TRƯỜNG ĐẠI HỌC ĐẠI NAM

KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN

HỌC PHẦN: DỮ LIỆU LỚN

ĐỀ TÀI: DỰ ĐOÁN GIÁ BẤT ĐỘNG SẢN TẠI USA

Giảng viên: Trần Quý Nam

TT	Mã sv	Họ và Tên	Ngày Sinh	Lớp
1	1671020357	Phan Đình Quang Vinh	14/10/2004	CNTT16-02
2	1671020250	Vũ Triệu Phú	16/12/2004	CNTT16-02
3	1671020162	Nguyễn Đình Khánh	01/08/2004	CNTT16-02
4	1671020127	Ngô Hữu Hoàng	19/05/2004	CNTT16-02

Hà Nội, năm 2025

TRƯỜNG ĐẠI HỌC ĐẠI NAM

KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN

HỌC PHẦN: DỮ LIỆU LỚN

ĐỀ TÀI: DỰ ĐOÁN GIÁ BẤT ĐỘNG SẢN TẠI USA

TT	Mã sv	Họ và Tên	Ngày Sinh	Điểm	
				Bằng Số	Bằng Chữ
1	1671020357	Phan Đình Quang Vinh	14/10/2004		
2	1671020250	Vũ Triệu Phú	16/12/2004		
3	1671020162	Nguyễn Đình Khánh	01/08/2004		
4	1671020127	Ngô Hữu Hoàng	19/05/2004		

CÁN BỘ CHẤM THI 1 **CÁN BỘ CHẤM THI 2**

Trần Quý Nam **Lê Thị Thùy Trang**

Hà Nội, năm 2025

LỜI NÓI ĐẦU

Trong những năm gần đây, thị trường bất động sản tại Mỹ đã có nhiều biến động mạnh mẽ do ảnh hưởng của các yếu tố kinh tế, xã hội và công nghệ. Việc dự đoán giá nhà đất không chỉ là một bài toán quan trọng đối với các nhà đầu tư, môi giới bất động sản mà còn hỗ trợ các cá nhân và tổ chức đưa ra quyết định mua bán, cho thuê một cách chính xác hơn.

Nhờ sự phát triển của khoa học dữ liệu và trí tuệ nhân tạo, các mô hình máy học ngày càng trở nên hiệu quả trong việc phân tích các yếu tố ảnh hưởng đến giá nhà, bao gồm vị trí địa lý, diện tích, số phòng, tình trạng nhà ở, xu hướng thị trường và nhiều yếu tố khác. Sự kết hợp giữa dữ liệu lớn và thuật toán hiện đại giúp cải thiện độ chính xác của dự đoán, mang lại những lợi ích thiết thực trong việc hoạch định chiến lược đầu tư bất động sản.

Mục tiêu của nghiên cứu này là xây dựng một mô hình dự đoán giá nhà đất tại Mỹ bằng cách sử dụng các thuật toán học máy tiên tiến. Bằng cách phân tích dữ liệu thực tế, tối ưu hóa mô hình và đánh giá hiệu suất, nghiên cứu này sẽ cung cấp một cách tiếp cận khoa học để dự báo giá nhà, giúp người dùng có cái nhìn rõ ràng hơn về xu hướng thị trường và đưa ra quyết định hợp lý.

MỤC LỤC

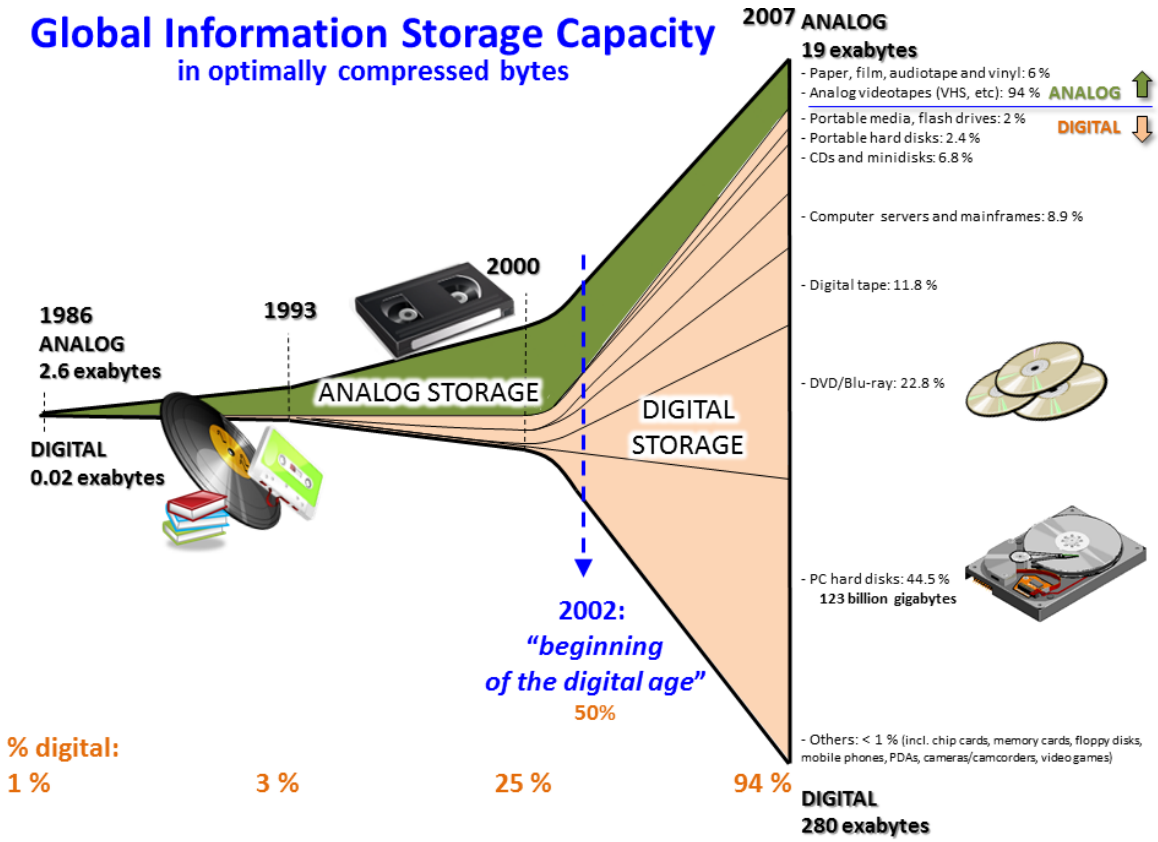
MỤC LỤC HÌNH ẢNH

CHƯƠNG 1. TỔNG QUAN CƠ SỞ LÝ THUYẾT

1.1 Dữ liệu lớn

1.1.1 Khái niệm của dữ liệu lớn

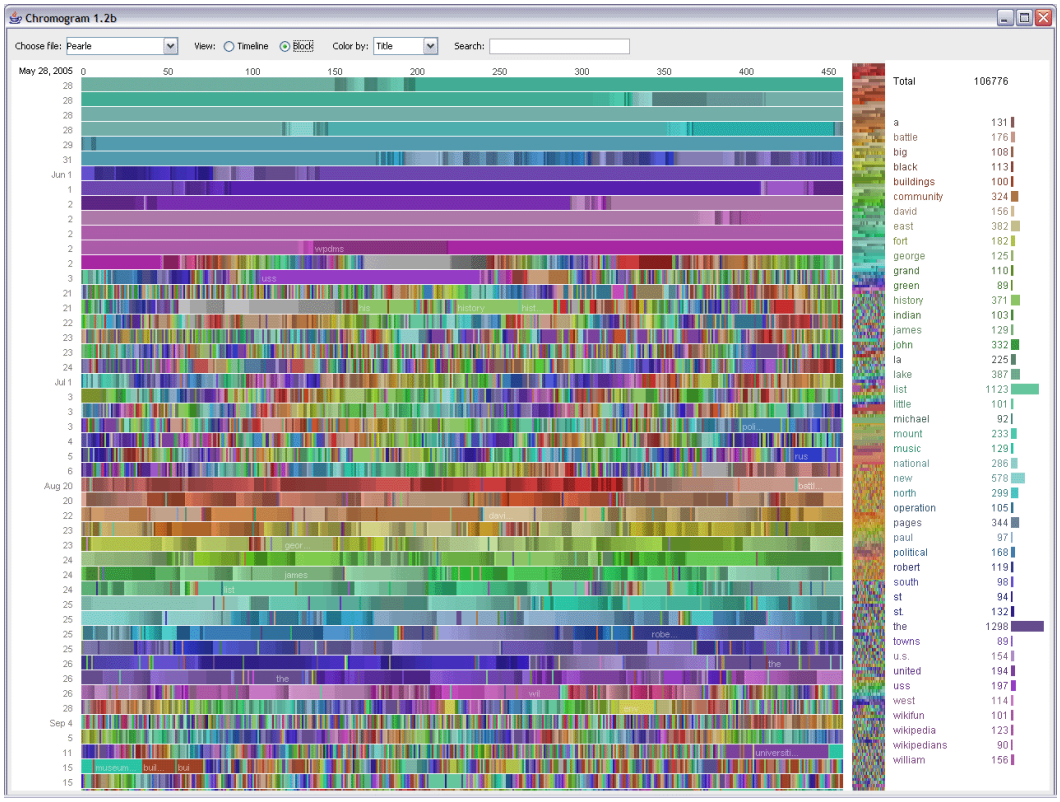
Dữ liệu lớn là một thuật ngữ cho việc xử lý một tập hợp dữ liệu rất lớn và phức tạp mà các ứng dụng xử lý dữ liệu truyền thống không xử lý được. Dữ liệu lớn bao gồm các thách thức như phân tích, thu thập, giám sát dữ liệu, tìm kiếm, chia sẻ, lưu trữ, truyền nhận, trực quan, truy vấn và tính riêng tư. Thuật ngữ này thường chỉ đơn giản đề cập đến việc việc sử dụng các phân tích dự báo, phân tích hành vi người dung.



