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 «РаботасJupyterNotebook, JupyterLab, GoogleColab» по дисциплине «Искусственный интеллект и машинное обучение»

Выполнил:
Гончаров Серафим Ростиславович 2 курс, группа ИВТ-б-о-23-1,09.03.01 «Информатика и вычислительная техника», направленность (профиль) «Автоматизированные системы обработки информации и управления», очная форма обучения

Руководитель практики:
Воронкин Роман Александрович

Ставрополь 2025

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.

Порядок выполнения лабораторной работы:

GoncharovSerafim/Lab5

Задание 1. Различие между Series и DataFrame.

```
[1]: import pandas as pd
     s = pd.Series([10, 20, 30, 40], index=["a", "b", "c", "d"])
     print(s)

a    10
b    20
c    30
d    40
dtype: int64
```

```
[3]: data = {
     "Возраст": [25, 30, 22],
     "Город": ["Москва", "СПб", "Казань"]
     }
     df = pd.DataFrame(data, index=["Анна", "Иван", "Ольга"])
     print(df)

       Возраст   Город
Анна        25  Москва
Иван        30     СПб
Ольга       22  Казань
```

```
[5]: print(df["Возраст"])

Анна     25
Иван     30
Ольга    22
Name: Возраст, dtype: int64
```

```
[7]: s = pd.Series([10, 20, 30], index=["a", "b", "c"])
     df = s.to_frame(name="Значение")
     print(df)

   Значение
a        10
b        20
c        30
```

```
[9]: s = df["Значение"]
     print(s)

a    10
b    20
c    30
Name: Значение, dtype: int64
```

Рисунок 1 – Series и DataFrame

Задание 2. Создание DataFrame из различных источников данных.

[27]:
```
import pandas as pd
df = pd.read_csv("data.csv")
print(df.head()) # Выведем первые 5 строк
```

```
         Дата  Цена
0  2024-03-01   100
1  2024-03-02   110
2  2024-03-03   105
3  2024-03-04   120
4  2024-03-05   115
```

[29]:
```
df = pd.read_csv("data.csv", sep=";", index_col=0, na_values=["?", "N/A"])
```

[31]:
```
df.to_csv("output.csv", index=False) # index=False – не сохранять индекс
```

[ ]:
```
df = pd.read_excel("data.xlsx", sheet_name="Лист1")
print(df.head())
```

[36]:
```
df.to_excel("output.xlsx", sheet_name="Результаты", index=False)
```

[38]:
```
pip install openpyxl
```

```
Requirement already satisfied: openpyxl in c:\users\hesko\anaconda3\lib\site-packages (3.1.5)
Requirement already satisfied: et-xmlfile in c:\users\hesko\anaconda3\lib\site-packages (from openpyxl) (1.1.0)
Note: you may need to restart the kernel to use updated packages.
```

[233]:
```
df.to_sql("users", conn, if_exists="replace", index=False )
```

[233]:
```
3
```

[51]:
```
df.to_json("output.json", orient="records", indent=4)
```

[59]:
```
!pip install pyarrow
```

```
Requirement already satisfied: pyarrow in c:\users\hesko\anaconda3\lib\site-packages (16.1.0)
Requirement already satisfied: numpy>=1.16.6 in c:\users\hesko\anaconda3\lib\site-packages (from pyarrow) (1.26.4)
```

[ ]:
```
df = pd.read_parquet("data.parquet")
print(df.head())
```

[63]:
```
df.to_parquet("output.parquet", engine="pyarrow", index=False)
```

Рисунок 2 – DataFrame из различных источников данных

Задание 3. Основные методы для первичного анализа данных DataFrame.

[65]:
```
import pandas as pd
# Создадим небольшой DataFrame
data = {
"Имя": ["Анна", "Иван", "Ольга", "Петр", "Мария", "Сергей"],
"Возраст": [25, 30, 22, 40, 35, 28],
"Город": ["Москва", "СПб", "Казань", "Новосибирск",
"Екатеринбург", "Сочи"]
}
df = pd.DataFrame(data)
# Выведем первые 3 строки
print(df.head(3))
```

```
     Имя  Возраст   Город
0   Анна       25  Москва
1   Иван       30     СПб
2  Ольга       22  Казань
```

[67]:
```
print(df.tail(2)) # Последние 2 строки
```

```
      Имя  Возраст         Город
4   Мария       35  Екатеринбург
5  Сергей       28          Сочи
```

[69]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Имя      6 non-null      object
 1   Возраст  6 non-null      int64
 2   Город    6 non-null      object
dtypes: int64(1), object(2)
memory usage: 276.0+ bytes
```

[71]:
```
print(df.describe())
```

```
        Возраст
count   6.00000
mean   30.00000
std     6.60303
min    22.00000
25%    25.75000
50%    29.00000
```

Рисунок 3 – Первичный анализ данных DataFrame

Задание 4. Доступ к данным в DataFrame.

[75]:
```
import pandas as pd
data = {
"Имя": ["Анна", "Иван", "Ольга", "Петр"],
"Возраст": [25, 30, 22, 40],
"Город": ["Москва", "СПб", "Казань", "Новосибирск"]
}
df = pd.DataFrame(data, index=["a", "b", "c", "d"])
print(df)
```

```
     Имя  Возраст        Город
a   Анна       25       Москва
b   Иван       30          СПб
c  Ольга       22       Казань
d   Петр       40  Новосибирск
```

[77]:
```
print(df.loc["b"])
```

