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 6

Выполнил:
Якушенко Антон Андреевич
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:
Ассистент департамента цифровых, робототехнических систем и электроники Хацукова А.И

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

ТЕМА: ВВЕДЕНИЕ В PANDAS: ИЗУЧЕНИЕ СТРУКТУРЫ DATAFRAME И БАЗОВЫХ ОПЕРАЦИЙ

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

Ссылка на репозиторий: https://github.com/Yakush766/LB5AI.git

Порядок выполнения работы:

1. Перед выполнением заданий установил окружение conda с версией python 3.12.7

```
(base) C:\Users\anton\Documents\Икусственный интелект и машинное обучение\LB5AI>conda create --name myenv python=3.12.7
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\anton\.conda\envs\myenv

  added / updated specs:
    - python=3.12.7
```

Рисунок 1. Установка окружения conda

2. Далее активировал окружение и добавил его как ядро в jupyter lab

```
(base) C:\Users\anton\Documents\Икусственный интелект и машинное обучение\LB5AI>conda activate myenv

(myenv) C:\Users\anton\Documents\Икусственный интелект и машинное обучение\LB5AI>
```

Рисунок 2. Активация окружения

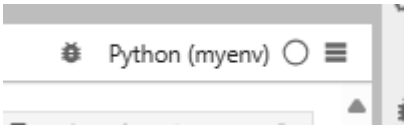

Рисунок 3. Добавление окружения в качестве ядра

Задание 1. Создание DataFrame разными способами

```python
import pandas as pd
import numpy as np

# Данные из таблицы для создания словаря списков
data_dict = {
    '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]
}

# 1. Создание DataFrame из словаря списков
df_from_dict = pd.DataFrame(data_dict)
print("DataFrame из словаря списков:\n", df_from_dict)

# 2. Создание DataFrame из списка словарей
list_of_dicts = [dict(zip(data_dict.keys(), values)) for values in zip(*data_dict.values())]
df_from_list_of_dicts = pd.DataFrame(list_of_dicts)
print("\nDataFrame из списка словарей:\n", df_from_list_of_dicts)

# 3. Создание DataFrame из массива NumPy со случайными числами (возраст сотрудников)
np.random.seed(0)  # для воспроизводимости
random_ages = np.random.randint(20, 61, size=20)  # 20 случайных чисел от 20 до 60
df_numpy = pd.DataFrame({
    'ID': data_dict['ID'],
    'Возраст_случайный': random_ages
})
print("\nDataFrame из массива NumPy:\n", df_numpy)

# 4. Проверка типов данных в каждом столбце
print("\nТипы данных в DataFrame из словаря списков:\n")
print(df_from_dict.info())
```

Рисунок 4. Код для выполнения задания

DataFrame из словаря списков:

	ID	Имя	Возраст	Должность	Отдел	Зарплата	Стаж работы
0	1	Иван	25	Инженер	IT	60000	2
1	2	Ольга	30	Аналитик	Маркетинг	75000	5
2	3	Алексей	40	Менеджер	Продажи	90000	15
3	4	Мария	35	Программист	IT	80000	7
4	5	Сергей	28	Специалист	HR	50000	3
5	6	Анна	32	Разработчик	IT	85000	6
6	7	Дмитрий	45	HR	HR	48000	12
7	8	Елена	29	Маркетолог	Маркетинг	70000	4
8	9	Виктор	31	Юрист	Юридический	95000	10
9	10	Алиса	27	Дизайнер	Дизайн	62000	5
10	11	Павел	33	Администратор	Администрация	55000	7
11	12	Светлана	26	Тестировщик	Тестирование	67000	2
12	13	Роман	42	Финансист	Финансы	105000	20
13	14	Татьяна	37	Редактор	Редакция	72000	9
14	15	Николай	39	Логист	Логистика	75000	11
15	16	Валерия	24	SEO-специалист	SEO	64000	3
16	17	Григорий	50	Бухгалтер	Бухгалтерия	110000	25
17	18	Юлия	45	Директор	Финансы	150000	20
18	19	Степан	41	Экономист	Экономика	98000	14
19	20	Василиса	38	Проект-менеджер	Продажи	88000	8