Hình 1.1 Sự tăng trưởng, số hóa khả năng lưu trữ thông tin toàn cầu

Dữ liệu lớn thường bao gồm tập hợp dữ liệu với kích thước vượt xa khả năng của các công cụ phần mềm thông thường để thu thập, hiển thị, quản lý và xử lý dữ liệu trong một thời gian có thể chấp nhận được. Kích thước dữ liệu lớn là một mục tiêu liên tục thay đổi. Như năm 2012 thì phạm vi một vài tá terabytes `tới nhiều petabytes dữ liệu. Dữ liệu lớn yêu

cầu một tập các kỹ thuật và công nghệ được tích hợp theo hình thức mới để khai phá từ tập dữ liệu đa dạng, phức tạp, và có quy mô lớn.

i) 

Hình 1.2 Hình ảnh trực quan của dữ liệu lớn

1.1.2 Đặc trưng của dữ liệu lớn

Big Data được mô tả bởi những đặc trưng sau:

Volume (Dung lượng) :

Số lượng dữ liệu được tạo ra và lưu trữ. Kích thước của dữ liệu xác định giá trị và tiềm năng insight- và liệu nó có thể thực sự được coi là dữ liệu lớn hay không.

Variety (Tính đa dạng) :

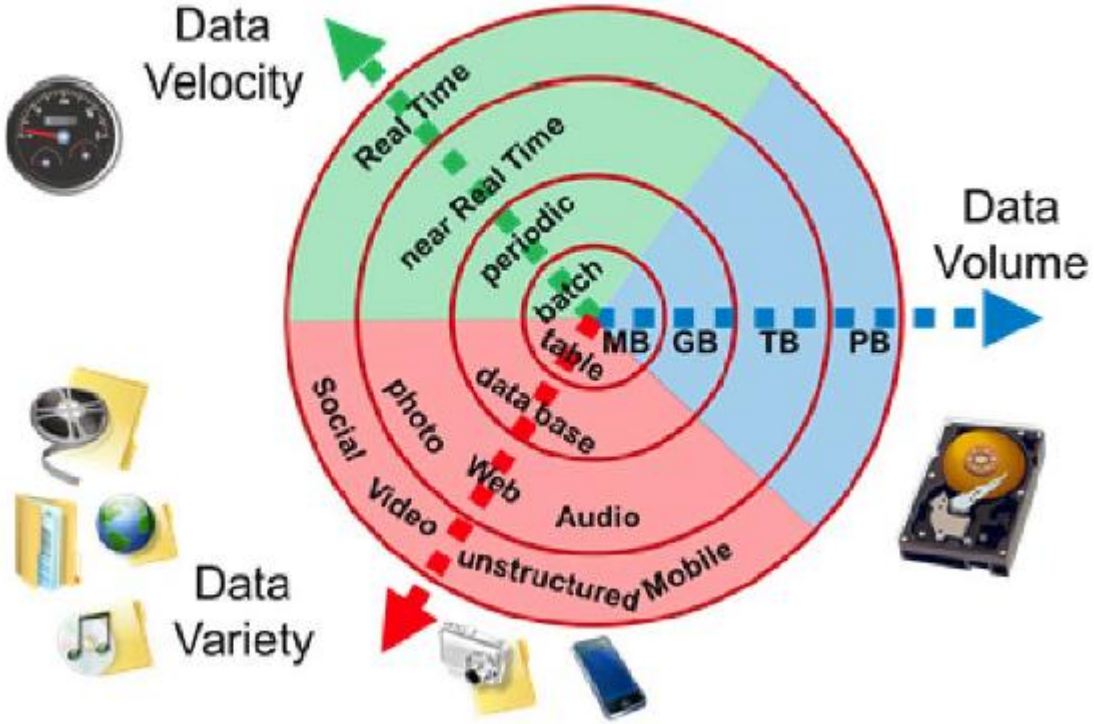
Các dạng và kiểu của dữ liệu. Dữ liệu được thu thập từ nhiều nguồn khác nhau và các kiểu dữ liệu cũng có rất nhiều cấu trúc khác nhau.

Velocity (Vận tốc) :

Trong trường hợp này nghĩa là tốc độ các dữ liệu được tạo ra và xử lý để đáp ứng các nhu cầu và thách thức trên con đường tăng trưởng và phát triển.

Veracity (Tính xác thực) :

Chất lượng của dữ liệu thu được có thể khác nhau rất nhiều, ảnh hưởng đến sự phân tích chính xác.



Hình 1.3 Đặc trưng

Nhà máy và các hệ thống không thực-ảo có thể có một hệ thống 6C bao gồm:

• Kết nối (cảm biến và mạng)

• Đám mây (tính toán và dữ liệu theo yêu cầu)

• Nội dung ảo (mẫu và bộ nhớ)

• Nội dung / ngữ cảnh (ý nghĩa và tương quan)

• Cộng đồng (chia sẻ và cộng tác)

• Tuỳ chỉnh (cá nhân hoá và giá trị)

Dữ liệu phải được xử lý bằng các công cụ tiên tiến (phân tích và thuật toán) để cho ra các thông tin có ý nghĩa. Ví dụ, để quản lý một nhà máy phải xem xét cả hai vấn đề hữu hình và vô hình với các thành phần khác nhau. Các thuật toán tạo thông tin phải phát hiện và giải quyết các vấn đề không nhìn thấy được như sự xuống cấp của máy, mài mòn linh kiện, vv. trong nhà máy.

1.1.3 Lịch sử hình thành của dữ liệu lớn .

Nguồn dữ liệu lớn đã tồn tại dưới nhiều hình thức, thường được xây dựng bởi các công ty cho những nhu cầu đặc biệt. Bắt đầu từ những năm 1990, các nhà cung cấp thương mại tham gia cung cấp các hệ thống quản lý cơ sở dữ liệu song song cho các dữ liệu lớn. Trong nhiều năm, WinterCorp là công ty phát hành báo cáo lớn nhất về cơ sở dữ liệu.

Năm 1984, Tập đoàn Teradata đưa ra thị trường hệ thống xử lý dữ liệu song song DBC 1012. Các hệ thống của Teradata là những hệ thống đầu tiên lưu trữ và phân tích đến 1 terabyte dữ liệu vào năm 1992. Ổ đĩa cứng đã đạt đến mức dung lượng 2.5GB vào năm 1991 nên định nghĩa dữ liệu lớn liên tục phát triển theo quy luật Kryder. Teradata đã cài đặt hệ thống đầu tiên dựa trên RDBMS có thể phân tích hàng petabytes dữ liệu vào năm 2007. Đến năm 2017, có hàng chục các cơ sở dữ liệu dựa trên hệ thống của Teradata có dung lượng hàng petabyte, trong đó dữ liệu lớn nhất vượt quá 50 petabytes. Cho đến năm 2008, 100% hệ thống đều xử lý các dữ liệu quan hệ có cấu trúc. Do đó, Teradata đã thêm các kiểu dữ liệu phi cấu trúc bao gồm XML, JSON và Avro.

