

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 9**

Выполнил:  
Кравчук Мирослав Витальевич  
2 курс, группа ИТС-б-о-23-1,  
11.03.02 «Инфокоммуникационные  
технологии и системы связи», 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 работы:** “Введение в pandas: изучение структуры DataFrame и базовых операций”.

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

Ссылка на git репозиторий: <https://github.com/miron2314/DLab-5.git>

### **Порядок выполнения работы:**

#### **1.Создание DataFrame разными способами.**

```
data = {
    "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 = pd.DataFrame(data)
display(df)
```

Рисунок 1. Создание df с помощью dict

```
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. Создание df с помощью dict's list

```
data_full = {
    "ID": list(range(1, 21)),
    "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_full = pd.DataFrame(data_full)
```

Рисунок 3. Создание df с помощью пр массива

## 2. Чтение данных из файлов (CSV , Excel , JSON ).

```
df_full.to_csv("table1.csv", index=False)
df_csv = pd.read_csv("table1.csv")
```

Рисунок 4. Сохранение df в csv таблицу и загрузка из csv

```
data_full_2 = {
    "ID": list(range(1, 21)),
    "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 = pd.DataFrame(data_full_2)
file_name = 'Клиенты.xlsx'
df.to_excel(file_name, index=False, sheet_name='Клиенты')
```

Рисунок 5. Сохранение таблицы 1 df в csv таблицу и загрузка из csv

```
json_file = 'table_1.json'
df_full.to_json(json_file, orient='records', force_ascii=False)
df_from_json = pd.read_json(json_file)
df_from_json.head()
```

Рисунок 6. Экспорт таблицы 1 в json файл

## 3. Доступ к данным ( .loc , .iloc , .at , .iat ).

```
df_1 = pd.DataFrame(data_full)
print(df_1.loc[df_1["ID"] == 5])
```

Рисунок 7. Получение информации о сотруднике с ID =5

```
print(df_1.loc[2]["Возраст"])
```

Рисунок 8. Возраст третьего сотрудника в таблице

```
print(df_1.at[3, "Отдел"])
```

Рисунок 9. Название отдела для сотрудника "Мария"

```
print(df_1.iat[3,5])
```

Рисунок 10. Зарплата сотрудника, находящегося в четвертой строке и пятом столбце

#### 4.Добавление новых столбцов и строк.

```
conditions = [  
    (df_1['Зарплата'] < 60000),  
    (df_1['Зарплата'] >= 60000) & (df_1['Зарплата'] < 100000),  
    (df_1['Зарплата'] >= 100000)  
]  
categories = ["Низкая", "Средняя", "Высокая"]  
df_1["Категория зарплаты"] = np.select(conditions, categories)  
display(df_1)
```

Рисунок 11. Добавление столбца в таблицу с категориями зарплат

```
df_1.loc[20] = [21, "Антон", 32, "Разработчик", "IT", 85000, 6, "Средняя"]  
display(df_1.tail())
```

Рисунок 12. Добавление сотрудника в таблицу

```
new_data = {  
    "ID": [22, 23],  
    "Имя": ["Антон", "Марина"],  
    "Возраст": [32, 27],  
    "Должность": ["Разработчик", "Менеджер"],  
    "Отдел": ["IT", "Продажи"],  
    "Зарплата": [85000, 75000],  
    "Стаж работы": [6, 5],  
    "Категория зарплаты": ["Средняя", "Средняя"]  
}  
new_df = pd.DataFrame(new_data)  
df_1 = pd.concat([df_1, new_df], ignore_index=True)  
display(df_1.tail())
```

Рисунок 13. Добавление нескольких сотрудника в таблицу

## 5. Удаление строк и столбцов.