```
Имя        Иван
Возраст      30
Город       СПб
Name: b, dtype: object
```

[79]:
```
print(df.loc["c", "Город"]) # Получим город Ольги
```

```
Казань
```

[81]:
```
print(df.loc[["a", "c"]]) # Выбор строк "a" и "c"
```

```
     Имя  Возраст   Город
a   Анна       25  Москва
c  Ольга       22  Казань
```

[83]:
```
print(df.loc[:, ["Имя", "Город"]]) # Выбор всех строк, но только двух столбцов
```

```
     Имя        Город
a   Анна       Москва
b   Иван          СПб
c  Ольга       Казань
d   Петр  Новосибирск
```

[85]:
```
print(df.loc[df["Возраст"] > 25]) # Выведем строки, где возраст больше 25
```

```
    Имя  Возраст        Город
b  Иван       30          СПб
d  Петр       40  Новосибирск
```

Рисунок 4 – Доступ к данным в DataFrame

Задание 5. Добавление строк и столбцов в DataFrame.

[101]:
```
import pandas as pd
data = {
"Имя": ["Анна", "Иван", "Ольга"],
"Возраст": [25, 30, 22]
}
df = pd.DataFrame(data)
df["Город"] = ["Москва", "СПб", "Казань"] # Добавляем новый столбец
print(df)
```

```
     Имя  Возраст   Город
0   Анна       25  Москва
1   Иван       30     СПб
2  Ольга       22  Казань
```

[103]:
```
df["Страна"] = "Россия"
print(df)
```

```
     Имя  Возраст   Город  Страна
0   Анна       25  Москва  Россия
1   Иван       30     СПб  Россия
2  Ольга       22  Казань  Россия
```

[105]:
```
df["Возраст_в_месяцах"] = df["Возраст"] * 12
print(df)
```

```
     Имя  Возраст   Город  Страна  Возраст_в_месяцах
0   Анна       25  Москва  Россия                300
1   Иван       30     СПб  Россия                360
2  Ольга       22  Казань  Россия                264
```

[107]:
```
df = df.assign(Зарплата=[50000, 60000, 45000])
print(df)
```

```
     Имя  Возраст   Город  Страна  Возраст_в_месяцах  Зарплата
0   Анна       25  Москва  Россия                300     50000
1   Иван       30     СПб  Россия                360     60000
2  Ольга       22  Казань  Россия                264     45000
```

[109]:
```
df["Категория возраста"] = df["Возраст"].apply(lambda x: "Молодой" if
x < 30 else "Взрослый")
print(df)
```

```
     Имя  Возраст   Город  Страна  Возраст_в_месяцах  Зарплата  \
0   Анна       25  Москва  Россия                300     50000
1   Иван       30     СПб  Россия                360     60000
```

Рисунок 5 – Добавление строк и столбцов в DataFrame

Задание 6. Удаление строк и столбцов в DataFrame.

[129]:
```
import pandas as pd
data = {
"Имя": ["Анна", "Иван", "Ольга"],
"Возраст": [25, 30, 22],
"Город": ["Москва", "СПб", "Казань"],
"Зарплата": [50000, 60000, 45000]
}
df = pd.DataFrame(data)
# Удаляем столбец "Зарплата"
df = df.drop(columns=["Зарплата"])
print(df)
```

```
     Имя  Возраст   Город
0   Анна       25  Москва
1   Иван       30     СПб
2  Ольга       22  Казань
```

[131]:
```
df = df.drop(columns=["Возраст", "Город"])
print(df)
```

```
     Имя
0   Анна
1   Иван
2  Ольга
```

[133]:
```
del df["Имя"]
print(df)
```

```
Empty DataFrame
Columns: []
Index: [0, 1, 2]
```

[135]:
```
df["Имя"] = ["Анна", "Иван", "Ольга"] # Вернем столбец для примера
удаленный_столбец = df.pop("Имя")
print(удаленный_столбец)
```

```
0     Анна
1     Иван
2    Ольга
Name: Имя, dtype: object
```

[137]:
```
df = pd.DataFrame(data) # Восстанавливаем DataFrame
# Удаляем строку с индексом 1 (Иван)
df = df.drop(index=1)
print(df)
```

Рисунок 6 – Удаление строк и столбцов в DataFrame

Задание 7. Фильтрация и условная индексация в DataFrame.

```
import pandas as pd
data = {
"Имя": ["Анна", "Иван", "Ольга", "Петр", "Мария"],
"Возраст": [25, 30, 22, 40, 35],
"Город": ["Москва", "СПб", "Казань", "Новосибирск", "СПб"],
"Зарплата": [50000, 60000, 45000, 70000, 65000]
}
df = pd.DataFrame(data)
print(df)
```

```
     Имя  Возраст        Город  Зарплата
0   Анна       25       Москва     50000
1   Иван       30          СПб     60000
2  Ольга       22       Казань     45000
3   Петр       40  Новосибирск     70000
4  Мария       35          СПб     65000
```

```
df_filtered = df.query("Возраст > 30")
print(df_filtered)
```

```
     Имя  Возраст        Город  Зарплата
3   Петр       40  Новосибирск     70000
4  Мария       35          СПб     65000
```

```
df_filtered = df.query("Город == 'СПб' and Зарплата > 60000")
print(df_filtered)
```

```
     Имя  Возраст Город  Зарплата
4  Мария       35   СПб     65000
```

```
min_age = 25
df_filtered = df.query("Возраст > @min_age")
print(df_filtered)
```

```
     Имя  Возраст        Город  Зарплата
1   Иван       30          СПб     60000
3   Петр       40  Новосибирск     70000
4  Мария       35          СПб     65000
```

```
df_filtered = df[df["Город"].isin(["Москва", "СПб"])]
print(df_filtered)
```

```
     Имя  Возраст   Город  Зарплата
0   Анна       25  Москва     50000
1   Иван       30     СПб     60000
4  Мария       35     СПб     65000
```

Рисунок 7 – Фильтрация и условная индексация в DataFrame

Задание 8. Подсчет значений в DataFrame.