Рисунок 5. Выполненное задание

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

```python
import pandas as pd
import json

# Создаём DataFrame с вашими данными

data_employees = {
    "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]
}

data_clients = {
    "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",
                          "Средняя", "Отличная", "Средняя", "Хорошая", "Средняя", "Плохая", "Отличная", "Хорошая", "Средняя", "Плохая"]
}
```

```python
# Создаём DataFrame
df_employees = pd.DataFrame(data_employees)
df_clients = pd.DataFrame(data_clients)

# 1. Сохранение таблицы в CSV и чтение обратно
csv_filename = 'Таблица1.csv'
df_employees.to_csv(csv_filename, index=False)

# Чтение из CSV
df_csv = pd.read_csv(csv_filename)
print("Первые 5 строк из CSV файла:")
print(df_csv.head())

# 2. Запись таблицы в Excel и чтение
excel_filename = 'Таблица2.xlsx'
df_employees.to_excel(excel_filename, index=False, sheet_name='Employees')

# Чтение из Excel
df_excel = pd.read_excel(excel_filename, sheet_name='Employees')
print("\nПервые 5 строк из Excel файла:")
print(df_excel.head())

# 3. Экспорт в JSON и чтение
json_filename = 'Таблица1.json'
df_employees.to_json(json_filename, orient='records')

# Чтение из JSON
with open(json_filename, 'r', encoding='utf-8') as f:
    data_json = json.load(f)

print("\nПервые 5 элементов из JSON файла:")
for item in data_json[:5]:
    print(item)
```

Рисунок 6. Код для выполнения задания

```
Первые 5 строк из CSV файла:
   ID      Имя  Возраст    Должность      Отдел  Зарплата  Стаж работы
0   1     Иван       25      Инженер         IT     60000            2
1   2    Ольга       30    Аналитик  Маркетинг     75000            5
2   3  Алексей       40    Менеджер    Продажи     90000           15
3   4    Мария       35  Программист         IT     80000            7
4   5   Сергей       28   Специалист         HR     50000            3

Первые 5 строк из Excel файла:
   ID      Имя  Возраст    Должность      Отдел  Зарплата  Стаж работы
0   1     Иван       25      Инженер         IT     60000            2
1   2    Ольга       30    Аналитик  Маркетинг     75000            5
2   3  Алексей       40    Менеджер    Продажи     90000           15
3   4    Мария       35  Программист         IT     80000            7
4   5   Сергей       28   Специалист         HR     50000            3

Первые 5 элементов из JSON файла:
{'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}
```

Рисунок 7. Выполненное задание

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

```python
import pandas as pd

# Создаём таблицу
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)

# Устанавливаем ID как индекс
df.set_index('ID', inplace=True)

# 1. Получить информацию о сотруднике с ID=5
employee_id_5 = df.loc[5]
print("Информация о сотруднике с ID=5:\n", employee_id_5)

# 2. Вывести возраст третьего сотрудника (позиция 2)
age_third_employee = df.iloc[2]['Возраст']
print("Возраст третьего сотрудника:", age_third_employee)

# 3. Название отдела для сотрудника "Мария"
row_maria = df[df['Имя'] == 'Мария']
department_maria = df.at[row_maria.index[0], 'Отдел']
print("Отдел Марии:", department_maria)

# 4. Зарплата в 4-й строке и 5-м столбце (нумерация с 0)
salary = df.iat[3, 4]
print("Зарплата в 4-й строке, 5-м столбце:", salary)

# 5. Объяснение разницы между методами:
print("""
.loc[] - по меткам (имена индексов и колонок)
.iloc[] - по числовым позициям
.at[] - быстрый доступ к одному элементу по метке
.iat[] - быстрый доступ к одному элементу по позиции
""")
```

Рисунок 8. Код для выполнения задания

```
Информация о сотруднике с ID=5:
 Имя                Сергей
Возраст                28
Должность      Специалист
Отдел                  HR
Зарплата            50000
Стаж работы             3
Name: 5, dtype: object
Возраст третьего сотрудника: 40
Отдел Марии: IT
Зарплата в 4-й строке, 5-м столбце: 80000

.loc[] - по меткам (имена индексов и колонок)
.iloc[] - по числовым позициям
.at[] - быстрый доступ к одному элементу по метке
.iat[] - быстрый доступ к одному элементу по позиции
```