Năm 2000, Seisint Inc. (nay là Tập đoàn LexisNexis) đã phát triển một khung chia sẻ tệp dựa trên cấu cấu trúc C++ để lưu trữ và truy vấn dữ liệu. Hệ thống này lưu trữ và phân phối dữ liệu có cấu trúc, bán cấu trúc, và phi cấu trúc trên nhiều máy chủ. Người dùng có thể truy vấn bằng một phương ngữ C ++ gọi là ECL. ECL sử dụng phương thức "áp dụng giản đồ khi truy cập dữ liệu" để suy luận cấu trúc dữ liệu được lưu trữ khi nó được truy vấn, thay vì khi nó được lưu trữ. Năm 2004, LexisNexis mua lại Seisint Inc. và trong năm 2008 đã mua lại ChoicePoint, Inc.[16] cùng với nền tảng xử lý song song tốc độ cao của họ. Hai nền tảng đã được sáp nhập vào hệ thống HPCC (High-Performance Computing Cluster) và HPCC có mã nguồn mở dựa trên giấy phép Apache v2.0 vào năm 2011. Khoảng cùng thời điểm đó, hệ thống Quantcast File đã được phát hành.

Năm 2004, Google xuất bản một bài báo về một quá trình gọi là MapReduce sử dụng một kiến trúc tương tự. MapReduce cung cấp một mô hình xử lý song song, và phát hành những ứng dụng liên quan để xử lý lượng dữ liệu khổng lồ. Với MapReduce, các truy vấn được chia nhỏ và truyền đi qua các nút mạng song song và được xử lý song song (bước Map). Các kết quả sau đó được thu thập và phân phối (Bước Reduce). Khuôn mẫu này rất thành

công[18] nên những công ty khác cũng muốn sao chép các thuật toán của nó. Do đó, Google đã triển khai khuôn mẫu MapReduce thông qua dự án mã nguồn mở Apache Hadoop.

Các nghiên cứu vào năm 2012 cho thấy cấu trúc nhiều lớp là một lựa chọn để giải quyết các vấn đề của xử lý dữ liệu lớn. Một kiến trúc phân tán song song phân tán dữ liệu trên nhiều máy chủ; những môi trường thực hiện song song này có thể cải thiện đáng kể tốc độ xử lý dữ liệu. Kiểu cấu trúc này chèn dữ liệu vào một DBMS song song, thực hiện việc sử dụng các khung nền MapReduce và Hadoop. Loại khung nền này sẽ tăng sức mạnh xử lý thông suốt đến người dùng cuối bằng cách sử dụng một máy chủ ứng dụng đầu cuối.

Phân tích dữ liệu lớn ứng dụng vào việc sản xuất được giới thiệu như một cấu trúc 5C (connection - kết nối, conversion - chuyển đổi, cyber - không gian mạng, cognition - nhận thức và configuration - cấu hình).

Hồ dữ liệu cho phép một tổ chức thay đổi định hướng từ mô hình kiểm soát tập trung sang mô hình chia sẻ thông tin để năng động đáp ứng với sự thay đổi của việc quản lý thông tin. Điều này cho phép phân tách nhanh chóng dữ liệu vào hồ dữ liệu, do đó làm giảm thời gian xử lý thông tin.

1.1.4 Tầm quan trọng của dữ liệu lớn.

Dữ liệu lớn đã trở thành một trong những yếu tố quan trọng nhất trong thời đại công nghệ số, mang lại những lợi ích to lớn trong nhiều lĩnh vực:

Trong kinh doanh và doanh nghiệp:

Hỗ trợ ra quyết định chiến lược: Phân tích dữ liệu giúp doanh nghiệp dự đoán xu hướng, tối ưu chiến lược tiếp thị và nâng cao hiệu suất hoạt động.

Cải thiện trải nghiệm khách hàng: Dữ liệu lớn giúp cá nhân hóa dịch vụ, dựa trên hành vi và sở thích của từng khách hàng.

Quản lý rủi ro và gian lận: Ngành tài chính và ngân hàng sử dụng Big Data để phát hiện giao dịch đáng ngờ, giảm thiểu gian lận.

Trong y tế và chăm sóc sức khỏe:

Chẩn đoán và điều trị bệnh: Phân tích dữ liệu từ bệnh nhân giúp bác sĩ đưa ra phác đồ điều trị phù hợp hơn.

Dự đoán dịch bệnh: Dữ liệu lớn hỗ trợ dự báo sự lây lan của dịch bệnh, giúp chính phủ và tổ chức y tế có biện pháp phòng chống kịp thời.

Nghiên cứu thuốc và vắc xin: Big Data giúp tăng tốc quá trình phát triển thuốc và thử nghiệm lâm sàng.

Trong giao thông và thành phố thông minh:

Tối ưu giao thông: Hệ thống dữ liệu lớn giúp điều phối giao thông, giảm tắc đường và tối ưu hóa vận tải công cộng.

Quản lý năng lượng: Big Data giúp thành phố thông minh kiểm soát tiêu thụ điện năng hiệu quả hơn.

Trong khoa học và giáo dục:

Phân tích dữ liệu nghiên cứu: Big Data hỗ trợ các nhà khoa học trong nghiên cứu thiên văn, sinh học, môi trường, v.v.

Cải thiện phương pháp giảng dạy: Dữ liệu giúp theo dõi tiến độ học tập của sinh viên và đề xuất lộ trình học phù hợp.

Trong truyền thông và giải trí:

Gợi ý nội dung cá nhân hóa: Big Data giúp nền tảng như Netflix, YouTube đề xuất nội dung phù hợp với từng người dùng.

Phân tích xu hướng truyền thông: Hỗ trợ các công ty truyền thông theo dõi xu hướng và điều chỉnh nội dung phù hợp.

1.2 Công nghệ của dữ liệu lớn.

1.2.1. Kiến trúc xử lý dữ liệu lớn.

Kiến trúc xử lý dữ liệu lớn được thiết kế để có thể thu thập, lưu trữ, xử lý và phân tích một lượng dữ liệu khổng lồ một cách hiệu quả. Một hệ thống Big Data điển hình bao gồm ba tầng chính:

1.2.1.a Tầng thu thập dữ liệu

Tầng này đảm nhận việc thu thập dữ liệu từ nhiều nguồn khác nhau, bao gồm:

Dữ liệu cảm biến IoT: Hệ thống cảm biến trong nhà máy, thành phố thông minh.

Dữ liệu mạng xã hội: Facebook, Twitter, YouTube, TikTok.

Dữ liệu giao dịch tài chính: Hệ thống thanh toán điện tử, ngân hàng.

Dữ liệu log hệ thống: Nhật ký máy chủ, truy cập web.

1.2.1.b Tầng lưu trữ và xử lý

Dữ liệu sau khi thu thập sẽ được lưu trữ và xử lý bằng các hệ thống chuyên dụng:

Lưu trữ dữ liệu: Sử dụng HDFS, NoSQL (MongoDB, Cassandra), hoặc các nền tảng đám mây (AWS S3, Google Cloud Storage).

Xử lý dữ liệu hàng loạt (Batch Processing): Sử dụng Apache Hadoop, Spark để xử lý dữ liệu theo lô.

Xử lý dữ liệu thời gian thực (Real-time Processing): Sử dụng Apache Kafka, Apache Flink để phân tích dữ liệu ngay khi nó được tạo ra.

1.2.1.c Tầng phân tích và trực quan hóa

Sau khi dữ liệu được xử lý, nó cần được phân tích và hiển thị trực quan để hỗ trợ ra quyết định

Công cụ phân tích dữ liệu: Sử dụng SQL, Apache Hive, Google BigQuery để truy vấn dữ liệu.

Trí tuệ nhân tạo và học máy: Dùng AI để phân tích xu hướng, dự đoán tương lai.

1.2.2 Các công nghệ và nền tảng chính

1.2.2.a Hệ thống lưu trữ phân tán

Hệ thống lưu trữ phân tán giúp dữ liệu lớn được lưu trữ trên nhiều máy chủ thay vì tập trung trên một hệ thống đơn lẻ. Một số công nghệ phổ biến bao gồm:

- Hadoop Distributed File System (HDFS): Hệ thống tệp phân tán phổ biến giúp lưu trữ dữ liệu lớn với khả năng chịu lỗi cao.

- Amazon S3: Dịch vụ lưu trữ đám mây của Amazon hỗ trợ khả năng mở rộng lớn.
- Google Cloud Storage: Cung cấp khả năng lưu trữ dữ liệu lớn với tốc độ truy xuất cao.

1.2.2.b Cơ sở dữ liệu NoSQL

Do tính đa dạng và khối lượng lớn của dữ liệu, cơ sở dữ liệu NoSQL được sử dụng thay vì các hệ quản trị cơ sở dữ liệu quan hệ truyền thống (SQL). Một số hệ thống phổ biến:

- MongoDB: Hệ cơ sở dữ liệu tài liệu NoSQL phổ biến.
- Cassandra: Hệ cơ sở dữ liệu phân tán, hiệu suất cao do Facebook phát triển.
- HBase: Hệ cơ sở dữ liệu NoSQL hoạt động trên HDFS.

