Контрольное задание

In [89]:
```
import pandas as pd
import numpy as np
```

Task 1.

Задание:

1. Создайте Series из последовательности 15 значений, равномерно разбивающих отрезок [0, 20] (воспользуйтесь функцией linspace)
2. Определите отношение элементов полученной серии к их предыдущим элементам (*).
3. В результате необходимо получить среднее полученного вектора, оставив в нём только те значения, которые не более чем 1.5 (**).

Выберите из ответов тот, который максимально близок к полученному (с точки зрения абсолютной разницы).

Варианты ответов:

- 1) 1.24
- 2) 1.18
- 3) 0.71
- 4) 1.13

Пояснения:

(*) Если было бы необходимо найти последовательность из 3-х значений, равномерно разбивающих отрезок [0,1], то это были бы значения [0, 0.5, 1].

(**) Если был бы дан список элементов a = [1,2,3,12], отношения элементов к предыдущим будут равны [NaN, 2, 1.5, 4].
А на последнем этапе в таком примере останется только [1.5] и среднее значение будет также 1.5.

In [90]:
```
### Type your code here

# Создаём Series с 15 значениям (равномерно разделяющие отрезо [0, 20])
series = pd.Series(np.linspace(0, 20, num=15))

# Определяем отношение значений из серии с их предыдущими элементами
ratio = series / series.shift(1)

# Отфильтруем полученную серию на значения,
# не больше чем 1.5 и получим среднее арифметическое
ratio[ratio <= 1.5].mean()

# 1.18167... ~ 1.81 второго варианта
# Ответ: 2
```

Out[90]: np.float64(1.181677812927813)

Task 2.

Выберите все верные ответы касательно следующих 3-х Series:

- pd.Series('abcde'); (1)
- pd.Series(['abcde']); (2)
- pd.Series(list('abcde')); (3)
- pd.Series("abcde"); (4)

Пояснения:

- функция list: в строке каждый символ - это отдельный элемент для list
- квадратные скобки: в квадратных скобках списку передается множество элементов по отдельности через запятую

Вопросы:

```
- 1) Серия (1) совпадает с серией (2), так как в каждом из
случаев серия создаётся из списка строк
- 2) Серия (2) совпадает с серией (3), так как в каждом из
случаев серия создаётся из списка символов
- 3) Серия (1) не совпадает с серией (4), так как в (4)
используются двойные кавычки "" вместо одинарных ''
```

In [91]:
```
### Type your code here

first = pd.Series('abcde')
second = pd.Series(['abcde'])
third = pd.Series(list('abcde'))  # Единственный, где буквы - отдельные э
fourth = pd.Series("abcde")

# Первый вопрос правильный, так как функции подаются списком с единствен
# Второй вопрос не правильный, так как в третьей Series буквы расположены
# так как функция list разделяет строку на отдельные буквы
# Третий вопрос  не правильный, так как кавычки не влияют на строку
# Ответ: 1
```

Task 3.

По клиенту получены зашумленные данные (объект s типа Series) по его транзакциям.

Для заданного ниже объекта s проделайте следующее:

1. Создайте новый Series, значения которого совпадают со значениями s, а индексы - целочисленные значения от 2 до 12, не включая 12.
2. Выберите из новой серии элементы с индексами 3 и 5, после чего просуммируйте их, сохранив результат (1).
3. Выберите из s только целочисленные элементы и вычислите их дисперсию (2). (*)

Все полученные результаты округлите до 2-х знаков после запятой.

Выберите все верные пункты:

```
- 1) Ответ (1) - 642.52
- 2) Ответ (1) - 91.78
- 3) Ответ (1) - не может быть определён (укажите причину)
- 4) Ответ (2) - 57591.19
- 5) Ответ (2) - 210.12
- 6) Ответ (2) - не может быть определён
```

Пояснения:

(*) Целочисленные значения - значения, имеющие тип int.

- Дисперсия рассчитывается с помощью функции из библиотеки numpy: np.var(, ddof=0) или встроенной в python функции: .var(ddof=0)

In [92]:
```
s = pd.Series(data=['1', 2, 3.1, 'hi!', 5, -512, 12.42, 'sber', 10.10, 98
              index=range(6, 26, 2))

### Type your code here

# Создаём новый s, меняем индексы...
sn = s
sn.index = [i for i in range(2, 12)]

# Пытаемся сложить элементы с индексами 3 и 5
try:
    value = sn[3] + sn[5]
except TypeError:
    print("Элементы не могут быть суммированы, т.к. они разного типа (str

# Вычисляем дисперсию только для целых чисел
s[s.apply(lambda x: type(x) == int)].var(ddof=0)

# Ответ: 3, 4
```

```
Элементы не могут быть суммированы, т.к. они разного типа (str и int)
```

Out[92]:
```
np.float64(57591.1875)
```

Task 4.

