

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 3

Выполнил:
Борцов Богдан Михайлович
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/REPONCFU/ai-jlab5>

Порядок выполнения работы:

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

```
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

```
# 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

```

import numpy as np

# Создаем массив NumPy с случайными возрастaми 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

```

# 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

```

# 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

```

# 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

```

# 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.

```

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": ["Артём", "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.

```

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

```

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

Листинг программы:

```
[26]: import pandas as pd
from datetime import datetime
import os

DATA_FILE = "data/students.parquet"

def load_data():
    if os.path.exists(DATA_FILE):
        return pd.read_parquet(DATA_FILE)
    else:
        return pd.DataFrame(columns=["ФИО", "Группа", "Математика", "Физика", "Информатика", "Дата ввода"])

def save_data(df):
    os.makedirs("data", exist_ok=True)
    df.to_parquet(DATA_FILE, index=False)

def add_student(fio, group, math, physics, informatics):
    df = load_data()
    new_row = {
        "ФИО": fio,
        "Группа": group,
        "Математика": int(math),
        "Физика": int(physics),
        "Информатика": int(informatics),
        "Дата ввода": datetime.now()
    }
    df = pd.concat([df, pd.DataFrame([new_row])], ignore_index=True)
    df.sort_values("ФИО", inplace=True)
    save_data(df)
    print("Студент добавлен.")

def show_bad_students():
    df = load_data()
    mask = (df[["Математика", "Физика", "Информатика"]] == 2).any(axis=1)
    bad_students = df[mask][["ФИО", "Группа"]]
    if bad_students.empty:
        print("Студентов с оценкой 2 нет.")
    else:
        print("Студенты с хотя бы одной двойкой:")
        print(bad_students.to_string(index=False))

def delete_column(column_name):
    df = load_data()
    if column_name in df.columns:
        df.drop(columns=[column_name], inplace=True)
        save_data(df)
        print(f"Колонка '{column_name}' удалена.")
```

```
df = load_data()
delete_student("Иванов И.И.")
```

✓ Студент 'Иванов И.И.' удалён.

```
show_bad_students()
```

Студентов с оценкой 2 нет.

```
!pytest test_main.py
```

```
===== test session starts =====
platform linux -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0
rootdir: /home/lyash/workspace/test1
plugins: anyio-4.2.0
collected 1 item
```

```
test_main.py . [100%]
```

```
===== warnings summary =====
```

```
test_main.py::test_add_student
/home/ user /workspace/test1/app.py:27: FutureWarning: The behavior of DataFrame concatenation with empty or all-NA entries
is deprecated. In a future version, this will no longer exclude empty or all-NA columns when determining the result dtypes.
To retain the old behavior, exclude the relevant entries before the concat operation.
    df = pd.concat([df, pd.DataFrame([new_row])], ignore_index=True)
```

```
-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
```

```
===== 1 passed, 1 warning in 0.33s =====
```

```

        print(f"Колонка '{column_name}' удалена.")
    else:
        print("Колонка не найдена.")

def delete_student(fio: str):
    global df
    initial_len = len(df)

    df = df[df["ФИО"] != fio]

    if len(df) < initial_len:
        print(f"Студент '{fio}' удалён.")
        save_data(df)
    else:
        print(f"Студент '{fio}' не найден.")

# Пример использования
add_student("Иванов И.И.", "ИКТ-101", 3, 2, 5)
show_bad_students()
delete_column("Дата ввода")

Студент добавлен.
Студенты с хотя бы одной двойкой:
    ФИО    Группа
Иванов И.И. ИКТ-101
Колонка 'Дата ввода' удалена.

```

Использование модульного теста(pytest):

```

!pytest test_main.py

===== test session starts =====
platform linux -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0
rootdir: /home/lyash/workspace/test1
plugins: anyio-4.2.0
collected 1 item

test_main.py . [100%]

===== warnings summary =====
test_main.py::test_add_student
/home/ user /workspace/test1/app.py:27: FutureWarning: The behavior of DataFrame concatenation with empty or all-NA entries
is deprecated. In a future version, this will no longer exclude empty or all-NA columns when determining the result dtypes.
To retain the old behavior, exclude the relevant entries before the concat operation.
    df = pd.concat([df, pd.DataFrame([new_row])], ignore_index=True)

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 1 passed, 1 warning in 0.33s =====

```

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.