Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по домашнему заданию на тему «Введение в Data Analysis на Python с помощью Pandas»

Выполнил:
студент группы ИУ5-35Б
Хрипков Т.А.
Подпись и дата:

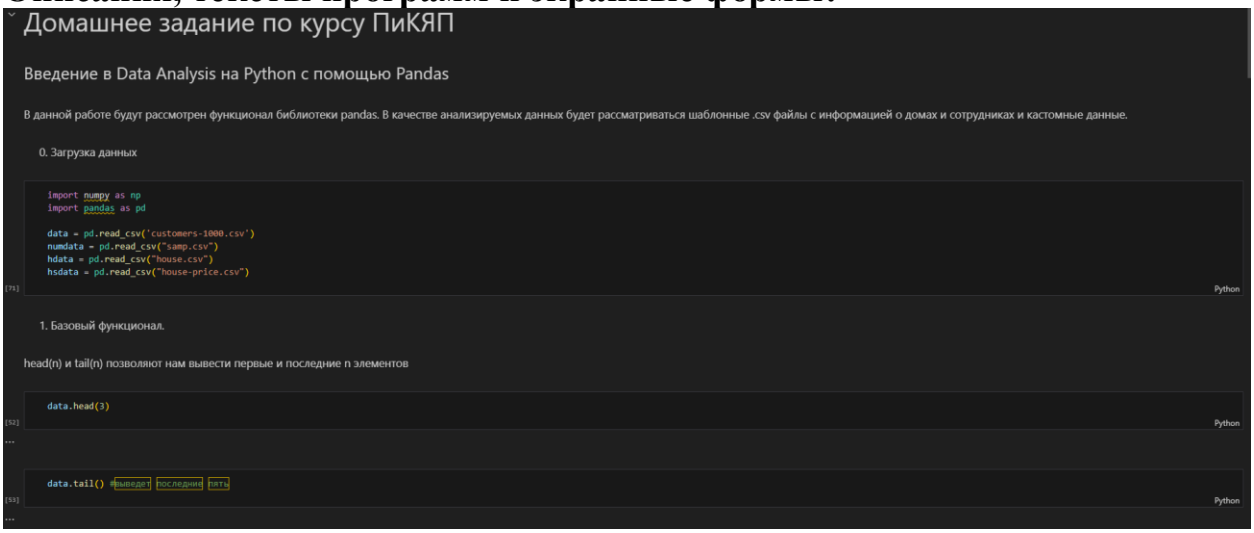
Проверил:
Подпись и дата:

Москва, 2024 г

Задание:

1. Выберите язык программирования (который Вы ранее не изучали) и (1) напишите по нему реферат с примерами кода или (2) реализуйте на нем небольшой проект (с детальным текстовым описанием).
2. Реферат (проект) может быть посвящен отдельному аспекту (аспектам) языка или содержать решение какой-либо задачи на этом языке.
3. Необходимо установить на свой компьютер компилятор (интерпретатор, транспилятор) этого языка и произвольную среду разработки.
4. В случае написания реферата необходимо разработать и откомпилировать примеры кода (или модифицировать стандартные примеры).
5. В случае создания проекта необходимо детально комментировать код.
6. При написании реферата (создании проекта) необходимо изучить и корректно использовать особенности парадигмы языка и основных конструкций данного языка.
7. Приветствуется написание черновика статьи по результатам выполнения ДЗ. Черновик статьи может быть подготовлен группой студентов, которые исследовали один и тот же аспект в нескольких языках или решили одинаковую задачу на нескольких языках.

Описания, тексты программ и экранные формы:



info() отображает количество строк со столбцами, значение каждого стобца и количество non-null(ненулевые значения), типы данных, которые хранятся в каждом столбце и объем памяти

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Index             1000 non-null   int64 
 1   Customer Id       1000 non-null   object
 2   First Name        1000 non-null   object
 3   Last Name         1000 non-null   object
 4   Company           1000 non-null   object
 5   City              1000 non-null   object
 6   Country           1000 non-null   object
 7   Phone 1           1000 non-null   object
 8   Phone 2           1000 non-null   object
 9   Email             1000 non-null   object
 10  Subscription Date  1000 non-null   object
 11  Website           1000 non-null   object
dtypes: int64(1), object(11)
memory usage: 93.9+ KB
```

Также можно использовать shape, чтобы просто отобразить кол-во строк и столбцов,

```
data.shape
```

```
(1000, 12)
```

а columns для названий столбцов

```
data.columns
```

```
Index(['Index', 'Customer Id', 'First Name', 'Last Name', 'Company', 'City',
       'Country', 'Phone 1', 'Phone 2', 'Email', 'Subscription Date',
       'Website'],
      dtype='object')
```

2. Уникальные типы данных.

В Pandas существуют три уникальных типа данных:

- Series - выражается одномерным массивом неизменного размера. Напоминает структуру с однородными данными.

Каждому элементу в Series соответствует метка, доступ к которой можно получить через атрибут index. Если рассматривать простейший пример работы с этим типом, то это напоминает dict, где index - ключ словаря. Вот пример вне нашей таблицы о сотрудниках:

+ Code + Markdown

```
series = pd.Series([1, 2, 3, 4,5,6,7], index=["a", "b", "c", "e", "c","d", "e"], dtype=np.uint8)
series
```

```
a    1
b    2
c    3
e    4
c    5
d    6
e    7
dtype: uint8
```

- DataFrames - двумерная табличная структура. Поддерживает изменение размера. Столбцы в ней неоднородно типизированны.

Каждый столбец в DataFrame является объектом типа Series. Вместе они формируют двумерную таблицу с общим индексом. В DataFrame присутствуют две оси индексации: index для строк и columns для столбцов. Метки столбцов — это их названия. То есть, вся наша таблица это объект типа DataFrame.