1. Сгенерируйте Series из 100 значений нормально распределённой СВ (np.random.normal с дефолтными параметрами - нулевым средним и единичной дисперсией).
2. Возведите каждое значение серии в 3 степень, а значения индекса увеличьте в 3 раза.
3. Ответьте на следующие вопросы через запятую (без пробелов) (*)
    - A. Выведите сумму элементов, строго меньших 2.6, имеющих нечётные значения индекса.
    - B. Выведите количество значений серии меньше нуля.

Пояснения:

(*) Если получились ответы 4.32 и 3, то необходимо вывести их в виде "4.32 , 3". То есть вещественные числа необходимо разделять точками. Не забудьте про фиксированный seed (его менять не нужно)!

- Определенное значение seed нужно, чтобы ответы у всех выполняющих это задание были одинаковые и их можно было проверить (так как генерируются одинаковые series).
- Следует внимательнее использовать [ ] для выбора данных по нескольким условиям: либо выбирать данные последовательно, либо сразу по нескольким условиям, но через оператор &. Отличие оператора and от оператора &: and - выводит последнее проверенное значение, & - выводит пересечение значений. Пример: s[_ & _].sum()

In [93]:
```
np.random.seed(242)

### Type your code here

# Создаём Series со 100 случайными нормально-распредилёными числами
series = pd.Series(np.random.normal(size=100))
# Увеличиваем значения на 3 степени и их индексы в 3 раза
series = series ** 3
series.index = series.index * 3

# Фильтруем серию по условиям: чётный индекс и значение строго меньше 2.6
ns = series[(series.index % 2 == 0) & (series < 2.6)]
# Получаем сумму Series и количество значений, меньше нуля
sum = ns.sum()
count = ns[ns < 0].count()
# Принтуем ответ...
print(f"{sum}, {count}")

# Ответ: -19.455490619511657, 23
```

```
-19.455490619511657, 23
```

Информация для последующих заданий

- Для всех последующих заданий будем использовать обезличенные транзакционные банковские данные. Для этого считайте в переменные tr_mcc_codes, tr_types, transactions и gender_train из одноимённых таблиц из папки data. Для таблицы transactions используйте только первые n=1000000 строк. Обратите внимание на разделители внутри каждого из файлов - они могут различаться!

In [94]:
```
### Type your code here

# transactions = pd.read_csv('data/transactions.csv', sep=',')
# 1) Кол-во транзакций ограничено заданием на 1000000,
# однако всего их будет...
# print(len(transactions))
# Ответ: 6849346

# 2) А клиентов...
# print(transactions["customer_id"].nunique())
```

```
# Ответ: 15000

# 3) У какого (у каких) клиентов наибольшее число
# приходных транзакций, величина которых
# выше среднего значения
# всех приходных транзакций?
'''
income = transactions[transactions["amount"] > 0]
mean_income = income["amount"].mean()
only_larger = income[income["amount"] > mean_income]
biggest_customer = {
    "customer_id": 0,
    "count": 0
}
for customer in income["customer_id"].unique():
    # Старый способ, время выполнения 1 минута
    # count = income[income["customer_id"] == customer]
    # count = len(count[count["amount"] > mean_income])

    # Новый способ, время выполнения: 20 секунд
    count = len(only_larger[only_larger["customer_id"] == customer])
    if count > biggest_customer["count"]:
        biggest_customer["customer_id"] = customer
        biggest_customer["count"] = count
    elif count == biggest_customer["count"]:
        print("Одинаковое значение! Оставлю предыдущее...")
print(f"{biggest_customer["customer_id"]}, ({biggest_customer["count"]})"
'''
# Ответ: 84219086, (2510)

# А это переменные для следующих заданий:

tr_mcc_codes = pd.read_csv('data/tr_mcc_codes.csv', sep=';')
tr_types = pd.read_csv('data/tr_types.csv', sep=';')
transactions = pd.read_csv('data/transactions.csv', sep=',', nrows=100000
gender_train = pd.read_csv('data/gender_train.csv', sep=',')
```

Описание данных

Таблица `transactions.csv`

Описание

Таблица содержит историю транзакций клиентов банка за один год и три месяца.

Формат данных

```
customer_id,tr_datetime,mcc_code,tr_type,amount,term_id
111111,15 01:40:52,1111,1000,-5224,111111
111112,15 15:18:32,3333,2000,-100,11122233
...
```

Описание полей

- `customer_id` — идентификатор клиента;

- `tr_datetime` — день и время совершения транзакции (дни нумеруются с начала данных);
- `mcc_code` — mcc-код транзакции;
- `tr_type` — тип транзакции;
- `amount` — сумма транзакции в условных единицах со знаком; `+` — начисление средств клиенту (приходная транзакция), `-` — списание средств (расходная транзакция);
- `term_id` — идентификатор терминала;

Таблица `gender_train.csv`

Описание

Данная таблица содержит информацию по полу для части клиентов, для которых он известен. Для остальных клиентов пол неизвестен.

