|  |  |
| --- | --- |
| **bỘ GIÁO DỤC VÀ ĐÀO TẠO** | **BỘ NÔNG NGHIỆP VÀ PTNT** |

TRƯỜNG ĐẠI HỌC THỦY LỢI



họ và tên: TRẦN THỊ VUI

DỰ BÁO MỰC NƯỚC TẠI TRẠM THAKLEK TRÊN SÔNG MEKONG DÙNG RANDOM FOREST

ĐỒ ÁN TỐT NGHIỆP

HÀ NỘI, NĂM 2021

|  |  |
| --- | --- |
| **bỘ GIÁO DỤC VÀ ĐÀO TẠO** | **BỘ NÔNG NGHIỆP VÀ PTNT** |

TRƯỜNG ĐẠI HỌC THỦY LỢI

HỌ VÀ TÊN: TRẦN THỊ VUI

DỰ BÁO MỰC NƯỚC TẠI TRẠM THAKLEK TRÊN SÔNG MEKONG DÙNG RANDOM FOREST

|  |  |
| --- | --- |
| Ngành: | Công nghệ thông tin |
| Mã số: | 7480201 |

|  |  |
| --- | --- |
| NGƯỜI HƯỚNG DẪN | 1. PGS.TS NGUYỄN THANH TÙNG |
|  |  |

HÀ NỘI, NĂM 2021

|  |  |
| --- | --- |
|  | CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM  **Độc lập - Tự do - Hạnh phúc**  ----------★----------  **NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP** |

|  |  |
| --- | --- |
| Họ tên sinh viên: **Trần Thị Vui** | Hệ đào tạo : **Đại học chính quy** |
| Lớp: **59TH2** | Ngành: **Công nghệ thông tin** |
| Khoa: **Công nghệ thông tin** |  |

1. TÊN ĐỀ TÀI:

DỰ BÁO MỰC NƯỚC TẠI TRẠM THAKLEK TRÊN SÔNG MEKONG DÙNG RANDOM FOREST

1. CÁC TÀI LIỆU CƠ BẢN:
2. [Machine Learning | Coursera](https://www.coursera.org/learn/machine-learning/home/welcome) (coursera.com)
3. https://www.python.org/
4. https://www.djangoproject.com/start/
5. https://codelearn.io/sharing/web-step-by-step-voi-django
6. https://machinelearningmastery.com/random-forest-for-time-series-forecasting/
7. NỘI DUNG CÁC PHẦN THUYẾT MINH VÀ TÍNH TOÁN:

* Chương 1: Giới thiệu (20%)
  + Lý do chọn đề tài
  + Mục tiêu đề tài
  + Đối tượng và phạm vi nghiên cứu
* Chương 2: Tiếp cận cơ sở lý thuyết (20%)
  + Học máy
  + Khai phá dữ liệu
  + Bootstrap và bagging
  + Cây hồi quy
  + Random Forest
  + Lỗi của mô hình
  + Phương pháp đánh giá độ chính xác của mô hình
* Chương 3: Phương pháp và xây dựng mô hình (30%)
  + Dữ liệu
  + Công cụ
  + Ứng dụng dữ liệu và công cụ để xây dựng mô hình và triển khai trên web
* Chương 4: Đánh giá kết quả đạt được (30%)
  + Huấn luyện mô hình
  + Kết quả và kiểm định mô hình
  + Kết luận & kiến nghị

1. GIÁO VIÊN HƯỚNG DẪN TỪNG PHẦN

|  |  |
| --- | --- |
| **Phần** | **Họ tên giáo viên hướng dẫn** |
| Tiếp cận cơ sở lý thuyết | Nguyễn Thanh Tùng |
| Ứng dụng phương pháp và xây dựng mô hình |

1. NGÀY GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

Ngày .... tháng …. năm 2021.

|  |  |  |
| --- | --- | --- |
| **Trưởng Bộ môn**  *(Ký và ghi rõ Họ tên)* |  | **Giáo viên hướng dẫn chính**  *(Ký và ghi rõ Họ tên)* |

Nhiệm vụ Đồ án tốt nghiệp đã được Hội đồng thi tốt nghiệp của Khoa thông qua.

|  |
| --- |
| Ngày …. tháng …. năm 2021.  **Chủ tịch Hội đồng**  *(Ký và ghi rõ Họ tên)* |

Sinh viên đã hoàn thành và nộp bản Đồ án tốt nghiệp cho Hội đồng thi.

|  |
| --- |
| Ngày .... tháng …. năm 2021.  **Sinh viên làm Đồ án tốt nghiệp**  *(Ký và ghi rõ Họ tên)* |

|  |  |
| --- | --- |
| logo | TRƯỜNG ĐẠI HỌC THUỶ LỢI  **KHOA CÔNG NGHỆ THÔNG TIN**  BẢN TÓM TẮT ĐỀ CƯƠNG ĐỒ ÁN TỐT NGHIỆP |

TÊN ĐỀ TÀI: Dự báo mực nước tại trạm Thakhek trên sông Mekong sử dụng Random Forest.

*Sinh viên thực hiện*: Trần Thị Vui

*Lớp*: 59TH2

*Giáo viên hướng dẫn*: PGS.TS Nguyễn Thanh Tùng

**TÓM TẮT ĐỀ TÀI**

Lũ lụt là một hiện tượng thời tiết phức tạp, để lại nhiều thiệt hại nặng nề về tài sản và tính mạng tại khu vực trực tiếp chịu thiên tai. Việt Nam là một trong những nước có tần suất lũ lụt cao, các khu vực vùng Đồng bằng sông Cửu Long, Đồng bằng duyên hải miền Trung, Đồng bằng Bắc Bộ nhiều năm gần đây chứng kiến nhiều trận lũ lịch sử đỉnh điểm là trận lũ miền Trung diễn ra vào tháng 10 và 11 của năm 2020. Vì thế việc dự báo mực nước tại các trạm trên sông Mekong là cần thiết để có thể hỗ trợ cảnh báo lũ cho vùng đồng bằng sông Cửu Long, giảm thiểu một cách tối ra thiệt hại về người và tài sản.

Random Forest (RF), một trong những phương pháp phi tuyến nổi trội được dùng phổ biến trong lĩnh vực học máy và khai thác dữ liệu trên thế giới những năm gần đây, RF hoạt động tốt trên các bài toán phân loại và hồi quy và ít bị ảnh hưởng bởi nhiễu (noise). Vì vậy, RF có thể ứng dụng vào xử lý cho bài toán dự báo mực nước tại trạm Thakhek trên sông Mekong.

**CÁC MỤC TIÊU CHÍNH**

* Tìm hiểu các phương pháp học máy, phương pháp học máy kết hợp (ensemble learning).
* Tìm hiểu phương pháp học máy rừng ngẫu nhiên (RF -Random Forest) và các phiên bản cải tiến của RF.
* Ứng dụng RF dự mực nước tại trạm Thakhek trên sông Mekong.

**KẾT QUẢ DỰ KIẾN**

* Mô hình học máy RF dự báo mực nước tại trạm Thakhek trên sông Mekong.
* Xây dựng được một website để dự báo mực nước sử dụng RF tại trạm Thakhek trên sông Mekong.
* Viết báo cáo và tổng kết.

LỜI CAM ĐOAN

Tác giả xin cam đoan đây là Đồ án tốt nghiệp của bản thân tác giả. Các kết quả trong Đồ án tốt nghiệp này là trung thực, và không sao chép từ bất kỳ một nguồn nào và dưới bất kỳ hình thức nào. Việc tham khảo các nguồn tài liệu (nếu có) đã được thực hiện trích dẫn và ghi nguồn tài liệu tham khảo đúng quy định.

|  |  |
| --- | --- |
|  | **Tác giả ĐATN**  *Chữ ký*  **Trần Thị Vui** |

LỜI CẢM ƠN

Sau hơn 4 năm học tập tại trường Đại học Thủy Lợi nói chung và khoa Công nghệ thông tin nói riêng, em đã nhận được sự chỉ bảo và giúp đỡ của các thầy cô giáo và các bạn rất nhiều trong lĩnh vực học tập và cuộc sống.

Đầu tiên, em xin chân thành cảm ơn các thầy cô giáo Trường đại học Thủy Lợi và đặc biệt là các thầy cô giáo khoa Công nghệ thông tin đã dạy cho em có được những kiến thức để phục vụ cho việc thực hiện đồ án. Đặc biệt, trong 14 tuần làm đồ án, em đã được sự hướng dẫn nhiệt tình của Phó giáo sư, tiến sĩ, thầy giáo Nguyễn Thanh Tùng. Em xin gửi lời cảm ơn chân thành tới thầy cô đã giúp đỡ, bổ sung cho em những kiến thức, cho em những lời khuyên và sự góp ý để em có thể hoàn thành đồ án một cách nhanh chóng và hiệu quả nhất.

Trong suốt thời gian học tập và hoàn thành đồ án em đã may mắn được thầy cô chỉ bảo, dìu dắt và được gia đình bạn bè quan tâm, động viên luôn bên cạnh và tạo mọi điều kiện thuận lợi đẻ cho em có thể hoàn thành đồ án này. Trong suốt quá trình làm đò án với đề tài “**Dự báo mực nước tại trạm ThakLek trên sông MeKong dùng Random Forest**”, em đã cố gắng hết sức để xây dựng và hoàn thiện đồ án một cách tốt nhất, nhưng do kiến thức còn hạn chế, thời gian làm đồ án có hạn và kinh nghiệm thực tế chưa có nên cũng không thể tránh được những sai sót. Một lần nữa, em xin chân thành cảm ơn thầy cô giáo, bạn bè và gia đình đã giúp đỡ em trong suốt thời gian qua.

MỤC LỤC

[DANH MỤC CÁC HÌNH ẢNH v](#_Toc91858529)

[DANH MỤC BẢNG BIỂU vi](#_Toc91858530)

[DANH MỤC CÁC TỪ VIẾT TẮT VÀ GIẢI THÍCH CÁC THUẬT NGỮ vii](#_Toc91858531)

[CHƯƠNG 1 GIỚI THIỆU 1](#_Toc91858532)

[1.1 Lý do chọn đề tài 1](#_Toc91858533)

[1.2 Mục tiêu đề tài 2](#_Toc91858534)

[1.3 Đối tượng và phạm vi nghiên cứu 2](#_Toc91858535)

[1.3.1 Đối tượng nghiên cứu 2](#_Toc91858536)

[1.3.2 Phạm vi nghiên cứu 2](#_Toc91858537)

[CHƯƠNG 2 TIẾP CẬN CƠ SỞ LÝ THUYẾT 4](#_Toc91858538)

[2.1 Học máy (Machine Learning) 4](#_Toc91858539)

[2.1.1 Tổng quan 4](#_Toc91858540)

[2.1.2 Nhận xét ưu nhược điểm của học có giám sát và không giám sát 6](#_Toc91858541)

[2.2 Khai phá dữ liệu 7](#_Toc91858542)

[2.2.1 Tổng quan 7](#_Toc91858543)

[2.2.2 Nhiệm vụ chính của khai phá dữ liệu 9](#_Toc91858544)

[2.2.3 Qúa trình khai phá dữ liệu 9](#_Toc91858545)

[2.2.4 Một số kỹ thuật khai phá dữ liệu 10](#_Toc91858546)

[2.3 Bootstrap và bagging 13](#_Toc91858547)

[2.3.1 Bootstrap 13](#_Toc91858548)

[2.3.2 Bagging 13](#_Toc91858549)

[2.4 Cây hồi quy 13](#_Toc91858550)

[2.4.1 Xây dựng cây hồi quy 14](#_Toc91858551)

[2.4.2 Dự đoán dùng cây hồi quy 14](#_Toc91858552)

[2.5 Random Forest 15](#_Toc91858553)

[2.5.1 Giới thiệu 15](#_Toc91858554)

[2.5.2 Rừng ngẫu nhiên hồi quy 18](#_Toc91858555)

[2.6 Lỗi của mô hình 19](#_Toc91858556)

[2.7 Phương pháp đánh giá độ chính xác của mô hình hồi quy 20](#_Toc91858557)

[CHƯƠNG 3 ỨNG DỤNG PHƯƠNG PHÁP VÀ XÂY DỰNG MÔ HÌNH 21](#_Toc91858558)

[3.1 Dữ liệu phục vụ xây dựng mô hình 21](#_Toc91858559)

[3.1.1 Mô tả dữ liệu 21](#_Toc91858560)

[3.1.2 Tiền xử lý dữ liệu 22](#_Toc91858561)

[3.2 Các công cụ xây dựng mô hình RF 25](#_Toc91858562)

[3.2.1 Ngôn ngữ lập trình Python và thư viện 25](#_Toc91858563)

[3.2.2 Xây dựng web sử django 30](#_Toc91858564)

[3.3 Ứng dụng dữ liệu và công cụ vào để xây dựng mô hình và web 35](#_Toc91858565)

[3.3.1 Xây dựng mô hình học máy 35](#_Toc91858566)

[3.3.2 Deploy mô hình học máy lên web sử dụng django 40](#_Toc91858567)

[CHƯƠNG 4 KẾT QUẢ ỨNG DỤNG MÔ HÌNH 52](#_Toc91858568)

[4.1 Qúa trình huấn luyện mô hình 52](#_Toc91858569)

[4.1.1 Các thông số huấn luyện 52](#_Toc91858570)

[4.1.2 Môi trường huấn luyện 52](#_Toc91858571)

[4.1.3 Đánh giá mô hình 52](#_Toc91858572)

[4.1.4 Quá trình huấn luyện 53](#_Toc91858573)

[4.2 Kết quả ứng dụng mô hình 53](#_Toc91858574)

[4.2.1 Kết quả huấn luyện 53](#_Toc91858575)

[4.2.2 Kết quả deploy mô hình nên web 54](#_Toc91858576)

[4.3 Kết luận & kiến nghị 56](#_Toc91858577)

[TÀI LIỆU THAM KHẢO 56](#_Toc91858578)

[PHỤ LỤC 58](#_Toc91858579)

DANH MỤC CÁC HÌNH ẢNH

[Hình 2.1 Minh họa học máy có giám sát 5](file:///D:\DOAN\DeployMode-Excel\Model%20and%20data\DoAn_TranThiVui.docx#_Toc91858148)

[Hình 2.2 Qúa trình khai phá dữ liệu 10](file:///D:\DOAN\DeployMode-Excel\Model%20and%20data\DoAn_TranThiVui.docx#_Toc91858149)

[Hình 2.3 Minh họa rừng ngẫu nhiên 17](file:///D:\DOAN\DeployMode-Excel\Model%20and%20data\DoAn_TranThiVui.docx#_Toc91858150)

[Hình 3.1 Một phần bản ghi của dữ liệu 22](file:///D:\DOAN\DeployMode-Excel\Model%20and%20data\DoAn_TranThiVui.docx#_Toc91858086)

[Hình 3.2 Chuyển bộ dữ liệu từ Time Series sang Supervised Learning 24](file:///D:\DOAN\DeployMode-Excel\Model%20and%20data\DoAn_TranThiVui.docx#_Toc91858087)