```
import pandas as pd
data = {"Имя": ["Анна", "Иван", "Ольга", "Петр", None],
"Возраст": [25, 30, 22, None, 35],
"Город": ["Москва", "СПб", "Казань", "Новосибирск", "СПб"]
}
df = pd.DataFrame(data)
print(df)
```

```
     Имя  Возраст        Город
0   Анна     25.0       Москва
1   Иван     30.0          СПб
2  Ольга     22.0       Казань
3   Петр      NaN  Новосибирск
4   None     35.0          СПб
```

```
print(df.count())
```

```
Имя        4
Возраст    4
Город      5
dtype: int64
```

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

```
Город
СПб           2
Москва        1
Казань        1
Новосибирск   1
Name: count, dtype: int64
```

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

```
Город
Москва        1
СПб           2
Казань        1
Новосибирск   1
Name: count, dtype: int64
```

```
print(df["Имя"].value_counts(dropna=False))
```

```
Имя
Анна    1
Иван    1
```

Рисунок 8 – Подсчет значений в DataFrame

Задание 9. Сортировка данных в DataFrame.

[185]:
```
import pandas as pd
data = {
"Имя": ["Анна", "Иван", "Ольга", "Петр", "Мария"],
"Возраст": [25, 30, 22, 40, 35],
"Зарплата": [50000, 60000, 45000, 70000, 65000]
}
df = pd.DataFrame(data)
print(df)
```

```
     Имя  Возраст  Зарплата
0   Анна       25     50000
1   Иван       30     60000
2  Ольга       22     45000
3   Петр       40     70000
4  Мария       35     65000
```

[187]:
```
df_sorted = df.sort_values(by="Возраст")
print(df_sorted)
```

```
     Имя  Возраст  Зарплата
2  Ольга       22     45000
0   Анна       25     50000
1   Иван       30     60000
4  Мария       35     65000
3   Петр       40     70000
```

[189]:
```
df_sorted = df.sort_values(by="Возраст", ascending=False)
print(df_sorted)
```

```
     Имя  Возраст  Зарплата
3   Петр       40     70000
4  Мария       35     65000
1   Иван       30     60000
0   Анна       25     50000
2  Ольга       22     45000
```

[193]:
```
df.loc[5] = ["Елена", None, 55000] # Добавляем строку с NaN в
"Возраст"
df_sorted = df.sort_values(by="Возраст", na_position="first")
print(df_sorted)
```

```
     Имя Возраст  Зарплата
5  Елена    None     55000
2  Ольга      22     45000
0   Анна      25     50000
```

Рисунок 9 – Сортировка данных в DataFrame

Задание 10. Задание 1. Создание DataFrame разными способами

1. Создайте DataFrame из словаря списков с данными о пяти сотрудниках (возьмите из таблицы 1).
2. Создайте DataFrame из списка словарей с теми же данными.
3. Создайте DataFrame из массива NumPy, заполненного случайными числами от 20 до 60 (для возраста сотрудников).
4. Проверьте типы данных в каждом столбце с помощью .info()

[17]:
```
import pandas as pd
import numpy as np

data_dict = {
    '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]
}
df1 = pd.DataFrame(data_dict)
print("1. DataFrame из словаря списков:")
print(df1)

data_list = [
    {'Имя': 'Иван', 'Возраст': 25, 'Должность': 'Инженер', 'Отдел': 'IT', 'Зарплата': 60000, 'Стаж работы': 2},
    {'Имя': 'Ольга', 'Возраст': 30, 'Должность': 'Аналитик', 'Отдел': 'Маркетинг', 'Зарплата': 75000, 'Стаж работы': 5},
    {'Имя': 'Алексей', 'Возраст': 40, 'Должность': 'Менеджер', 'Отдел': 'Продажи', 'Зарплата': 90000, 'Стаж работы': 15},
    {'Имя': 'Мария', 'Возраст': 35, 'Должность': 'Программист', 'Отдел': 'IT', 'Зарплата': 80000, 'Стаж работы': 7},
    {'Имя': 'Сергей', 'Возраст': 28, 'Должность': 'Специалист', 'Отдел': 'HR', 'Зарплата': 50000, 'Стаж работы': 3}
]
df2 = pd.DataFrame(data_list)
print("\n2. DataFrame из списка словарей:")
print(df2)

np_ages = np.random.randint(20, 61, size=5)
df3 = pd.DataFrame(np_ages, columns=['Возраст'])
print("\n3. DataFrame из массива NumPy:")
print(df3)
```

```python
print("\n4. Типы данных в df1:")
df1.info()

print("\nТипы данных в df2:")
df2.info()

print("\nТипы данных в df3:")
df3.info()
```

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

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

3. DataFrame из массива NumPy:
   Возраст
0       52
1       25
2       48
3       38
4       37

4. Типы данных в df1:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Имя          5 non-null      object
 1   Возраст      5 non-null      int64
 2   Должность    5 non-null      object
 3   Отдел        5 non-null      object
 4   Зарплата     5 non-null      int64
 5   Стаж работы  5 non-null      int64
```

Рисунок 10 – Создание DataFrame разными способами

Задание 11. Задание 2. Чтение данных из файлов (CSV , Excel , JSON)

1. Сохраните Таблицу 1 в CSV и затем загрузите ее обратно в DataFrame .
2. Запишите Таблицу 2 в Excel (data.xlsx , лист "Клиенты") и прочитайте ее в DataFrame .
3. Экспортируйте Таблицу 1 в формат JSON и затем прочитайте ее обратно вDataFrame .