Рисунок 9. Выполненное задание

Задание 4. Добавление новых столбцов и строк

```python
import pandas as pd

# Данные из "Таблицы 1" (используем тот же DataFrame, что и раньше, или инициализируем заново)
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
    'Отдел': ['IT', 'Маркетинг', 'Продажи', 'IT', 'HR', 'IT', 'HR', 'Маркетинг', 'Юридический', 'Дизайн', 'Администрация', 'Тестирование', 'Финансы', 'Редакция', 'Логистика', 'SEO', 'Бухгалтерия', 'Финансы', 'Экономика',
    '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. Добавление столбца "Категория зарплаты"
def categorize_salary(salary):
    if salary < 60000:
        return "Низкая"
    elif 60000 <= salary < 100000:
        return "Средняя"
    else:
        return "Высокая"

df['Категория зарплаты'] = df['Зарплата'].apply(categorize_salary)

# 2. Добавление нового сотрудника
# Нужно создать новый DataFrame, а затем объединить его с существующим
new_employee_data = {'ID': [21], 'Имя': ['Антон'], 'Возраст': [32], 'Должность': ['Разработчик'], 'Отдел': ['IT'], 'Зарплата': [85000], 'Стаж работы': [6], 'Категория зарплаты': categorize_salary(85000)}
new_employee_df = pd.DataFrame(new_employee_data)

df = pd.concat([df, new_employee_df], ignore_index=True)  # ignore_index=True для сброса индексов

# 3. Добавление двух новых сотрудников с использованием pd.concat()
new_employees_data = {
    'ID': [22, 23],
    'Имя': ['Елена', 'Игорь'],
    'Возраст': [28, 35],
    'Должность': ['Аналитик', 'Менеджер'],
    'Отдел': ['Маркетинг', 'Продажи'],
    'Зарплата': [70000, 120000],
    'Стаж работы': [3, 8],
    'Категория зарплаты': [categorize_salary(70000), categorize_salary(120000)]
}
new_employees_df = pd.DataFrame(new_employees_data)

df = pd.concat([df, new_employees_df], ignore_index=True)

# 4. Вывод обновленного DataFrame
print(df)
```

Рисунок 10. Код для выполнения задания

```
    ID        Имя  Возраст        Должность          Отдел  Зарплата
0    1       Иван       25          Инженер             IT     60000
1    2      Ольга       30        Аналитик      Маркетинг     75000
2    3   Алексей        40        Менеджер        Продажи     90000
3    4     Мария        35     Программист             IT     80000
4    5    Сергей        28      Специалист             HR     50000
5    6      Анна        32     Разработчик             IT     85000
6    7   Дмитрий        45              HR             HR     48000
7    8     Елена        29      Маркетолог      Маркетинг     70000
8    9    Виктор        31          Юрист     Юридический     95000
9   10     Алиса        27       Дизайнер         Дизайн     62000
10  11     Павел        33  Администратор  Администрация     55000
11  12   Светлана       26    Тестировщик    Тестирование     67000
12  13     Роман        42      Финансист         Финансы    105000
13  14    Татьяна       37       Редактор        Редакция     72000
14  15   Николай        39         Логист       Логистика     75000
15  16   Валерия        24  SEO-специалист             SEO     64000
16  17  Григорий        50       Бухгалтер      Бухгалтерия    110000
17  18     Юлия         45       Директор         Финансы    150000
18  19    Степан        41      Экономист       Экономика     98000
19  20  Василиса        38  Проект-менеджер        Продажи     88000
20  21     Антон        32     Разработчик             IT     85000
21  22     Елена        28        Аналитик      Маркетинг     70000
22  23     Игорь        35        Менеджер        Продажи    120000

    Стаж работы Категория зарплаты
0             2              Средняя
1             5              Средняя
2            15              Средняя
3             7              Средняя
4             3               Низкая
5             6              Средняя
6            12               Низкая
7             4              Средняя
8            10              Средняя
9             5              Средняя
10            7               Низкая
11            2              Средняя
12           20              Высокая
13            9              Средняя
14           11              Средняя
15            3              Средняя
16           25              Высокая
17           20              Высокая
18           14              Средняя
19            8              Средняя
20            6              Средняя
21            3              Средняя
22            8              Высокая
```

