IT", "HR", "Маркетинг",
              "Юридический", "Дизайн", "Администрация", "Тестирование", "Финансы",
              "Редакция", "Логистика", "SEO", "Бухгалтерия", "Финансы",
              "Экономика", "Продажи"],
    "Зарплата": [60000, 75000, 90000, 80000, 50000, 85000, 48000, 70000, 95000,
                 62000, 55000, 67000, 105000, 72000, 75000, 64000, 110000,
                 150000, 98000, 88000],
    "Стаж работы": [2, 5, 15, 7, 3, 6, 12, 4, 10, 5, 7, 2, 20, 9, 11, 3, 25, 20, 14, 8]
}

df_employees = pd.DataFrame(employees_data)

# 1. Экспортируем DataFrame в JSON файл
df_employees.to_json("employees_table1.json", orient="records", force_ascii=False, indent=4)
print("Таблица успешно экспортирована в employees_table1.json")

# 2. Читаем данные обратно в DataFrame
df_loaded = pd.read_json("employees_table1.json")

# 3. Выводим первые 5 строк загруженного DataFrame
print("\nПервые 5 строк загруженной таблицы сотрудников:")
display(df_loaded.head())
```

Рисунок 7. Задание №2.3

10.Выполнение задания №3.

```
df = pd.DataFrame(data)

# 1. Получаем информацию о сотруднике с ID = 5 с помощью .loc[]
employee_id5 = df.loc[df['ID'] == 5]
print("1. Информация о сотруднике с ID = 5:")
display(employee_id5)

# 2. Выводим возраст третьего сотрудника в таблице с .iloc[]
age_third_employee = df.iloc[2, 2]  # Индексация начинается с 0, третья строка - индекс 2
print("\n2. Возраст третьего сотрудника:", age_third_employee)

# 3. Выводим название отдела для сотрудника "Мария" с .at[]
# Сначала находим индекс строки с именем "Мария"
maria_index = df[df['Имя'] == 'Мария'].index[0]
department_maria = df.at[maria_index, 'Отдел']
print("\n3. Отдел сотрудника Мария:", department_maria)

# 4. Выводим зарплату сотрудника в четвертой строке и пятом столбце с .iat[]
# Четвертая строка - индекс 3, пятый столбец - индекс 4 (нумерация с 0)
salary = df.iat[3, 4]
print("\n4. Зарплата сотрудника в 4 строке и 5 столбце:", salary)
```

Рисунок 8. Задание №3

11. Выполнение задания №4.

```python
df = pd.DataFrame(data)

# 1. Добавляем столбец "Категория зарплаты"
df["Категория зарплаты"] = pd.cut(df["Зарплата"],
                                  bins=[0, 60000, 100000, float('inf')],
                                  labels=["Низкая", "Средняя", "Высокая"])
print("1. DataFrame с добавленным столбцом 'Категория зарплаты':")
display(df)

# 2. Добавляем нового сотрудника с ID 21
new_id = max(df['ID']) + 1
df.loc[len(df)] = [new_id, "Антон", 32, "Разработчик", "IT", 85000, 6, "Средняя"]
print("\n2. DataFrame после добавления нового сотрудника:")
display(df.tail(1))  # Показываем только последнюю добавленную строку

# 3. Добавляем двух новых сотрудников с помощью pd.concat()
new_employees = pd.DataFrame({
    "ID": [22, 23],
    "Имя": ["Артем", "Екатерина"],
    "Возраст": [29, 36],
    "Должность": ["Аналитик", "Менеджер"],
    "Отдел": ["Маркетинг", "Продажи"],
    "Зарплата": [72000, 95000],
    "Стаж работы": [4, 8],
    "Категория зарплаты": ["Средняя", "Средняя"]
})

df = pd.concat([df, new_employees], ignore_index=True)
print("\n3. DataFrame после добавления двух новых сотрудников:")
display(df.tail(2))  # Показываем две последние добавленные строки

# Обновленный DataFrame целиком
print("\nПолный обновленный DataFrame:")
display(df)
```

Рисунок 9. Задание №4

12. Выполнение задания №5.