[Hình 4. 1 Đường quá trình dự báo mực nước và thực đo tại trạm ThahLek 53](file:///D:\DOAN\DeployMode-Excel\Model%20and%20data\DoAn_TranThiVui.docx#_Toc91858417)

[Hình 4. 2 Độ chính xác mô hình 53](#_Toc91858418)

[Hình 4.3 Giao diện web dùng dự đoán khi người dùng nhập dữ liệu và giao diện kết quả dự đoán 54](#_Toc91858419)

[Hình 4. 4 Giao diện web dùng dự đoán khi người dùng tải file dữ liệu và kết quả dự đoán trên web 55](#_Toc91858420)

DANH MỤC BẢNG BIỂU

[Bảng 2.1 So sánh học có giám sát với học không giám sát 7](#_Toc91422930)

[Bảng 4.1 Các thông số huấn luyện 52](#_Toc91858435)

DANH MỤC CÁC TỪ VIẾT TẮT VÀ GIẢI THÍCH CÁC THUẬT NGỮ

**ĐATN:** Đồ án tốt nghiệp

**RF:** Random Forest

**DEPLOY**: Triển khai mô hình

**TEST ERROR**: Lỗi kiểm thử

**UNSUPERVISED LEARNING**: Học không giám sát

**VALIDATION ERROR:** Lỗi kiểm định

# GIỚI THIỆU

## Lý do chọn đề tài

Lũ lụt là một hiện tượng thời tiết phức tạp, để lại nhiều thiệt hại nặng nề về tài sản và tính mạng tại khu vực trực tiếp chịu thiên tai. Theo thống kê của Tổ chức Hợp tác và Phát triển Kinh tế (Organization for Economic Co-operation and Development – OECD), mỗi năm toàn thế giới chịu thiệt hại hơn 40 tỉ đô la Mỹ, ảnh hưởng đến xấp xỉ 250 triệu người, tần suất xuất hiện lũ lụt đã tăng gần gấp đôi trong giai đoạn 2000-2009 so với thập kỷ trước đó và số lần lũ lụt trong khoảng thời gian 2010 đến 2013 nhiều hơn tổng số lần lũ lụt của cả thập niên 80. Việt Nam là một trong những nước có tần suất lũ lụt cao; các khu vực vùng Đồng bằng sông Cửu Long, Đồng bằng duyên hải miền Trung, Đồng bằng Bắc Bộ nhiều năm gần đây chứng kiến nhiều trận lũ lịch sử đỉnh điểm là trận lũ miền Trung diễn ra vào tháng 10 và 11 của năm 2020.

Vì thế việc dự báo mực nước tại các trạm trên sông Mekong là cần thiết để có thể hỗ trợ cảnh báo lũ cho vùng đồng bằng sông Cửu Long, giảm thiểu một cách tối ra thiệt hại về người và tài sản. Xây dựng mô hình dự báo mực nước đáp ứng tối ưu việc cập nhật, xác định và đánh giá về lũ lụt xảy ra trong suốt thời gian lũ lụt từ đó có những giải pháp cứu trợ, tiếp cận kịp thời khi xuất hiện những trận lũ lụt gây ra hơn là các đánh giá dự báo thông thường. Với sự phát triển của lĩnh vực công nghệ thông tin, trí tuệ nhân tạo mà đặc biệt trong lĩnh vực học máy, học sâu những năm gần đây đang thực sự được quan tâm và ứng dụng vào thực tiễn các lĩnh vực, ngành nghề trong cuộc sống. Bài toán xây dựng mô hình random forest dự báo mực nước là một minh chứng cho sự phát triển của công nghệ thông tin dựa trên các phương pháp, thuật toán đã có thể xác định, cập nhật tính ưu việt trong công tác dự báo, đánh giá với mức độ chính xác cao nhằm góp phần thiết thực vào thực tiễn trong lĩnh vực quản lý tài nguyên nước và phòng chống thiên tai tại Việt Nam. Nghiên cứu xây dựng mô hình rừng ngẫu nhiên (random forest) dự báo mực nước có vai trò vô cùng quan trọng trong việc đánh giá tác động của trận lũ lịch sử khi các tài liệu đầu vào để đánh giá theo phương pháp truyền thống còn nhiều hạn chế tại thời điểm tức thời, thời gian thực khi lũ xuất hiện, đây là một ý nghĩa quan trọng trong quản lý tài nguyên nước, phòng chống thiên tai cho khu vực nghiên cứu. Vì những lý do nêu trên, đề tài nghiên cứu khoa học: “Dự báo mực nước tại trạm ThakLek trên sông Mekong sử dụng Random Forest” nơi thể hiện sự đóng góp quan trọng của lưu vực thuộc phía trung-nam nước Cộng hòa Dân chủ Nhân dân Lào cho dòng chính sông Mekong.

## Mục tiêu đề tài

Dựa trên lý do chọn đề tài, trong nghiên cứu này đồ án thực hiện có các mục tiêu chính sau:

* Tiếp cận cơ sở lý thuyết về mạng trí tuệ nhân tạo nói chung và mô hình máy học RF.
* Ứng dụng RF trong việc xây dựng mô hình dự báo mực nước tại ThakLek trên sông Mekong bằng ngôn ngữ Python.
* Đánh giá kết quả từ mô hình đạt được, từ đó đưa ra kết luận và kiến nghị.
* Xây dựng một website để dự báo mực nước sử dụng RF tại trạm Thakhek trên sông Mekong.

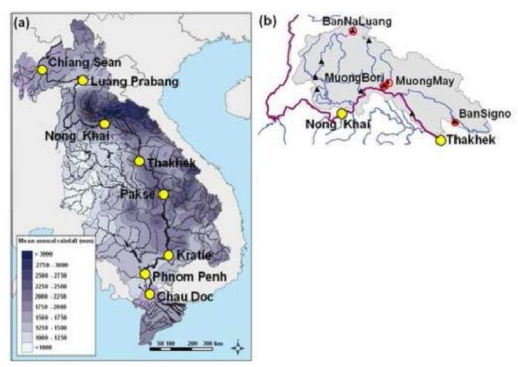
## Đối tượng và phạm vi nghiên cứu­

### Đối tượng nghiên cứu

Xây dựng mô hình dự đoán và đánh giá mực nước tại trạm ThakLek trên sông Mekong dùng RF.

### Phạm vi nghiên cứu

Vùng nghiên cứu là trạm ThakLek trên sông Mekong nơi thể hiện sự đóng góp quan trọng của lưu vực thuộc phía trung-nam nước Cộng hòa Dân chủ Nhân dân Lào cho dòng chính sông Mekong. Vị trí nghiên cứu được minh họa tại [Hình 1.1](#hinh11).



Hình 1. Vị trí lưu vực nghiên cứu

# TIẾP CẬN CƠ SỞ LÝ THUYẾT

## Học máy (Machine Learning)

### Tổng quan

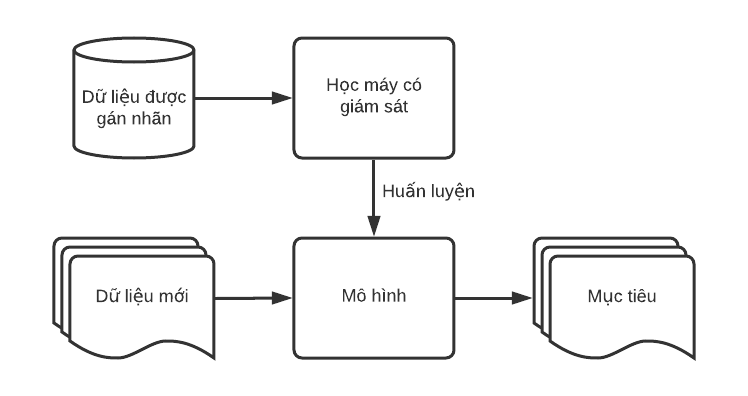
Học máy là một lĩnh vực của trí tuệ nhân tạo (trí thông minh được thể hiện bằng máy móc) liên quan đến việc nghiên cứu và xây dựng các kĩ thuật giúp máy tính “học” tự động từ dữ liệu cho trước để giải quyết những bài toán cụ thể. Machine learning còn cung cấp một phương pháp hiệu quả để học hỏi từ dữ liệu thay vì dựa vào con người để phân tích và dự đoán.

Học máy hiện nay được áp dụng rộng rãi nhiều lĩnh vực: chuẩn đoán y khoa, phân loại ảnh, nhận dạng vật thể, nhận dạng tiếng nói và chữ viết, phân tích câu và dịch tự động, chơi trò chơi, xe tự lái, gợi ý bán hàng, …

Học máy có thể được phân loại bằng phương thức học: học có giám sát – supervised learning, học không giám sát – unsupervised learning, học nửa giám sát – semi-supervised learning, học tăng cường – reinforcement learning. Do nội dung đồ án tập trung vào bài toán dự đoán mực nước tại trạm ThakLek trên sông Mekong sử dụng RF, vì thế ta có thể chỉ quan tâm đến hình thức học đầu tiên đó là học có giám sát mà không dành nhiều quan tâm đến các hình thức học còn lại.

#### Học có giám sát (Supervised Learning)

Đây là phương pháp học máy giải quyết bài toán dự đoán mục tiêu với đầu vào cho trước. Các mục tiêu được gọi là “nhãn”, thường được ký hiệu là . Dữ liệu đầu vào thường được ký hiệu là . Mỗi một cặp được gọi là một mẫu và được cung cấp cho quá trình huấn luyện mô hình. Nói ngắn gọn: học có giám sát là hình thức học mà dữ liệu dùng để xây dựng mô hình đã được gán nhãn từ trước.



Hình 2. Minh họa học máy có giám sát

Học có giám sát giải quyết một số bài toán sau:

* Hồi quy (regression): là bài toán tìm nghiệm tối ưu có đầu ra là một số thực, quan hệ giữa đầu vào với mục tiêu là 1 hàm số , thường là tuyến tính hoặc logistic. Mục tiêu của bài toán là tìm ra nghiệm gần đúng với . Hai hàm mất mát phổ biến là và giả sử dữ liệu có nhiễu Gauss. Ví dụ: dự đoán giá bất động sản.
* Phân loại (classification): là bài toán dự đoán nhãn lớp giữa những nhãn lớp có sẵn. Mô hình được xây dựng dựa trên một tập dữ liệu được gán nhãn trước. Đầu ra của mô hình là một mảng chứa các xác suất của các lớp tương ứng với dữ liệu đầu vào. Hàm mất mát phổ biến là entropy chéo. Bài toán phân lớp có các loại sau: phân lớp nhị phân (phân loại đúng/sai), phân lớp đa trị, phân lớp đa lớp. Ví dụ: đưa hình của một chú chó vào một mô hình phân loại giữa {chó, mèo, gà} ta được khả năng bức hình đó là chó, mèo và gà theo mô hình đó là {0.6, 0.3, 0.1}.

##### Hàm mất mát (Loss Function)

Hàm mất mát là một hàm để đánh giá một mô hình học máy xem mô hình đang tốt hay tệ, hàm mất mát so sánh giá trị đầu ra của hàm mục tiêu so với tập dữ liệu huấn luyện. Theo quy ước, giá trị hàm mất mát càng thấp thì mô hình càng tốt.

Hàm bình phương sai số (Squared error) là hàm mất mát phổ biến nhất cho giá trị số, hàm entropy chéo (Cross entropy) giá trị là nhãn.

Để tránh việc mô hình quá khớp khi huấn luyện ta thường chia dữ liệu ban đầu thành hai: tập dữ liệu huấn luyện để học mô hình tập này cho ra lỗi huấn luyện và tập dữ liệu kiểm định để đánh giá mô hình, cho ra lỗi kiểm thử.

#### Học không giám sát (Unsupervised Learning)

Đây là phương pháp học máy không biết được mục tiêu mà bài toán đang nhắm đến hay nhãn mà chỉ có dữ liệu đầu vào. Học không giám sát sẻ dụng những dữ liệu chưa được gán nhãn để suy luận ra mỗi quan hệ. Phương pháp này thường được sử dụng để tìm cấu trúc của tập dữ liệu. Tuy nhiên lại không có phương pháp đánh giá được cấu trúc tìm ra được là đúng hay sai.

Các bài toán học không giám sát được chia làm hai loại:

* Phân cụm: phân tập dữ liệu thành cụm/thể loại ( cho trước) dựa trên sự liên quan giữa các dữ liệu trong mỗi nhóm. Ví dụ: phân nhóm khách hàng dựa trên hành vi mua hàng, lịch sử ghé thăm và đặt mua các món hàng.
* Luật kết hợp: Là bài toán khi chúng ta muốn khám phá ra một quy luật dựa trên nhiều dữ liệu cho trước. Ví dụ: nhóm khách hàng mua xà phòng, dầu gội đầu thì có thường mua thêm kem đánh răng hay không? Những khách hàng nam mua quần áo thường có xu hướng mua thêm đồng hồ hoặc thắt lưng..

### Nhận xét ưu nhược điểm của học có giám sát và không giám sát

Sau khi tìm hiểu 2 dạng học máy thì chúng ta có thể đưa ra so sánh ưu nhược điểm giữa chúng như trong bảng sau:

Bảng 2. So sánh học có giám sát với học không giám sát

|  |  |  |
| --- | --- | --- |
|  | **Ưu điểm** | **Nhược điểm** |
| Học có giám sát | * Học dựa trên nhãn và có mục tiêu rõ ràng. * Cho phép thu thập dữ liệu hoặc tạo đầu ra bằng kinh nghiệm học hỏi trước đó. * Giải quyết vấn đề tính toán trong thế giới thực. * Phương pháp đơn giản dễ sử dụng. * Kết quả đầu ra có thể đo lường độ chính xác | * Dữ liệu cần để mô hình học phải đạt nhiều yêu cầu: đủ nhãn cho mỗi lớp, ít bị nhiễu, đầu ra phải chính xác. * Phân loại dữ liệu lớn có thể là một thách thức lớn * Cần nhiều thời gian tính toán |
| Học không giám sát | * Tìm ra được mối quan hệ ẩn trong dữ liệu. * Tìm ra những đặc trưng để phân thể loại dữ liệu. * Xử lý trong thời gian thực, dữ liệu đầu vào có thể được phân tích và phân loại ngay. * Dễ thu thập dữ liệu chưa gán nhãn hơn là gán nhãn | * Cần một tập dữ liệu đủ lớn để có thể phân loại chính xác * Không thể đo lường độ tin cậy của kết quả. * Không biết số lớp. |

Để phục vụ cho bài toán dự báo mực nước tại trạm ThakLek trên sông Mekong, em đánh giá phương pháp học có giám sát là phương pháp học máy phù hợp do có những ưu điểm được kể trong bảng ở trên. Ngoài ra đề tài cũng có mục tiêu rõ ràng cho đầu ra của mô hình là dự đoán mực nước tại trạm ThakLek. Do đó, em đã chọn học máy có giám sát là phương pháp học máy để phục vụ đề tài này.

