УДК 62-9

Ким Е. Э.
студент
4 курс, факультет «Математический»
Тверской Государственный Университет
Россия, г. Тверь

РЕАЛИЗАЦИЯ АЛГОРИТМА ПОСТРОЕНИЯ ПРОФИЛЯ ПОТРЕБЛЕНИЯ ЭЛЕКТРОЭНЕРГИИ НА PYTHON

***Аннотация**: Статья посвящена реализации алгоритма построения профиля потребления электроэнергии. Имея исходные данные, программа строит профиль потребления с заданной частотой на заданной диапазоне. Алгоритм реализован на Python.*

***Ключевые слова**: Электроэнергетика, анализ данных электропотребления, алгоритмы, Python.*

***Annotation**: The article is devoted to the implementation of an algorithm for constructing an electricity consumption profile. Having the initial data, the program builds a consumption profile with a given frequency on a given range. The algorithm is implemented in Python.*

***Key words**: power industry, analysis of power consumption data, algorithms, Python.*

Профиль потребления позволяет понимать характер потребления в целом. Чтобы не каждый день отслеживать, а знать, что в целом на заводе/предприятии/офисе такое распределение во времени на протяжении суток/месяца/года. Тогда можно и нагрузку балансировать и понимать, как в целом работы идут или понимать время работы объекта, т.к. часы открытия и

закрытия коррелируют с профилем потребления энергии, и тд. Также опираясь на профиль потребления можно подбирать тариф электропотребления, который является более выгодным для объекта.

Для реализации алгоритма будем использовать Python библиотеку pandas. Наши данные хранятся в файлах Excel. Нам нужно их считать в Python для дальнейшей работы. Для этого данные нужно перевести в формат DataFrame. Это позволит производить различные манипуляции с данными средствами библиотеки pandas.

DateFrame - это табличная структура данных. Ее главная задача — позволить использовать многомерные Series. DataFrame состоит из упорядоченной коллекции колонок, каждая из которых содержит значение разных типов.

DataFrame - быстрый и эффективный инструмент для манипулирования данными со встроенной индексацией.

```
path = './Гостинница Северная'  # название папки, в которой хранятся все файлы
fds = os.listdir(path)

df = pd.concat([pd.read_excel(path + '/' + file, header=8) for file in fds])
df.drop(columns=['Начало среза', 'Конец среза', 'A-', 'R+', 'R-'], inplace=True)
df.columns = ['date', 'consumption']


def main_fun(df, freq, range1):
    df.set_index('date', inplace=True)
    df = df.resample(freq).sum()
    df = df.reset_index(drop=False)

    df['year'] = df['date'].apply(lambda x:x.strftime('%m%d'))
    df['week'] = df['date'].dt.dayofweek

    if range1 == 'day':
        df['date_str'] = df['date'].astype(str)
        split_hour = df.date_str.str.split(' ', expand=True).astype(str)
        print(split_hour)
        df['day'] = split_hour[1]

    df.groupby(range1)['consumption'].mean().plot(kind = 'bar')
    plt.show()
    print(df)


main_fun(df, '60T', 'day')
```

Рис. 1. Код алгоритма.

Для считывания данных с эксель файла мы используем метод read_excel. Так как у нас 12 эксель файлов, то есть в одном файле хранится информация за один месяц, то мы используем функцию concat, чтобы соединить все 12 файлов в один.

Отбрасываем ненужные колонки с помощью df.drop.

Даем понятные названия колонкам: date, consumption.

Изначально у нас индексация идет таким образом: 0, 1, 2, 3… и так далее. Но для использования функции resample, нам необходимо индексами сделать колонку date. Таким образом получаем временной ряд.

Функция resample позволяет сделать выборку данных по временному интервалу (частоте).

То есть, например, если у нас есть данные с частотой полчаса. То с помощью функции resample, можно эти данные преобразовать в данные с частотой час.

В функцию resample передаем задаваемый аргумент. Этот аргумент будет означать частоту, с которой мы хотим получить данные. Аргумент может принимать следующие значения: месяц, день, час, полчаса | M, D, 60T, 30T и так далее.

Для дальнейшей работы нам понадобится индексация по умолчанию. Для этого используем метод reset_index и в параметрах используем drop=False, чтобы колонку с датой не удалило.

Далее создаем колонку в DataFrame под названием year.

Pandas.apply позволяет передавать функцию и применять ее к каждому значению серии DataFrame. Мы передаем функцию, которая будет возвращать день и месяц.

Далее создаем колонку week, в котором хранится номера дней недели (пн – 0, вск – 6). Делаем это с помощью библиотеки datetime.

Создаем колонку day. Для этого сначала дату переводим в формат строки, чтобы к ней можно было применить функцию split, которая отделит дату от времени на отдельные составляющие. И оттуда выбираем колонку 1.