[27]:
```
data1 = {
    '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]
}

data2 = {
    'ID': [1, 2, 3, 4, 5],
    'Имя': ['Иван', 'Ольга', 'Алексей', 'Мария', 'Сергей'],
    'Возраст': [34, 27, 45, 38, 29],
    'Город': ['Москва', 'Санкт-Петербург', 'Казань', 'Новосибирск', 'Екатеринбург'],
    'Баланс на счете': [120000, 80000, 150000, 200000, 95000],
    'Кредитная история': ['Хорошая', 'Средняя', 'Плохая', 'Хорошая', 'Средняя']
}

df1 = pd.DataFrame(data1)
df1.to_csv('employees.csv', index=False, encoding='utf-8')

df1_csv = pd.read_csv('employees.csv')
print("1. Данные из CSV (первые 5 строк):")
print(df1_csv.head())

df2 = pd.DataFrame(data2)
with pd.ExcelWriter('data.xlsx') as writer:
    df2.to_excel(writer, sheet_name='Клиенты', index=False)

df2_excel = pd.read_excel('data.xlsx', sheet_name='Клиенты')
print("\n2. Данные из Excel (первые 5 строк):")
print(df2_excel.head())

df1.to_json('employees.json', orient='records', force_ascii=False)

df1_json = pd.read_json('employees.json')
print("\n3. Данные из JSON (первые 5 строк):")
print(df1_json.head())
```

```
1. Данные из CSV (первые 5 строк):
   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

2. Данные из Excel (первые 5 строк):
   ID      Имя  Возраст            Город  Баланс на счете Кредитная история
0   1     Иван       34           Москва           120000           Хорошая
1   2    Ольга       27  Санкт-Петербург            80000           Средняя
2   3  Алексей       45           Казань           150000            Плохая
3   4    Мария       38      Новосибирск           200000           Хорошая
4   5   Сергей       29     Екатеринбург            95000           Средняя

3. Данные из JSON (первые 5 строк):
   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
```

Рисунок 11 – Чтение данных из файлов

Задание 12. Задание 3. Доступ к данным (.loc , .iloc , .at , .iat)

1. Получите информацию о сотруднике с ID = 5 с помощью .loc[] .
2. Выведите возраст третьего сотрудника в таблице с .iloc[] .
3. Выведите название отдела для сотрудника "Мария" с .at[] .
4. Выведите зарплату сотрудника, находящегося в четвертой строке и пятом столбце, используя .iat[] .

[35]:
```
df = pd.DataFrame(data1)
employee_5 = df.loc[df['ID'] == 5]
print("1. Сотрудник с ID = 5:")
print(employee_5)

age_3rd = df.iloc[2]['Возраст']  # или df.iloc[2, 2]
print("\n2. Возраст третьего сотрудника:", age_3rd)

position = df.at[3, 'Должность']
print("\n3. Должность Марии:", position)

salary = df.iat[3, 5]
print("\n4. Зарплата сотрудника в 4 строке:", salary)
```

```
1. Сотрудник с ID = 5:
   ID     Имя  Возраст   Должность Отдел  Зарплата  Стаж работы
4   5  Сергей       28  Специалист    HR     50000            3

2. Возраст третьего сотрудника: 40

3. Должность Марии: Программист

4. Зарплата сотрудника в 4 строке: 80000
```

Рисунок 12 – Доступ к данным

Задание 13. Задание 4. Добавление новых столбцов и строк

1. Добавьте новый столбец "Категория зарплаты" :
"Низкая" – если Зарплата < 60000
"Средняя" – если 60000 ≤ Зарплата < 100000
"Высокая" – если Зарплата ≥ 100000
2. Добавьте нового сотрудника с данными:
df.loc[21] = ["Антон", 32, "Разработчик", "IT", 85000, 6]
3. Добавьте двух новых сотрудников одновременно, используя pd.concat()

[51]:
```
def salary_category(salary):
    if salary < 60000:
        return "Низкая"
    elif 60000 <= salary < 100000:
        return "Средняя"
    else:
        return "Высокая"

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

df.loc[6] = [6, "Антон", 32, "Разработчик", "IT", 85000, 10, "Средняя"]

new_employees = pd.DataFrame({
    'ID': [7, 8],
    'Имя': ['Елена', 'Сергей'],
    'Возраст': [27, 45],
    'Должность': ['Бухгалтер', 'Руководитель отдела'],
    'Отдел': ['Финансы', 'Администрация'],
    'Зарплата': [65000, 150000],
    'Категория зарплаты': ['Средняя', 'Высокая']
})

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

print("Обновленная таблица сотрудников:")
print(df)
```

```
Обновленная таблица сотрудников:
    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    8   Сергей        45  Руководитель отдела  Администрация    150000
6    7    Елена        27            Бухгалтер        Финансы     65000
7    8   Сергей        45  Руководитель отдела  Администрация    150000
8    7    Елена        27            Бухгалтер        Финансы     65000
9    8   Сергей        45  Руководитель отдела  Администрация    150000
10   7    Елена        27            Бухгалтер        Финансы     65000
11   8   Сергей        45  Руководитель отдела  Администрация    150000
12   6    Антон        32          Разработчик             IT     85000
13   7    Елена        27            Бухгалтер        Финансы     65000
14   8   Сергей        45  Руководитель отдела  Администрация    150000

    Стаж работы Категория зарплаты
0           2.0             Средняя
1           5.0             Средняя
2          15.0             Средняя
3           7.0             Средняя
4           3.0              Низкая
5           NaN             Высокая
6           NaN             Средняя
7           NaN             Высокая
8           NaN             Средняя
9           NaN             Высокая
10          NaN             Средняя
11          NaN             Высокая
12         10.0             Средняя
13          NaN             Средняя
14          NaN             Высокая
```