```
df_1=df_1.drop(columns=['Категория зарплаты'])  
display(df_1.head())
```

Рисунок 14. Удаление столбца "Категория зарплаты"

```
df_1_1 = df_1  
df_1_1 = df_1_1[df_1_1['ID'] != 10]  
display(df_1_1)
```

Рисунок 15. Удаление строки с ID =10

```
df_1_1 = df_1_1[df_1_1['Стаж работы'] >= 3]  
display(df_1_1)
```

Рисунок 16. Удаление строк, где стаж работы <3

```
df_1_1 = df_1_1[['Имя', 'Должность', 'Зарплата']]  
display(df_1_1)
```

Рисунок 17. Удаление всех столбцов кроме Имя, Должность, Зарплата

## 5. Фильтрация данных (query, isin, between).

```
df_2=pd.DataFrame(data_full_2)  
filters = ["Москва", "Санкт-Петербург"]  
filtered_df = df[df['Город'].isin(filters)]  
display(filtered_df)
```

Рисунок 18. Выбор всех клиентов из "Москва" или "Санкт-Петербург"

```
filtered_df=df_2[df_2['Баланс на счете'].between(100000, 250000)]  
display(filtered_df)
```

Рисунок 19. Выбор клиентов, у которых Баланс на счете от 100000 до 250000

```
filtered_df_2 = df.query('`Кредитная история` == "Хорошая" and `Баланс на счете` > 150000')
display(filtered_df_2)
```

Рисунок 20. Фильтр клиентов, у которых "Кредитная история" "Хорошая" и "Баланс на счете»> 150000

6.Подсчет значений (count, value\_counts, nunique).

```
count_per_column = df_2.count()
print(count_per_column)
```

Рисунок 21. Подсчет количества непустых значений в столбцах

```
count_cities = df["Город"].value_counts()
print(count_cities)
```

Рисунок 22. Подсчет частоты встречаемых городов

```
names = ["Город" , "Возраст" , "Баланс на счете"]
unique = df[names].nunique()
print(unique)
```

Рисунок 23. Количество уникальных значений в "Город" , "Возраст" , "Баланс на счете"

6.Обнаружение пропусков (isna, notna ).

```
data_3 = {
    "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", пр.nan, "Дмитрий", "Елена", "Виктор", "Алиса",
           "Павел", пр.nan, "Роман", "Татьяна", "Николай", "Валерия", "Григорий", "Юлия", "Степан", "Василиса"],
    "Возраст": [34, 27, пр.nan, 38, 29, 50, 31, 40, пр.nan, 33, 46, 37, 41, 25, пр.nan, 42, 49, пр.nan, 30, 35],
    "Город": ["Москва", "Санкт-Петербург", "Казань", пр.nan, "Екатеринбург", "Воронеж", пр.nan, "Краснодар",
             "Ростов-на-Дону", "Уфа", "Омск", "Пермь", пр.nan, "Саратов", "Самара", пр.nan, "Барнаул",
             "Иркутск", "Хабаровск", "Томск"],
    "Баланс на счете": [120000, пр.nan, 150000, 200000, пр.nan, 300000, 140000, 175000, 110000, пр.nan,
                       250000, 210000, 135000, 155000, 125000, пр.nan, 275000, 320000, 105000, 90000],
    "Кредитная история": ["Хорошая", "Средняя", "Плохая", "Хорошая", пр.nan, "Отличная", "Средняя",
                          "Хорошая", пр.nan, "Средняя", "Хорошая", "Отличная", "Средняя", пр.nan,
                          "Средняя", "Плохая", "Отличная", "Хорошая", "Средняя", "Плохая"]
}

df_3 = pd.DataFrame(data)
```

Рисунок 24. df таблицы 3

```
nan_counts = df.isna().sum()
print("Количество NaN в каждом столбце:\n", nan_counts)
```

Рисунок 25. Количество NaN в каждом столбце

```
filled_counts = df.notna().sum()
print("\nКоличество заполненных значений в каждом столбце:\n", filled_counts)
```

Рисунок 26. количество заполненных значений в каждом столбце