```python
df = pd.DataFrame(data)

# 1. Удаляем столбец "Категория зарплаты"
print("1. Удаление столбца 'Категория зарплаты':")
df = df.drop(columns=["Категория зарплаты"])
display(df.head(3))

# 2. Удаляем строку с ID = 10
print("\n2. Удаление строки с ID = 10:")
df = df[df["ID"] != 10]
display(df[df["ID"].isin([9, 10, 11])])  # Проверка удаления

# 3. Удаляем все строки, где Стаж работы < 3 лет
print("\n3. Удаление строк с стажем работы < 3 лет:")
df = df[df["Стаж работы"] >= 3]
# Проверка что строк с стажем < 3 нет
print("Строки с стажем < 3 (должно быть пусто):")
display(df[df["Стаж работы"] < 3])
print("Минимальный стаж в таблице:", df["Стаж работы"].min())

# 4. Удаляем все столбцы, кроме Имя, Должность, Зарплата
print("\n4. Удаление всех столбцов, кроме Имя, Должность, Зарплата:")
df = df[["Имя", "Должность", "Зарплата"]]
display(df.head())

# Итоговый DataFrame
print("\nИтоговый DataFrame:")
display(df)
```

Рисунок 10. Задание №5

13. Выполнение задания №6

```python
df = pd.DataFrame(data)

# 1. Фильтрация клиентов из Москвы или Санкт-Петербурга с .isin()
print("1. Клиенты из Москвы или Санкт-Петербурга:")
msk_spb = df[df["Город"].isin(["Москва", "Санкт-Петербург"])]
display(msk_spb)

# 2. Фильтрация клиентов с балансом от 100000 до 250000 с .between()
print("\n2. Клиенты с балансом от 100000 до 250000:")
balance_filter = df[df["Баланс на счете"].between(100000, 250000)]
display(balance_filter)

# 3. Фильтрация клиентов с хорошей кредитной историей и балансом > 150000 с .query()
print("\n3. Клиенты с хорошей кредитной историей и балансом > 150000:")
good_history = df.query("`Кредитная история` == 'Хорошая' and `Баланс на счете` > 150000")
display(good_history)
```

Рисунок 11. Задание №6

14. Выполнение задания №7

```
df = pd.DataFrame(data)

# 1. Количество непустых значений в каждом столбце
print("1. Количество непустых значений в каждом столбце:")
non_null_counts = df.count()
display(non_null_counts)

# 2. Частота встречаемости значений в столбце "Город"
print("\n2. Частота встречаемости городов:")
city_counts = df["Город"].value_counts()
display(city_counts)

# 3. Количество уникальных значений в указанных столбцах
print("\n3. Количество уникальных значений:")
unique_counts = df[["Город", "Возраст", "Баланс на счете"]].nunique()
display(unique_counts)
```

Рисунок 12. Задание №7

15. Выполнение задания №8

```
df = pd.DataFrame(data)

# 1. Подсчет количества NaN в каждом столбце
print("1. Количество пропущенных значений (NaN) в каждом столбце:")
nan_counts = df.isna().sum()
display(nan_counts)

# 2. Подсчет количества заполненных значений в каждом столбце
print("\n2. Количество заполненных значений в каждом столбце:")
notna_counts = df.notna().sum()
display(notna_counts)

# 3. DataFrame без пропущенных значений
print("\n3. DataFrame без пропущенных значений:")
df_clean = df.dropna()
display(df_clean)
```

Рисунок 13. Задание №8

16. Выполнение индивидуального задания

```
import pandas as pd
from datetime import datetime
import ipywidgets as widgets
from IPython.display import display, clear_output
import os
```

Рисунок 14. Необходимые зависимости

17. Листинг программы:

```
class StudentPerformanceJupyter:
    def __init__(self):
        self.df = pd.DataFrame(columns=[
            'Фамилия и инициалы',
            'Номер группы',
            'Математика',
            'Физика',
            'Программирование',
            'Средний балл'
        ])
        self.setup_ui()

    def setup_ui(self):
        """Настройка пользовательского интерфейса"""
        # Создаем виджеты
        self.create_widgets()

        # Собираем интерфейс
        self.assemble_interface()

        # Отображаем интерфейс
        self.display_interface()

    def create_widgets(self):
        """Создание всех виджетов"""
        # Виджеты для добавления студента
        self.name_input = widgets.Text(description="ФИО:", layout={'width': '400px'})
        self.group_input = widgets.Text(description="Группа:")
        self.math_input = widgets.FloatSlider(
            description="Математика:",
            min=1, max=5, step=0.5,
            value=3,
            style={'description_width': 'initial'}
        )
        self.physics_input = widgets.FloatSlider(
            description="Физика:",
            min=1, max=5, step=0.5,
            value=3,
            style={'description_width': 'initial'}
        )
        self.programming_input = widgets.FloatSlider(
            description="Программирование:",
            min=1, max=5, step=0.5,
            value=3,
            style={'description_width': 'initial'}
        )
        self.add_button = widgets.Button(
            description="Добавить студента",
            button_style='success'
        )
        self.add_button.on_click(self.add_student_handler)
```

```
        # Виджеты для управления данными
        self.show_button = widgets.Button(
            description="Показать успешных студентов (средний балл > 4.0)",
            button_style='info'
        )
        self.show_button.on_click(self.show_top_students_handler)

        self.save_input = widgets.Text(description="Имя файла для сохранения:")
        self.save_button = widgets.Button(
            description="Сохранить в Parquet",
            button_style='primary'
        )
        self.save_button.on_click(self.save_handler)

        self.load_input = widgets.Text(description="Имя файла для загрузки:")
        self.load_button = widgets.Button(
            description="Загрузить из Parquet",
            button_style='primary'
        )
        self.load_button.on_click(self.load_handler)

        self.delete_input = widgets.Text(description="Группа для удаления:")
        self.delete_button = widgets.Button(
            description="Удалить записи группы",
            button_style='danger'
        )
        self.delete_button.on_click(self.delete_handler)

        # Выходная область
        self.output = widgets.Output()

        # Горизонтальные разделители
        self.hr = widgets.HTML(value="<hr>")
```

```python
def assemble_interface(self):
    """Компоновка интерфейса"""
    self.form = widgets.VBox([
        widgets.Label("Добавление нового студента:"),
        self.name_input,
        self.group_input,
        self.math_input,
        self.physics_input,
        self.programming_input,
        self.add_button,
        self.hr,
        self.show_button,
        self.hr,
        widgets.Label("Сохранение/загрузка данных:"),
        self.save_input,
        self.save_button,
        self.load_input,
        self.load_button,
        self.hr,
        widgets.Label("Удаление данных:"),
        self.delete_input,
        self.delete_button,
        self.hr,
        widgets.Label("Результаты:"),
        self.output
    ], layout=widgets.Layout(width='100%'))

def display_interface(self):
    """Отображение интерфейса"""
    display(self.form)

# Обработчики событий
def add_student_handler(self, button):
    """Обработчик добавления студента"""
    with self.output:
        clear_output()
        try:
            self.add_student(
                name=self.name_input.value,
                group=self.group_input.value,
                math=self.math_input.value,
                physics=self.physics_input.value,
                programming=self.programming_input.value
            )
            print("✅ Студент успешно добавлен!")
            display(self.df.tail())
        except Exception as e:
            print(f"❌ Ошибка: {str(e)}")
```