1.2.2.c Điện toán đám mây

Các nền tảng điện toán đám mây giúp mở rộng khả năng xử lý và lưu trữ dữ liệu lớn:

- Amazon Web Services (AWS): Cung cấp dịch vụ lưu trữ, xử lý dữ liệu lớn như AWS Lambda, EMR, Redshift.
- Google Cloud Platform (GCP): Hỗ trợ các dịch vụ BigQuery, Dataflow.
- Microsoft Azure: Cung cấp dịch vụ phân tích dữ liệu lớn như Azure Synapse Analytics.

1.2.2.d Công cụ xử lý dữ liệu

- Một số công cụ phổ biến để xử lý dữ liệu lớn:
- Apache Hadoop: Khung xử lý dữ liệu lớn mã nguồn mở phổ biến.
- Apache Spark: Công cụ xử lý dữ liệu nhanh hơn Hadoop nhờ khả năng xử lý trong bộ nhớ.
- Flink: Công cụ xử lý dữ liệu streaming mạnh mẽ.

1.2.3 Ứng dụng của Dữ liệu lớn

- Ngành Y tế Phân tích dữ liệu bệnh nhân để dự đoán dịch bệnh. Sử dụng AI kết

hợp Big Data để chẩn đoán bệnh.

Hình 1.4 Y tế

- Tài chính – Ngân hàng Phát hiện gian lận tài chính qua phân tích dữ liệu giao dịch. Dự đoán xu hướng thị trường chứng khoán.

Hình 1.5 Tài chính ngân hàng

- Thương mại điện tử Cá nhân hóa quảng cáo dựa trên hành vi khách hàng. Tối ưu hóa chuỗi cung ứng và quản lý hàng tồn kho.

Hình 1.6 Thương mại điện tử

- Giao thông – Thành phố thông minh Dự báo tắc đường và tối ưu hóa giao thông. Quản lý camera an ninh bằng AI kết hợp Big Data.

Hình 1.7 Giao thông

CHƯƠNG 2. MÔ TẢ TẬP DỮ LIỆU VÀ CÔNG NGHỆ SỬ DỤNG

2.1 Tập dữ liệu USA Real Estate Dataset

Tập dữ liệu bất động sản Hoa Kỳ là một bộ dữ liệu toàn diện gồm hơn 1 triệu bản ghi, cung cấp thông tin chi tiết về thị trường bất động sản trên toàn lãnh thổ Hoa Kỳ. Đây là nguồn dữ liệu quý giá cho các nhà nghiên cứu, nhà phân tích, nhà đầu tư và các bên liên quan trong ngành bất động sản.

2.1.1 Cấu Trúc Dữ Liệu

Các trường dữ liệu chính:

brokered_by: Mã số của đại lý/nhà môi giới bất động sản

status: Trạng thái của bất động sản

price: Giá niêm yết của bất động sản (USD)

bed: Số phòng ngủ

bath: Số phòng tắm

acre_lot: Diện tích đất (đơn vị: acre)

street: Mã số đường/phố

city: Thành phố

state: Tiểu bang/vùng lãnh thổ

zip_code: Mã bưu chính

house_size: Diện tích nhà (đơn vị: foot vuông)

prev_sold_date: Ngày bán trước đây

2.2 Apache Spark

Trong kỷ nguyên dữ liệu số, khối lượng dữ liệu tạo ra hàng ngày tăng lên nhanh chóng, đòi hỏi các hệ thống xử lý mạnh mẽ, linh hoạt và có khả năng mở rộng. Apache Spark ra đời như một giải pháp xử lý dữ liệu lớn (Big Data) hiện đại, cung cấp khả năng tính toán phân tán hiệu suất cao, vượt trội hơn so với Hadoop MapReduce.

Hình 2.1 Spark

2.2.1 Kiến trúc của Apache Spark

Apache Spark được thiết kế theo kiến trúc phân tán với các thành phần chính như sau:

Spark Core: Thành phần trung tâm của hệ thống, chịu trách nhiệm quản lý bộ nhớ, thực hiện các tác vụ phân tán và xử lý lỗi.

Spark SQL: Hỗ trợ xử lý dữ liệu có cấu trúc thông qua ngôn ngữ SQL, giúp truy vấn dữ liệu dễ dàng.

Spark Streaming: Cho phép xử lý dữ liệu thời gian thực từ các nguồn như Apache Kafka, Flume.

MLlib (Machine Learning Library): Thư viện máy học giúp xây dựng các mô hình dự đoán trên dữ liệu lớn.

GraphX: Công cụ phân tích dữ liệu dạng đồ thị, hữu ích trong các bài toán mạng xã hội hoặc phân tích quan hệ.

Apache Spark hoạt động dựa trên Resilient Distributed Dataset (RDD) – một mô hình dữ liệu giúp tăng tốc độ xử lý nhờ khả năng lưu trữ trong bộ nhớ và chịu lỗi cao.

2.2.2 Đặc điểm nổi bật của Apache Spark

Apache Spark sở hữu nhiều ưu điểm vượt trội so với các nền tảng xử lý dữ liệu lớn khác:

Hiệu suất cao: Với khả năng tính toán trong bộ nhớ (in-memory computing), Spark nhanh hơn Hadoop MapReduce từ 10 đến 100 lần.

Hỗ trợ đa dạng ngôn ngữ: Người dùng có thể lập trình bằng Python, Java, Scala hoặc R.

Tích hợp tốt với hệ sinh thái dữ liệu lớn: Spark có thể chạy trên Hadoop YARN, Kubernetes và tích hợp với HDFS, Amazon S3, Cassandra.

Linh hoạt và dễ mở rộng: Apache Spark có thể hoạt động trên một cụm máy tính (cluster) lớn với hàng nghìn nút, giúp xử lý dữ liệu khổng lồ mà không làm giảm hiệu suất.

2.2.3 Ứng dụng thực tế của Apache Spark

Apache Spark được ứng dụng rộng rãi trong nhiều lĩnh vực:

Phân tích log hệ thống: Giúp phát hiện lỗi trong các hệ thống máy chủ lớn.

Xử lý dữ liệu IoT: Dùng để thu thập và phân tích dữ liệu từ các thiết bị cảm biến trong nhà máy hoặc thành phố thông minh.

Học máy và AI: Spark MLlib hỗ trợ huấn luyện mô hình máy học trên tập dữ liệu khổng lồ.

Tài chính và ngân hàng: Phát hiện gian lận trong giao dịch tài chính bằng cách phân tích dữ liệu thời gian thực.

2.3 Giới thiệu về Apache Hadoop

Apache Hadoop là một hệ sinh thái phần mềm mã nguồn mở được phát triển bởi Apache Software Foundation để hỗ trợ việc lưu trữ và xử lý dữ liệu lớn theo cách phân tán. Được thiết kế để chạy trên phần cứng thông thường, Hadoop giúp các tổ chức xử lý khối lượng dữ liệu khổng lồ một cách hiệu quả và tiết kiệm chi phí. Nhờ khả năng mở rộng cao và chịu lỗi tốt, Hadoop đã trở thành một trong những công nghệ chủ chốt trong lĩnh vực Big Data.

Hình 2.2 Hadoop

2.3.1 Kiến trúc của Apache Hadoop

Hadoop bao gồm bốn thành phần chính:

Hadoop Distributed File System (HDFS)

HDFS là hệ thống tệp phân tán cho phép lưu trữ dữ liệu lớn trên nhiều máy chủ khác nhau. Dữ liệu được chia thành các khối (blocks) và phân phối trên các nút trong hệ thống để đảm bảo tính khả dụng và độ tin cậy.

NameNode: Quản lý metadata của hệ thống tệp và theo dõi các vị trí lưu trữ dữ liệu.

DataNode: Lưu trữ dữ liệu thực tế và thực hiện các tác vụ đọc/ghi theo yêu cầu từ NameNode.

Secondary NameNode: Hỗ trợ NameNode trong việc quản lý metadata và sao lưu thông tin.

Yet Another Resource Negotiator (YARN)

YARN là hệ thống quản lý tài nguyên trong Hadoop, chịu trách nhiệm lập lịch và phân bổ tài nguyên cho các ứng dụng xử lý dữ liệu.

ResourceManager: Điều phối tài nguyên trên cụm Hadoop.

NodeManager: Giám sát tài nguyên của từng nút trong hệ thống.

ApplicationMaster: Quản lý vòng đời của từng ứng dụng chạy trên Hadoop.

MapReduce

MapReduce là mô hình lập trình được sử dụng để xử lý dữ liệu theo cách phân tán. Nó gồm hai giai đoạn chính:

Map: Chia nhỏ dữ liệu đầu vào thành các cặp khóa-giá trị và phân tán chúng cho nhiều nút tính toán.

Reduce: Tổng hợp kết quả từ giai đoạn Map để tạo ra kết quả cuối cùng.

Hadoop Common

Hadoop Common chứa các thư viện và tiện ích hỗ trợ cho các thành phần khác trong hệ sinh thái Hadoop. Nó cung cấp các giao diện lập trình API để các ứng dụng có thể giao tiếp với hệ thống.

