Практическая работа № 1. Наборы данных. Извлечение и визуализация данных.

Тема работы: изучение методов извлечения данных из наборов с использованием средств библиотеки `pandas Python`.

Цель работы: Научиться использовать на практике методы программирования в программной среде `Python` на примере использования методов извлечения данных, сведений о данных и визуализации данных.

Методические указания по выполнению лабораторной работы

Pandas — библиотека для обработки и анализа данных, которая предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временными рядами (уровень абстракции данных для объединения и преобразования данных). Работа Pandas с данными строится поверх библиотеки NumPy.

Источниками данных для структур библиотеки обычно служат или файлы с данными или базы данных. Наиболее распространённые форматы для хранения данных – табличные файлы csv и Excel.

Для работы с данными в Pandas используются 2 структуры:

- Series (серии),
- DataFrame (фреймы данных).

Структура или объект Series - одномерный массив или список с ассоциированными метками (индексами), т.е. этот объект подобен ассоциативному массиву или словарю в Python.

Конструктор класса Series выглядит следующим образом:

```
pandas.Series(data=None, index=None, dtype=None, name=None, copy=False,
fastpath=False)
```

- data – массив, словарь или скалярное значение, на базе которого будет построен Series;
- index – список меток, который будет использоваться для доступа к элементам Series. Длина списка должна быть равна длине data;
- dtype – объект numpy.dtype, определяющий тип данных;
- copy – создает копию массива данных, если параметр равен True в ином случае ничего не делает.

Создать структуру Series можно на базе различных типов данных:

- словари Python;
- списки Python;
- массивы из numpy: ndarray;
- скалярные величины.

Объект DataFrame является табличной структурой данных, в нем есть строки и столбцы. Столбцами в объекте DataFrame выступают объекты Series, строки которых являются их непосредственными элементами.

Объект DataFrame имеет индексы по строкам и по столбцам. Если индекс по строкам явно не задан (например, колонка по которой нужно их строить), то Pandas задаёт целочисленный индекс RangeIndex от 0 до N-1, где N это количество строк в таблице.

Конструктор класса DataFrame выглядит так:

```
class pandas.DataFrame(data=None, index=None, columns=None, dtype=None,
copy=False)
```

- data – массив ndarray, словарь (dict) или другой DataFrame;
- index – список меток для записей (имена строк таблицы);
- columns – список меток для полей (имена столбцов таблицы);
- dtype – объект numpy.dtype, определяющий тип данных;
- copy – создает копию массива данных, если параметр равен True в ином случае ничего не делает.

Структуру DataFrame можно создать на базе:

- словаря (dict) в качестве элементов которого должны выступать: одномерные ndarray, списки, другие словари, структуры Series;
- двумерные ndarray;
- структуры Series;
- структурированные ndarray;
- другие DataFrame.

Загрузка данных из табличного файла

CSV (от англ. Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделённых запятыми.

С файлом можно работать через текстовый или табличный редактор (например MS Excel).

Для загрузки данных их табличных файлов можно использовать объект DataFrame. Для этого, в первую очередь, подключаем библиотеку:

```
import pandas as pd
```

Затем используем метод read_csv():

```
fix_df = pd.read_csv('c://bikes.csv',
sep=';',
encoding='latin1')
```

В методе требуется указать первым параметром путь до имени файла, далее по необходимости параметры sep - задаёт символ-разделитель полей в файле (по умолчанию разделитель запятая), names - список названий колонок, если он не задан в файле, index_col -- номер колонки с индексом, decimal - символ-разделитель для знаков после запятой, encoding – кодировку файла и т.д.

Каждая строчка набора данных является одним наблюдением или одним объектом в задачи и их требуется различать. Для этого используется индекс, по умолчанию, если индекс не указан каждая строка нумеруется. В качестве индекса может быть использован один из столбцов таблицы:

```
fix_df1 = pd.read_csv('d://bikes.csv',
sep=';',
encoding='latin1',
index_col='Date' )
```

Можно указать какой тип данных должен быть у столбца:

```
fix_df2 = pd.read_csv('d://bikes.csv',
sep=';', encoding='latin1',
dtype={'Date':str} )
```

Также можно указать какой столбец должен считываться как дата:

```
fix_df3 = pd.read_csv('d://bikes.csv',
sep=';', encoding='latin1',
parse_dates=['Date'],
dayfirst=True,
index_col='Date')
```

Экспорт (выгрузка) данных

Данные датафрейма могут быть изменены в процессе обработки и их потребуется сохранить в табличный файл. Для этого можно использовать команду:

```
dataframe.to_csv('file_name.csv')
```

Метод `.to_csv` имеет ряд входных параметров, которые могут указывать формат выходного файла csv:

- кодировка, например encoding='utf-8',
- разделитель значений sep, по умолчанию ‘,’,
- запись имен строк (индексы) по умолчанию True,

- и другие.