```
df_cleaned = df.dropna()
print("\nDataFrame без пропущенных значений:")
display(df_cleaned)
```

Рисунок 27. Строки, где нет пропущенных значений.

7.Выполнил индивидуальное задание.

### **Индивидуальное задание. Вариант 9**

Написать программу на языке программирования Python для решения поставленной задачи (в репозитории должны присутствовать настройки требуемых пакетов для выбранного менеджера пакетов). Приложение должно использовать интерфейс командной строки (модуль `argparse` ) или графический интерфейс пользователя (модули `tkinter` , `PySide2` , `Kivy` и т. д.). При работе с датой и временем использовать пакет `datetime` . Организовать чтение и сохранение данных из/в формат `Parquet`. Выполнить валидацию сохраненных данных с помощью сторонних приложений для работы с форматом `Parquet`, например <https://github.com/mukunku/ParquetViewer>. Организовать также удаление данных по одной из колонок `DataFrame` на выбор обучающихся. Номер варианта определяется по согласованию с преподавателем.Использовать `DataFrame` , содержащий следующие колонки: название начального пункта маршрута; название конечного пункта маршрута; номер маршрута. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных и добавление строк в `DataFrame` ; записи должны быть упорядочены по номерам маршрутов; вывод на экран информации о маршруте, номер которого введен с

клавиатуры; если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

### Листинг программы:

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

class RouteManager:
    def __init__(self):
        self.df = pd.DataFrame(columns=['start_point', 'end_point', 'route_number', 'created_at'])

    def add_route(self, start: str, end: str, number: int):
        """Добавление нового маршрута с автоматической сортировкой"""
        new_route = pd.DataFrame({
            'start_point': [start],
            'end_point': [end],
            'route_number': [number],
            'created_at': [datetime.now()]
        })
        self.df = pd.concat([self.df, new_route], ignore_index=True)
        self.df = self.df.sort_values(by='route_number').reset_index(drop=True)
        return f"✅ Маршрут №{number} добавлен: {start} → {end}"

    def find_route(self, number: int):
        """Поиск маршрута по номеру"""
        result = self.df[self.df['route_number'] == number]
        return result

    def delete_by_column(self, column: str, value: str):
        """Удаление записей по значению в колонке"""
        if column in self.df.columns:
            before = len(self.df)
            self.df = self.df[self.df[column] != value]
            return f"🗑 Удалено {before - len(self.df)} записей где {column} = {value}"
        return "❌ Ошибка: неверное название колонки"

    def save_to_parquet(self, filename: str):
        """Сохранение данных в Parquet-формат"""
        self.df.to_parquet(filename, engine='pyarrow')
        return f"💾 Данные сохранены в {filename}"

    def load_from_parquet(self, filename: str):
        """Загрузка данных из Parquet-файла"""
        if os.path.exists(filename):
            self.df = pd.read_parquet(filename, engine='pyarrow')
            self.df = self.df.sort_values(by='route_number').reset_index(drop=True)
            return f"📂 Данные загружены из {filename}\nВсего маршрутов: {len(self.df)}"
        return f"❌ Файл {filename} не найден"

    def show_all_routes(self):
        """Показать все маршруты"""
        if self.df.empty:
            return "🕒 Нет данных о маршрутах"
        return self.df[['start_point', 'end_point', 'route_number']]
```



```

def create_ui():
    """Создание пользовательского интерфейса"""
    manager = RouteManager()
    output = widgets.Output()

    # Виджеты для добавления маршрута
    start_input = widgets.Text(placeholder='Москва', description='Откуда:')
    end_input = widgets.Text(placeholder='Санкт-Петербург', description='Куда:')
    number_input = widgets.IntText(description='Номер:', value=1)
    add_button = widgets.Button(description='Добавить', button_style='success')

    # Виджеты для поиска маршрута
    search_input = widgets.IntText(description='Поиск:', value=1)
    search_button = widgets.Button(description='Найти', button_style='info')

    # Виджеты для удаления маршрутов
    column_dropdown = widgets.Dropdown(
        options=['Начальный пункт', 'Конечный пункт', 'Номер маршрута'],
        value='Номер маршрута',
        description='Удалить по:'
    )
    value_input = widgets.Text(description='Значение:', value='')
    delete_button = widgets.Button(description='Удалить', button_style='danger')

    # Виджеты для работы с файлами
    filename_input = widgets.Text(value='routes.parquet', description='Файл:')
    save_button = widgets.Button(description='Сохранить', button_style='primary')
    load_button = widgets.Button(description='Загрузить', button_style='primary')

    # Обработчики событий
    def on_add_button_clicked(b):
        with output:
            clear_output()
            try:
                result = manager.add_route(
                    start_input.value,
                    end_input.value,
                    number_input.value
                )
                print(result)
                display(manager.show_all_routes())
            except Exception as e:
                print(f"⊖ Ошибка: {str(e)}")

    def on_search_button_clicked(b):
        with output:
            clear_output()
            try:
                routes = manager.find_route(search_input.value)
                if not routes.empty:
                    print(f"🔍 Найдены маршруты №{search_input.value}:")
                    display(routes[['start_point', 'end_point', 'route_number']])
                else:
                    print(f"⊖ Маршрутов с номером {search_input.value} не найдено")
            except Exception as e:
                print(f"⊖ Ошибка: {str(e)}")

    def on_delete_button_clicked(b):
        with output:

```

```

clear_output()
try:
    # Преобразование русских названий в английские
    column_map = {
        'Начальный пункт': 'start_point',
        'Конечный пункт': 'end_point',
        'Номер маршрута': 'route_number'
    }
    column = column_map[column_dropdown.value]

    # Преобразование значения для номера маршрута
    if column == 'route_number':
        value = int(value_input.value)
    else:
        value = value_input.value

    result = manager.delete_by_column(column, value)
    print(result)
    display(manager.show_all_routes())
except Exception as e:
    print(f"⊖ Ошибка: {str(e)}")

def on_save_button_clicked(b):
    with output:
        clear_output()
        try:
            print(manager.save_to_parquet(filename_input.value))
        except Exception as e:
            print(f"⊖ Ошибка: {str(e)}")

def on_load_button_clicked(b):
    with output:
        clear_output()
        try:
            result = manager.load_from_parquet(filename_input.value)
            print(result)
            display(manager.show_all_routes())
        except Exception as e:
            print(f"⊖ Ошибка: {str(e)}")

# Привязка обработчиков
add_button.on_click(on_add_button_clicked)
search_button.on_click(on_search_button_clicked)
delete_button.on_click(on_delete_button_clicked)
save_button.on_click(on_save_button_clicked)
load_button.on_click(on_load_button_clicked)

# Группировка виджетов
add_box = widgets.VBox([
    widgets.HTML("<h3>Добавить маршрут</h3>"),
    widgets.HBox([start_input, end_input, number_input, add_button])
])

search_box = widgets.VBox([
    widgets.HTML("<h3>Поиск маршрута</h3>"),
    widgets.HBox([search_input, search_button])
])

delete_box = widgets.VBox([
    widgets.HTML("<h3>Удалить маршруты</h3>"),
    widgets.HBox([column_dropdown, value_input, delete_button])
])

```

```

file_box = widgets.VBox([
    widgets.HTML("<h3>Работа с файлами</h3>"),
    widgets.HBox([filename_input, save_button, load_button])
])
# Отображение интерфейса
display(widgets.VBox([
    widgets.HTML("<h2 style='color: #1a73e8;'>🚌 Управление маршрутами</h2>"),
    add_box,
    search_box,
    delete_box,
    file_box,
    widgets.HTML("<h3>Результат:</h3>"),
    output
]))
# Запуск интерфейса
create_ui()

```

**Управление маршрутами**

**Добавить маршрут**

Откуда:  Куда:  Номер:

**Поиск маршрута**

Поиск:

**Удалить маршруты**

Удалить по:  Значение:

**Работа с файлами**

Файл:

**Результат:**

🔍 Найденные маршруты №1:

start_point	end_point	route_number
0		1

Рисунок 28. Интерфейс для программы

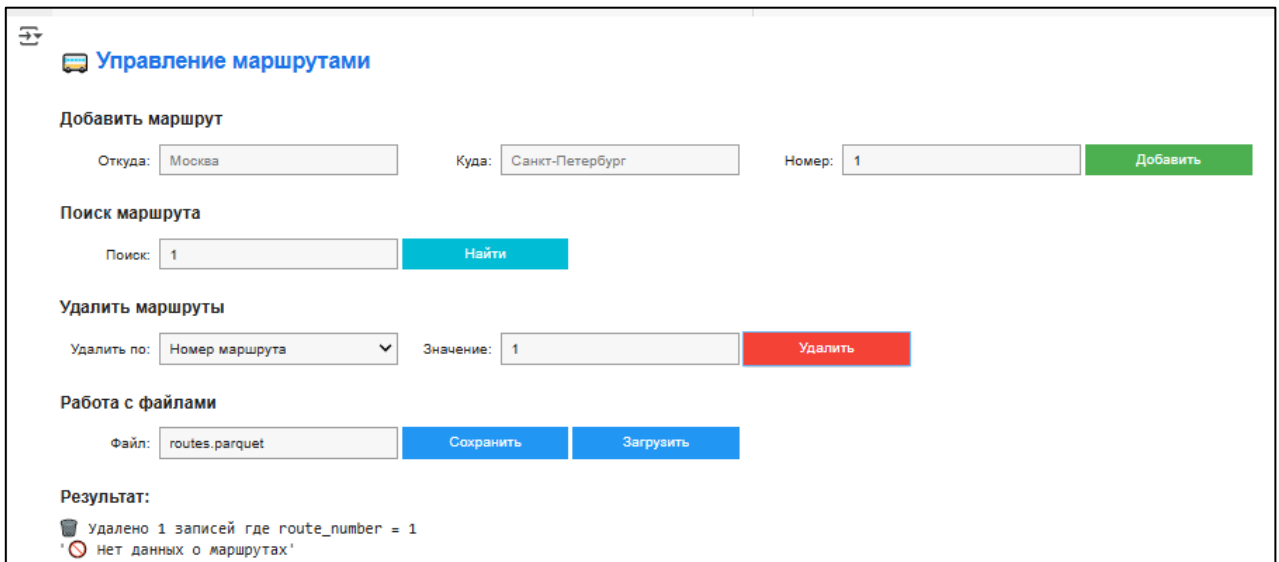


Рисунок 29. Результат работы программы

## 8. Сохранил изменения в репозитории

```
PS C:\Users\USER\DLab-5> git commit -m "commit"
[main fc4d468] commit
PS C:\Users\USER\DLab-5> git pull
Already up to date.
PS C:\Users\USER\DLab-5> git push
Enumerating objects: 11, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 12 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 11.61 KiB | 5.80 MiB/s, done.
Total 7 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/miron2314/DLab-5.git
 52eb3f2..fc4d468  main -> main
PS C:\Users\USER\DLab-5>
```

Рисунок 30. Добавление изменений в репозиторий

## Ответы на контрольные вопросы:

### 1. Как создать pandas.DataFrame из словаря списков?

Чтобы создать DataFrame из словаря списков, используйте конструкцию `pd.DataFrame()`:

```
import pandas as pd
data = {'Имя': ['Анна', 'Иван', 'Мария'], 'Возраст': [23, 35, 29]}
df = pd.DataFrame(data)
```

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

**Список словарей:** каждый словарь представляет строку данных.

Столбцы будут выведены на основе ключей словаря.

```
data = [{'Имя': 'Анна', 'Возраст': 23}, {'Имя': 'Иван', 'Возраст': 35}]
df = pd.DataFrame(data)
```

**Словарь списков:** ключи словаря — это названия столбцов, а значения — это списки данных, где каждый элемент в списке соответствует строке.

```
data = {'Имя': ['Анна', 'Иван'], 'Возраст': [23, 35]}
df = pd.DataFrame(data)
```