```python
def show_top_students_handler(self, button):
    """Обработчик показа успешных студентов"""
    with self.output:
        clear_output()
        result = self.show_top_students()
        print(result)
        if not self.df.empty:
            print("\nТекущие данные:")
            display(self.df)

def save_handler(self, button):
    """Обработчик сохранения данных"""
    with self.output:
        clear_output()
        try:
            filename = self.save_input.value.strip()
            if not filename:
                raise ValueError("Имя файла не может быть пустым")

            result = self.save_to_parquet(filename)
            print(f"✅ {result}")
        except Exception as e:
            print(f"❌ Ошибка при сохранении: {str(e)}")

def load_handler(self, button):
    """Обработчик загрузки данных"""
    with self.output:
        clear_output()
        try:
            filename = self.load_input.value.strip()
            if not filename:
                raise ValueError("Имя файла не может быть пустым")

            result = self.load_from_parquet(filename)
            print(f"✅ {result}")
            if not self.df.empty:
                display(self.df)
        except Exception as e:
            print(f"❌ Ошибка при загрузке: {str(e)}")
```

```python
def delete_handler(self, button):
    """Обработчик удаления данных"""
    with self.output:
        clear_output()
        try:
            group = self.delete_input.value.strip()
            if not group:
                raise ValueError("Номер группы не может быть пустым")

            result = self.delete_by_group(group)
            print(f"✅ {result}")
            if not self.df.empty:
                display(self.df)
            else:
                print("В DataFrame нет данных")
        except Exception as e:
            print(f"❌ Ошибка при удалении: {str(e)}")

# Основные методы
def add_student(self, name, group, math, physics, programming):
    """Добавление студента в DataFrame"""
    avg_score = (math + physics + programming) / 3
    new_row = {
        'Фамилия и инициалы': name,
        'Номер группы': group,
        'Математика': math,
        'Физика': physics,
        'Программирование': programming,
        'Средний балл': round(avg_score, 2)
    }

    self.df = pd.concat([self.df, pd.DataFrame([new_row])], ignore_index=True)
    self.df.sort_values(by='Номер группы', inplace=True)
    self.df.reset_index(drop=True, inplace=True)

def show_top_students(self):
    """Показать студентов со средним баллом > 4.0"""
    top_students = self.df[self.df['Средний балл'] > 4.0]

    if top_students.empty:
        return "Нет студентов со средним баллом больше 4.0"
    else:
        result = ["Студенты со средним баллом больше 4.0:"]
        for _, row in top_students.iterrows():
            result.append(f"{row['Фамилия и инициалы']}, группа {row['Номер группы']}, средний балл: {row['Средний балл']}")
        return "\n".join(result)
```

```python
    def save_to_parquet(self, filename):
        """Сохранение данных в Parquet"""
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        full_filename = f"{filename}_{timestamp}.parquet"
        self.df.to_parquet(full_filename)
        return f"Данные сохранены в файл: {full_filename}"

    def load_from_parquet(self, filename):
        """Загрузка данных из Parquet"""
        if not os.path.exists(filename):
            raise FileNotFoundError(f"Файл {filename} не найден")

        self.df = pd.read_parquet(filename)
        return f"Загружено {len(self.df)} записей из файла {filename}"

    def delete_by_group(self, group):
        """Удаление записей по номеру группы"""
        initial_count = len(self.df)
        self.df = self.df[self.df['Номер группы'] != group]
        removed_count = initial_count - len(self.df)
        return f"Удалено {removed_count} записей группы {group}"

# Запускаем приложение
if __name__ == "__main__":
    app = StudentPerformanceJupyter()
```

Ответы на контрольные вопросы:

1. Как создать pandas.DataFrame из словаря списков?

```
import pandas as pd
data = {'Имя': ['Анна', 'Борис', 'Мария'], 'Возраст': [25, 30, 28]}
df = pd.DataFrame(data)
```

2. В чем отличие создания DataFrame из списка словарей и словаря списков?

Словарь списков: ключи - названия столбцов, значения - данные столбцов

Список словарей: каждый словарь - строка, ключи - названия столбцов

3. Как создать pandas.DataFrame из массива NumPy?

```
import numpy as np
arr = np.array([[1, 2], [3, 4]])
df = pd.DataFrame(arr, columns=['A', 'B'])
```

4. Как загрузить DataFrame из CSV-файла, указав разделитель ;?