Вывод информации о наборе данных

Операция	Метод
Для быстрого просмотра первых 5 строк набора данных можно использовать метод head():	`fix_df.head()`
Для получения последних 5 строк – метод tail():	`fix_df.tail()`
Для получения определённого количества случайных строк из набора данных используют метод sample(количество):	`fix_df.sample(2)`
Для вывода определённого количества строк можно указать число:	`fix_df[:3]`
Для ввода данных из определённого столбца можно указать его название и количество выводимых значений:	`fix_df['Berri 1'][:10]`
Размерность фрейма данных можно получить, используя свойство фрейма данных shape:	`fix_df.shape`
Количество строк можно найти, применив функцию len() длина, к индексу фрейма данных или к фрейму:	`len(fix_df.index)` `len(fix_df)`
Для получения информации о типах столбцов фрейма данных используется dtypes:	`fix_df.dtypes`
Для получения информации о статистических оценках значений фрейма используют метод describe():	`fix_df.describe(include='all')`

Обработка пустых значений

Пустое значение для обработки данных является особым случаем, требующем обработки: исключения или заполнения. Если пропущенных значений не очень много, можно их заполнить. Если пропущенных значений в каком-то столбце много, то следует исключить столбец из рассмотрения, так как данных по нему недостаточно.

Для работы с пропечёнными значениями в библиотеке есть методы:

- isnull() — генерирует булеву маску для отсутствующих значений,
- dropna() — фильтрация данных по отсутствующим значениям,
- fillna() — замена пропусков, аргументы method=’ffill’ и method=’bfill’ определяют какими значениями будут заполняться пропуски (предыдущими или последующими в массиве).

Методы доступны как для объектов Series так и для dataFrame (с выбором столбца).

Для получения количества пустых значений в столбцах можно использовать метод isnull():

```
# Возвращает количество пропущенных значений, содержащихся в каждом
столбце
df.isnull ()df.isnull (). sum ()
```

Метод fillna() не изменяет текущую структуру, он возвращает структуру DataFrame, созданную на базе существующей, с заменой NaN значений на те, что переданы в метод в качестве аргумента. Операция по умолчанию df.fillna () (inplace = False) не является внутренней, то есть она не изменяет напрямую исходный фрейм данных, а создает копию и изменяет копию. В качестве значения для заполнения могут использоваться:

- ближайшее ненулевое значение,
- скользящее среднее,
- следующе или предыдущее значение и др.

Для удаления объектов, которые содержат значения *NaN,* используется метод *dropna()*, в зависимости от параметров метода можно удалять весть столбец с пстыми значениями, только строки с пустыми значениями, указывать ограничение на возможное число пустых значений в строке.

```
# Удалить строки с пропущенными значениями напрямую
df.dropna ()
```

```
# Прямое удаление столбцов с пропущенными значениями
df.dropna (axis = 1)
# Удалять только строки с пропущенными значениями
df.dropna (how = 'all')
# Сохранять строки с как минимум 4 пропущенными значениями
df.dropna (thresh = 4)
```

Работа с индексами набора данных

При больших объемах данных сортировка и поиск требуют достаточного времени. Для уменьшения времени для выполнения этих операций используется механизм индексов, аналогично индексам в базах данных.

Индекс можно указать сразу, при создании набора данных с помощью параметра index_col:

```
fix_df1 = pd.read_csv('d://bikes.csv',
sep=';',
encoding='latin1',
index_col='Date' )
```

Или индекс можно установить позже с помощью метода set_index():

```
df_with_index = df.set_index(['key'])
index_moved_to_col.set_index('Sector').head()
```

В случае, если индекс устанавливается для нескольких полей, он имеет структуру – иерархию (слои), и имеет значение порядок следования полей. Можно просмотреть количество уровней в индексе и изменить порядок слоев.