Рисунок 11. Выполненное задание

Задание 5. Удаление строк и столбцов

```python
import pandas as pd

# Создаем DataFrame с данными
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', 'Экономика', '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. Удалите столбец "Категория зарплаты" с '.drop()'
# В исходных данных такого столбца нет, предположим, что он есть, добавим его для примера
# Или, если его нет, пропустим этот шаг
# Для примера добавим его:
df['Категория зарплаты'] = ['Высокая', 'Высокая', 'Высокая', 'Высокая', 'Низкая', 'Высокая', 'Низкая', 'Средняя', 'Высокая', 'Средняя',
                            'Низкая', 'Средняя', 'Высокая', 'Средняя', 'Средняя', 'Средняя', 'Высокая', 'Высокая', 'Средняя', 'Средняя']

df = df.drop(columns=['Категория зарплаты'])

# 2. Удалите строку с ID = 10
df = df[df['ID'] != 10]

# 3. Удалите все строки, где стаж работы < 3 лет
df = df[df['Стаж работы'] >= 3]

# 4. Удалите все столбцы, кроме 'Имя', 'Должность', 'Зарплата'
df = df[['Имя', 'Должность', 'Зарплата']]

# Выводим итоговый DataFrame
print(df)
```

Рисунок 12. Код для выполнения задания

	Имя	Должность	Зарплата
1	Ольга	Аналитик	75000
2	Алексей	Менеджер	90000
3	Мария	Программист	80000
4	Сергей	Специалист	50000
5	Анна	Разработчик	85000
6	Дмитрий	HR	48000
7	Елена	Маркетолог	70000
8	Виктор	Юрист	95000
10	Павел	Администратор	55000
12	Роман	Финансист	105000
13	Татьяна	Редактор	72000
14	Николай	Логист	75000
15	Валерия	SEO-специалист	64000
16	Григорий	Бухгалтер	110000
17	Юлия	Директор	150000
18	Степан	Экономист	98000
19	Василиса	Проект-менеджер	88000

Рисунок 13. Выполненное задание

Задание 6. Фильтрация данных (query , isin , between)

```python
import pandas as pd

# Создаем DataFrame с данными из таблицы
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, 9800,
                        250000, 210000, 135000, 155000, 125000, 180000, 275000, 320000, 105000, 90000],
    'Кредитная история': ['Хорошая', 'Средняя', 'Плохая', 'Хорошая', 'Средняя', 'Отличная', 'Средняя', 'Хорошая', 'Плохая', 'Средняя',
                          'Средняя', 'Отличная', 'Средняя', 'Хорошая', 'Средняя', 'Плохая', 'Отличная', 'Хорошая', 'Средняя', 'Плохая']
}

df = pd.DataFrame(data)

# 1. Выбираем всех клиентов из "Москва" или "Санкт-Петербург"
clients_moscow_spb = df[df['Город'].isin(['Москва', 'Санкт-Петербург'])]

# 2. Выбираем клиентов, у которых баланс на счете от 100000 до 250000
clients_balance = df[df['Баланс на счете'].between(100000, 250000)]

# 3. Отфильтровать клиентов, у которых "Кредитная история" = "Хорошая" и "Баланс на счете" > 150000
clients_credit_balance = df[
    (df['Кредитная история'] == 'Хорошая') & (df['Баланс на счете'] > 150000)
]

# Вывод результатов
print("Клиенты из Москвы или Санкт-Петербурга:")
print(clients_moscow_spb)

print("\nКлиенты с балансом от 100000 до 250000:")
print(clients_balance)

print("\nКлиенты с хорошей кредитной историей и балансом > 150000:")
print(clients_credit_balance)
```