2.3.2 Ưu điểm của Apache Hadoop

Khả năng mở rộng cao: Hadoop có thể mở rộng từ vài nút máy chủ đến hàng nghìn nút mà không làm giảm hiệu suất.

Chi phí thấp: Có thể triển khai trên phần cứng thông thường thay vì các hệ thống lưu trữ đắt tiền.

Chịu lỗi tốt: HDFS tự động sao chép dữ liệu để đảm bảo rằng nếu một nút bị lỗi, dữ liệu vẫn có thể được khôi phục từ bản sao dự phòng.

Xử lý song song hiệu quả: Nhờ MapReduce, Hadoop có thể xử lý lượng lớn dữ liệu cùng lúc mà không ảnh hưởng đến hiệu suất tổng thể.

2.3.3 Thách thức của Apache Hadoop

Tốc độ xử lý hạn chế: Do sử dụng lưu trữ trên ổ cứng, MapReduce có thể chậm hơn các giải pháp xử lý dữ liệu trong bộ nhớ như Apache Spark.

Độ phức tạp cao: Việc triển khai và quản lý Hadoop đòi hỏi kiến thức chuyên sâu về hệ thống phân tán.

Khó khăn trong lập trình: MapReduce yêu cầu người dùng viết code phức tạp để xử lý dữ liệu, thay vì sử dụng SQL như trong các hệ thống truyền thống.

2.3.4 Ứng dụng thực tế của Apache Hadoop

Apache Hadoop được sử dụng rộng rãi trong nhiều lĩnh vực:

Lưu trữ và phân tích dữ liệu lớn: Các công ty như Facebook, Twitter, và Yahoo sử dụng Hadoop để lưu trữ và phân tích dữ liệu từ người dùng.

Tìm kiếm và lập chỉ mục web: Các công cụ tìm kiếm như Google và Bing sử dụng Hadoop để xử lý dữ liệu web.

Phân tích dữ liệu y tế: Trong ngành y tế, Hadoop hỗ trợ phân tích dữ liệu bệnh nhân và nghiên cứu gen.

Tài chính và ngân hàng: Các tổ chức tài chính sử dụng Hadoop để phân tích giao dịch, phát hiện gian lận và quản lý rủi ro.

CHƯƠNG 3. KẾT QUẢ XỬ LÝ, PHÂN TÍCH DỮ LIỆU

3.1 Import các thư viện cần thiết

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, count, when, isnan
from pyspark.sql.types import IntegerType
from pyspark.ml.feature import StringIndexer

# Khởi tạo Spark Session
print("Initializing Spark Session...")
spark = SparkSession.builder \
    .appName("Real Estate Analysis") \
    .config("spark.driver.memory", "8g") \
    .config("spark.jars.packages", "ml.dmlc:xgboost4j-spark_2.12:1.5.2")
    .master("local[*]") \
    .getOrCreate()
```

Hình 3.1 Import thư viện

from pyspark.sql import SparkSession

Import SparkSession, là điểm khởi đầu để làm việc với Spark SQL.

from pyspark.sql.functions import col, count, when, isnan

Import các hàm:

col: Truy xuất cột từ DataFrame.

count: Đếm số lượng hàng.

when: Tạo điều kiện trong DataFrame.

isnan: Kiểm tra giá trị NaN.

from pyspark.sql.types import IntegerType

Import IntegerType để xác định kiểu dữ liệu số nguyên trong DataFrame.

from pyspark.ml.feature import StringIndexer

Import StringIndexer, một công cụ trong thư viện ML của Spark để chuyển đổi dữ liệu dạng chuỗi thành số.

print("Initializing Spark Session...")

In ra thông báo để biết quá trình khởi tạo SparkSession đang diễn ra.

spark = SparkSession.builder \

Tạo một phiên làm việc với Spark.

.appName("Real Estate Analysis") \

Đặt tên ứng dụng là "Real Estate Analysis".

.config("spark.driver.memory", "8g") \

Cấu hình bộ nhớ của driver Spark là 8GB.

.config("spark.jars.packages", "ml.dmlc:xgboost4j-spark_2.12:1.5.2") \

Thêm thư viện XGBoost để sử dụng trong Spark ML.

.master("local[*]") \

Chạy ứng dụng Spark trên máy cục bộ, sử dụng tất cả lõi CPU có sẵn ([*]).

.getOrCreate()

Tạo hoặc lấy một phiên SparkSession đã tồn tại.

3.2 Tiền xử lý dữ liệu

```
Số lượng giá trị null:
prev_sold_date    32.981627
house_size        25.533983
bath              22.986666
bed               21.618797
acre_lot          14.624130
street             0.488056
brokered_by        0.203604
price              0.069215
city               0.063197
zip_code           0.013430
state              0.000359
status             0.000000
dtype: float64
```

Hình 3.2 Số lượng giá trị null

Bộ dữ liệu được sử dụng trong nghiên cứu này bao gồm các thuộc tính quan trọng như diện tích nhà, số phòng tắm, số phòng ngủ, diện tích đất, ngày bán trước đó, địa chỉ, mã bưu điện và giá nhà. Sau khi kiểm tra dữ liệu, tỷ lệ giá trị bị thiếu trong một số cột cụ thể như sau:

- prev_sold_date (32.98%)
- house_size (25.53%)
- bath (22.99%)
- bed (21.62%)
- acre_lot (14.62%)
- Một số cột khác có tỷ lệ thiếu dưới 1%.

3.2.1 Phương pháp xử lí giá trị thiếu

3.2.1.a Loại bỏ dữ liệu

Nếu một cột có tỷ lệ dữ liệu thiếu quá cao (trên 30-40%), có thể cân nhắc loại bỏ cột đó nếu nó không đóng vai trò quan trọng trong mô hình dự đoán. Trong trường hợp này, prev_sold_date có thể bị loại bỏ nếu không thực sự cần thiết.

Cụ thể, việc loại bỏ các cột không cần thiết được thực hiện bằng đoạn mã sau:

```
# Loại bỏ các cột không cần thiết
df = df.drop("city", "zip_code", "prev_sold_date")
```

Hình 3.3 Bỏ cột ko cần thiết

3.2.1.b Xử lí giá trị thiếu bằng Xgboost

```python
import xgboost as xgb
import pandas as pd
from pyspark.sql.functions import monotonically_increasing_id, when

print("\nĐang điền giá trị thiếu cho acre_lot bằng XGBoost...")

# Kiểm tra nếu có giá trị thiếu
if df.filter(col("acre_lot").isNull()).count() > 0:

    # Chuyển dữ liệu có acre_lot sang Pandas
    filled_pdf = filled_df.toPandas()
    missing_pdf = missing_df.toPandas()

    # Kiểm tra nếu có dữ liệu để huấn luyện
    if not filled_pdf.empty and not missing_pdf.empty:
        # Chuẩn bị dữ liệu huấn luyện
        X_train_fill = filled_pdf.drop(columns=["acre_lot"])
        y_train_fill = filled_pdf["acre_lot"]

        X_test_fill = missing_pdf.drop(columns=["acre_lot"])

        # Huấn luyện mô hình XGBoost
        model_fill = xgb.XGBRegressor(
            n_estimators=100,
            learning_rate=0.1,
            max_depth=3,
            subsample=0.8,
            colsample_bytree=0.8
        )
        model_fill.fit(X_train_fill, y_train_fill)

        # Dự đoán giá trị thiếu
        preds = model_fill.predict(X_test_fill)

        # Chuyển kết quả về DataFrame PySpark
        preds_df = pd.DataFrame({"acre_lot": preds})
        preds_spark = spark.createDataFrame(preds_df)

        # Thêm ID để kết hợp lại dữ liệu
        df = df.withColumn("id", monotonically_increasing_id())
        missing_df = missing_df.withColumn("id", monotonically_increasing_id())
        preds_spark = preds_spark.withColumn("id", monotonically_increasing_id())

        # Kết hợp giá trị dự đoán vào tập dữ liệu gốc
        df = df.join(preds_spark, "id", "left_outer").drop("id")
        df = df.withColumn("acre_lot", when(col("acre_lot").isNull(), col("prediction")).otherwise(col("acre_lot"))).drop("prediction")

        print(f"Đã điền {len(preds)} giá trị thiếu cho acre_lot bằng XGBoost!")

    else:
        print("Không có đủ dữ liệu để huấn luyện XGBoost. Bỏ qua việc điền giá trị thiếu.")