Пример:

```
# индексируем датафрейм data по столбцам Sector и Symbol
multi_fi = reindexed.set_index(['Sector', 'Symbol'])
len(multi_fi.index.levels)
# изменение порядка уровней индекса
multi_fi.reorder_levels([1, 0], axis=0).head()
```

Индекс может быть сброшен с помощью метода reset_index():

```
index_moved_to_col = sp500.reset_index()
```

Задание:

1. Выберите источник данных, предварительно рассмотрите ресурсы с открытыми наборами данных (Приложение А, или выбранный самостоятельно источник данных).

2. Для выбранного набора данных реализовать:

- чтение набора в таблицу записями типа DataFrame;
- получение информации о наборе данных, полях, типах данных, статистиках;
- вывести данные в консоль;
- преобразовать записи в массив;
- выбрать способ визуализации данных и представить их графически.

Пример выполнения задания:

```
# Этап 1. Получение данных
# Изучим данные, предоставленные сервисом
# Импорт библиотек
# <импорт библиотеки pandas>
importpandasaspd

# <Чтение данных из файла в таблицу записями типа DataFrame>
# Вариант 1
values_df = pd.read_csv('metal.csv', delimiter=';')
# Вариант 2
#values_df = pd.read_table('metal.csv', sep=';')
```

```
# Этап 2. Извлечение сведений о данных
# <получение общей информации о данных в таблице>
print('Общая информаця о данных в таблице')
print(values_df.info())
# <вывод данных таблицы>
print('Вывод данных таблицы')
print(values_df)
# <перечень названий столбцов таблицы values_df способ 1>
print('Вывод названий столбцов таблицы. способ 1')
print(values_df.columns)
# <перечень названий столбцов таблицы values df способ 2>
print('Вывод названий столбцов таблицы. способ 2')
print(list(values_df))
# Проверим данные на наличие пропусков
# <суммарное количество пропусков, выявленных методом isnull() в таблице
values df>
print('Количество пустых строк')
print(values_df.isnull().sum())
# <получение суммарного количества дубликатов в таблице values_df>
# для временного ряда - не совсем корректная информация
print('Количество строк-дубликатов')
print(values_df.duplicated().sum())
# Метод describe() позволяет собрать некоторую статистику по каждому
числовому признаку
print('Статистика по признакам')
print(values_df.describe())
# <получение первых 10 строк таблицы values_df>
print('1-е 10 значений таблицы')
print(values_df.head(10))

# Этап 3. Преобразование данных в массив
# <импорт библиотеки numpy>
importnumpyasnp
# <приведение типа DataFrame к типу ndArray и выбор 2-го столбца данных
('Silver')>
values_arr2 = np.array(values_df)[:, 2]
# <вывод данных массива>
print('Вывод данных массива')
print(values_arr2)
# <приведение типа DataFrame к типу ndArray и выбор 2-го и 3-го столбцов
данных ('Silver', 'Platinum')>
values_arr23 = np.array(values_df)[:, 2:4]
# <вывод данных массива>
print('Вывод данных массива')
print(values_arr23)

# Этап 4. Визуализация данных массива
# Построение графиков
# <импорт библиотеки matplotlib>
importmatplotlib
importmatplotlib.pyplotasplt
frompylabimportplot

# <задание типа графика>
matplotlib.style.use('ggplot')
importseabornassns
#sns.set(); # другой вид графиков по умолчанию

plt.figure()
x = np.linspace(values_arr2[0], values_arr2[values_arr2.size-1] ,
values_arr2.size)
plot(x, values_arr2,'g')
plt.show()
```