## Khai phá dữ liệu

### Tổng quan

Trong cuộc sống hiện nay sự phát triển mạnh mẽ của công nghệ thông tin và truyền thông, nhu cầu lưu trữ dữ liệu và trao đổi thông tin trong xã hội ngày càng tăng lên mạnh mẽ. Tuy nhiên, đi cùng với lượng dữ liệu và thông tin ngày càng khổng lồ mà chúng ta có được thì việc biến đổi những dữ liệu thô có sẵn đó thành tri thức trở thành một đòi hỏi tất yếu trong đời sống hàng ngày. Từ nhu cầu thực tế trên, đòi hỏi chúng ta phải tìm kiếm và ứng dụng các kỹ thuật nhằm “khai phá” những thông tin hữu ích, những tri thức có ích từ những nguồn dữ liệu khổng lồ hiện có.

Phát hiện tri thức và khai phá dữ liệu (Knowledge Discovery and Data Mining - KDD) là những công việc liên quan đến việc trích, lọc những thông tin có ích từ các nguồn dữ liệu. Khai phá dữ liệu là một tập các kỹ thuật được sử dụng một cách tự động nhằm khám phá những tri thức có ích ở dạng tiềm năng trong nguồn dữ liệu đã có.

Ở đây chúng ta có thể coi khai phá dữ liệu là cốt lõi của quá trình phát hiện tri thức. Quá trình phát hiện tri thức gồm các bước:

Bước 1: Trích chọn dữ liệu (data selection): Là bước trích chọn những tập dữ liệu cần được khai phá từ các tập dữ liệu lớn (databases, data ware houses).

Bước 2: Tiền xử lý dữ liệu (data preprocessing): Là bước làm sạch dữ liệu (xử lý dữ liệu không đầy đủ, dữ liệu nhiễu, dữ liệu không nhất quán, …), rút gọn dữ liệu (sử dụng các phương pháp thu gọn dữ liệu, histograms, lấy mẫu…), rời rạc hóa dữ liệu (dựa vào histograms, entropy, phân khoảng, ...). Sau bước này, dữ liệu sẽ nhất quán, đầy đủ, được rút gọn và được rời rạc hóa.

Bước 3: Biến đổi dữ liệu (data transformation): Là bước chuẩn hóa và làm mịn dữ liệu để đưa dữ liệu về dạng thuận lợi nhất nhằm phục vụ cho các kỹ thuật khai thác ở bước sau.

Bước 4: Khai phá dữ liệu (data mining): Đây là bước quan trọng và tốn nhiều thời gian nhất của quá trình khám phá tri thức, áp dụng các kỹ thuật khai phá (phần lớn là các kỹ thuật của machine learning) để khai phá, chọn lựa được các mẫu (pattern) thông tin, các mối liên hệ đặc biệt trong dữ liệu.

Bước 5: Đánh giá và biểu diễn tri thức (knowledge representation & evaluation): Dùng các kỹ thuật hiển thị dữ liệu để trình bày các mẫu thông tin (tri thức) và mối liên hệ đặc biệt trong dữ liệu đã được khai thác ở bước trên biểu diễn theo dạng gần gũi với người sử dụng như đồ thị, cây, bảng biểu, luật, …. Đồng thời bước này cũng đánh giá những tri thức khám phá được theo những tiêu chí nhất định. Trong giai đoạn khai phá dữ liệu, có thể cần sự tương tác của người dùng để điều chỉnh và rút ra các tri thức cần thiết nhất. Các tri thức nhận được cũng có thể được lưu và sử dụng lại.

Việc khai phá dữ liệu có thể được tiến hành trên một lượng lớn dữ liệu có trong CSDL, các kho dữ liệu hoặc trong các loại lưu trữ thông tin khác.

### Nhiệm vụ chính của khai phá dữ liệu

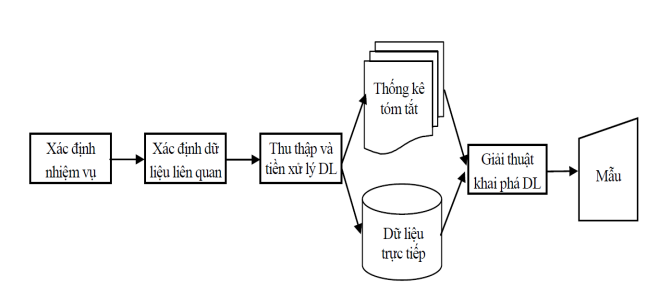
Giảm chiều dữ liệu: Giảm chiều dữ liệu là việc giảm chiều của không gian tìm kiếm dữ liệu, giảm chi phí thu thập và lưu trữ dữ liệu, nâng cao hiệu quả của việc khai phá dữ liệu và làm đơn giản hóa các kết quả khai phá dữ liệu.

Phân nhóm và phân loại: Phân loại và phân nhóm là hai nhiệm vụ có mối quan hệ tương đối gần nhau trong khai phá dữ liệu. Một lớp là một tập các đối tượng có cùng một số đặc điểm hoặc mối quan hệ nào đó, tất cả các đối tượng trong lớp này được phân vào trong cùng một loại tên nhằm mục đích là để phân biệt với các lớp khác. Một cụm là một tập các đối tượng tương tự nhau về mặt vị trí. Các cụm thường được tạo ra nhằm mục đích để sau đó tiến hành phân loại các đối tượng.

Trích chọn luật: Trích chọn luật tìm kiếm và đưa ra dữ liệu bằng cách tất cả các dữ liệu được đưa ra dựa trên các suy diễn/các quyết định mà các suy diễn/quyết định này được xây dựng từ các tri thức thu thập được từ dữ liệu đó. Đối với người sử dụng các kết quả của khai phá dữ liệu họ chỉ mong muốn có một cách giải thích đơn giản là tại sao có các kết quả phân loại đó, thuộc tính nào ảnh hưởng đến kết quả khai phá dữ liệu…Tuy nhiên, bằng các tham số phân loại rất khó để có thể diễn giải các tri thức đó theo cách mà người sử dụng có thể dễ dàng hiểu được.

### Qúa trình khai phá dữ liệu

Các giải thuật khai phá dữ liệu thường được miêu tả như những chương trình hoạt động trực tiếp trên tệp dữ liệu. Với các phương pháp học máy và thống kê trước đây, thường thì bước đầu tiên là các giải thuật nạp toàn bộ tệp dữ liệu vào trong bộ nhớ. Khi chuyển sang các ứng dụng công nghiệp liên quan đến việc khai phá các kho dữ liệu lớn, mô hình này không thể đáp ứng được. Không chỉ bởi vì nó không thể nạp hết dữ liệu vào trong bộ nhớ mà còn vì khó có thể chiết xuất dữ liệu ra các tệp đơn giản để phân tích được. Quá trình khai phá dữ liệu được thể hiện bởi mô hình sau:



Hình 2.2 Qúa trình khai phá dữ liệu

### Một số kỹ thuật khai phá dữ liệu

Mục đích của khai phá dữ liệu là chiết xuất ra các tri thức có lợi cho kinh doanh hay cho nghiên cứu khoa học…

Do đó, ta có thể xem mục đích của khai phá dữ liệu sẽ là mô tả các sự kiện và dự đoán. Các mẫu khai phá dữ liệu phát hiện được nhằm vào mục đích này. Dự đoán liên quan đến việc sử dụng các biến hoặc các đối tượng (bản ghi) trong cơ sở dữ liệu để chiết xuất ra các mẫu, dự đoán được những giá trị chưa biết hoặc những giá trị tương lai của các biến đáng quan tâm. Mô tả tập trung vào việc tìm kiếm các mẫu mô tả dữ liệu mà con người có thể hiểu được. Để đạt được những mục đích này, nhiệm vụ chính của khai phá dữ liệu bao gồm như sau:

#### Phân nhóm dữ liệu

Phân nhóm là kỹ thuật khai phá dữ liệu. Sự phân nhóm dữ liệu là quá trình lọc không được giám sát, là quá trình nhóm những đối tượng vào trong những lớp tương đương, đến những đối tượng trong một nhóm là tương đương nhau, chúng phải khác với những đối tượng trong những nhóm khác. Trong phân loại dữ liệu, một bản ghi thuộc về lớp nào là phải xác định trước, trong khi phân nhóm không xác định trước. Trong phân nhóm, những đối tượng được nhóm lại cùng nhau dựa vào sự giống nhau của chúng. Sự giống nhau giữa những đối tượng được xác định bởi những chức năng giống nhau. Thông thường những sự giống về định lượng như khoảng cách hoặc độ đo khác được xác định bởi những chuyên gia trong lĩnh vực của mình. Đa số các ứng dụng phân nhóm được sử dụng trong sự phân chia thị trường. Với sự phân nhóm khách hàng vào trong từng nhóm, những doanh nghiệp có thể cung cấp những dịch vụ khác nhau tới nhóm khách hàng một cách thuận lợi. Ví dụ, dựa vào chi tiêu, số tiền trong tài khoản và việc rút tiền của khách hàng, một ngân hàng có thể xếp những khách hàng vào những nhóm khác nhau. Với mỗi nhóm, ngân hàng có thể cho vay những khoản tiền tương ứng cho việc mua nhà, mua xe, … Trong trường hợp này ngân hàng có thể cung cấp những dịch vụ tốt hơn và cũng chắc chắn rằng tất cả các khoản tiền cho vay đều có thể thu hồi được. Ta có thể tham khảo một khảo sát toàn diện về kỹ thuật và thuật toán phân nhóm trong.

* Hồi qui (Regression): Là việc xây dựng mô hình máy tính từ một tập dữ liệu với biến đích có giá trị thực. Bài toán hồi qui tương tự như phân loại, điểm khác nhau là biến đích có dạng số trong khi bài toán phân loại có biến đích kiểu rời rạc. Việc dự báo các giá trị số thường được làm bởi các phương pháp thống kê cổ điển chẳng hạn như hồi qui tuyến tính. Tuy nhiên, phương pháp mô hình hóa cũng được sử dụng Hồi quy được ứng dụng trong nhiều lĩnh vực, ví dụ: dự đoán số lượng sinh vật phát quang hiện thời trong khu rừng bằng cách dò tìm vi sóng bằng thiết bị cảm biến từ xa; dự đoán khả năng tử vong của bệnh nhân khi biết các kết quả xét nghiệm chẩn đoán; dự đoán nhu cầu tiêu thụ một sản phẩm mới bằng một hàm chi tiêu quảng cáo….
* Tổng hợp (summarization): Là công việc liên quan đến các phương pháp tìm kiếm một mô tả cô đọng cho tập con dữ liệu. Các kỹ thuật tổng hợp thường được áp dụng trong việc phân tích dữ liệu có tính thăm dò và báo cáo tự động.
* Mô hình hóa phụ thuộc (dependency modeling): Là việc tìm kiếm mô tả các phụ thuộc quan trọng giữa các biến. Mô hình phụ thuộc tồn tại hai mức:
* Mức cấu trúc của mô hình (thường dưới dạng đồ thị) xác định các biến phụ thuộc cục bộ vào các biến khác.
* Mức định lượng của mô hình xác định mức độ phụ thuộc của biến. Những phụ thuộc này thường được biểu thị dưới dạng luật. Quan hệ phụ thuộc cũng có thể biểu diễn dưới dạng mạng tin cậy. Đó là đồ thị có hướng không có dạng chu trình, các nút biểu diễn thuộc tính và trọng số chỉ liên kết phụ thuộc giữa các nút đó.
* Phát hiện sự thay đổi và độ lệch (change and deviation dectection): Nhiệm vụ này tập trung vào khám phá những thay đổi có ý nghĩa trong dữ liệu dựa vào các giá trị chuẩn hay độ đo đã biết trước, phát hiện độ lệch đáng kể giữa nội dung của tập con dữ liệu và nội dung mong đợi. Hai mô hình độ lệch thường dùng là lệch theo thời gian và lệch theo nhóm. Độ lệch theo thời gian là sự thay đổi có nghĩa của dữ liệu theo thời gian. Độ lệch theo nhóm là sự khác nhau giữa dữ liệu trong hai tập con dữ liệu, tính cả trường hợp tập con của đối tượng này thuộc tập con kia, nghĩa là xác định dữ liệu trong một nhóm con của đối tượng có khác nhau đáng kể so với toàn bộ đối tượng.

#### Phân loại dữ liệu

Khái niệm phân loại dữ liệu được Han và Kamber đưa ra năm 2000. Phân loại dữ liệu là xây dựng một mô hình mà có thể phân các đối tượng thành những lớp để dự đoán giá trị bị mất tại một số thuộc tính của dữ liệu hay tiên đoán giá trị của dữ liệu sẽ xuất hiện trong tương lai. Quá trình phân loại dữ liệu được thực hiện qua hai bước:

* Bước thứ nhất: Dựa vào tập hợp dữ liệu huấn luyện, xây dựng một mô hình mô tả những đặc trưng của những lớp dữ liệu hoặc những khái niệm, đây là quá trình học có giám sát, học theo mẫu được cung cấp trước.
* Bước thứ hai: Từ những lớp dữ liệu hoặc những khái niệm đã được xác định trước, dự đoán giá trị của những đối tượng quan tâm. Một kỹ thuật phân loại dữ liệu được Han và Kamber đưa ra là cây quyết định. Mỗi nút của cây đại diện một quyết định dựa vào giá trị thuộc tính tương ứng. Kỹ thuật này đã được nhiều tác giả nghiên cứu và đưa ra nhiều thuật toán.

## Bootstrap và bagging

### Bootstrap

Là một phương pháp rất nổi tiếng trong thống kê được giới thiệu bởi Bradley Efron vào năm 1979. Phương pháp này chủ yếu dùng để ước lượng lỗi chuẩn (standard errors), độ lệch (bias) và tính khoảng tin cậy (confidence interval) cho các tham số. Phương pháp này được thực hiện như sau: Từ một quần thể ban đầu lấy ra một mẫu L= (x1, x2, ...xn) gồm n thành phần, tính toán các tham số mong muốn. Trong bước lặp lại b lần việc tạo ra mẫu Lb cũng gồm n phần tử từ L bằng cách lấy lại mẫu với sự thay thế các thành phần trong mẫu ban đầu sau đó tính toán các tham số mong muốn.

### Bagging

Phương pháp này được xem như là một phương pháp tổng hợp kết quả có được từ các bootstrap. Dựa trên cách phân tích hiệu quả của giải thuật học, (Breiman, 1996) đề xuất giải thuật học Bagging (Bootstrap Aggregating) nhằm giảm lỗi của mô hình dự báo. Giải thuật có thể được tóm tắt như sau:

* Từ tập dữ liệu học LS có m phần tử, xây dựng T mô hình cơ sở độc lập nhau
* Mô hình thứ t được xây dựng trên tập mẫu Bootstrap thứ t (lấy mẫu m phần tử có hoàn lại
* Kết thúc quá trình xây dựng T mô hình cơ sở, dùng chiến lược bình chọn số đông để phân lớp một phần tử x mới đến hoặc giá trị trung bình (regression) cho bài toán hồi quy.

## Cây hồi quy

Mô hình cây hồi quy tách đệ quy theo hàng của tập dữ liệu đầu vào ℒ thành các tập dữ liệu nhỏ hơn, hình thành nút và lá của cây. Tại mỗi lần tách nút, một thuộc tính và giá trị tách của thuộc tính này được chọn để chia nút thành 2 nút con, nút con trái và nút con phải.