	0	1
0	2020-01-01	00:00:00
1	2020-01-01	01:00:00
2	2020-01-01	02:00:00
3	2020-01-01	03:00:00
4	2020-01-01	04:00:00
...	...	...
8780	2020-12-31	20:00:00
8781	2020-12-31	21:00:00
8782	2020-12-31	22:00:00
8783	2020-12-31	23:00:00
8784	2021-01-01	00:00:00

Рис. 2. Результат работы функции split.

Таким образом, у нас в исходный DataFrame добавилось 3 колонки. И в функцию df.groupby аргументом мы передаем 3-й параметр функции main_fun, что и позволяет построить профиль потребления на заданном диапазоне.

И таким образом происходит группировка методом groupby. После чего находится среднее значение и строится график с помощью библиотеки matplotlib.

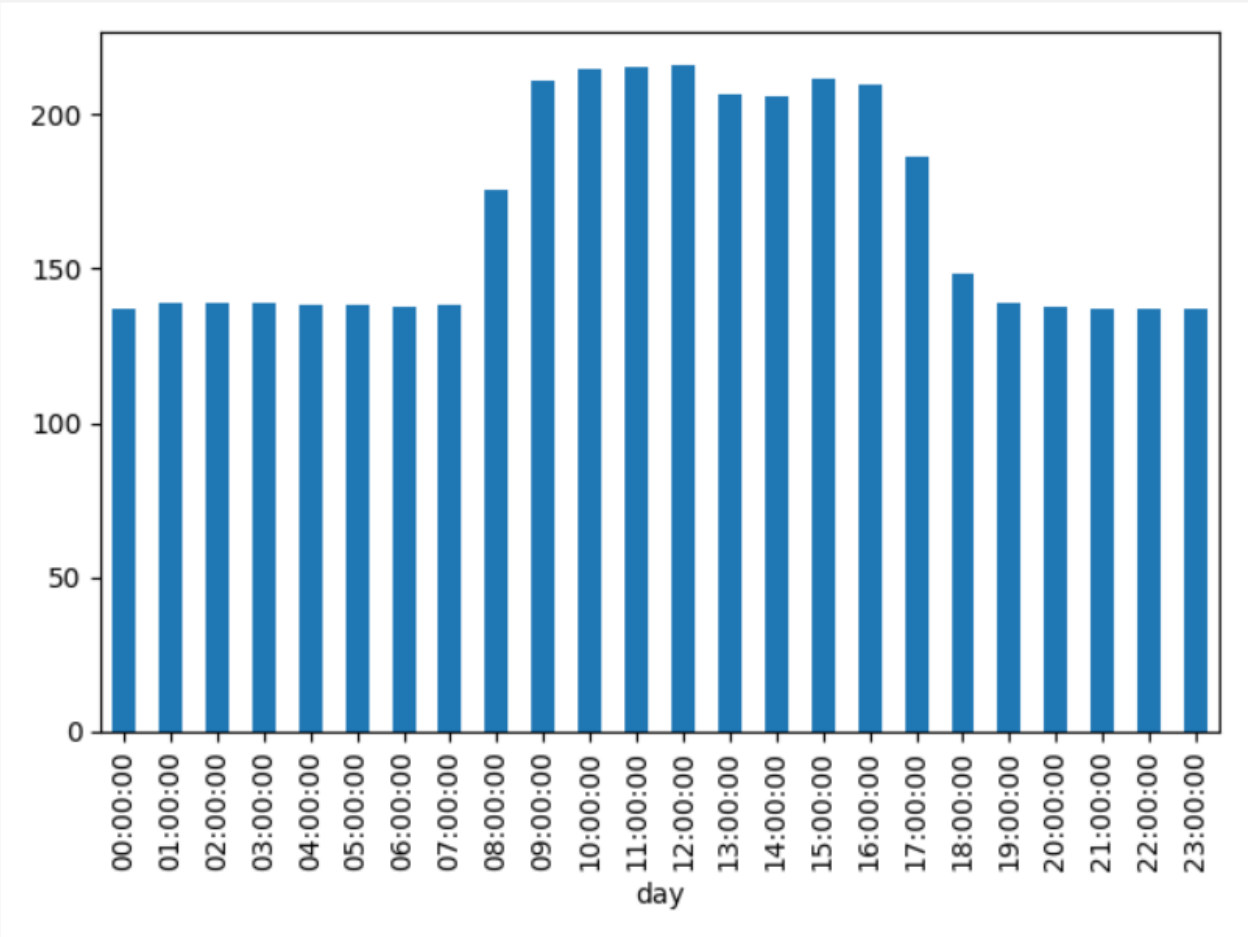

Рис. 3. Суточный профиль потребления.

Использованные источники:

1. Matplotlib: Visualization with Python. [Электронный ресурс]. URL: https://matplotlib.org/ (дата обращения 29.06.2021).

2. Всё о Python. Программирование на Python 3. [Электронный ресурс]. URL: https://all-python.ru/osnovy/modul-datetime.html (дата обращения 29.06.2021).

3. Pandas documentation [Электронный ресурс]. URL: https://pandas.pydata.org/pandas-docs/stable/index.html(дата обращения 29.06.2021).