Результат выполнения:

```
C:\Python\Projects\venv\Scripts\python.exe "C:/Python/Projects/Practic 1.1.py"
Общая информаця о данных в таблице
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 912 entries, 0 to 911
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       912 non-null    object
 1   Gold       912 non-null    float64
 2   Silver     912 non-null    float64
 3   Platinum   912 non-null    float64
 4   Palladium  912 non-null    float64
dtypes: float64(4), object(1)
memory usage: 32.1+ KB
None
Вывод данных таблицы
                   Date     Gold  Silver  Platinum  Palladium
0    18.09.2021 0:00:00  4120.07   53.68   2213.89    4770.70
1    17.09.2021 0:00:00  4148.59   54.66   2193.70    4732.06
2    16.09.2021 0:00:00  4219.32   55.83   2190.00    4609.54
3    15.09.2021 0:00:00  4181.70   55.17   2235.04    4811.41
4    14.09.2021 0:00:00  4200.92   55.41   2241.62    5023.67
..                   ...      ...     ...       ...        ...
907  16.01.2018 0:00:00  2433.40   31.02   1802.86    2043.84
908  13.01.2018 0:00:00  2425.60   30.95   1810.69    1976.29
909  12.01.2018 0:00:00  2418.56   31.40   1788.47    1982.71
910  11.01.2018 0:00:00  2416.66   31.19   1760.87    2009.54
911  10.01.2018 0:00:00  2411.72   31.49   1764.38    2022.99

[912 rows x 5 columns]
Вывод названий столбцов таблицы. способ 1
Index(['Date', 'Gold', 'Silver', 'Platinum', 'Palladium'], dtype='object')
Вывод названий столбцов таблицы. способ 2
['Date', 'Gold', 'Silver', 'Platinum', 'Palladium']
Количество пустых строк
Date         0
Gold         0
Silver       0
Platinum     0
Palladium    0
dtype: int64
Количество строк-дубликатов
0
Статистика по признакам
GoldSilverPlatinumPalladium
count   912.000000  912.000000   912.000000   912.000000
mean   3408.429342   42.478015  2029.763575  3979.578542
std     806.406054   13.233652   377.442826  1607.828190
min    2379.100000   29.690000  1522.760000  1684.600000
25%    2653.975000   31.660000  1763.462500  2603.762500
50%    3091.840000   35.755000  1846.370000  3632.340000
75%    4243.552500   58.620000  2239.035000  5566.380000
max    4887.700000   72.200000  3095.570000  7227.220000
1-е 10 значенийтаблицы
                 Date     Gold  Silver  Platinum  Palladium
0  18.09.2021 0:00:00  4120.07   53.68   2213.89    4770.70
1  17.09.2021 0:00:00  4148.59   54.66   2193.70    4732.06
2  16.09.2021 0:00:00  4219.32   55.83   2190.00    4609.54
3  15.09.2021 0:00:00  4181.70   55.17   2235.04    4811.41
4  14.09.2021 0:00:00  4200.92   55.41   2241.62    5023.67
5  11.09.2021 0:00:00  4210.48   56.20   2308.88    5183.86
6  10.09.2021 0:00:00  4221.14   56.77   2311.18    5301.84
7  09.09.2021 0:00:00  4245.35   57.41   2363.58    5565.39
8  08.09.2021 0:00:00  4260.95   57.05   2386.08    5638.12
9  07.09.2021 0:00:00  4277.87   57.95   2413.54    5690.23
Вывод данных массива
[53.68 54.66 55.83 55.17 55.41 56.2 56.77 57.41 57.05 57.95 56.34 56.6

… … … …

30.56 29.98 29.97 30.23 30.45 30.59 30.44 30.33 29.69 30.04 30.21 30.48
 30.54 30.44 30.22 30.12 30.0 30.35 29.99 30.19 29.72 29.83 30.11 29.84
 29.86 29.99 30.18 30.38 30.5 30.48 30.17 30.75 30.51 30.51 30.57 30.95
 30.77 31.04 31.2 30.97 31.17 31.24 31.37 31.49 31.45 30.91 30.78 30.9
 31.02 31.09 31.41 31.1 31.41 31.02 30.95 31.4 31.19 31.49]
```

```
Выводданныхмассива
[[53.68 2213.89]
 [54.66 2193.7]
 [55.83 2190.0]
 ...
 [31.4 1788.47]
 [31.19 1760.87]
 [31.49 1764.38]]

Process finished with exit code 0
```

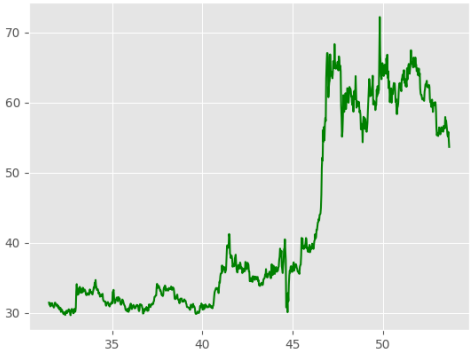

Рис. Пример визуализации