Формат данных

```
customer_id,gender
111111,0
111112,1
...
```

Описание полей

- `customer_id` — идентификатор клиента;
- `gender` — пол клиента;

Таблица `tr_mcc_codes.csv`

Описание

Данная таблица содержит описание mcc-кодов транзакций.

Формат данных

```
mcc_code;mcc_description
1000;словесное описание mcc-кода 1000
2000;словесное описание mcc-кода 2000
...
```

Описание полей

- `mcc_code` – mcc-код транзакции;
- `mcc_description` — описание mcc-кода транзакции.

Таблица `tr_types.csv`

Описание

Данная таблица содержит описание типов транзакций.

Формат данных

```
tr_type;tr_description
1000;словесное описание типа транзакции 1000
2000;словесное описание типа транзакции 2000
...
```

Описание полей

- `tr_type` – тип транзакции;
- `tr_description` — описание типа транзакции;

Task 5.

1. В tr_types выберите произвольные 100 строк с помощью метода sample (указав при этом random_seed равный 242)
2. В полученной на предыдущем этапе подвыборке найдите долю наблюдений (стобец tr_description), в которой содержится подстрока 'плата' (в любом регистре). (*)

Выведите ответ в виде вещественного числа, округлённого до двух знаков после запятой, отделив дробную часть точкой в формате "123.45"

Пояснения:

(*) Строки "ПлатА за аренду", "ПлатАза аренду", "ПЛАТА" удовлетворяют условию, так как будучи переведёнными в нижний регистр содержат подстроку "плата".

In [95]:
```python
### Type your code here

np.random.seed(242)

# Выбераем 100 произвольных строк
tr_types_sample = tr_types.sample(100)

# Ищем в полученной подвыборке строки, содержащие слово плата
tr_types_plata = tr_types_sample[\
    tr_types_sample["tr_description"]\
    .str.contains("плата", False)\
]

# Ищем пропорцию
proportion = len(tr_types_plata) / len(tr_types_sample)

print(f"Ответ: {proportion:.2f}")
# Ответ: 0.26
```

```
Ответ: 0.26
```

Task 6.

1. Для поля tr_type датафрейма transactions посчитайте частоту встречаемости всех типов транзакций tr_type в transactions.
2. Из перечисленных вариантов выберите те, которые попали в топ-5 транзакций по частоте встречаемости.

Выберите все верные пункты:

- 1) Выдача наличных в АТМ Сбербанк России
- 2) Комиссия за обслуживание ссудного счета
- 3) Списание по требованию
- 4) Оплата услуги. Банкоматы СБ РФ
- 5) Погашение кредита (в пределах одного филиала)
- 6) Покупка. POS ТУ СБ РФ

In [96]:
```
### Type your code here

# Ищем частоту типов транзакций, через функцию value_counts()
# Флаг sort=True сразу сортирует полученные частоты
tr_type_freq = transactions["tr_type"].value_counts(sort=True)

# Соединяем полученные частоты и типы транзакций через "tr_type"
tr_count_desc = pd.merge(
    tr_types,
    tr_type_freq.head(5),
    on="tr_type",
    how="right"
)

print(tr_count_desc)
# Ответ: 6, 1, 4
```

```
  tr_type                                  tr_description   count
0    1010                              Покупка. POS ТУ СБ РФ  231117
1    2010                Выдача наличных в АТМ Сбербанк России  151166
2    7070  Перевод на карту (с карты) через Мобильный бан...  149006
3    1110                             Покупка. POS ТУ Россия  137658
4    1030                   Оплата услуги. Банкоматы СБ РФ  118975
```

Task 7.

1. В датафрейме transactions задайте столбец customer_id в качестве индекса.
2. Выделите клиента с максимальной суммой транзакции (то есть с максимальным приходом на карту). (*)
3. Найдите у него наиболее часто встречающийся модуль суммы приходов/расходов. (**)

Выберите все верные пункты:

- 1) 1122957.89
- 2) 15721.41
- 3) 22459.16
- 4) 13475494.63
- 5) 107407.78
- 6) 65019.26

Пояснения:

(*) Если у клиента были транзакции [-10000, 10, 0, -10], то максимумом будет являться значение 10.
(**) Если у клиента были транзакции [-10000, 10, 0, -10], то наиболее встречающийся модуль суммы транзакций равен 10, и встретился он 2 раза.