Рисунок 13 – Добавление новых столбцов и строк

Задание 14. Задание 5. Удаление строк и столбцов

1. Удалите столбец "Категория зарплаты" с .drop() .
2. Удалите строку с ID = 10 .
3. Удалите все строки, где Стаж работы < 3 лет .
4. Удалите все столбцы, кроме Имя , Должность , Зарплата .

[53]:
```
print("Исходник:")
print(df)

df = df.drop('Категория зарплаты', axis=1)
print("\n1. После удаления столбца 'Категория зарплаты':")
print(df)

df = df[df['ID'] != 6]
print("\n2. После удаления строки с ID = 6:")
print(df)

df = df[df['Стаж работы'] >= 3]
print("\n3. После удаления сотрудников со стажем < 3 лет:")
print(df)

df = df[['Имя', 'Должность', 'Зарплата']]
print("\n4. После удаления всех столбцов, кроме Имя, Должность, Зарплата:")
print(df)
```

```
Исходник:
    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    8   Сергей        45  Руководитель отдела  Администрация    150000
6    7    Елена        27             Бухгалтер        Финансы     65000
7    8   Сергей        45  Руководитель отдела  Администрация    150000
8    7    Елена        27             Бухгалтер        Финансы     65000
9    8   Сергей        45  Руководитель отдела  Администрация    150000
10   7    Елена        27             Бухгалтер        Финансы     65000
11   8   Сергей        45  Руководитель отдела  Администрация    150000
12   6    Антон        32          Разработчик             IT     85000
13   7    Елена        27             Бухгалтер        Финансы     65000
14   8   Сергей        45  Руководитель отдела  Администрация    150000
```

```
    Стаж работы Категория зарплаты
0           2.0              Средняя
1           5.0              Средняя
2          15.0              Средняя
3           7.0              Средняя
4           3.0               Низкая
5           NaN              Высокая
6           NaN              Средняя
7           NaN              Высокая
8           NaN              Средняя
9           NaN              Высокая
10          NaN              Средняя
11          NaN              Высокая
12         10.0              Средняя
13          NaN              Средняя
14          NaN              Высокая

1. После удаления столбца 'Категория зарплаты':
    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    8   Сергей       45  Руководитель отдела  Администрация    150000
6    7    Елена       27           Бухгалтер        Финансы     65000
7    8   Сергей       45  Руководитель отдела  Администрация    150000
8    7    Елена       27           Бухгалтер        Финансы     65000
9    8   Сергей       45  Руководитель отдела  Администрация    150000
10   7    Елена       27           Бухгалтер        Финансы     65000
11   8   Сергей       45  Руководитель отдела  Администрация    150000
12   6    Антон       32         Разработчик             IT     85000
13   7    Елена       27           Бухгалтер        Финансы     65000
14   8   Сергей       45  Руководитель отдела  Администрация    150000

    Стаж работы
0           2.0
1           5.0
2          15.0
3           7.0
4           3.0
5           NaN
6           NaN
7           NaN
8           NaN
```

```
2. После удаления строки с ID = 6:
    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    8   Сергей       45  Руководитель отдела  Администрация    150000
6    7    Елена       27            Бухгалтер       Финансы     65000
7    8   Сергей       45  Руководитель отдела  Администрация    150000
8    7    Елена       27            Бухгалтер       Финансы     65000
9    8   Сергей       45  Руководитель отдела  Администрация    150000
10   7    Елена       27            Бухгалтер       Финансы     65000
11   8   Сергей       45  Руководитель отдела  Администрация    150000
13   7    Елена       27            Бухгалтер       Финансы     65000
14   8   Сергей       45  Руководитель отдела  Администрация    150000

    Стаж работы
0           2.0
1           5.0
2          15.0
3           7.0
4           3.0
5           NaN
6           NaN
7           NaN
8           NaN
9           NaN
10          NaN
11          NaN
13          NaN
14          NaN

3. После удаления сотрудников со стажем < 3 лет:
   ID      Имя  Возраст    Должность      Отдел  Зарплата  Стаж работы
1   2    Ольга       30     Аналитик  Маркетинг     75000          5.0
2   3  Алексей       40     Менеджер    Продажи     90000         15.0
3   4    Мария       35  Программист         IT     80000          7.0
4   5   Сергей       28   Специалист         HR     50000          3.0

4. После удаления всех столбцов, кроме Имя, Должность, Зарплата:
       Имя    Должность  Зарплата
1    Ольга     Аналитик     75000
2  Алексей     Менеджер     90000
3    Мария  Программист     80000
```

Рисунок 14 – Удаление строк и столбцов

Задание 15. Задание 6. Фильтрация данных (query , isin , between)

1. Выберите всех клиентов из "Москва" или "Санкт-Петербург" , используя .isin() .
2. Выберите клиентов, у которых Баланс на счете от 100000 до 250000, используя .between() .
3. Отфильтруйте клиентов, у которых "Кредитная история" "Хорошая" и "Баланс на счете" > 150000 , используя .query() .