### 3. Как создать `pandas.DataFrame` из массива `NumPy`?

Для создания `DataFrame` из массива `NumPy`, передайте его в `pd.DataFrame()`:

```
import numpy as np
import pandas as pd
data = np.array([[1, 2], [3, 4], [5, 6]])
df = pd.DataFrame(data, columns=['A', 'B'])
```

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

Для загрузки данных из CSV с разделителем ; используйте параметр `sep`:

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

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

Для загрузки данных из Excel и указания конкретного листа:

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

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

**JSON:** используется для работы с текстовыми данными в формате JSON.

```
df = pd.read_json('data.json')
```

**Parquet:** это бинарный формат, который более эффективен по скорости и использует схемы данных. Он поддерживает типизацию и сжимаемые данные.

```
df = pd.read_parquet('data.parquet')
```

### 7. Как проверить типы данных в DataFrame после загрузки?

Для проверки типов данных столбцов используйте атрибут `.dtypes`:

```
df.dtypes
```

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

Для получения размера используйте атрибут `.shape`:

```
df.shape # (строки, столбцы)
```

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

`.loc[]` используется для выбора по меткам (индексам).

`.iloc[]` используется для выбора по позициям (индексам по порядку).

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

Используйте `.iloc[]` с позициями:

```
df.iloc[2, 1]
```

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

Используйте `.loc[]`:

```
df.loc[df['Имя'] == 'Мария']
```

### 12. Чем `.at[]` отличается от `.loc[]`?

`.at[]` используется для получения одного значения по меткам, быстрее, чем `.loc[]`, но работает только для одного значения (не для целых строк или столбцов).

`.loc[]` позволяет выбрать более сложные срезы данных.

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

`.iat[]` работает быстрее, чем `.iloc[]`, если нужно получить одно значение по позиции, так как `.iat[]` более специализирован для этого.

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

Для выбора строк с `.isin()`:

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

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

Для фильтрации с `.between()`:

```
df[df['Возраст'].between(25, 35)]
```

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

`.query()` позволяет использовать строковые выражения для фильтрации, что делает код более читаемым и удобным.

`.loc[]` требует использования явных условий, часто более гибкое, но менее удобное для простых фильтров.

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

Для использования переменных в `.query()` передайте их через параметр `local_dict`:

```
возраст_min = 25
возраст_max = 35
df.query('Возраст >= @возраст_min and Возраст <= @возраст_max')
```

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

Используйте `.isna()` и `.sum()`:

```
df.isna().sum()
```

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

`.isna()` возвращает True для пропущенных значений (NaN).

`.notna()` возвращает True для непустых значений.

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

Для вывода строк без пропущенных значений:

```
df.dropna()
```

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

Добавьте новый столбец:

```
df['Категория'] = 'Неизвестно'
```

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

Используйте .loc[] для добавления строки по новому индексу:

```
df.loc[len(df)] = ['Новый человек', 28]
```

## 23. Как удалить столбец "Возраст" из DataFrame?

Для удаления столбца используйте drop():

```
df = df.drop(columns=['Возраст'])
```

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

Для удаления строк с NaN:

```
df = df.dropna()
```

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

Для удаления столбцов с NaN:

```
df = df.dropna(axis=1)
```



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

Для подсчета непустых значений используйте `.count()`:

```
df.count()
```

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

`.value_counts()` возвращает количество уникальных значений в столбце.

`.nunique()` возвращает количество уникальных значений, но не учитывает их частоту.

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

Используйте `.value_counts()`:

```
df['Город'].value_counts()
```

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

`display(df)` в Jupyter отображает DataFrame в виде таблицы с форматированием, что делает данные более читаемыми, в отличие от `print(df)`, который просто выводит их в виде текста.

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

Чтобы изменить максимальное количество строк, отображаемых в DataFrame в Jupyter Notebook, можно использовать параметр `pd.set_option()`.

**Вывод по заданию:** Выполнение задания позволило изучить основные функциональные возможности библиотеки pandas при работе с DataFrame.

**Вывод:** в ходе работы были получены навыки работы с библиотекой pandas, в частности, со структурой данных DataFrame