In [97]:
```python
### Type your code here

# Поменял индекс DataFrame на customer_id
transactions_cid = transactions.set_index("customer_id")

# Нашёл максимальную сумму транзакции
max_transactions_cid = transactions_cid\
    [transactions_cid["amount"] > 0]\
    ["amount"].max()

# Нашёл клиента, совершивший транзакцию на данную сумму
customer_max = transactions_cid\
    [transactions_cid["amount"] == max_transactions_cid]

# Нашёл все транзакции, которые он совершил
customer_max_transactions = transactions_cid\
    .loc[customer_max.index]

# Нашёл модули всех полученнх транзакции
abs_customer_max_transactions = customer_max_transactions\
    ["amount"].abs()

# Нашёл наиболее часто встречающейся модуль (мода)
mode = abs_customer_max_transactions.mode().iloc[0]

print(f"Ответ: {mode:.2f}")
# Ответ: 3
```

```
Ответ: 22459.16
```

Task 8.

1. Найдите максимальную разницу между медианами суммы транзакций, посчитанными при заданных ниже условиях по полю amount из таблицы transactions (*):
    - Медиана суммы транзакций
    - Медиана суммы транзакций по тем строкам, которые ни в одном из своих столбцов не содержат пустые значения
    - Медиана суммы транзакций по строкам, отсортированным по полю amount в порядке возрастания, и из которых удалены дублирующиеся по столбцам [mcc_code, tr_type] строки, причём при удалении соответстующих дублей остаются только последние из дублирующихся строк (keep='last')

Выведите ответ в виде вещественного числа, округлённого до двух знаков после запятой, отделив дробную часть точкой в формате "123.45"

Пояснения:

(*) Для вычисления максимальной разницы между значениями списка можно использовать функцию np.ptp
(**) Если в результате получились значения [1,3,5], то макимальная разница между ними 4 == 5-1.

In [98]:
```python
### Type your code here

# Ищем медиану суммы транзакций
std_sum = transactions["amount"].median()

# Ищем то же самое, но где нет строк с пустыми данными
nona_sum = transactions.dropna()["amount"].median()

# Затем медиану, где мы убрали дубликатов между "mcc_code" и "tr_type"
# Предварительно отсортировав таблицу
sort_sum = transactions.sort_values("amount")\
    .drop_duplicates(subset=["mcc_code", "tr_type"], keep='last')\
    ["amount"].median()

# Ищем максимальную разницу
max_median_diff = np.ptp(np.array([std_sum, nona_sum, sort_sum]))

print(f"Ответ: {max_median_diff:.2f}")
# Ответ: 4693.96
```

```
Ответ: 4693.96
```

Творческое задание:

Определите топ-5 MCC-кодов по объему расходных операций (транзакций со знаком -) среди мужчин. (*) В ответе выведите таблицу с описанием MCC-кода транзакции и объём операции.

(*) - gender_train["gender"] == 0, если нет информации о поле клиента - не считаем его в расчётах

In [130…
```python
### Мой собственный код решения

# Фильтруем транзакции на расходные операции
transactions_expence = transactions[transactions["amount"] < 0]

# Фильтруем полученные транзакции на клиентов мужского пола
# Это можно сделать функцией merge
transactions_expence_men = pd.merge(
    transactions_expence,
    gender_train[gender_train["gender"] == 0], # здесь идёт фильтрация на
    how='inner', on='customer_id'
)
# Мы получили таблицу расходных операций, сделанные мужчинами

# Теперь ищем общий объём расходных операций по каждому MCC-коду
# Для этого группируем таблицу относительно mcc_code
transactions_grouped_mcc = transactions_expence_men.groupby("mcc_code")

# И потом суммируем все транзакции, сортируем и берём первые 5
transactions_grouped_mcc = transactions_grouped_mcc["amount"]\
    .apply(lambda x: abs(x).sum())\
    .sort_values(ascending=False)\
    .head(5)\
    .reset_index(name='total_expense')

# Наконец, выводим таблицу с суммами и описанием MCC
transactions_grouped_desc = pd.merge(transactions_grouped_mcc,
```

```
    tr_mcc_codes, how="left", on="mcc_code")\
    .drop(axis=1, labels="mcc_code")

# Простая таблица, без форматирования
# print(transactions_grouped_desc)


# Более красивая, но не обязательная
print("Топ-5 MCC-кодов по расходам среди мужчин:")
print("=" * 80)
for _, row in transactions_grouped_desc.iterrows():
    print(f"{row['mcc_description'][:40]:40} | {row['total_expense']:12,.
```

```
Топ-5 MCC-кодов по расходам среди мужчин:
================================================================================
======
Финансовые институты — снятие наличности | 4,958,077,150.17 ₽
Денежные переводы                        | 3,513,863,834.98 ₽
Бакалейные магазины, супермаркеты        | 578,215,765.79 ₽
Финансовые институты — снятие наличности | 261,273,107.84 ₽
Звонки с использованием телефонов, считы | 185,845,134.88 ₽
```