else:
    print("Không có giá trị thiếu trong acre_lot. Không cần điền.")
```

Hình 3.4 Xử lí giá trị thiếu bằng Xgboost

Đoạn code này thực hiện một kỹ thuật quan trọng trong xử lý dữ liệu: điền giá trị thiếu (imputation) cho cột "acre_lot" bằng mô hình học máy XGBoost. Đây là phương pháp điền giá trị thiếu nâng cao, vượt trội hơn các phương pháp truyền thống như điền bằng giá trị trung bình, trung vị hay mode.

Quy trình xử lý

Kiểm tra sự tồn tại của giá trị thiếu:

Code sử dụng phương thức filter để xác định xem có giá trị null trong cột "acre_lot" hay không.

Tách dữ liệu thành hai tập

Dữ liệu có giá trị "acre_lot" (dùng làm tập huấn luyện)

Dữ liệu thiếu giá trị "acre_lot" (cần dự đoán)

Kiểm tra điều kiện huấn luyện:

Xác minh tập huấn luyện không trống và tập cần dự đoán có dữ liệu cần điền

Chuẩn bị dữ liệu huấn luyện:

Tạo biến đầu vào bằng cách loại bỏ cột "acre_lot" từ tập huấn luyện

Tạo biến mục tiêu là giá trị "acre_lot" từ tập huấn luyện

Chuẩn bị dữ liệu kiểm tra từ tập cần dự đoán

Huấn luyện mô hình XGBoost:

Thiết lập mô hình hồi quy XGBoost với các tham số phù hợp

Huấn luyện mô hình với dữ liệu đã chuẩn bị

Dự đoán và tích hợp kết quả:

Dự đoán giá trị "acre_lot" thiếu

Chuyển kết quả về DataFrame PySpark

Thêm ID để kết hợp dữ liệu

Hợp nhất kết quả dự đoán với dữ liệu gốc

Xử lý cuối cùng:

Thay thế giá trị null trong "acre_lot" bằng kết quả dự đoán

Ưu điểm của phương pháp

Độ chính xác cao hơn: XGBoost học các mối quan hệ phức tạp giữa các biến, tạo ra giá trị dự đoán chính xác hơn so với điền đơn giản bằng giá trị trung bình hay trung vị.

Bảo tồn mối quan hệ giữa các biến: Phương pháp này giữ nguyên cấu trúc tương quan giữa các biến, rất quan trọng trong phân tích thống kê và học máy.

Linh hoạt với nhiều loại dữ liệu: XGBoost xử lý tốt cả dữ liệu số và phân loại.

Khả năng mở rộng: Code được thiết kế để vận hành trong môi trường PySpark, phù hợp cho xử lý dữ liệu lớn.

Một số cải tiến có thể thực hiện

Tối ưu hóa siêu tham số: Có thể thêm GridSearchCV để tìm các tham số tối ưu cho mô hình XGBoost.

Xử lý các biến phân loại: Thêm mã hóa one-hot cho các biến phân loại trước khi huấn luyện.

Đánh giá chất lượng điền giá trị: Có thể thêm đánh giá cross-validation để kiểm tra hiệu quả của kỹ thuật điền giá trị.

Xử lý ngoại lệ: Thêm xử lý cho trường hợp giá trị dự đoán âm hoặc không hợp lý đối với "acre_lot".

3.2.1.c Xử lí giá trị ngoại lai

```
# Xử lý outlier cho cột price
q95 = df.approxQuantile("price", [0.95], 0.01)[0]
q25 = df.approxQuantile("price", [0.25], 0.01)[0]
iqrMax = q95 + q25
print(f"\nThreshold cho price: {iqrMax}")
percent_outliers = df.filter(col("price") > 3150000.0).count() / df.count() * 100
print(f"Tỷ lệ outliers trong price: {percent_outliers:.2f}%")
df = df.filter(col("price") <= 3150000.0)

# Xử lý outlier cho acre_lot
percent_outliers = df.filter(col("acre_lot") > 200).count() / df.count() * 100
print(f"\nTỷ lệ outliers trong acre_lot: {percent_outliers:.2f}%")
df = df.filter(col("acre_lot") <= 200)

# Xử lý outlier cho house_size
percent_outliers = df.filter(col("house_size") >= 20000).count() / df.count() * 100
print(f"\nTỷ lệ outliers trong house_size: {percent_outliers:.2f}%")
df = df.filter(col("house_size") < 20000)

# Kiểm tra lại giá trị null sau khi xử lý outliers
null_counts = df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns])
print("\nSố lượng giá trị null sau khi xử lý outliers:")
null_counts.show()

# Hiển thị thống kê mô tả
df.describe().show()
```

Hình 3.5 Xử lí giá trị ngoại lai

- Xử lý outlier cho cột price:

Lấy giá trị phân vị thứ 95 (q95) và phân vị thứ 25 (q25) của cột price.

Tính ngưỡng iqrMax bằng cách cộng q95 và q25.

ác định phần trăm giá trị ngoại lai (outliers) bằng cách đếm số dòng có price > 315000 rồi chia cho tổng số dòng.

In ra tỷ lệ ngoại lai.

Loại bỏ những dòng có price > 315000.

- Xử lý outlier cho cột acre_lot

Tính tỷ lệ giá trị ngoại lai cho acre_lot (giá trị lớn hơn 200).

Loại bỏ các dòng có giá trị acre_lot > 200.

- Xử lý outlier cho cột house_size

Tính tỷ lệ outliers của house_size với giá trị lớn hơn 20000.

Loại bỏ các dòng có house_size >= 20000.

3.3 Thêm các đặc trưng

```python
from pyspark.sql.functions import col, when, expr

print("Đang tạo các đặc trưng mới...")

# Tránh chia cho 0 bằng cách sử dụng F.when
df = df.withColumn("bed_bath_ratio", col("bed") / when(col("bath") == 0, 1).otherwise(col("bath")))
df = df.withColumn("total_rooms", col("bed") + col("bath"))
df = df.withColumn("room_density", col("total_rooms") / when(col("house_size") == 0, 1).otherwise(col("house_size")))
df = df.withColumn("house_size_per_bed", col("house_size") / when(col("bed") == 0, 1).otherwise(col("bed")))
df = df.withColumn("house_size_per_bath", col("house_size") / when(col("bath") == 0, 1).otherwise(col("bath")))
df = df.withColumn("lot_to_house_ratio", col("acre_lot") / when(col("house_size") == 0, 1).otherwise(col("house_size")))
df = df.withColumn("size_by_state", col("house_size") * col("state_numeric"))
df = df.withColumn("rooms_by_state", col("total_rooms") * col("state_numeric"))

# Điền giá trị thiếu cho các cột số (tránh lỗi)
numeric_columns = [c for c, t in df.dtypes if t in ('int', 'double')]
df = df.fillna(0, subset=numeric_columns)

print("Hoàn tất tạo đặc trưng mới!")
```

Hình 3.6 Thêm các đặc trưng

Đoạn code này thực hiện kỹ thuật "Feature Engineering" (tạo đặc trưng mới) - một bước quan trọng trong tiền xử lý dữ liệu bất động sản. Thay vì chỉ sử dụng các biến gốc có sẵn, code tạo ra nhiều đặc trưng mới có ý nghĩa phân tích sâu sắc hơn.

Cụ thể, đoạn code tạo ra 7 đặc trưng mới có giá trị phân tích cao:

1. **bed_bath_ratio**: Tỷ lệ giữa số phòng ngủ và phòng tắm, giúp đánh giá sự cân đối trong thiết kế nhà. Code thông minh thay thế giá trị 0 trong cột 'bath' bằng 1 để tránh lỗi chia cho 0.
2. **total_rooms**: Tổng số phòng chính (phòng ngủ + phòng tắm), là chỉ số đơn giản về kích thước tổng thể của bất động sản và thường có tương quan mạnh với giá.
3. **room_density**: Số phòng trên mỗi đơn vị diện tích, giúp phân biệt giữa nhà có nhiều phòng nhỏ và nhà có ít phòng lớn. Đây là chỉ số về hiệu quả sử dụng không gian.
4. **house_size_per_bed**: Diện tích trung bình dành cho mỗi phòng ngủ, là chỉ số về mức độ sang trọng/rộng rãi của bất động sản, giúp phân biệt bất động sản cao cấp và bình dân.
5. **house_size_per_bath**: Tương tự, nhưng tính cho phòng tắm. Phòng tắm lớn thường là dấu hiệu của bất động sản cao cấp.

6. **lot_to_house_ratio**: Tỷ lệ giữa diện tích đất và diện tích xây dựng, giúp phát hiện bất động sản có sân vườn lớn và phân biệt nhà ở thành thị (tỷ lệ thấp) với ngoại ô/nông thôn (tỷ lệ cao).
7. **size_by_state** và **rooms_by_state**: Các biến tương tác giữa kích thước/số phòng và vị trí địa lý, giúp nắm bắt sự khác biệt về giá trị của các đặc điểm vật lý theo từng tiểu bang.

Cuối cùng, code xử lý các giá trị NaN có thể xuất hiện trong quá trình tính toán bằng cách điền giá trị trung bình: df = df.fillna(df.mean()).

Việc tạo đặc trưng mới này mang lại nhiều lợi ích: nâng cao hiệu suất mô hình dự đoán, phát hiện insights mới về thị trường, phân khúc thị trường hiệu quả hơn, và cải thiện khả năng diễn giải kết quả phân tích cho người không chuyên.

3.4 Huấn luyện mô hình

```python
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import GBTRegressor
from pyspark.ml.evaluation import RegressionEvaluator

print("\nChuẩn bị dữ liệu cho mô hình...")