### Xây dựng cây hồi quy

Gọi *t* là nút cha để tách nhánh trên cây hồi quy. Việc tách nhánh trên thuộc tính X được xác định bởi việc giảm sự hỗn tạp tại nút *t*, ký hiệu , xem Breiman và đồng nghiệp [13]. Kỳ vọng của Y ở nút *t* được tối thiểu hóa nhờ hàm lỗi bình phương sai số được định nghĩa như sau:

Trong đó *N(t)* là tổng số mẫu hiện tại ở nút t và Ӯt là trung bình mẫu của *Y* tại t.

Gọi là giá trị chia tách thuộc tính tại nút thành nút con trái và nút con phải phụ thuộc vào hoặc , và . Độ biến thiên của các mẫu cho mỗi nút con là:

Trong đó là trung bình mẫu của và là kích thước mẫu của . Tương tự, và là trung bình mẫu và kích thước mẫu của .

Như vậy, việc giảm độ hỗn tạp theo việc chia tách đối với được tính như sau:

(1)

Trong đó và là các tỷ lệ quan sát trong và . Điểm chia tách được chọn trên thuộc tính cho mỗi nút chính là giá trị làm cho đạt cực đại.

### Dự đoán dùng cây hồi quy

Khi xây dựng cây hồi quy, ta cần phải tính toán giá trị cho nút lá của cây, quá trình này được mô tả sau đây. Gọi là véc-tơ chứa tham số ngẫu nhiên để xác định việc xây dựng cây. Trong mỗi cây hồi quy, ta tính toán trọng số dương cho mỗi mẫu . Đặt là nút lá trong cây hồi quy. Các mẫu được gán các trọng số , trong đó là số mẫu trong . Nghĩa là việc dự đoán dùng cây hồi quy đơn giản là tính giá trị trung bình của các mẫu tại nút lá của cây.

Với dữ liệu thử nghiệm , là giá trị dự đoán của cây hồi quy được tính như sau:

## Random Forest

### Giới thiệu

Random Forest (rừng ngẫu nhiên) được đề xuất và phát triển bởi Leo Breiman tại đại học California, Berkeley. Breiman cũng đồng thời là tác giả của phương pháp CART (Classification and Regression Trees) được đánh giá là một trong mười phương pháp khai phá dữ liệu kinh điển. Random Forest được xây dựng dựa trên 3 thành phần chính là: (1) CART, (2) học toàn bộ, hội đồng các chuyên gia, kết hợp các mô hình và (3) tổ hợp bootstrap (bagging).

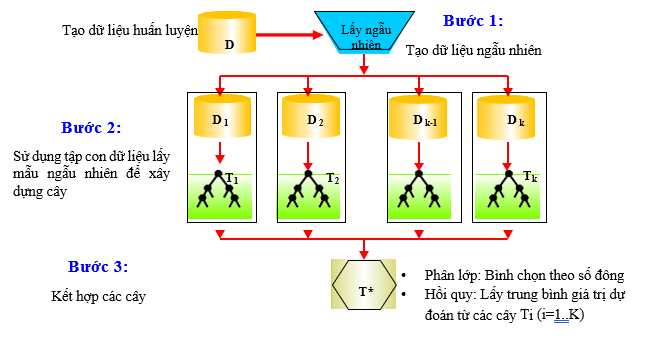
Random Forest (RF) là phương pháp cải tiến của phương pháp tổng hợp bootstrap (bagging) và có thể được sử dụng trong bài toán phân loại và hồi quy. RF sử dụng 2 bước ngẫu nhiên, một là ngẫu nhiên theo mẫu (sample) dùng phương pháp bootstrap có hoàn lại (with replacement), hai là lấy ngẫu nhiên một lượng thuộc tính từ tập thuộc tính ban đầu. Các tập con (sub-dataset) được tạo ra từ 2 lần ngẫu nhiên này có tính đa dạng cao, ít liên quan đến nhau, giúp giảm lỗi phương sai (variance). Các cây CART được xây dựng từ tập các tập dữ liệu con này tạo thành rừng. Khi tổng hợp kết quả, RF dùng phương pháp bỏ phiếu (voting) cho bài toán phân loại và lấy giá trị trung bình (average) cho bài toán hồi quy. Việc kết hợp các mô hình CART này để cho kết quả cuối cùng nên RF được gọi là phương pháp học tập thể.

Đối với bài toán phân loại, cây CART sử dụng công thức Gini như là một hàm điều kiện để tính toán điểm tách nút của cây. Số lượng cây là không hạn chế, các cây trong RF được xây dựng với chiều cao tối đa.

Trong những năm gần đây, RF được sử dụng khá phổ biến bởi những điểm vượt trội của nó so với các thuật toán khác: xử lý với dữ liệu có số lượng các thuộc tính lớn, có khả năng ước lượng độ quan trọng các thuộc tính, thường có độ chính xác cao trong phân loại (hoặc hồi quy), quá trình học nhanh. Trong RF, mỗi cây chỉ chọn một tập nhỏ các thuộc tính trong quá trình xây dựng, cơ chế này làm cho RF thực thi với tập dữ liệu có số lượng thuộc tính lớn trong thời gian chấp nhận được khi tính toán. Người dùng có thể đặt mặc định số lượng các thuộc tính để xây dựng cây trong rừng, thông thường giá trị mặc định tối ưu là cho bài toán phân loại và *p*/3 với các bài toán hồi quy (p là số lượng tất cả các thuộc tính của tập dữ liệu ban đầu). Số lượng các cây trong rừng cần được đặt đủ lớn để đảm bảo tất cả các thuộc tính đều được sử dụng một số lần. Thông thường là 500 cây cho bài toán phân loại, 1000 cây cho bài toán hồi quy. Do sử dụng phương pháp bootstrap lấy mẫu ngẫu nhiên có hoàn lại nên các tập dữ liệu con có khoảng 2/3 các mẫu không trùng nhau dùng để xây dựng cây, các mẫu ngày được gọi là in-bag. Khoảng 1/3 số mẫu còn lại gọi là out-of-bag, do không tham gia vào việc xây dựng cây nên RF dùng luôn các mẫu out-of-bag này để kiểm thử và tính toán độ quan trọng thuộc tính của các cây CART trong rừng

Tóm tắt giải thuật RF:

* Bước 1: Từ tập dữ liệu huấn luyện D, ta tạo dữ liệu ngẫu nhiên (mẫu bootstrap)
* Bước 2: Sử dụng các tập con dữ liệu lấy mẫu ngẫu nhiên *D1, D2, ..., Dk* xây dựng nên các cây *T1, T2, ..., Tk*.
* Bước 3: Kết hợp các cây: sử dụng chiến lược bình chọn theo số đông với bài toán phân loại hoặc lấy trung bình các giá trị dự đoán từ các cây với bài toán hồi quy.



Hình 2.3 Minh họa rừng ngẫu nhiên

### Rừng ngẫu nhiên hồi quy

Rừng ngẫu nhiên hồi quy (RF) gồm tập hợp các cây hồi quy đã trình bày ở mục[***2.2***](#_Cây_hồi_quy)***.*** Từ tập dữ liệu đầu vào ℒ, RF dùng kỹ thuật lấy mẫu bootstrap có hoàn lại tạo ra nhiều tập dữ liệu khác nhau. Trên mỗi tập dữ liệu con này, lấy ngẫu nhiên một lượng cố định thuộc tính thường gọi là mtry (max\_feature) để xây dựng cây. Mỗi cây hồi quy được xây dựng không cắt nhánh với chiều cao tối đa. Việc lấy hai lần ngẫu nhiên cả mẫu và thuộc tính đã tạo ra các tập dữ liệu con khác nhau giúp RF giảm độ dao động (variance) của mô hình học.

#### Dự đoán bằng rừng ngẫu nhiên hồi quy

Việc xây dựng rừng ngẫu nhiên hồi quy và dự đoán mẫu mới được mô tả như sau. Đặt Θ = {*θk*}1K là tập gồm K các vectơ tham số ngẫu nhiên cho rừng được sinh ra từ ℒ, trong đó *θk* là một vectơ tham số ngẫu nhiên để xác định độ lớn của cây thứ k trong rừng (k = 1... K). Gọi ℒk là tập dữ liệu thứ k sinh ra từ ℒ dùng kỹ thuật bootstrap, trong mỗi cây hồi quy Tk từ ℒk, ta tính trọng số dương *wi* (*xi*, *θk*) cho từng mẫu *xi* ∈ ℒ. Đặt *l* (*x, θk*, t) là nút lá t trong cây *Tk*. Mẫu *xi* ∈ *l* (*x, θk*, t) được gán cùng một trọng số *wi* (*xi*, *θk*) = 1/*N(t),* trong đó *N(t)* là các mẫu trong *l* (*x, θk*, t). Trong trường hợp này, tất cả các mẫu trong ℒk được gán trọng số dương và các mẫu không trong ℒk được gán bằng 0.

Với một cây hồi quy *Tk*, khi có giá trị thử nghiệm *X=x* thì giá trị dự đoán *Ŷk* tương ứng:

Trọng số *wi* (*x*) được tính bởi rừng ngẫu nhiên là giá trị trung bình của các trọng số dự đoán của tất cả các cây trong rừng. Công thức tính như sau:

Cuối cùng, giá trị dự đoán của rừng ngẫu nhiên hồi quy được cho bởi:

#### Độ đo sự quan trọng thuộc tính

Khi cây hồi quy phân chia tập dữ liệu đầu vào thành các vùng không giao nhau (theo hàng), giá trị dự đoán là giá trị trung bình được gán vào các vùng tương ứng (lá của cây).

Với mô hình rừng ngẫu nhiên, độ đo sự quan trọng của thuộc tính X được tính bằng cách lấy giá trị trung bình của tất cả các độ đo của các cây hồi quy độc lập. Có một điểm lợi trong việc tính độ đo sự quan trọng của thuộc tính dùng mô hình rừng ngẫu nhiên là độ đo của các biến có tương tác lẫn nhau đều được xem xét một cách tự động, điều này khác hẳn với những phương pháp tính tương quan tuyến tính như Kendall, Pearson. Độ đo sự quan trọng của thuộc tính X còn được tính theo cách khác là dùng phương pháp lặp hoán vị cho kết quả chính xác hơn, tuy nhiên thời gian tính toán lâu hơn do chạy nhiều lần rừng ngẫu nhiên trong tập dữ liệu mở rộng cỡ 2M chứa các biến giả.

Gọi *ISK*(*Xj*), *ISXj* lần lượt là độ đo sự quan trọng của thuộc tính *Xj* trong một cây hồi quy Tk (k=1, …, K) và trong một rừng ngẫu nhiên. Từ công thức (12) ta tính độ đo sự quan trọng *Xj* từ cây hồi quy độc lập như sau:

và từ rừng ngẫu nhiên là:

## Lỗi của mô hình

* Lỗi huấn luyện (training error): là lỗi của mô hình được tính toán trên tập huấn luyện.
* Lỗi kiểm định (validation error): là lỗi của mô hình được tính toán trên tập kiểm định (tập dữ dùng để đo lỗi của mô hình trong quá trình học nhưng không tham gia vào quá trình tính toán trọng số để học của mô hình).
* Lỗi kiểm thử (test error): là lỗi của mô hình được tính toán trên tập kiểm thử (tập dữ liệu đánh giá mô hình sau khi đã hoàn thành quá trình học).
* Lỗi khái quát (generalization error): là lỗi kỳ vọng của mô hình khi áp dụng trên một luồng vô hạn dữ liệu mới được lấy từ cùng một phân phối với các mẫu ban đầu.

## Phương pháp đánh giá độ chính xác của mô hình hồi quy

 Đánh giá độ chính xác của mô hình là một phần thiết yếu của quá trình tạo mô hình học máy để mô tả mô hình hoạt động tốt như thế nào trong các dự đoán của nó. Để đánh giá tính hiệu quả của mô hình, chúng tôi dùng các phương pháp sau: căn bình phương sai số (Root mean square error - RMSE), sai số tuyệt đối trung bình (mean absolute error - MAE) và hệ số xác định bội (coefficient of determination - R2). Trong đó:

* MAE thể hiện sự khác biệt giữa giá trị gốc và giá trị dự đoán được trích xuất bằng cách lấy trung bình chênh lệch tuyệt đối trên tập dữ liệu.
* MSE đại diện cho sự khác biệt giữa giá trị gốc và giá trị dự đoán được trích xuất bằng bình phương sự khác biệt trung bình trên tập dữ liệu.
* RMSE là tỷ lệ lỗi tính theo căn bậc hai của MSE.
* R2đại diện cho hệ số về mức độ phù hợp của các giá trị so với các giá trị ban đầu. Giá trị từ 0 đến 1 được hiểu là tỷ lệ phần trăm. Giá trị càng cao thì mô hình càng tốt.

Các chỉ số trên được diễn đạt thành các công thức:

Trong đó : Yi, và chỉ giá trị thực, giá trị dự đoán và giá trị trung bình của mẫu thứ i tương ứng. Mô hình hồi quy cho kết quả tốt là mô hình đạt được sai số RMSE và MAE nhỏ. Giá trị R2 càng cao cho thấy mô hình sử dụng để phân tích có khả năng giải thích càng tốt các khác biệt về mực nước. Giá trị MAE, RMSE càng thấp thì khả năng dự báo của mô hình càng cao.

# ỨNG DỤNG PHƯƠNG PHÁP VÀ XÂY DỰNG MÔ HÌNH

## Dữ liệu phục vụ xây dựng mô hình

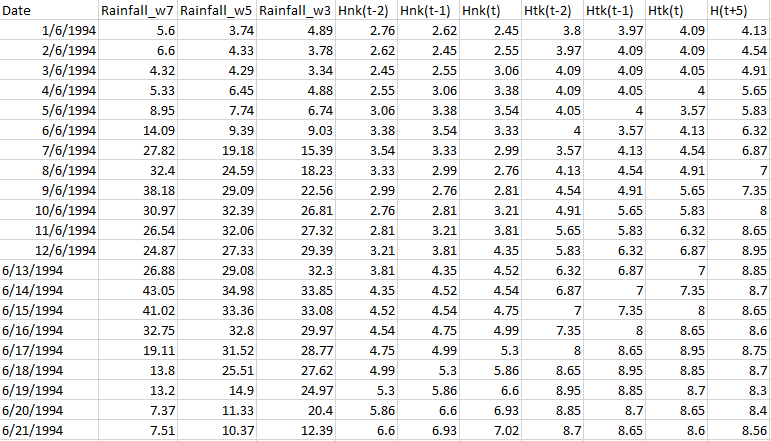
### Mô tả dữ liệu

Dữ liệu được dùng để xây dựng mô hình dự đoán mực nước tại trạm Thakhek được thu thập là bộ dữ liệu mùa lũ các năm từ gồm 1071 bản ghi.