[57]:
```
cities = ['Москва', 'Санкт-Петербург']
clients_city = df2[df2['Город'].isin(cities)]
print("1. Клиенты из Москвы или Санкт-Петербурга:")
print(clients_city)

clients_balance = df2[df2['Баланс на счете'].between(100000, 250000)]
print("\n2. Клиенты с балансом от 100000 до 250000:")
print(clients_balance)

clients_filtered = df2.query("`Кредитная история` == 'Хорошая' and `Баланс на счете` > 150000")
print("\n3. Клиенты с хорошей кредитной историей и балансом > 150000:")
print(clients_filtered)
```

```
1. 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

2. 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

3. Клиенты с хорошей кредитной историей и балансом > 150000:
   ID    Имя  Возраст        Город  Баланс на счете Кредитная история
3   4  Мария       38  Новосибирск           200000           Хорошая
```

Рисунок 15 – Фильтрация данных

Задание 16. Задание 7. Подсчет значений (count , value_counts , nunique)

1. Подсчитайте количество непустых значений в каждом столбце (.count()).
2. Определите частоту встречаемости значений в "Город" (.value_counts()).
3. Найдите количество уникальных значений в "Город" , "Возраст" , "Баланс на счете" (.nunique()).

[61]:
```
print("1. Количество непустых значений:")
print(df2.count())

print("\n2. Частота встречаемости городов:")
print(df2['Город'].value_counts(dropna=False))

print("\n3. Количество уникальных значений:")
print("Город:", df2['Город'].nunique(dropna=True))
print("Возраст:", df2['Возраст'].nunique())
print("Баланс на счете:", df2['Баланс на счете'].nunique())
```

```
1. Количество непустых значений:
ID                  5
Имя                 5
Возраст             5
Город               5
Баланс на счете     5
Кредитная история   5
dtype: int64

2. Частота встречаемости городов:
Город
Москва             1
Санкт-Петербург    1
Казань             1
Новосибирск        1
Екатеринбург       1
Name: count, dtype: int64

3. Количество уникальных значений:
Город: 5
Возраст: 5
Баланс на счете: 5
```

Рисунок 16 – Подсчет значений

Задание 17. Задание 8. Обнаружение пропусков (isna , notna)

1. Подсчитайте количество NaN в каждом столбце (.isna().sum()).
2. Подсчитайте количество заполненных значений в каждом столбце (.notna().sum()).
3. Выведите DataFrame , оставив только строки, где нет пропущенных значений.

```
data2 = {
    'ID': [1, 2, 3, 4, 5],
    'Имя': ['Иван', 'Ольга', 'Алексей', 'Мария', 'Сергей'],
    'Возраст': [34, 27, np.nan, 38, 29],
    'Город': ['Москва', 'Санкт-Петербург', 'Казань', np.nan, 'Екатеринбург'],
    'Баланс на счете': [120000, np.nan, 150000, 200000, np.nan],
    'Кредитная история': ['Хорошая', 'Средняя', 'Плохая', 'Хорошая', np.nan]
}
df2 = pd.DataFrame(data2)
print("Исходная таблица:")
print(df2)

nan_counts = df2.isna().sum()
print("\n1. Количество пропущенных значений (NaN) в каждом столбце:")
print(nan_counts)

notna_counts = df2.notna().sum()
print("\n2. Количество заполненных значений в каждом столбце:")
print(notna_counts)

df_clean = df2.dropna()
print("\n3. Таблица без пропущенных значений:")
print(df_clean)
```

```
Исходная таблица:
   ID      Имя  Возраст            Город  Баланс на счете Кредитная история
0   1     Иван     34.0           Москва         120000.0            Хорошая
1   2    Ольга     27.0  Санкт-Петербург              NaN           Средняя
2   3  Алексей      NaN           Казань         150000.0            Плохая
3   4    Мария     38.0              NaN         200000.0            Хорошая
4   5   Сергей     29.0     Екатеринбург              NaN               NaN

1. Количество пропущенных значений (NaN) в каждом столбце:
ID                   0
Имя                  0
Возраст              1
Город                1
Баланс на счете      2
Кредитная история    1
dtype: int64

2. Количество заполненных значений в каждом столбце:
ID                   5
Имя                  5
Возраст              4
Город                4
Баланс на счете      3
Кредитная история    4
dtype: int64

3. Таблица без пропущенных значений:
   ID   Имя  Возраст   Город  Баланс на счете Кредитная история
0   1  Иван     34.0  Москва         120000.0            Хорошая
```