# Xác định cột đặc trưng (bỏ cột price)
feature_cols = [c for c in df.columns if c != "price"]

# Dùng VectorAssembler để gộp tất cả đặc trưng vào một cột duy nhất
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
df = assembler.transform(df).select("features", "price")

# Chia dữ liệu thành tập huấn luyện và kiểm tra (80/20)
train_df, test_df = df.randomSplit([0.8, 0.2], seed=42)

print(f"Kích thước tập huấn luyện: {train_df.count()} mẫu, tập kiểm tra: {test_df.count()} mẫu")
```

Hình 3.7 Chuẩn bị dữ liệu

Đoạn mã trên thực hiện các bước chuẩn bị dữ liệu trước khi huấn luyện mô hình Gradient Boosted Trees Regression (GBTRegressor) trong PySpark.

Trước tiên, nó tạo danh sách các cột đặc trưng bằng cách lấy tất cả các cột trong DataFrame và loại bỏ cột "price" vì đây là nhãn cần dự đoán.

Tiếp theo, nó sử dụng VectorAssembler để gộp các cột đặc trưng thành một cột duy nhất có tên "features", vì mô hình học máy trong Spark yêu cầu dữ liệu đầu vào phải ở dạng vector. Sau khi chuyển đổi, DataFrame chỉ còn hai cột: "features" (chứa vector đặc trưng) và "price" (giá trị mục tiêu).

Cuối cùng, dữ liệu được chia thành hai tập:

80% dùng để huấn luyện (train_df).

20% dùng để kiểm tra (test_df) .

Việc đặt seed=42 giúp đảm bảo dữ liệu được chia một cách nhất quán giữa các lần chạy.

Sau đó, chương trình in ra số lượng mẫu trong mỗi tập để kiểm tra kích thước dữ liệu.

```
from pyspark.ml.regression import GBTRegressor
from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit
from pyspark.ml.evaluation import RegressionEvaluator

print("\nBắt đầu tìm kiếm siêu tham số tối ưu...")

# Sử dụng GBTRegressor với các tham số được tối ưu
gbt = GBTRegressor(
    featuresCol="features",
    labelCol="price",
    maxBins=32,         # Tăng maxBins để cải thiện hiệu suất
    maxIter=100,        # Số lần lặp
    stepSize=0.1,       # Learning rate
    maxDepth=5,         # Độ sâu cây
    subsamplingRate=0.8  # Subsampling rate
)

# Giảm tham số trong lưới tìm kiếm để tránh timeout
param_grid = ParamGridBuilder() \
    .addGrid(gbt.maxDepth, [3]) \
    .addGrid(gbt.stepSize, [0.1]) \
    .build()

# Đánh giá mô hình
evaluator = RegressionEvaluator(
    labelCol="price",
    predictionCol="prediction",
    metricName="r2"
)

# Sử dụng TrainValidationSplit với tỷ lệ train cao hơn
tvs = TrainValidationSplit(
    estimator=gbt,
    estimatorParamMaps=param_grid,
    evaluator=evaluator,
    trainRatio=0.9,  # 90% huấn luyện, 10% kiểm định
    seed=42
)

# Huấn luyện mô hình
print("Đang huấn luyện mô hình, vui lòng đợi...")
model = tvs.fit(train_df)

# Lấy mô hình tốt nhất
best_model = model.bestModel
print("\nĐã tìm thấy mô hình tốt nhất!")
print(f"Best maxDepth: {best_model.getMaxDepth()}")
print(f"Best maxIter: {best_model.getMaxIter()}")
print(f"Best stepSize: {best_model.getStepSize()}")
```

Hình 3.8 Huấn luyện mô hình

Đoạn mã trên thực hiện việc huấn luyện mô hình Gradient Boosted Trees Regression (GBTRegressor) trên tập dữ liệu và tìm kiếm siêu tham số tối ưu bằng cách sử dụng TrainValidationSplit trong PySpark.

Khởi tạo mô hình GBTRegressor

featuresCol="features": Cột chứa đặc trưng đầu vào.

labelCol="price": Cột mục tiêu cần dự đoán.

maxBins=32: Tăng số lượng bins để cải thiện hiệu suất chia nhánh.

maxIter=100: Số lần lặp tối đa để mô hình hội tụ.

stepSize=0.1: Tốc độ học của thuật toán.

maxDepth=5: Độ sâu tối đa của cây quyết định.

subsamplingRate=0.8: Chỉ sử dụng 80% dữ liệu để xây dựng mỗi cây nhằm tránh overfitting.

Xây dựng lưới siêu tham số (ParamGridBuilder)

Do giới hạn về thời gian huấn luyện, chỉ thử nghiệm với maxDepth = 3 và stepSize = 0.1.

Nếu mở rộng lưới tìm kiếm, có thể thử thêm maxIter hoặc subsamplingRate.

Đánh giá mô hình bằng RegressionEvaluator

Chỉ số đánh giá là R^2 (coefficient of determination), thể hiện mức độ giải thích biến mục tiêu dựa trên mô hình.

Sử dụng TrainValidationSplit

90% dữ liệu dùng để huấn luyện, 10% để kiểm tra.

Không sử dụng CrossValidation để tránh thời gian chạy quá lâu.

Huấn luyện mô hình

tvs.fit(train_df): Tiến hành huấn luyện và chọn ra mô hình tốt nhất.

Lấy và in ra mô hình tốt nhất

Hiển thị các siêu tham số tối ưu: maxDepth, maxIter, stepSize.

```
Đã tìm thấy mô hình tốt nhất!
Best maxDepth: 3
Best maxIter: 100
Best stepSize: 0.1
```

Hình 3.9 Tham số tốt nhất cho mô hình

3.5 Dự đoán

```python
from pyspark.ml.evaluation import RegressionEvaluator

print("\nĐánh giá mô hình trên tập kiểm tra...")

# Dự đoán
predictions = best_model.transform(test_df)

# Khởi tạo bộ đánh giá
evaluator_r2 = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="r2")
evaluator_rmse = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="rmse")
evaluator_mae = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="mae")

# Tính toán các chỉ số đánh giá
r2 = evaluator_r2.evaluate(predictions)
rmse = evaluator_rmse.evaluate(predictions)
mae = evaluator_mae.evaluate(predictions)