Рисунок 14. Код для выполнения задания

```
Клиенты из Москвы или Санкт-Петербурга:
   ID    Имя  Возраст            Город  Баланс на счете Кредитная история
0   1   Иван       34           Москва           120000           Хорошая
1   2  Ольга       27  Санкт-Петербург            80000           Средняя

Клиенты с балансом от 100000 до 250000:
    ID       Имя  Возраст            Город  Баланс на счете Кредитная история
0    1      Иван       34           Москва           120000           Хорошая
2    3   Алексей       45           Казань           150000            Плохая
3    4     Мария       38      Новосибирск           200000           Хорошая
6    7   Дмитрий       31       Челябинск           140000           Средняя
7    8     Елена       40       Краснодар           175000           Хорошая
8    9    Виктор       28   Ростов-на-Дону           110000            Плохая
10  11     Павел       46            Омск           250000           Средняя
11  12  Светлана       37           Пермь           210000          Отличная
12  13     Роман       41          Тюмень           135000           Средняя
13  14   Татьяна       25         Саратов           155000           Хорошая
14  15   Николай       39          Самара           125000           Средняя
15  16   Валерия       42       Волгоград           180000            Плохая
18  19    Степан       30       Хабаровск           105000           Средняя

Клиенты с хорошей кредитной историей и балансом > 150000:
    ID      Имя  Возраст        Город  Баланс на счете Кредитная история
3    4    Мария       38  Новосибирск           200000           Хорошая
7    8    Елена       40    Краснодар           175000           Хорошая
13  14  Татьяна       25     Саратов           155000           Хорошая
17  18     Юлия       50     Иркутск           320000           Хорошая
```

Рисунок 15. Выполненное задание

Задание 7. Подсчет значений (count , value_counts , nunique)

```python
import pandas as pd

# Создаем DataFrame с данными из таблицы
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, 9800,
                        250000, 210000, 135000, 155000, 125000, 180000, 275000, 320000, 105000, 90000],
    'Кредитная история': ['Хорошая', 'Средняя', 'Плохая', 'Хорошая', 'Средняя', 'Отличная', 'Средняя', 'Хорошая', 'Плохая', 'Средняя',
                          'Средняя', 'Отличная', 'Средняя', 'Хорошая', 'Средняя', 'Плохая', 'Отличная', 'Хорошая', 'Средняя', 'Плохая']
}

df = pd.DataFrame(data)

# 1. Подсчет количества пропущенных значений в каждом столбце
null_counts = df.isnull().sum()

# 2. Определение частоты встречаемости значений в столбце "Город"
city_counts = df['Город'].value_counts()

# 3. Подсчет количества уникальных значений в столбцах "Город", "Возраст" и "Баланс на счете"
unique_counts = {
    'Город': df['Город'].nunique(),
    'Возраст': df['Возраст'].nunique(),
    'Баланс на счете': df['Баланс на счете'].nunique()
}

# Вывод результатов
print("Количество пропущенных значений в каждом столбце:")
print(null_counts)

print("\nЧастота встречаемости значений в столбце 'Город':")
print(city_counts)

print("\nКоличество уникальных значений:")
for key, value in unique_counts.items():
    print(f"{key}: {value}")
```

Рисунок 16. Код для выполнения задания

```
Количество пропущенных значений в каждом столбце:
ID                  0
Имя                 0
Возраст             0
Город               0
Баланс на счете     0
Кредитная история   0
dtype: int64

Частота встречаемости значений в столбце 'Город':
Город
Москва              1
Санкт-Петербург     1
Казань              1
Новосибирск         1
Екатеринбург        1
Воронеж             1
Челябинск           1
Краснодар           1
Ростов-на-Дону      1
Уфа                 1
Омск                1
Пермь               1
Тюмень              1
Саратов             1
Самара              1
Волгоград           1
Барнаул             1
Иркутск             1
Хабаровск           1
Томск               1
Name: count, dtype: int64

Количество уникальных значений:
Город: 20
Возраст: 19
Баланс на счете: 20
```