Рисунок 17 – Обнаружение пропусков

Задание 18. Индивидуальное задание

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 маршрутах, которые начинаются или оканчиваются в пункте, название которого введено с клавиатуры; если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

```python
import pandas as pd
import tkinter as tk
from tkinter import ttk, messagebox
from datetime import datetime

class RouteManagerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Менеджер маршрутов")
        self.root.geometry("600x400")
        self.df = pd.DataFrame(columns=['start_point', 'end_point', 'route_number', 'created_at'])
        self.create_widgets()

    def create_widgets(self):
        # Вкладки
        self.notebook = ttk.Notebook(self.root)
        self.notebook.pack(fill='both', expand=True)

        # Вкладка добавления
        self.add_tab = ttk.Frame(self.notebook)
        self.notebook.add(self.add_tab, text="Добавить маршрут")
        self.create_add_tab()

        # Вкладка поиска
        self.search_tab = ttk.Frame(self.notebook)
        self.notebook.add(self.search_tab, text="Поиск маршрутов")
        self.create_search_tab()

        # Вкладка просмотра
        self.view_tab = ttk.Frame(self.notebook)
        self.notebook.add(self.view_tab, text="Все маршруты")
        self.create_view_tab()

    def create_add_tab(self):
        # Поля ввода
        ttk.Label(self.add_tab, text="Начальный пункт:").grid(row=0, column=0, padx=5, pady=5, sticky='e')
        self.start_entry = ttk.Entry(self.add_tab)
        self.start_entry.grid(row=0, column=1, padx=5, pady=5)

        ttk.Label(self.add_tab, text="Конечный пункт:").grid(row=1, column=0, padx=5, pady=5, sticky='e')
        self.end_entry = ttk.Entry(self.add_tab)
        self.end_entry.grid(row=1, column=1, padx=5, pady=5)

        ttk.Label(self.add_tab, text="Номер маршрута:").grid(row=2, column=0, padx=5, pady=5, sticky='e')
        self.num_entry = ttk.Entry(self.add_tab)
        self.num_entry.grid(row=2, column=1, padx=5, pady=5)

        # Кнопка добавления
        ttk.Button(self.add_tab, text="Добавить", command=self.add_route).grid(row=3, columnspan=2, pady=10)

    def create_search_tab(self):
        # Поле поиска
        ttk.Label(self.search_tab, text="Пункт для поиска:").pack(pady=5)
        self.search_entry = ttk.Entry(self.search_tab)
        self.search_entry.pack(pady=5)

        # Кнопка поиска
        ttk.Button(self.search_tab, text="Найти", command=self.search_routes).pack(pady=10)

        # Результаты поиска
        self.search_results = tk.Text(self.search_tab, height=10, state='disabled')
        self.search_results.pack(fill='both', expand=True, padx=5, pady=5)

    def create_view_tab(self):
        # Таблица маршрутов
        self.tree = ttk.Treeview(self.view_tab, columns=('start', 'end', 'num'), show='headings')
        self.tree.heading('start', text='Начальный пункт')
        self.tree.heading('end', text='Конечный пункт')
        self.tree.heading('num', text='Номер')
        self.tree.pack(fill='both', expand=True, padx=5, pady=5)

        # Кнопка обновления
        ttk.Button(self.view_tab, text="Обновить", command=self.update_table).pack(pady=10)
```

```
    def add_route(self):
        try:
            start = self.start_entry.get().strip()
            end = self.end_entry.get().strip()
            num = int(self.num_entry.get())

            if not start or not end:
                messagebox.showwarning("Ошибка", "Названия пунктов не могут быть пустыми")
                return

            new_route = pd.DataFrame({'start_point': [start],'end_point': [end],'route_number': [num],'created_at': [datetime.now()]})

            self.df = pd.concat([self.df, new_route], ignore_index=True)
            self.df.sort_values('route_number', inplace=True)

            # Очистка полей
            self.start_entry.delete(0, 'end')
            self.end_entry.delete(0, 'end')
            self.num_entry.delete(0, 'end')

            messagebox.showinfo("Успех", f"Маршрут {num} добавлен: {start} → {end}")
            self.update_table()

        except ValueError:
            messagebox.showerror("Ошибка", "Номер маршрута должен быть числом")

    def search_routes(self):
        point = self.search_entry.get().strip()
        if not point:
            messagebox.showwarning("Ошибка", "Введите пункт для поиска")
            return

        mask = (self.df['start_point'].str.lower() == point.lower()) | \
               (self.df['end_point'].str.lower() == point.lower())
        results = self.df[mask]

        self.search_results.config(state='normal')
        self.search_results.delete(1.0, 'end')

        if not results.empty:
            text = "Найденные маршруты:\n\n"
```

```
            for _, row in results.iterrows():
                text += f"{row['route_number']}: {row['start_point']} → {row['end_point']}\n"
        else:
            text = f"Маршруты через пункт '{point}' не найдены."

        self.search_results.insert('end', text)
        self.search_results.config(state='disabled')

    def update_table(self):
        # Очистка таблицы
        for item in self.tree.get_children():
            self.tree.delete(item)

        # Заполнение данными
        for _, row in self.df.iterrows():
            self.tree.insert('', 'end', values=(row['start_point'], row['end_point'], row['route_number']))

if __name__ == "__main__":
    root = tk.Tk()
    app = RouteManagerApp(root)
    root.mainloop()
```