print("\n🔍 Đánh giá mô hình trên dữ liệu thực tế:")
print(f"✅ R² Score: {r2:.4f}")
print(f"✅ RMSE: ${rmse:.2f}")
print(f"✅ MAE: ${mae:.2f}")
```

Hình 3.10 Dự đoán

Đầu tiên, em sử dụng mô hình tối ưu (best_model) đã được huấn luyện trước đó để dự đoán giá bất động sản trên tập dữ liệu kiểm tra X_test. Kết quả dự đoán được lưu vào biến y_pred.

Sau đó, em tính toán bốn chỉ số đánh giá hiệu suất phổ biến cho bài toán hồi quy:

1. **MAE (Mean Absolute Error)**: Sai số tuyệt đối trung bình - cho biết mức độ chênh lệch trung bình giữa giá dự đoán và giá thực tế (tính bằng đơn vị tiền tệ)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

2. **MSE (Mean Squared Error)**: Sai số bình phương trung bình - nhấn mạnh các sai số lớn do bình phương hóa các sai số

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

3. **RMSE (Root Mean Squared Error)**: Căn bậc hai của MSE - đưa đơn vị trở lại đơn vị gốc (đơn vị tiền tệ), giúp diễn giải kết quả dễ dàng hơn

$$RMSE = \sqrt{MSE}$$

4. **R^2 (R-squared)**: Hệ số xác định - cho biết phần trăm biến thiên của giá trị thực tế được giải thích bởi mô hình (từ 0 đến 1, càng gần 1 càng tốt)

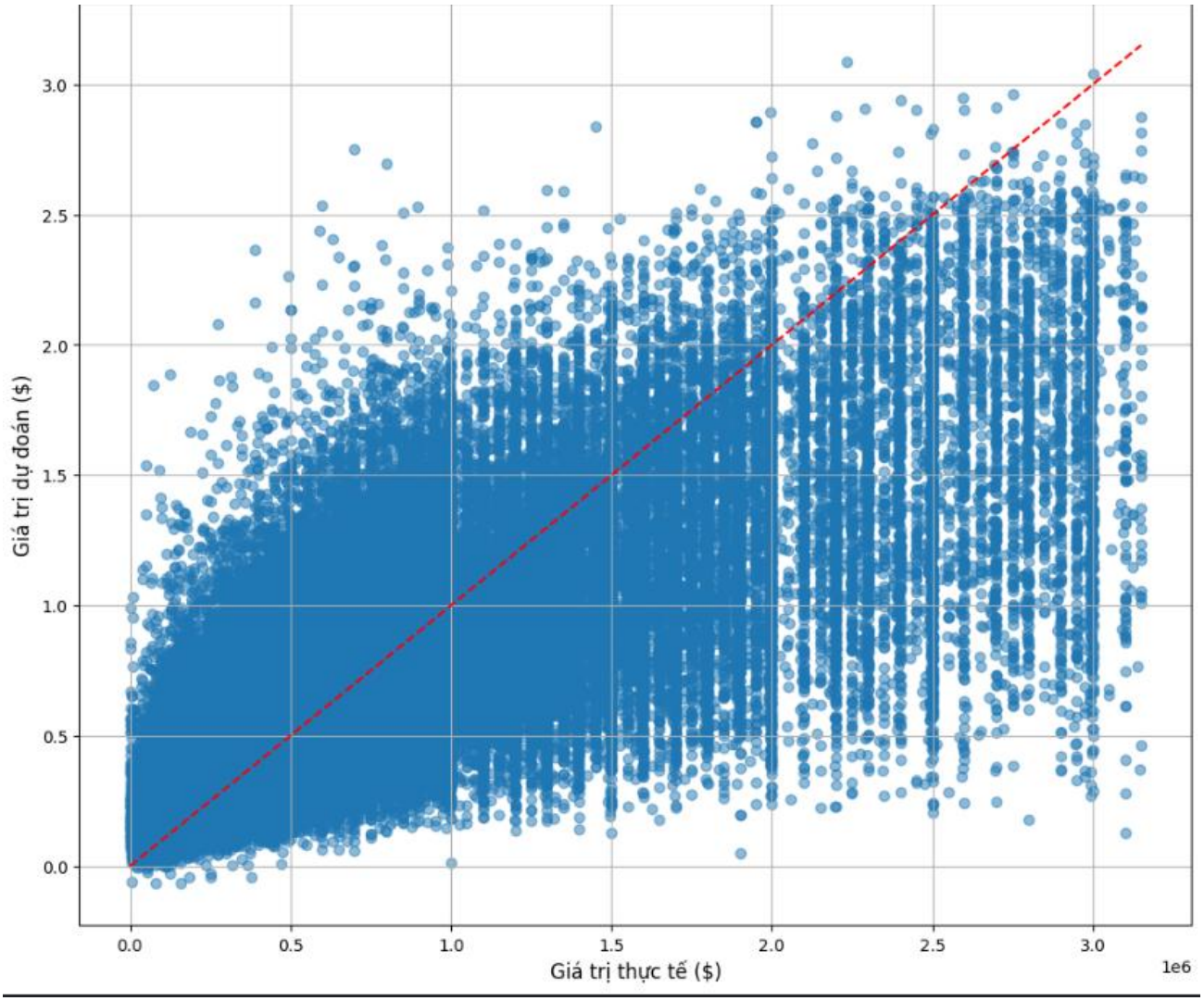
$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

Cuối cùng, em in ra tất cả các chỉ số này với định dạng rõ ràng, giới hạn ở 2 chữ số thập phân để dễ đọc. Dòng chú thích "# Đánh giá trên dữ liệu thục tế " chỉ ra rằng việc đánh giá được thực hiện trên thang đo giá gốc (USD), không qua bất kỳ phép biến đổi nào như logarithm.

Các chỉ số này cung cấp cái nhìn toàn diện về hiệu suất của mô hình. MAE và RMSE giúp hiểu mức độ sai lệch trung bình của dự đoán theo đơn vị tiền tệ, trong khi R^2 cho biết mô hình giải thích được bao nhiêu phần trăm biến thiên trong giá bất động sản. Đây là những thông tin quan trọng để đánh giá khả năng áp dụng thực tế của mô hình dự đoán.

🔍 Đánh giá mô hình trên dữ liệu thực tế:
✅ R² Score: 0.5381
✅ RMSE: $284943.18
✅ MAE: $172705.09

Hình 3.11 Kết quả



Hình 3.12 Giá trị thực tế với dự đoán

Biểu đồ này thể hiện kết quả dự đoán của mô hình XGBoost đối với giá bất động sản. Đây là một biểu đồ phân tán (scatter plot) so sánh giữa giá trị thực tế (trục hoành) và giá trị dự đoán (trục tung), với đơn vị là triệu đô la Mỹ.

Đường chéo màu đỏ đứt nét thể hiện đường chuẩn nơi các điểm dự đoán hoàn hảo sẽ nằm (khi giá trị dự đoán bằng chính xác giá trị thực tế). Các điểm dữ liệu màu xanh biểu thị cho mỗi bất động sản trong tập kiểm tra.

Từ biểu đồ này, em có thể quan sát được một số điểm quan trọng:

1. Độ phân tán tăng theo giá trị: Có thể thấy rõ rằng độ phân tán của các dự đoán tăng lên khi giá bất động sản tăng. Các bất động sản có giá thấp (dưới 500.000$) được dự đoán tương đối chính xác, trong khi các bất động sản có giá cao hơn có sai số dự đoán lớn hơn.
2. Mức độ tập trung: Phần lớn các điểm dữ liệu tập trung ở khu vực giá thấp đến trung bình (dưới 1,5 triệu đô), phản ánh sự phân bố thực tế của thị trường bất động sản.
3. Có hiện tượng dự đoán thấp hơn với bất động sản giá cao: Đối với nhiều bất động sản có giá trị cao (trên 2 triệu đô), mô hình thường dự đoán giá trị thấp hơn giá trị thực tế, thể hiện qua nhiều điểm nằm dưới đường chéo đỏ ở phần bên phải của biểu đồ.
4. Một số dự đoán cực đoan: Có một số trường hợp mô hình đưa ra dự đoán cực đoan - hoặc quá cao hoặc quá thấp so với giá trị thực tế, thể hiện qua các điểm nằm xa đường chéo.

Mặc dù mô hình có xu hướng nắm bắt được xu hướng chung của dữ liệu (các điểm tập trung xung quanh đường chéo), nhưng vẫn còn dư địa để cải thiện, đặc biệt là với các bất động sản giá cao. Hiện tượng này phổ biến trong dự đoán giá bất động sản, do các bất động sản cao cấp thường có những đặc điểm độc đáo khó nắm bắt hoàn toàn bằng các biến số thông thường.

KẾT LUẬN

Trong nghiên cứu này, chúng tôi đã áp dụng các phương pháp khoa học dữ liệu và học máy để xây dựng mô hình dự đoán giá nhà đất tại Mỹ. Quá trình phân tích bao gồm việc tiền xử lý dữ liệu, lựa chọn đặc trưng, huấn luyện mô hình và đánh giá hiệu suất. Kết quả cho thấy mô hình có khả năng dự đoán giá nhà với độ chính xác tương đối cao, giúp hỗ trợ các nhà đầu tư và cá nhân đưa ra quyết định hợp lý hơn trong thị trường bất động sản.

Việc sử dụng các thuật toán hiện đại như XGBoost cùng với các kỹ thuật tối ưu hóa giúp cải thiện chất lượng dự đoán, giảm thiểu sai số và tăng khả năng tổng quát hóa của mô hình. Ngoài ra, việc trực quan hóa dữ liệu và phân tích tầm quan trọng của từng đặc trưng cũng giúp hiểu rõ hơn về các yếu tố ảnh hưởng đến giá nhà, từ đó cung cấp những thông tin hữu ích cho người sử dụng.

Mặc dù mô hình đã đạt được kết quả khả quan, nhưng vẫn còn một số hạn chế cần được cải thiện, chẳng hạn như việc mở rộng phạm vi dữ liệu, thử nghiệm thêm các mô hình tiên tiến hơn, hoặc tích hợp thêm dữ liệu kinh tế vĩ mô để nâng cao độ chính xác của dự đoán. Trong tương lai, việc kết hợp các công nghệ mới như trí tuệ nhân tạo nâng cao và dữ liệu thời gian thực có thể giúp nâng cao chất lượng dự báo, mang lại nhiều giá trị hơn cho lĩnh vực bất động sản

DANH MỤC TÀI LIỆU THAM KHẢO

[1] https://www.kaggle.com/datasets/ahmedshahriarsakib/usa-real-estate-dataset

[2] https://vi.wikipedia.org/wiki/D%E1%BB%AF_li%E1%BB%87u_l%E1%BB%9Bn