Рисунок 17. Выполненное задание

Задание 8. Обнаружение пропусков (isna , notna)

```python
import pandas as pd

# Создаем DataFrame с данными из таблицы
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', 'Роман', 'Татьяна', 'Николай', 'Валерия', 'Григорий', 'Юлия', 'Степан', 'Василиса', 'Пустой'],
    'Возраст': [34.0, 27.0, 45.0, 38.0, 29.0, 50.0, 31.0, 40.0, 28.0, 33.0, 46.0, 37.0, 41.0, 25.0, 42.0, 49.0, 50.0, 30.0, 35.0, None],
    '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, 9800,
                        250000, 210000, 135000, 155000, 125000, 180000, 275000, 320000, 105000, 90000],
    'Кредитная история': ['Хорошая', 'Средняя', 'Плохая', 'Хорошая', 'Средняя', 'Отличная', 'Средняя', 'Хорошая', 'Плохая', 'Средняя',
                          'Средняя', 'Отличная', 'Средняя', 'Хорошая', 'Средняя', 'Плохая', 'Отличная', 'Хорошая', 'Средняя', 'Плохая']
}

df = pd.DataFrame(data)

# 1. Подсчет количества пропущенных значений в каждом столбце
null_counts = df.isna().sum()

# 2. Подсчет количества заполненных значений в каждом столбце
filled_counts = df.notna().sum()

# 3. Вывести строки, где нет пропущенных значений
rows_without_na = df[df.isna().sum(axis=1) == 0]

# Вывод результатов
print("Количество пропущенных значений в каждом столбце:")
print(null_counts)

print("\nКоличество заполненных значений в каждом столбце:")
print(filled_counts)

print("\nСтроки без пропущенных значений:")
print(rows_without_na)
```

Рисунок 18. Код для выполнения задания

```
Количество пропущенных значений в каждом столбце:
ID                   0
Имя                  0
Возраст              1
Город                0
Баланс на счете      0
Кредитная история    0
dtype: int64

Количество заполненных значений в каждом столбце:
ID                   20
Имя                  20
Возраст              19
Город                20
Баланс на счете      20
Кредитная история    20
dtype: int64

Строки без пропущенных значений:
    ID       Имя  Возраст            Город  Баланс на счете Кредитная история
0    1      Иван      34.0           Москва           120000            Хорошая
1    2     Ольга      27.0  Санкт-Петербург            80000            Средняя
2    3   Алексей      45.0           Казань           150000             Плохая
3    4     Мария      38.0       Новосибирск           200000            Хорошая
4    5    Сергей      29.0      Екатеринбург            95000            Средняя
5    6      Анна      50.0           Воронеж           300000           Отличная
6    7   Дмитрий      31.0         Челябинск           140000            Средняя
7    8     Елена      40.0         Краснодар           175000            Хорошая
8    9    Виктор      28.0   Ростов-на-Дону           110000             Плохая
9   10     Алиса      33.0               Уфа             9800            Средняя
10  11     Павел      46.0              Омск           250000            Средняя
11  12     Роман      37.0             Пермь           210000           Отличная
12  13   Татьяна      41.0            Тюмень           135000            Средняя
13  14   Николай      25.0          Саратов           155000            Хорошая
14  15   Валерия      42.0           Самара           125000            Средняя
15  16  Григорий      49.0         Волгоград           180000             Плохая
16  17      Юлия      50.0          Барнаул           275000           Отличная
17  18    Степан      30.0          Иркутск           320000            Хорошая
18  19  Василиса      35.0         Хабаровск           105000            Средняя
```

Рисунок 19. Выполненное задание

Индивидуальное задание:

```python
import pandas as pd

# Создаем начальный DataFrame
columns = ['Пункт назначения', 'Номер поезда', 'Время отправления']
train_data = pd.DataFrame(columns=columns)

# Функция для ввода данных
def input_train():
    destination = input("Введите пункт назначения: ")
    train_number = input("Введите номер поезда: ")
    departure_time = input("Введите время отправления (в формате НН:ММ): ")
    return {columns [0] : destination, columns [1] : train_number, columns [2] : departure_time}

# Ввод данных (например, 3 записи)
for _ in range(3):
    row = input_train()
    # Используем pd.DataFrame([row]) для добавления
    train_data = pd.concat([train_data, pd.DataFrame([row])], ignore_index=True)

# Сортируем по пункту назначения
train_data = train_data.sort_values(by='Пункт назначения')

# Ввод времени для фильтрации
time_threshold = input("Введите время для фильтрации (в формате НН:ММ): ")

# Преобразуем строки в datetime
train_data['Время отправления'] = pd.to_datetime(train_data['Время отправления'], format='%H:%M')
time_threshold_dt = pd.to_datetime(time_threshold, format='%H:%M')

# Фильтр по времени
filtered_trains = train_data[train_data['Время отправления'] > time_threshold_dt]

# Вывод результата
if filtered_trains.empty:
    print("Поездов, отправляющихся после указанного времени, нет.")
else:
    print("Поезда, отправляющиеся после указанного времени:")
    print(filtered_trains)
```

```
Введите пункт назначения:  Ставрополь
Введите номер поезда:  3
Введите время отправления (в формате НН:ММ):  12:30
Введите пункт назначения:  Зеленокумск
Введите номер поезда:  3
Введите время отправления (в формате НН:ММ):  ↑↓ for history. Search history with c-↑/c-↓
```

Рисунок 20. Выполненное задание

Ответы на контрольные вопросы:

1. Как создать pandas.DataFrame из словаря списков?