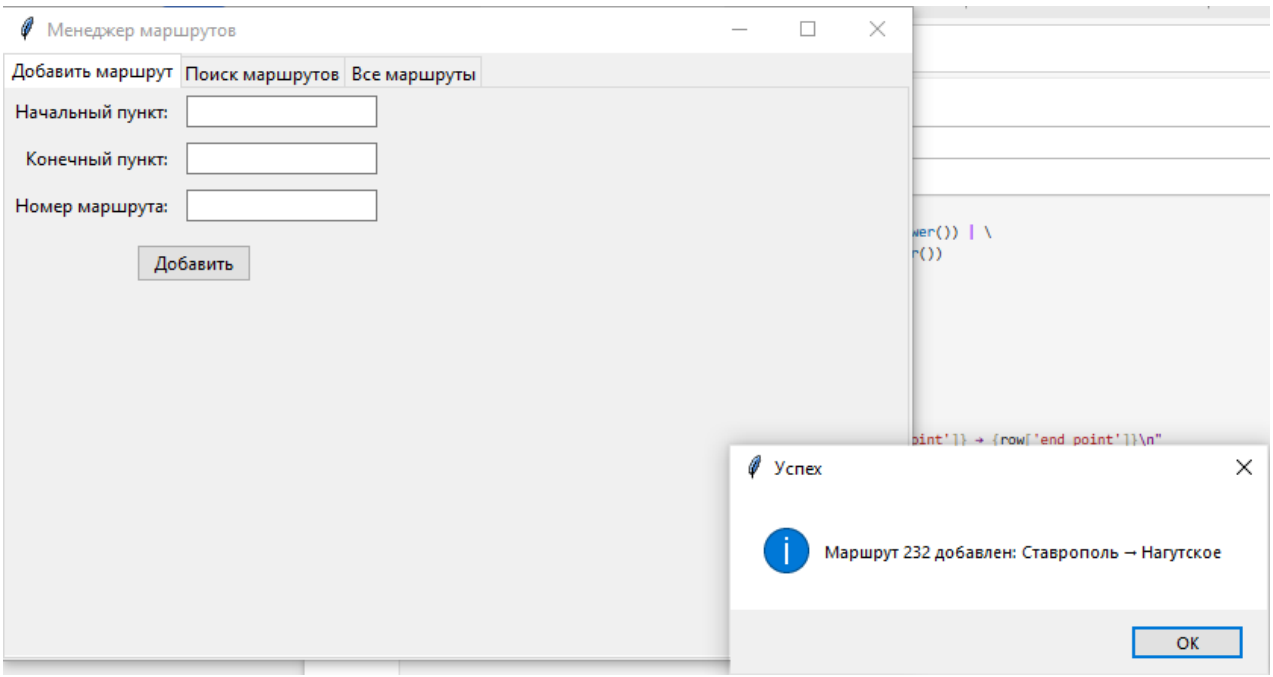

Рисунок 18 – Индивидуальное задание

https://github.com/GoncharovSerafim/

Ответы на контрольные вопросы:

1. Как создать DataFrame из словаря списков?

```
import pandas as pd
data = {
    'Имя': ['Анна', 'Борис', 'Мария'],
    'Возраст': [25, 30, 35],
    'Город': ['Москва', 'СПб', 'Казань']
}
df = pd.DataFrame(data)
```

2. Отличие создания DataFrame из списка словарей и словаря списков

Словарь списков (как выше) — ключи становятся столбцами, списки — значениями.

Список словарей — каждый словарь представляет строку:

```
data = [
    {'Имя': 'Анна', 'Возраст': 25},
```

```
    {'Имя': 'Борис', 'Возраст': 30}
]
df = pd.DataFrame(data)
```

3. Создание DataFrame из массива NumPy

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

4. Загрузка из CSV с разделителем ;

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

5. Загрузка из Excel с выбором листа

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

6. Чтение JSON vs Parquet

JSON — текстовый формат, медленнее для больших данных.

Parquet — бинарный, оптимизирован для скорости и сжатия.

```
df_json = pd.read_json('data.json')
df_parquet = pd.read_parquet('data.parquet')
```

7. Проверка типов данных

```
df.dtypes
```

8. Размер DataFrame

```
df.shape  # (строки, столбцы)
```

9. Разница между .loc[] и .iloc[]

.loc[] — доступ по меткам (индексы и названия столбцов).

.iloc[] — доступ по позициям (как в NumPy).

10. Получение элемента (3-я строка, 2-й столбец)

```
df.iloc[2, 1]  # Индексация с 0
```

11. Получение строки с индексом "Мария"

```
df.loc['Мария']  # Если 'Мария' — индекс
# Или фильтрация:
df[df['Имя'] == 'Мария']
```

12. Отличие .at[] от .loc[]

.at[] — быстрый доступ к одному элементу (только скаляр).

.loc[] — поддерживает срезы и массивы.

13. Когда .iat[] быстрее .iloc[]

.iat[] — аналог .at[] для позиций, быстрее для точечного доступа.

14. Фильтрация с .isin()

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

15. Фильтрация с .between()

df[df['Возраст'].between(25, 35)]

16. Разница между .query() и .loc[]

.query() — запросы в виде строк (удобно для сложных условий).

.loc[] — классическая фильтрация.

17. Использование переменных в .query()

min_age = 25

df.query('Возраст > @min_age')

18. Количество пропусков в столбцах

df.isna().sum()

19. Разница .isna() и .notna()

.isna() — True для NaN.

.notna() — True для не-NaN.

20. Строки без пропусков

df.dropna()

21. Добавление столбца с фиксированным значением

df['Категория'] = 'Неизвестно'

22. Добавление строки через .loc[]

df.loc[3] = ['Иван', 40, 'Владивосток']

23. Удаление столбца

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

24. Удаление строк с NaN

df.dropna(how='any') # Любой NaN

25. Удаление столбцов с NaN

df.dropna(axis=1)

26. Количество непустых значений

df.count()

27. Разница .value_counts() и .nunique()

.value_counts() — частоты всех значений.

.nunique() — число уникальных значений.

28. Частота значений в столбце

df['Город'].value_counts()

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

display() выводит красивые таблицы с пагинацией.

print() — простой текст.

30. Изменение максимального числа строк в Jupyter

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

Вывод: в ходе лабораторной работы мы исследовали основы работы с библиотекой pandas, в частности, со структурой данных DataFrame