```
print(type(data)) #Тип нашей таблицы
print(type(data["Last Name"])) #Тип одной строки
```

Каждый столбец в DataFrame является объектом типа Series. Вместе они формируют двумерную таблицу с общим индексом. В DataFrame присутствуют две оси индексации: index для строк и columns для столбцов. Метки столбцов — это их названия. То есть, вся наша таблица это объект типа DataFrame.

```
print(type(data)) #Тип нашей таблицы
print(type(data["Last Name"])) #Тип одной строки
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

Представим, что наши сотрудники являются выпускниками ВУЗа "АБВГД" и они заранее отсортированы по самым успешным компаниям. Тогда мы можем получить топ 5 самых успешных компаний, в которых работают выпускники "АБВГД" как:

```
data["Company"].head()
```

```
0                 Stewart-Flynn
1    Terry, Proctor and Lawrence
2                  Bailey Group
3                  Moss-Maxwell
4               Mccarthy-Kelley
Name: Company, dtype: object
```

- Panel - трехмерный массив, который может меняться в размерах. Его в рамках темы рассматривать не нужно, но не будет лишним упомянуть.

3. Обработка данных.

В датасетах часто важно, чтобы не было повторяющихся значений. Для этого используется метод drop_duplicates(), рассмотрим его вне датасета, ибо в нём нет повторений.

```
dataframe = pd.DataFrame([[1, "Ivan", 9], [2, "Sergey", np.nan], [3, "Dmitry", 4.3], [4, "Dima", 5.4], [5, "Dimchik", np.nan]], columns=["#", "Name", "Score"])
ddataframe = pd.concat([dataframe, dataframe]) #С помощью concat дублируем созданный датафрейм и закидываем в ddataframe
ddataframe.shape
```

```
(10, 3)
```

```
ddataframe.drop_duplicates(inplace=True)
ddataframe.shape
ddataframe.info()
```

Python

```
<class 'pandas.core.frame.DataFrame'>
Index: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   #       5 non-null      int64
 1   Name    5 non-null      object
 2   Score   3 non-null      float64
dtypes: float64(1), int64(1), object(1)
memory usage: 160.0+ bytes
```

Видим, что при использовании concat() наши элементы удвоились, поэтому drop_duplicates сработал.

4. Статистические данные.

Если в датафрейме достаточно много числовых данных, то может быть полезно знать, например, если важна цена или количество парковок у дома(пример таблицы домов ниже)

```
hsdata.describe()
```

Python

+ Code + Markdown

5. Извлечение информации.

Мы можем извлечь значения только лишь столбцов/строк. Например, узнать имена участников игры из нашего самодельного датасета.

```
dataframe['Name'].tolist()
```

Python

```
['Ivan', 'Sergey', 'Dmitry', 'Dima', 'Dimchik']
```

Или также, в случаях, когда у таблицы более информативные индексы, вывести и их(в данном случае у нас действительно будут просто индексы)

```
dataframe.index.tolist()
```

Python

```
[0, 1, 2, 3, 4]
```

6. Обработка нулевых значений

Иногда нулевые значения в таблицах могут быть недопустимыми, и тогда их следует заменить либо убрать строки с ними. В данном случае чаще всего предусматривается значение вместо нуля(например, минимум за победу в игре всегда 1 балл, несмотря на результат).

Первым делом проверим какие конкретно строки содержат нулевые значения, посчитаем их общее количество

```
dataframe.isnull()
```

Python

...

А теперь посчитаем общее количество num ячеек

```
dataframe.isnull().sum()
```

Python

```
#       0
Name    0
Score   2
dtype: int64
```

Теперь можно заменить нулевые значения на единицу, как и условилось. Для этого создадим переменную типа series со всеми значениями баллов, заменим её и присвоим обратно в dataframe.

```
score = dataframe["Score"]
score.fillna(1, inplace=True)
```

Теперь можно заменить нулевые значения на единицу, как и условилось. Для этого создадим переменную типа series со всеми значениями баллов, заменим её и присвоим обратно в dataframe.

```
score = dataframe["Score"]
score.fillna(1, inplace=True)
```

Python

```
dataframe.isnull().sum()
```

Python

```
#       0
Name    0
Score   0
dtype: int64
```

Как видим, нулевых значений ячеек с баллами больше не осталось.

6. Выбор подходящего варианта

В случае, когда нам необходимо выбрать строки удовлетворяющие каким-то условиям, то мы можем воспользоваться булевым условием, например, нам нужны дома обязательно с кондиционером.

```
hsdata[hsdata["airconditioning"] == "yes"].head()
```

Python

И можно сделать проверку также и по наличии подвала

```
hsdata[(hsdata["airconditioning"] == "yes") & (hsdata["basement"] == "yes")].head()
```

Python

7. Присоединение к датафрейму нового столбца

7. Присоединение к датафрейму нового столбца

К примеру, нам нужно, чтобы в нашей игре мы могли знать кто прошёл, а кто нет. Тут мы просто передаём dataframe новое название и задаём ему начальное значение(в будущем можно, например, всем у кого 3 и меньше баллов ставить False)

```
[146] dataframe['Passed'] = True
```
Python

```
[147] dataframe.head
```
Python

```
... <bound method NDFrame.head of    #     Name  Score  Passed
    0  1     Ivan    9.0    True
    1  2   Sergey    1.0    True
    2  3   Dmitry    4.3    True
    3  4     Dima    5.4    True
    4  5  Dimchik    1.0    True>
```