Ответ:

Использовать конструктор pd.DataFrame() и передать словарь, где ключи — имена столбцов, а значения — списки данных. Например:

df = pd.DataFrame({'Столбец1': [1, 2], 'Столбец2': [3, 4]})

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

Ответ:

- Из списка словарей создается DataFrame, где каждый словарь — строка.

- Из словаря списков — создается DataFrame, где ключи — столбцы, а списки — данные по столбцам.

3. Как создать pandas.Numpy из массива NumPy?

Ответ:

```
import numpy as np
import pandas as pd
array = np.array([[1, 2], [3, 4]])
df = pd.DataFrame(array)
```

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

Ответ:

Использовать функцию pd.read_csv(), указав параметр sep. Например:

df = pd.read_csv('файл.csv', sep=';')

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

Ответ:

Использовать pd.read_excel(), указав название листа через параметр sheet_name:

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

6. Чем отличается чтение данных из JSON и Parquet?

Ответ:

- JSON — это текстовый формат, легко читаемый человеком и обычно используемый для обмена данными между системами.

- Parquet — это бинарный колонковый формат, предназначенный для эффективного хранения и быстрого чтения больших объемов данных.

7. Как проверить тип данных в DataFrame после загрузки?

Ответ:

Использовать type() или df.dtypes. Например:

type(df)

df.dtypes

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

Ответ:

Использовать df.shape. Например:

rows, columns = df.shape

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

Ответ:

- .loc[] — доступ по меткам индексов и названиям столбцов.

- .iloc[] — доступ по позициям (целым числам) строк и столбцов.

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

Ответ:

Использовать .loc[]:

df.loc['Мария']

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

Ответ:

df.loc['Паша']

12. Чем .at[] отличается от .loc[]?

Ответ:

- .at[] — быстрый доступ к одному элементу по метке.
- .loc[] — доступ к одному или нескольким элементам по меткам.

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

Ответ:

Когда нужно быстро получить один элемент по позиции (целым числом).

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

Ответ:

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

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

Ответ:

df[(df['Возраст'] >= 25) & (df['Возраст'] <= 35)]

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

Ответ:

- .query() использует строковый синтаксис, похожий на SQL.

- .loc[] — более универсален, позволяет фильтровать по меткам и условиям.

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

Ответ:

Использовать @ для вставки переменной:

Lua

age_limit = 30

df.query('Возраст > @age_limit')

18. Как узнать, сколько пропущенных значений в каждой колонке DataFrame?

Ответ:

Использовать .isna().sum():

df.isna().sum()

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

Ответ:

- .isna() — возвращает True для пропущенных значений (NaN).

- .notna() — возвращает True для заполненных значений.

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

Ответ:

df.dropna()

21. Как добавить новый столбец "Категория" с фиксированным значением "Неизвестно"?

Ответ:

df['Категория'] = 'Неизвестно'

22. Как удалить строку по индексу?

Ответ:

df.drop(index)

23. Как удалить столбец "Возраст"?

Ответ:

df.drop('Возраст', axis=1)

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

Ответ:

df.dropna(how='any')

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

Ответ:

df.dropna(axis=1, how='any')

26. Как посчитать количество пустых значений в каждом столбце?

Ответ:

Использовать .isna().sum(), как в вопросе 18.

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

Ответ:

- .value_counts() — считает количество уникальных значений и их частоты.

- .nunique() — возвращает число уникальных значений.

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

Ответ:

df['Город'].value_counts()

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

Ответ:

display() показывает DataFrame в виде хорошо отформатированной таблицы, а print() — в виде текста, что менее удобно для визуального восприятия.

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

Ответ:

Использовать настройку pandas:

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

Вывод: в ходе лабораторной работы познакомились с основами работы с библиотекой pandas, в частности, а также ознакомились со структурой данных Series.