Dữ liệu đầu vào L trong đó N là cỡ mẫu đo đạc được. Trong bài này mối quan hệ đầu vào – ra trong mô hình phi tuyền RF như sau:

HThakhek(t+5) = f{Htk(t), Htk(t-1), Htk(t-2), Hnk(t), Hnk(t-1), Hnk(t-2), Rainfall\_w3(t), Rainfall\_w5(t), Rainfall\_w7(t)}.

Biến đầu ra H(t+5) là mực nước dự báo cho 5 ngày tới tại trạm Thakhek. Htk(t), Htk(t-1) và Htk(t-2) lần lượt là mực nước đo được trong ngày hiện tại và hai ngày trước tại trạm Thakhek. Hnk(t), Hnk(t-1) và Hnk(t-2) lần lượt là mực nước đo được trong ngày hiện tại và hai ngày trước tại trạm Nông Khai, đây là trạm đo ở phía trên của Thakhek, cách Thakhek 300 km về phía thượng nguồn sông Mekong. Rainfall\_w3, Rainfall\_w5 và Rainfall\_w7 lần lượt là lượng mưa trung bình trên lưu vực gia nhập khu giữa Nông Khai và Thakhek cho các thời đoạn 3, 5 và 7 ngày gần đây nhất.

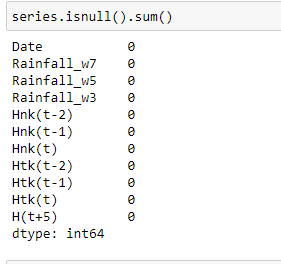


Hình 3.1 Một phần bản ghi của dữ liệu

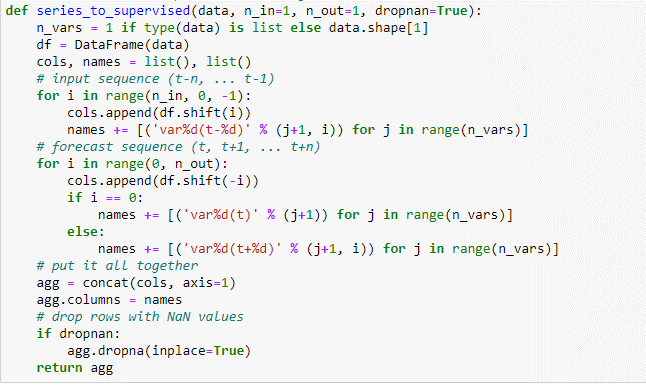
### Tiền xử lý dữ liệu

#### Chuẩn hóa dữ liệu đầu vào

Bộ dữ liệu trên qua kiểm tra cho thấy không có dữ liệu bị missing. Missing values hay còn gọi là những giá trị bị thiếu, không được điền hoặc không được cập nhật vào bộ dữ liệu.



#### Chuyển dữ liệu từ Time Series sang Supervised Learning



Hàm series\_to\_supervised (): nhận chuỗi thời gian đơn biến hoặc đa biến và đóng khung nó dưới dạng tập dữ liệu học tập có giám sát.

Hàm nhận bốn đối số:

+Dữ liệu: chuỗi quan sát dưới dạng danh sách hoặc mảng 2D NumPy.

+ n\_in: Số lần quan sát độ trễ dưới dạng đầu vào (X). Gía trị có thể nằm trong khoảng [1 … len (dữ liệu)]. Tùy chọn mặc định là 1.

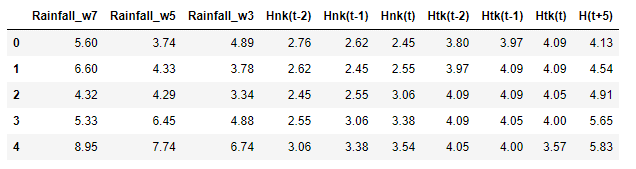
+ n\_out: Số lượng quan sát dưới dạng đầu ra (y). Gía trị nó có thể nằm trong khoảng [0 ...len (data-1)]. Và không bắt buộc mặc định là 1.

+ dropnan: Boolean có thả các hàng có giá trị NaN hay không không bắt buộc mặc định là True.

Hàm trả về một giá trị duy nhất:

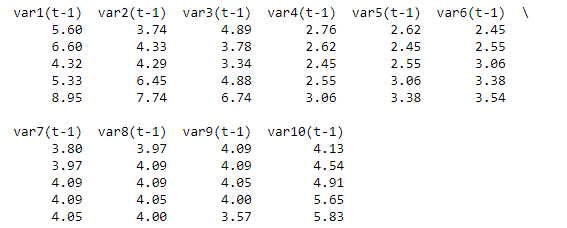
+ return: Pandas DataFrame của loạt series được đóng khung để học có giám sát.











Hình 3.2 Chuyển bộ dữ liệu từ Time Series sang Supervised Learning

Tập dữ liệu này gồm 9 biến đầu vào và 1 biến đầu ra. Sau quá trình tiền xử lý, tổng số bản ghi là 1071. Dữ liệu huấn luyện được lấy từ năm 1994-1997 gồm 612 bản ghi. Dữ liệu kiểm thử lấy từ năm 1998-2000 gồm 459 bản ghi được dùng để đánh giá hiệu quả của mô hình hồi quy phi tuyến RF.

## Các công cụ xây dựng mô hình RF

### Ngôn ngữ lập trình Python và thư viện

#### Giới thiệu Python

**Python** là một [ngôn ngữ lập trình](https://vi.wikipedia.org/wiki/Ng%C3%B4n_ng%E1%BB%AF_l%E1%BA%ADp_tr%C3%ACnh) bậc cao cho các mục đích lập trình đa năng, do [Guido van Rossum](https://vi.wikipedia.org/w/index.php?title=Guido_van_Rossum&action=edit&redlink=1) tạo ra và lần đầu ra mắt vào năm 1991. Python được thiết kế với ưu điểm mạnh là dễ đọc, dễ học và dễ nhớ. Python là ngôn ngữ có hình thức rất sáng sủa, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình và là ngôn ngữ lập trình dễ học; được dùng rộng rãi trong phát triển [trí tuệ nhân tạo](https://vi.wikipedia.org/wiki/Tr%C3%AD_tu%E1%BB%87_nh%C3%A2n_t%E1%BA%A1o). Cấu trúc của Python còn cho phép người sử dụng viết mã lệnh với số lần gõ phím tối thiểu. Vào tháng 7 năm 2018, van Rossum đã từ chức lãnh đạo trong cộng đồng ngôn ngữ Python sau 30 năm làm việc.

Python hoàn toàn [tạo kiểu động](https://vi.wikipedia.org/w/index.php?title=T%E1%BA%A1o_ki%E1%BB%83u_%C4%91%E1%BB%99ng&action=edit&redlink=1) và dùng cơ chế [cấp phát bộ nhớ tự động](https://vi.wikipedia.org/wiki/Qu%E1%BA%A3n_l%C3%BD_b%E1%BB%99_nh%E1%BB%9B); do vậy nó tương tự như [Perl](https://vi.wikipedia.org/wiki/Perl), [Ruby](https://vi.wikipedia.org/wiki/Ruby_(ng%C3%B4n_ng%E1%BB%AF_l%E1%BA%ADp_tr%C3%ACnh)), [Scheme](https://vi.wikipedia.org/wiki/Scheme), [Smalltalk](https://vi.wikipedia.org/wiki/Smalltalk), và [Tcl](https://vi.wikipedia.org/wiki/Tcl). Python được phát triển trong một dự án mã mở, do tổ chức phi lợi nhuận Python Software Foundation quản lý.

**Python là ngôn ngữ lập trình hướng đối tượng đơn giản, dễ học, mạnh mẽ, cấp cao.** Python có cấu trúc cú pháp ít hơn các ngôn ngữ khác.

* **Python được thông dịch**: Python được trình thông dịch xử lý trong thời gian chạy. Bạn không cần phải biên dịch chương trình của mình trước khi thực hiện nó. Nó tương tự với PERL và PHP.
* **Python là tương tác (Interactive)**: Tại một dấu nhắc Python (command line) bạn có thể tương tác trực tiếp với trình thông dịch để viết chương trình Python.
* **Python là hướng đối tượng**: Python hỗ trợ kỹ thuật lập trình hướng đối tượng hoặc kỹ thuật lập trình đóng gói mã trong các đối tượng.
* **Python là ngôn ngữ của người mới bắt đầu**: Python là ngôn ngữ tuyệt vời cho các lập trình viên mới bắt đầu và hỗ trợ phát triển một loạt các ứng dụng từ xử lý văn bản đơn giản, lập trình web, cho đến lập trình game.

#### Các thư viện sử dụng

Ngôn ngữ sử dụng là Python. Phiên bản 3.8.0



Tải python tại: <https://www.python.org/downloads/> chọn phiên bản cần cài đặt.

Ngôn ngữ python có hệ thống các gói rất phong phú, hỗ trợ nhiều lĩnh vực khác nhau, từ xây dựng ứng dụng, xử lý web, xử lý ảnh, xử lý text…

Sử dụng pip để tải các gói mới về từ internet.

Một số gói dành cho lập trình thông thường:

* Os: Xử lý file và tương tác với hệ điều hành.
* Networkx và igraph: làm việc với dữ liệu đồ thị, có thể làm việc với dữ liệu rất lớn (đồ thị hàng triệu đỉnh).
* Regular expressions: tìm kiếm mẫu trong dữ liệu text
* BeautifulSoup: trích xuất dữ liệu từ file HTML hoặc từ website
* NumPy (Numerical Python): là gói chuyên về xử lý dữ liệu số (nhiều chiều), gói cũng chứa các hàm số tuyến tính cơ bản, biến đổi fourier, sinh số ngẫu nhiên nâng cao, …
* SciPy (Scientific Python): dựa trên NumPy, cung cấp các công cụ mạnh cho khoa học và kỹ nghệ, chẳng hạn như biến đổi fourier rời rạc, đại số tuyến tính, tối ưu hóa và ma trận thưa.
* Matplotlib: chuyên sử dụng để vẽ biểu đồ, hỗ trợ rất nhiều loại biểu đồ khác nhau
* Pandas: chuyên sử dụng cho quản lý và tương tác với dữ liệu có cấu trúc, được sử dụng rộng rãi trong việc thu thập và tiền xử lý dữ liệu
* Sciket Learn: chuyên về học máy, dựa trên NumPy, SciPy và matplotlib, thư viện này có sẵn nhiều công cụ hiệu quả cho việc học máy và thiết lập mô hình thống kê chẳng hạn như các thuật toán phân lớp, hồi quy, phân cụm, giảm chiều dữ liệu.
* Statsmodels: cho phép người sử dụng khám phá dữ liệu, ước lượng mô hình thống kê và kiểm định
* Seaborn: dự trên matplotlib, cung cấp các công cụ diễn thị (visualization) dữ liệu thống kê đẹp và hiệu quả, mục tiêu của gói là sử dụng việc diễn thị như là trọng tâm của khám phá và hiểu dữ liệu
* Bokeh: để tạo ra các ô tương tác, biểu đồ tổng quan trên nền web, rất hiệu quả khi tương tác với dữ liệu lớn và trực tuyến
* Blaze: gói dựa trên NumPy và Pandas hướng đến dữ liệu phân tán hoặc truyền phát, là công cụ mạnh mẽ tạo diễn thị về dữ liệu cực lớn
* Scrapy: chuyên về thu thập thông tin trên web, rất phù hợp với việc lấy các dữ liệu theo mẫu
* SymPy: tính toán chuyên ngành dùng cho số học, đại số, toán rời rạc và vật lí lượng tử
* Theano: gói chuyên dùng tính toán hiệu quả các mảng nhiều chiều, sử dụng rộng rãi trong học máy
* TensorFlow: gói chuyên dùng cho học máy của Google, đặc biệt là các mạng thần kinh nhân tạo
* Keras: thư viện cấp cao chuyên về học máy, sử dụng Theano, TensorFlow hoặc CNTK làm phụ trợ.

##### Thư viện Numpy

NumPy là thư viện bổ sung của python, do không có sẵn ta phải cài đặt:

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy

pip install numpy



* Một số hệ thống python đã có sẵn numpy thì có thể bỏ qua bước này
* Để kiểm tra xem hệ thống đã cài numpy hay chưa là thử import gói xem có bị lỗi hay không: import numpy as np

Đặc điểm của Numpy:

* Kiểu dữ liệu phần tử con trong mảng phải giống nhau
* Mảng có thể có một hoặc nhiều chiều
* Các chiều được đánh thứ tự từ 0 trở đi
* Số chiều gọi là hạng (rank)
* Có đến 24 kiểu số khác nhau
* Kiểu ndarray là lớp chính xử lý dữ liệu mảng nhiều chiều
* Rất nhiều hàm và phương thức xử lí ma trận

##### Thư viện Matplotlib

“Matplotlib” là thư viện chuyên về vẽ biểu đồ, mở rộng từ numpy.

Có mục tiêu đơn giản hóa tối đa công việc vẽ biểu đồ để “chỉ cần vài dòng lệnh”

Hỗ trợ rất nhiều loại biểu đồ, đặc biệt là các loại được sử dụng trong nghiên cứu hoặc kinh tế như biểu đồ dòng, đường, tần suất (histograms), phổ, tương quan, errorcharts, scatterplots, …

Cấu trúc của matplotlib gồm nhiều phần, phục vụ cho các mục đích sử dụng khác nhau. Ngoài các API liên quan đến vẽ biểu đồ, matplotlib còn bao gồm một số interface: Object-Oriented API, The Scripting Interface (pyplot), The MATLAB Interface (pylab).

Các interface này giúp chúng ta thuận tiện trong việc thiết lập chỉ số trước khi thực hiện vẽ biểu đồ. Interface pylab hiện đã không còn được phát triển. Sử dụng Object-Oriented API hoặc trực tiếp các API của matplotlib sẽ cho phép can thiệp sâu hơn vào việc vẽ biểu đồ (hầu hết project sẽ không có nhu cầu này).