```
df = pd.read_csv('file.csv', sep=';')
```

5. Как загрузить данные из Excel в pandas.DataFrame и выбрать конкретный лист?

```
df = pd.read_excel('file.xlsx', sheet_name='Лист1')
```

6. Чем отличается чтение данных из JSON и Parquet в pandas?

JSON: текстовый формат, медленнее, больше места

Parquet: бинарный, сжатый, сохраняет типы данных, быстрее

7. Как проверить типы данных в DataFrame после загрузки?

```
df.dtypes
```

8. Как определить размер DataFrame (количество строк и столбцов)?

df.shape

9. В чем разница между .loc[] и .iloc[]?

.loc[] - выбор по меткам (индексам и названиям столбцов)

.iloc[] - выбор по позициям (целочисленным индексам)

10. Как получить данные третьей строки и второго столбца с .iloc[]?

value = df.iloc[2, 1]

11. Как получить строку с индексом "Мария" из DataFrame?

row = df.loc['Мария']

12. Чем .at[] отличается от .loc[]?

.at[] - доступ к одному элементу, быстрее

.loc[] - может выбирать несколько элементов

13. В каких случаях .iat[] работает быстрее, чем .iloc[]?

.iat[] быстрее при доступе к одному элементу по позиции

14. Как выбрать все строки, где "Город" равен "Москва" или "СПб", используя .isin()?

df[df['Город'].isin(['Москва', 'СПб'])]

15. Как отфильтровать DataFrame, оставив только строки, где "Возраст" от 25 до 35 лет, используя .between()?

df[df['Возраст'].between(25, 35)]

16. В чем разница между .query() и .loc[] для фильтрации данных?

.query() - синтаксис как в SQL, удобен для сложных условий

.loc[] - стандартный pandas-синтаксис

17. Как использовать переменные Python внутри .query()?

min_age = 25

df.query('Возраст > @min_age')

18. Как узнать, сколько пропущенных значений в каждом столбце DataFrame?

df.isna().sum()

19. В чем разница между .isna() и .notna()?

.isna() - True для пропущенных значений

.notna() - True для не пропущенных

20. Как вывести только строки, где нет пропущенных значений?

df.dropna()

21. Как добавить новый столбец "Категория" в DataFrame, заполнив его фиксированным значением "Неизвестно"?

df['Категория'] = 'Неизвестно'

22. Как добавить новую строку в DataFrame, используя .loc[]?

df.loc[новый_индекс] = {'Колонка1': знач1, 'Колонка2': знач2}

23. Как удалить столбец "Возраст" из DataFrame?

df.drop('Возраст', axis=1, inplace=True)

24. Как удалить все строки, содержащие хотя бы один NaN, из DataFrame?

df.dropna(how='any')

25. Как удалить столбцы, содержащие хотя бы один NaN, из DataFrame?

df.dropna(axis=1)

26. Как посчитать количество непустых значений в каждом столбце DataFrame?

df.count()

27. Чем .value_counts() отличается от .nunique()?

.value_counts() - частоты всех значений

.nunique() - количество уникальных значений

28. Как определить сколько раз встречается каждое значение в столбце "Город"?

df['Город'].value_counts()

29. Почему display(df) лучше, чем print(df), в Jupyter Notebook?

display() в Jupyter показывает красивое форматирование с возможностью сортировки

30. Как изменить максимальное количество строк, отображаемых в DataFrame в Jupyter Notebook?

pd.set_option('display.max_rows', 100)

Вывод: В ходе лабораторной работы были изучены основные методы работы с pandas DataFrame, включая создание, фильтрацию, модификацию и анализ данных. Освоены различные способы доступа к данным (loc, iloc, at, iat), а также методы обработки пропущенных значений и работы с разными форматами данных (CSV, Excel, JSON, Parquet). Полученные навыки позволяют эффективно работать с табличными данными в Python для решения практических задач анализа данных.