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

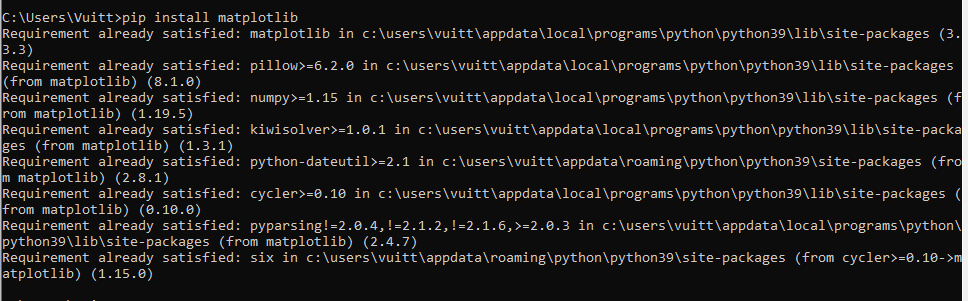
pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib

pip install matplotlib



##### Thư viện pandas

Để cài đặt thư viện ta gõ câu lệnh:

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

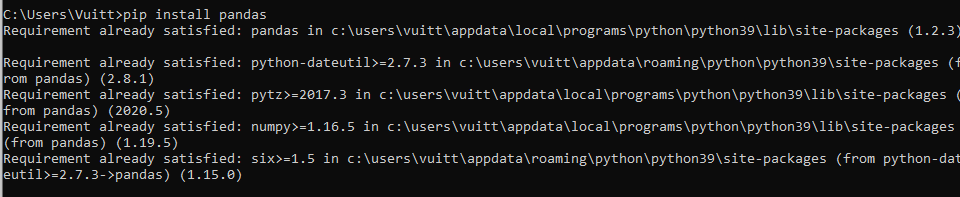
pip install pandas

pip install pandas

pip install pandas

pip install pandas

pip install pandas

“Pandas” là thư viện mở rộng từ numpy, chuyên để xử lý dữ liệu cấu trúc dạng bảng

Tên “pandas” là dạng số nhiều của “panel data”

Đặc điểm nổi bật của pandas:

* Đọc dữ liệu từ nhiều định dạng
* Liên kết dữ liệu và tích hợp xử lý dữ liệu bị thiếu
* Xoay và chuyển đổi chiều của dữ liệu dễ dàng
* Tách, đánh chỉ mục và chia nhỏ các tập dữ liệu lớn dựa trên nhãn
* Có thể nhóm dữ liệu cho các mục đích hợp nhất và chuyển đổi
* Lọc dữ liệu và thực hiện query trên dữ liệu
* Xử lý dữ liệu chuỗi thời gian và lấy mẫu

##### Thư viện scikit-learn

Scikit-learn xuất phát là một dự án trong một cuộc thi lập trình của Google vào năm 2007, người khởi xướng dự án là David Cournapeau.

Sau đó nhiều viện nghiên cứu và các nhóm ra nhập, đến năm 2010 mới có bản đầu tiên (v0.1 beta).

Scikit-learn cung cấp gần như tất cả các loại thuật toán học máy cơ bản (khoảng vài chục) và vài trăm biến thể của chúng, cùng với đó là các kĩ thuật xử lý dữ liệu đã được chuẩn hóa.

Để cài đặt ta gõ câu lệnh:

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

pip install scikit-learn

### Xây dựng web sử django

Sử dụng Framework Django đề lập trình đẩy mô hình học máy lên website.

#### Giới thiệu Django

**Django**là một **framework** bậc cao của Python có thể thúc đẩy việc phát triển phần mềm thần tốc và clean, thiết kế thực dụng. Được xây dựng bởi nhiều lập trình viên kinh nghiệm, Django tập trung lớn những vấn đề phát triển Web, bạn có thể phát triển trang web của bạn mà không cần xây dựng từ những căn bản. Đặc biệt nó **free**và **open source**.

Lịch sử của Django:

* **2003** - Bắt đầu bởi Adrian Holovaty và Simon Willison như một dự án nội bộ tại tờ báo Lawrence Journal-World.
* **2005** - Phát hành tháng 7 năm 2005 và đặt tên là Django, theo tên nghệ sĩ guitar jazz Django Reinhardt.
* **2005** - Đủ trưởng thành để xử lý một số trang web có lưu lượng truy cập cao.
* **Hiện tại** - Django hiện là một dự án mã nguồn mở với những người đóng góp trên khắp thế giới.

#### Những lợi thế của django

* **Hoàn thiện**: Django phát triển theo tư tưởng "Batteries included" (có thể hiểu ý nghĩa là tích hợp toàn bộ, chỉ cần gọi ra mà dùng). Nó cung cấp mọi thứ cho developer không cần phải nghĩ phải dùng cái ngoài. Chúng ta chỉ cần tập trung vào sản phẩm, tất cả đều hoạt động liền mạch với nhau.
* **Đa năng**: Django có thể được dùng để xây dựng hầu hết các loại website, từ hệ thống quản lý nội dung, cho đến các trang mạng xã hội hay web tin tức. Nó có thể làm việc với framework client-side, và chuyển nội dung hầu hết các loại format(HTML, RESS, JSON, XML, ...)
* **Bảo mật**: Django giúp các developer trang các lỗi bảo mật thông thường bằng cách cung cấp framework rằng có những kĩ thuật "phải làm như vậy" để bảo vệ website. Ví dụ: Django cung cấp bảo mật quản lý tên tài khoản và mật khẩu, tránh các lỗi cơ bản như để thông tin session lên cookie, mã hóa mật khẩu thay vì lưu thẳng.
* **Dễ Scale**: Django sử dụng kiến trúc [shared-nothing](https://www.howkteam.vn/redirect?Id=SDf1weZWe72xWllgybxU0So0lwQ5dDuM98%2bZWBcd5hQaIw80M%2b5bEuZHRjdRdnNDEJ%2brEMMWxL7VyCKgO1b%2bbw%3d%3d)dựa vào **component**(mỗi phần của kiến trúc sẽ độc lập với nhau, và có thể thay thế hoặc sửa đổi nếu cần thiết). Có sự chia tách rõ ràng giữa các phần nghĩa là nó có thể scale cho việc gia tăng traffic bằng cách thêm phần cứng ở mỗi cấp độ: caching, servers, database servers, hoặc application servers. Nhiều web về kinh doanh đã thành công khi Django được scale đáp ứng yêu cầu của họ
* **Dễ maintain**: code django được viết theo nguyên tắc thiết kế và pattern có thể khuyến khích ý tưởng bảo trì và tái sử dụng code. Trên thực tế, nó sự theo khái niệm Don't Repeat Yourself làm cho không có sự lặp lại không cần thiết, giảm một lượng code.
* **Tính linh động**: Django được viết bằng Python, nó có thể **chạy đa nền tảng.** Nó có nghĩa rằng bạn không ràng buộc một **platform server**cụ thể. Django được hỗ trợ tốt ở nhiều nhà cung cấp hosting, họ sẽ cung cấp hạ tầng và tài liệu cụ thể cho hosting web Django.

#### Cài đặt django

Thực hiện install Django thông qua trình quản lý package pip:

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

pip install django

Việc cài đặt có thể kéo dài trong một vài phút, tùy thuộc vào đường truyền mạng của chúng ta.

Sau khi thực hiện cài đặt xong, chúng ta kiểm tra lại xem việc cài đặt đã thành công chưa bằng câu lệnh:

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

pip freeze

#### Khởi tạo website với django

Bây giờ sẽ cài đặt website. Sau khi cài đặt xong django, chúng ta sẽ có một công cụ sử dụng trên command-line có tên là django-admin, công cụ này cho phép chúng ta thực hiện việc khởi tạo các website, application, migrate, shell, dbshell, ...

Để tạo ra một project, chúng ta dùng command:

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

django-admin startproject my\_url

Chúng ta sẽ có một thư mục my\_url bên trong chứa cấu trúc như bên dưới:



Folder my\_url/ ở bên ngoài là một thư mục chứa toàn bộ project của chúng ta. Tên của nó không quan trọng, Django framework không quản lý cái tên này, chúng ta có thể đổi tên nó thành các tên khác nhau nếu muốn.

manage.py: Django's command-line utility for administrative tasks. Nơi thực thi/truyền tải vào django-core các lệnh CLI từ django-admin.

Thư mục my\_url bên trong là thư mục chứa các module cần thiết để định nghĩa cho dự án của bạn.

my\_url/\_\_init\_\_.py: Là file trống để quy định my\_url là một package.

my\_url/settings.py: Chứa các cài đặt/cấu hình của một dự án Django.

my\_url/urls.py: Nơi khai báo các Path-URL chính của dự án Django.

my\_url/asgi.py: File cấu hình lưu trữ các thông tin dùng để thực hiện deloy lên website.

my\_url/wsgi.py: File cấu hình lưu trữ các thông tin dùng để thực hiện deploy lên webserver.

Với khung thông tin hiện tại, chúng ta đã có thể thực hiện start website của mình lên để kiểm tra. Câu lệnh khởi động là:

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

python manage.py runserver

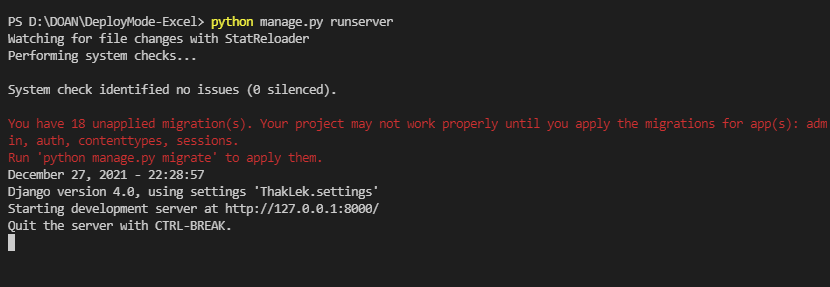
python manage.py runserver

python manage.py runserver

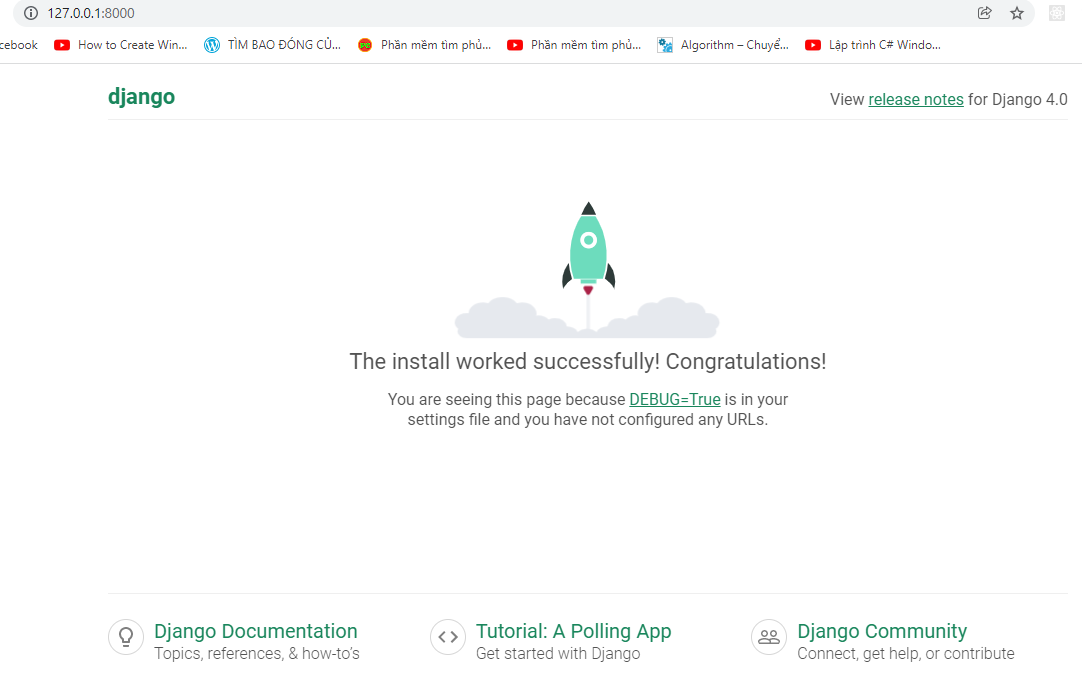
python manage.py runserver

python manage.py runserver

python manage.py runserver



Đến đây là chúng ta đã khởi tạo một website đang chạy trên localhost và cổng 8000.  
Chúng ta sẽ thực hiện truy cập vào địa chỉ URL phía trên (http://127.0.0.1:8000) và xem kết quả hiện ra.



Để thực hiện tắt ứng dụng, chúng ta thực hiện gõ CTRL - BREAK trên màn hình đen terminal.  
Để ý thấy giá trị 8000 là giá trị mặc định của server port khi thực hiện khởi tạo webserver, giá trị này có thể thay đổi được bằng cách khi khởi động webserver, thêm giá trị port vào phía sau:

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

python manage.py runserver 8080

## Ứng dụng dữ liệu và công cụ vào để xây dựng mô hình và web

### Xây dựng mô hình học máy

#### Import các thư viện sử dụng

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

#### Đọc dữ liệu và kiểm tra những dữ liệu bị missing

Dữ liệu đầu vào là một file excel có tên Thakek-Mekong.xlsx. Sau đó ta kiểm tra xem bộ dữ liệu có chỗ nào bị missing không, và đưa ra tổng số dữ liệu nếu bị missing.

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

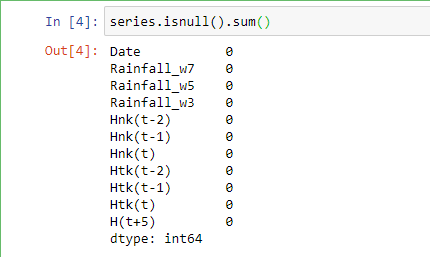
series.head()

#data process

series.isnull()

series.isnull().sum()

Sau khi kiểm tra bộ dữ liệu kết quả thu được là không có dữ liệu bị missing.



#### Chuyển đổi dữ liệu từ Timeseries sang Supervised Learing

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

e=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

#### Chia bộ dữ liệu thành train/test

Hàm này được dùng để chia bộ dữ liệu thành 2 phần là train và test

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

#### Trainning model RF và sử dụng mô hình đưa ra dự đoán

Hàm này để xây dựng mô hình RF để đưa ra dự đoán đối với dữ liệu đầu vào là dữ liệu test. Model RF được xây dựng với số lượng cây nhị phân hồi quy trong RF được đặt mặc định là 1000 cây, số lượng biến chọn ngẫu nhiên để tách nút trong cây hồi quy max\_features =3

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

#### Hàm tính toán độ chính xác của mô hình và giá trị dự đoán

Để kiểm tra độ chính xác của mô hình ta dùng các chỉ số MAE, RMSE, R2 và đưa ra các giá trị dự đoán cho tập dữ liệu được đưa vào để dự đoán.

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

#### Sử dụng các hàm trên

Đọc bộ dữ liệu và gọi hàm chuyển đổi dữ liệu từ chuỗi thời gian sang học có giám sát.

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

Gọi hàm để đưa ra độ chính xác của mô hình và show biểu đồ để quan sát giá trị đầu ra được mô hình dự đoán so với giá trị đầu ra thực tế.Các chỉ số để dự đoán độ chính xác của mô hình được làm tròn 3 chữ số sau dấu phẩy

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()

### Deploy mô hình học máy lên web sử dụng django

Sau khi đã xây dựng được mô hình ở phần [*3.1.1*](#_Xây_dựng_mô) chúng ta save weights mô hình bằng joblib. Được đặt tên là finalized\_model.sav

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

joblib.dump(model,filename)

model.fit(trainX, trainy)

filename ='finalized\_model.sav'

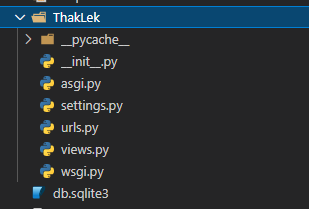
joblib.dump(model,filename)

model.fit(trainX, trainy)

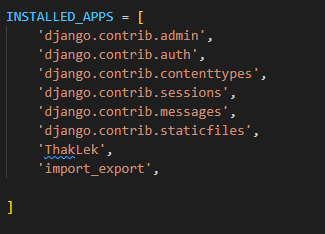
filename ='finalized\_model.sav'

joblib.dump(model,filename)

Sau đó tạo app có tên là ThakLek



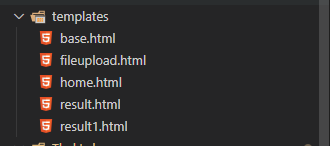
Ta vào thư mục settings.py, mở file **settings.py. Ở phần khai báo INSTALLED\_APPS** ta ghi thêm tên app vào trong.



Trong thư mục vừa tạo ta cần quan tâm chủ yếu tới 2 file là views.py và urls.py

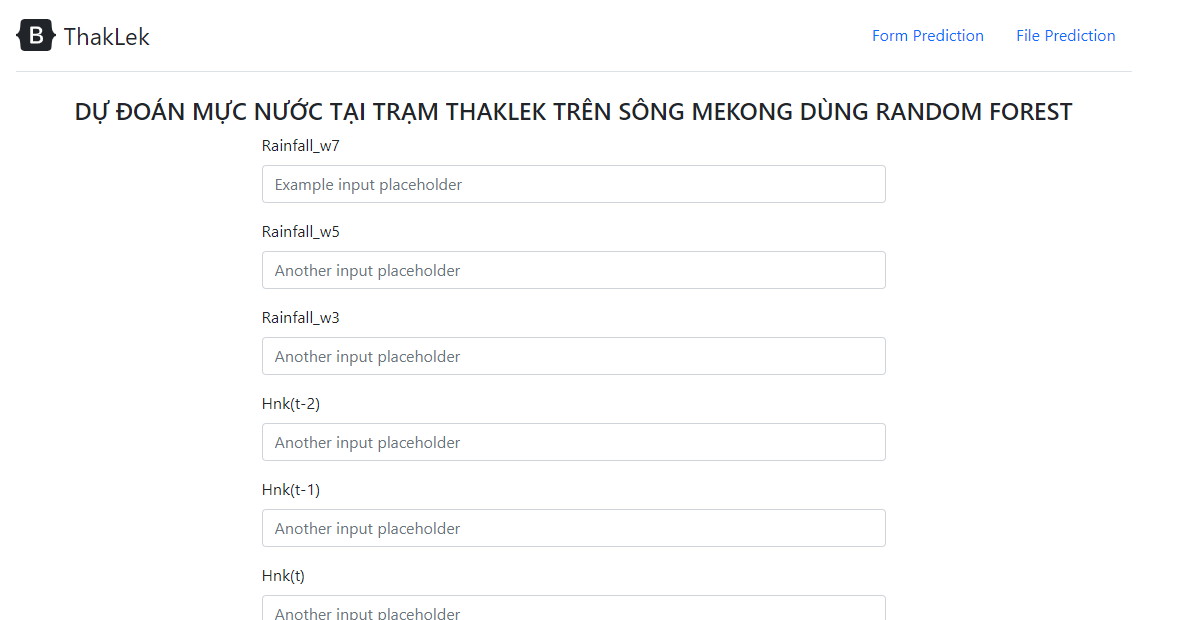
* views.py: Thực hiện chức năng như là 1 Controller trong MVC, nhận các yêu cầu, xử lý yêu cầu lấy data từ database sau đó trả về view (HTML, CSS, JS).
* url.py: Là file chứa các đường dẫn được chỉ định.

Chúng ta tạo thêm một thư mục đặt tên là templates là nơi chứa code giao hiện HMTL



* home.html: nơi chứa trang giao diện chính
* base.html: nơi chứa navigation của trang web
* result.html: giao diện khi kết quả được trả về sau khi nhập dữ liệu cần dữ đoán vào ô input
* result1.html: giao diện kết quả trả về đã được dự đoán khi chúng ta tải một file excel là dữ liệu đầu vào.

#### Xử lý đưa ra kết quả dự đoán khi người dùng nhập dữ liệu vào ô input



Đầu tiên chúng ta sẽ xây dựng giao diện để người dùng nhập dữ liệu vào ô input là dữ liệu đầu vào để mô hình đưa ra dự đoán mực nước tại trạm ThahLek. Trong file home.html. Sử dụng một form để có thể lấy được dữ liệu từ ô input khi người dùng ấn submit.



Sau khi người dùng nhập đầy đủ thông tin về dữ liệu đầu vào tại ô input và ấn submit thì dữ liệu sẽ được chuyển đến chuyển đến file views.py và gọi tới hàm result1 để xử lý

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

def result1(request):

cls=joblib.load('finalized\_model.sav')

lis=[]

lis.append(request.GET['w7'])

lis.append(request.GET['w5'])

lis.append(request.GET['w3'])

lis.append(request.GET['hnk2'])

lis.append(request.GET['hnk1'])

lis.append(request.GET['hnk'])

lis.append(request.GET['htk2'])

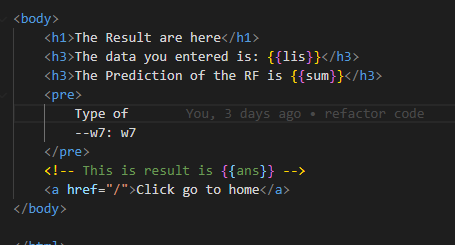
lis.append(request.GET['htk1'])

lis.append(request.GET['htk'])

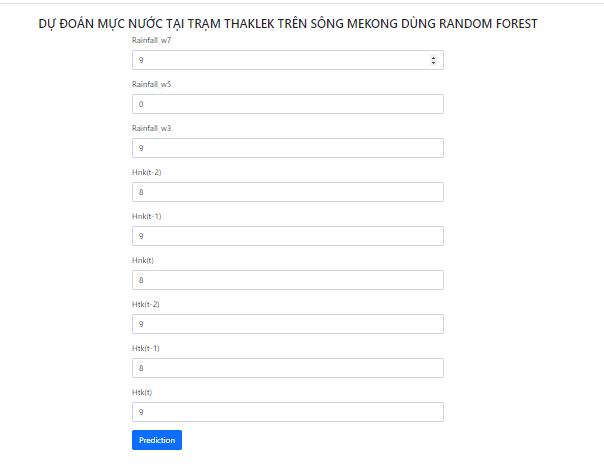
sumation=cls.predict([lis])

return render(request,"result1.html",{'something':True,'sum':sumation,'lis':lis})

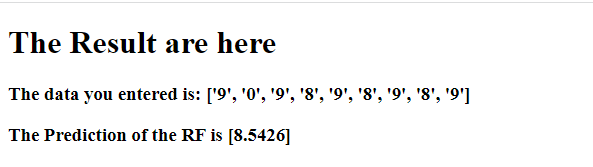
Để lấy được dữ liệu đã nhập từ ô input tại trang home.html. Chúng ta dùng phương thức GET và lưu các biến dữ liệu truyền lên vào trong một mảng lis. Sau đó sử dụng thư vện joblib để load file “sav” đã được lưu lại từ quá trình xây dựng mô hình học máy để đưa ra dự đoán cho dữ liệu mới. Kết quả đã dự đoán sẽ được trả về lên giao diện trong file result1.html



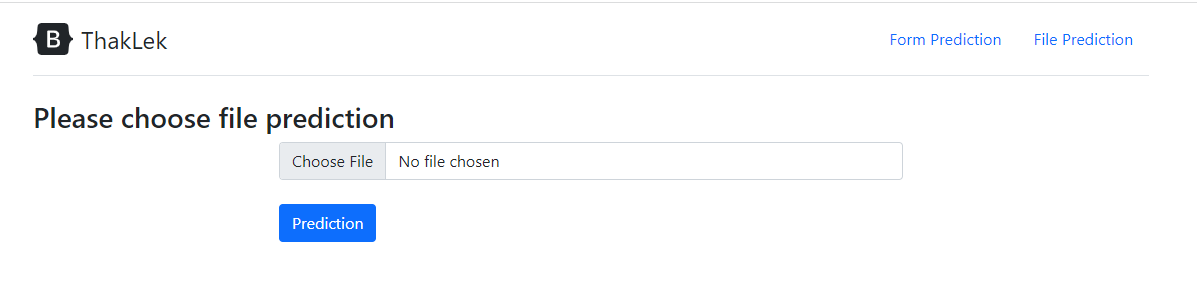
Demo kết quả thu được từ việc xử lý dữ liệu khi người dùng nhập dữ liệu từ ô input



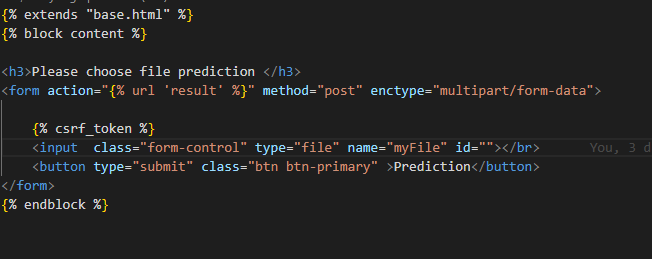
Sau đó ấn nút Prediction. Kết quả thu được



#### Xử lý đưa ra kết quả dự đoán khi người dùng tải lên file excel



Code giao diện cho phần xử lý khi người dùng muốn dự đoán bằng việc tải file excel được để trong “fileupload.html”.



Khi người dùng chọn đúng loại file là excel thì tại trang hiện tại có đường dẫn là [*http://127.0.0.1:8000/upload/*](http://127.0.0.1:8000/upload/) sau khi ấn nút “Prediction” sẽ được chuyển hướng đến trang có đường dẫn [*http://127.0.0.1:8000/result/*](http://127.0.0.1:8000/result/). Việc quy định đường dẫn đã được xử lý tại file “urls.py”

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

from django.contrib import admin

from django.urls import path,include

from .import views

urlpatterns = [

path('admin/', admin.site.urls),

path('',views.home),

path('upload/',views.upload,name='upload'),

path('result1/',views.result1,name='result1'),

path('result/',views.simple\_upload,name='result'),

]

Khi được chuyển đến đường dẫn [*http://127.0.0.1:8000/result/*](http://127.0.0.1:8000/result/)thì thực chất bên dưới nó là quá trình xử lý của file views.py và trả về kết quả hiển thị trên web có đường dẫn /result.

Bên trong file “views.py” quá trình đọc file excel và để đưa ra dự đoán từ dữ liệu đầu vào là file excel được thực hiện như sau:

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

class NumpyArrayEncoder(json.JSONEncoder):

def default(self, obj):

if isinstance(obj, np.integer):

return int(obj)

elif isinstance(obj, np.floating):

return float(obj)

elif isinstance(obj, np.ndarray):

return obj.tolist()

else:

return super(NumpyArrayEncoder, self).default(obj)

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

else:

return render(request,"result.html")

def simple\_upload(request):

if request.method == "POST":

file= request.FILES["myFile"]

# dataset=Dataset()

imported\_data = pd.read\_excel(file,usecols=[1,2,3,4,5,6,7,8,9],engine='openpyxl')

data\_excel= pd.read\_excel(file,engine='openpyxl')

values = imported\_data.values

# print(values)

# transform the time series data into supervised learning

data1 = series\_to\_supervised(values,1,1)

data1.drop(data1.columns[[9,10,11,12,13, 14, 15, 16,17]], axis=1, inplace=True)

if not file.name.endswith("xlsx"):

messages.info(request,'wrong format')

return render(request,"result.html")

data1 = data1.values

x1 = [x for x in data1]

cls=joblib.load('finalized\_model.sav')

predict=cls.predict(x1).round(decimals=2)

encodedNumpyData = json.dumps(predict, cls=NumpyArrayEncoder)

data\_predictions = json.loads(encodedNumpyData)

json\_records = data\_excel.reset\_index().to\_json(orient ='records')

data\_input = []

data\_input = json.loads(json\_records)

# print(type(data\_input))

mylist=zip(data\_input,data\_predictions)

print(mylist)

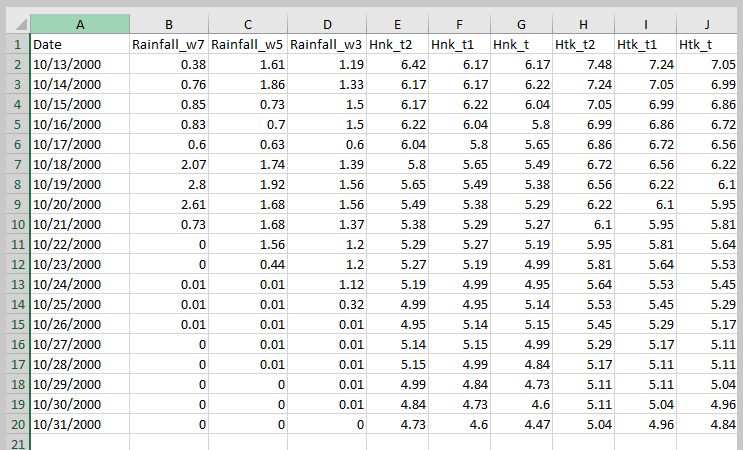
return render(request,"result.html",{'something':True,'predict':data\_predictions,'d':data\_input,'pr':mylist})

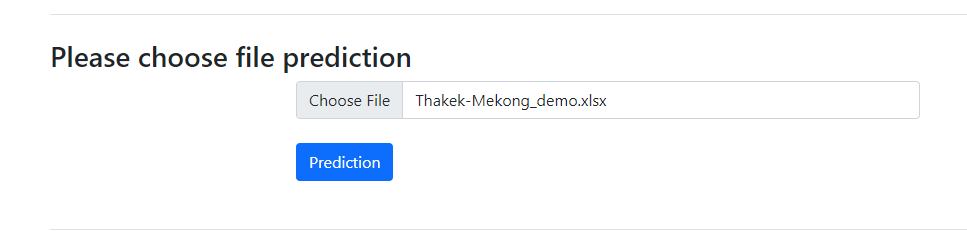
else:

return render(request,"result.html")

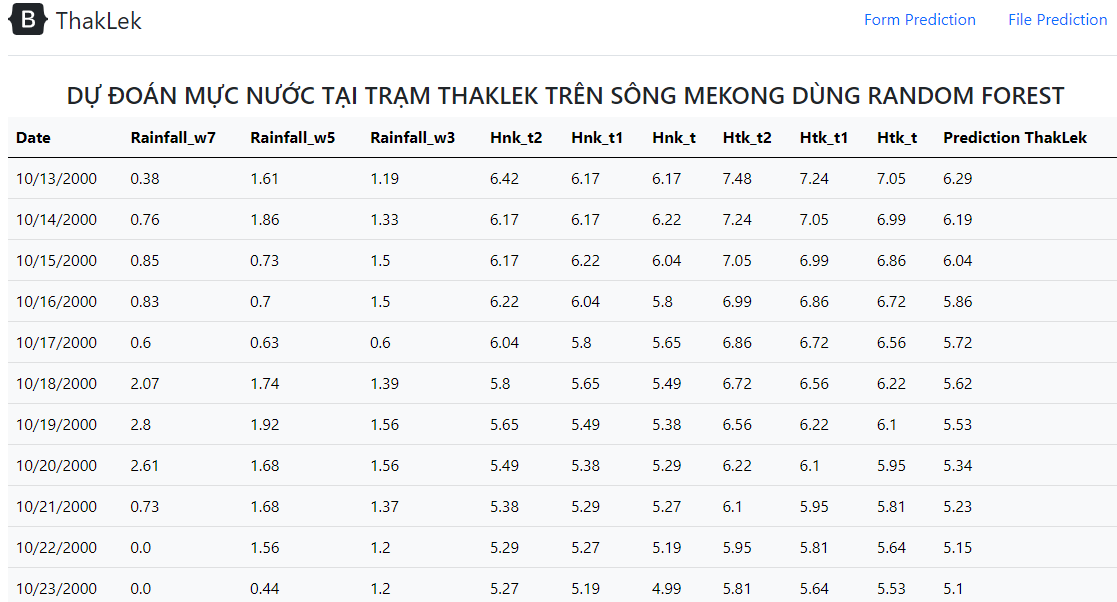
Demo kết quả thu được.

File người dùng tải nên là một file excel có dữ liệu như sau:





Khi người dùng ấn Prediction. Kết quả dự đoán mực nước tại trạm ThahLek thu được được hiển thị là cột cuối ở bảng bên dưới



# KẾT QUẢ ỨNG DỤNG MÔ HÌNH

## Qúa trình huấn luyện mô hình

### Các thông số huấn luyện

Dưới đây là các thông số được khai báo trước khi đưa vào chương trình python để huấn luyện mô hình. Các thông số này đã được điều chỉnh để tối ưu với bài toán dự báo mực nước.

Bảng 4. Các thông số huấn luyện

|  |  |  |
| --- | --- | --- |
| **Tên tham số** | **Giá trị** | **Mô tả** |
| K |  | Số lượng cây nhị phân hồi quy |
| Max\_feature |  | Số lượng biến chọn ngẫu nhiên để tách nút trong cây hồi quy |

### Môi trường huấn luyện

Môi trường được sử dụng để huấn luyện mô hình RF là Windows 10, ngôn ngữ Python 3.9.1 và trình biên tập lập trình VS Code 1.63.

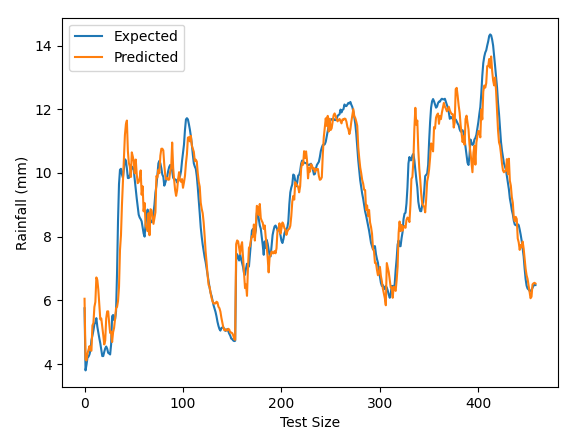
### Đánh giá mô hình

Các đại lượng được sử dụng để huấn luyện, kiểm định và kiểm tra mô hình là:

* Độ chính xác:

### Quá trình huấn luyện

Dưới đây là đồ thị quá trình huấn luyện của mô hình RF gồm đường mực nước dự báo và thực đo tại trạm ThahLek.



Hình 4. Đường quá trình dự báo mực nước và thực đo tại trạm ThahLek

## Kết quả ứng dụng mô hình

### Kết quả huấn luyện

Mô hình mất khoảng 15 phút để chuẩn hóa và học dữ liệu. Với độ chính xác các chỉ số thu được như sau: MAE =0.484, RMSE =0.649, R2=0.925. Ta có giá trị R2 thu được khá cao gần so với 1. Ngoài ra theo khuyến nghị của Ủy ban sông Mê-Kông [6], giá trị MAE chấp nhận được cho dự báo trước 5 ngày tại vị trí Thakhek là nhỏ hơn 0.75 mét. Từ giá trị MAE thu được < 0.75. Vì vậy cho thấy mô hình RF hoạt động tốt và cho ra kết quả có độ chính xác tương đối cao.

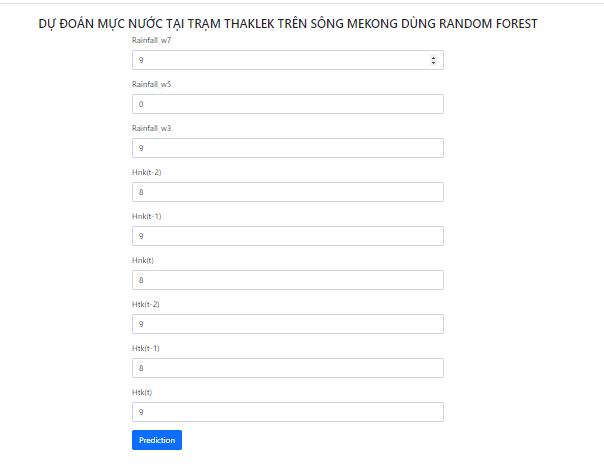


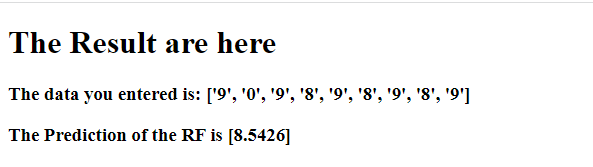
Hình 4. 2 Độ chính xác mô hình

### Kết quả deploy mô hình nên web

Mô hình sau khi được dùng để triển khai trên web có thể đưa ra dự đoán mực nước tại trạm ThahLek cho người dùng theo 2 cách.

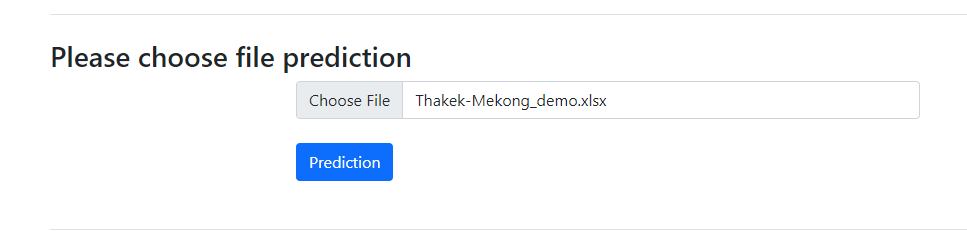
* Khi người dùng nhập dữ liệu đầu vào từ ô input

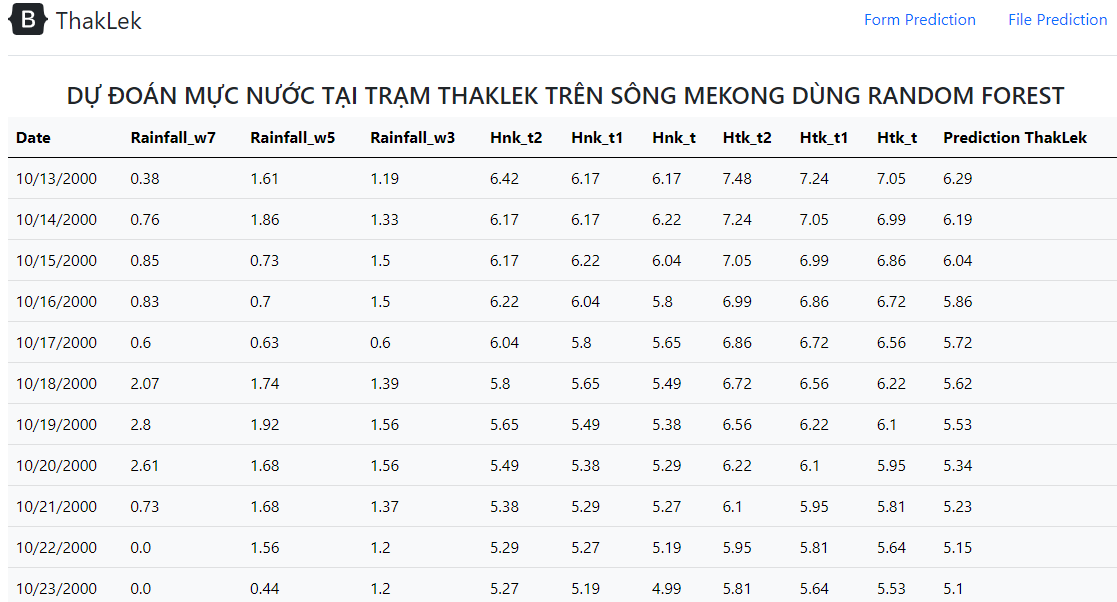




Hình 4.3 Giao diện web dùng dự đoán khi người dùng nhập dữ liệu và giao diện kết quả dự đoán

* Khi người dùng tải lên một file excel đã có dữ liệu gồm mực nước đo được trong ngày hiện tại và hai ngày trước tại trạm Thakhek. Hnk(t), Hnk(t-1) và Hnk(t-2) lần lượt là mực nước đo được trong ngày hiện tại và hai ngày trước tại trạm Nông Khai. Rainfall\_w3, Rainfall\_w5 và Rainfall\_w7 lần lượt là lượng mưa trung bình trên lưu vực gia nhập khu giữa Nông Khai và Thakhek cho các thời đoạn 3, 5 và 7 ngày gần đây nhất.





Hình 4. 4 Giao diện web dùng dự đoán khi người dùng tải file dữ liệu và kết quả dự đoán trên web

## Kết luận & kiến nghị

Mô hình dự báo rừng ngẫu nhiên (RF) được đề xuất trong nghiên cứu này nhằm dự báo mực nước tại trạm ThahLek trên sông MeKong và dựa trên phương pháp đánh giá R2, RMSE và MAE. Kết quả thực nghiệm cho thấy mô hình rừng ngẫu nhiên cho kết quả dự đoán với độ chính xác cao. Trong tương lai, chúng tôi sẽ áp dụng kết quả nghiên cứu mở rộng, áp dụng cùng với các mô hình học máy khác để tìm ra mô hình tối ưu cùng kết quả mong muốn có độ chính xác tương đối cao.

TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | P. K.-T. L. H.-C. Nguyen, "Flood forecasting in large rivers with data-driven models," *Natural hazards 71.1 ,* pp. 767-784, 2014. |
| [2] | V. V. LUÂN, "RỪNG NGẪU NHIÊN CẢI TIẾN," ĐHQG-Hà Nội, 2017. |
| [3] | L. Breiman, Bagging predictors., Bagging predictors. , 1996. |
| [4] | L. NikhilR.Pal, Advanced Techniques in Knowledge Discovery and DataMining, 2005. |
| [5] | L. Breiman, "Random forests. Machine Learning," vol. 45, no. 1, pp. 5-32, 2001. |
| [6] | Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen, Classification and regression trees, CRC press, 1984. |
| [7] | Thanh-Tung Nguyen, JoshuaZ. Huang, and ThuyThi Nguyen, "Two-level quantile regression forests for bias correction in range prediction. Machine Learning," 2014, pp. 1-19. |
| [8] | Thanh-Tung Nguyen, Joshua Z Huang, Qingyao Wu, Thuy T Nguyen, and Mark J Li, Genome-wide association data classification and snps selection using two-stage quality-based random forests, BMC Genomics, 2015. |
| [9] | V. H. Tiệp, Machine learning cơ bản, Nhà xuất bản Khoa học và Kỹ thuật, 2018. |
| [10] | T. X. Nam, "txnam.net," 2018. [Online]. Available: https://txnam.net/course/nhap-mon-lap-trinh-khoa-hoc-du-lieu-k58. |
| [11] | "Django," [Online]. Available: https://www.tutorialspoint.com/django/index.html. |
| [12] | "DataTechNotes," [Online]. Available: https://www.datatechnotes.com/2019/02/regression-model-accuracy-mae-mse-rmse.html. |
| [13] | "CODELEARN," [Online]. Available: https://codelearn.io/sharing/web-step-by-step-voi-django. |
| [14] | "pandas," [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html. |
| [15] | "Marchine Learning Mastery," [Online]. Available: https://machinelearningmastery.com/time-series-prediction-with-deep-learning-in-python-with-keras/. |
| [16] | "Python," [Online]. Available: https://www.python.org/. |

PHỤ LỤC

Các dòng lệnh chính của ngôn ngữ lập trình Python được sử dụng khi tiến hành thực nghiệm.

from numpy import asarray

from pandas import read\_excel

from pandas import DataFrame

from pandas import concat

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_error #RMSE

from sklearn.metrics import r2\_score #R-squared (R^2)

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',engine='openpyxl')

series.head()

#data process

series.isnull()

series.isnull().sum()

# convert series to supervised learning

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

# split a univariate dataset into train/test sets

def train\_test\_split(data, n\_test):

return data[:-n\_test, :], data[-n\_test:, :]

# fit an random forest model and make a one step prediction

def random\_forest\_forecast(train, testX):

# transform list into array

train = asarray(train)

# split into input and output columns

trainX, trainy = train[:, :-1], train[:, -1]

# fit model

model = RandomForestRegressor(n\_estimators=1000, max\_features=3)

# random\_state=0, n\_estimators=200, max\_depth=None, max\_features=1, min\_samples\_leaf=1, min\_samples\_split=2, bootstrap=False

model.fit(trainX, trainy)

# make a one-step prediction

yhat = model.predict([testX])

return yhat[0]

# walk-forward validation for univariate data

def walk\_forward\_validation(data, n\_test):

predictions = list()

# split dataset

train, test = train\_test\_split(data, n\_test)

# seed history with training dataset

history = [x for x in train]

# step over each time-step in the test set

for i in range(len(test)):

# split test row into input and output columns

testX, testy = test[i, :-1], test[i, -1]

# fit model on history and make a prediction

yhat = random\_forest\_forecast(history, testX)

# store forecast in list of predictions

predictions.append(yhat)

# add actual observation to history for the next loop

history.append(test[i])

# summarize progress

print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

# estimate prediction error (MAE)

error = mean\_absolute\_error(test[:, -1], predictions)

#caculator RMSE

mse=mean\_squared\_error(test[:, -1], predictions)

rmse = sqrt(mse)

#caculator R-squared (R^2)

global r2\_score

r2\_score=r2\_score(test[:, -1], predictions)

return error,rmse,r2\_score, test[:, -1], predictions

# load the dataset

series = read\_excel('Thakek-Mekong.xlsx',usecols=[1,2,3,4,5,6,7,8,9,10,11],engine='openpyxl')

series.head()

values = series.values

# transform the time series data into supervised learning

data = series\_to\_supervised(values,1,1)

data.drop(data.columns[[10,11,12,13, 14, 15, 16, 17, 18,19]], axis=1, inplace=True)

data = data.values

# evaluate

mae,rmse,r2\_score, y, yhat = walk\_forward\_validation(data,459) #int(len(data) \* 0.3)

print('MAE: %.3f' % mae)

print('Test RMSE: %.3f' % rmse)

print('Test R^2: %.3f' % r2\_score)

# plot expected vs predicted

pyplot.plot(y, label='Expected')

pyplot.plot(yhat, label='Predicted')

pyplot.legend()

pyplot.xlabel('Test Size')

pyplot.ylabel('Rainfall (mm)')

pyplot.show()