Rinnovation: Optimize Your Home Renovation

Aditya Agrawal, Andrew Selvia, Kaustubh Kulkarni and Pallavi Chaturvedi
Department of Software Engineering, San José State University
San José, CA
aditya.agrawal@sjsu.edu, andrew.selvia@sjsu.edu, kaustubh.kulkarni@sjsu.edu, pallavinirmal.chaturvedi@sjsu.edu

***Abstract*— Whether you're a young couple moving in to your first home or a seasoned property manager, you might be asking yourself, "How do I turn this fixer-upper into a dream home?". Rinnovation is here to help. Whether you're focused on location, renovation type, or ROI, Rinnovation empowers you to make the best decision based on local trends. By leveraging data from similar renovations, Rinnovation can predict how much of your project costs you will recoup.**

I. Introduction

Rinnovation is a tool that helps home buyers, home owners, and property managers plan renovation projects. Rinnovation serves historical predicted ROI for 43 renovation project types and 150 cities.

II. Architecture

There are 5 major components: Data, ETL, Machine Learning, Service, and Client.

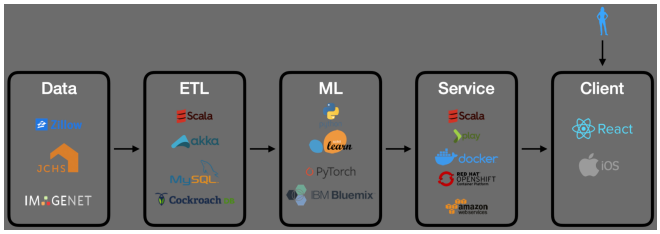

Fig. 1. Project Architecture

III. Data Analysis

The dataset for this project contains average costs for 43 renovation projects with the value those projects retain a resale in 150 United States markets. The data is categorized by region and year of project execution which includes city job costs, city resale value, city cost recouped, city-region comparison and city-national comparison. The dataset is publicly accessible from the author's website [1].

A. Data Distribution

The dataset available is in the form of pdf files. Each file contains data for renovation projects carried out in a particular city for a particular year. Each file has data for around 20 renovation projects including Bathroom Addition — Midrange, Bathroom Addition — Upscale, Bath Remodel — Midrange, Bath Remodel — Upscale, and Bath Remodel — Universal Design. Data for San Jose in 2019 is shown in Fig 2.

PROJECT TYPE	SAN JOSE Job Cost	Resale Value	Cost Recouped
Bathroom Addition | Midrange	$ 63,474	$ 41,905	66.0%
Bathroom Addition | Upscale	113,311	74,238	65.5%
Bath Remodel | Midrange	26,924	20,048	74.5%
Bath Remodel | Upscale	81,464	53,350	65.5%
Bath Remodel | Universal Design	42,549	27,111	63.7%
Minor Kitchen Remodel | Midrange	27,909	25,220	90.4%
Major Kitchen Remodel | Midrange	79,623	58,481	73.4%
Major Kitchen Remodel | Upscale	156,553	99,170	63.3%
Master Suite Addition | Midrange	173,834	137,476	79.1%
Master Suite Addition | Upscale	354,642	214,000	60.3%
Deck Addition | Composite	24,618	16,026	65.1%
Deck Addition | Wood	18,386	13,421	73.0%
Backyard Patio	69,780	42,095	60.3%
Entry Door Replacement | Steel	2,175	1,542	70.9%
Grand Entrance | Fiberglass	9,867	7,975	80.8%
Garage Door Replacement	3,970	3,700	93.2%
Window Replacement | Vinyl	19,102	16,236	85.0%
Window Replacement | Wood	23,317	19,024	81.6%
Siding Replacement	20,879	15,997	76.6%
Manufactured Stone Veneer	10,234	13,450	131.4%
Roofing Replacement | Asphalt Shingles	28,645	21,850	76.3%
Roofing Replacement | Metal	52,628	29,750	56.5%

Fig. 2. Renovation data for San Jose for year 2019

Figure 2 shows Job cost, Resale value and Cost Reoccupied for 22 renovation projects carried out in San Jose in year 2019.

IV. ETL

Extract, Transform, and Load (ETL) is a category of software focused on data transformations. Rinnovation extracts raw data in the form of PDF files by scraping `https://www.remodeling.hw.net/cost-vs-value` for each year between 2008 and 2019. That raw data is then transformed via two distinct processes. The first employs tabula-java[2] to retrieve the tables within the raw PDF documents and convert them to CSV files (via scala-csv[3]); 1 CSV file is produced for each (city, year) pair. The second merges the intermediate CSVs into 1 file per city.

Initially, the interface between the ETL and ML portions

of the project was poorly-defined. During development, it became clear that the second step of the ETL pipeline would be necessary, thus it was prioritized. The uncertainty regarding the interface was one of the key challenges we faced. We encourage you to focus on this problem in your own projects.

Eventually, the second stage of the ETL pipeline was expanded to load data in to a Postgres database hosted on Amazon Web Services (AWS)[4]. The data was also enriched with geospatial information (latitude, longitude, and human-readable English names) via the Google Maps Geocoding API[5].

Data combined for San Francisco is shown in Fig 3.

	year	Attic Bedrc	Backup Po	Backyard P	Basement	Bathroom	Bathroom	Bathroom	Bathroom	Bathroom	Deck Addit
0	2008	108.6	75.3		101.2	92.4	88.8	114.1		89.8	110.1
1	2009	102.6	53.1		97.2	80.6	75.1	95.2		78.1	92.7
2	2010	104.6	46.2		107.1	81	79.8	101.9		74.4	101.9
3	2011	101.6	46.4		85	75.9	77.9	93.3		85.5	84.1
4	2013	113.5	60.8		105.2	89.2	86.4	118.8		95.8	102.6
5	2014	135.2	92.9		117.8	111.1	99.2	136.5		118.2	126.2
6	2015	133	88.9		124.4	101.2	97.7	119.9		95.7	114.4
7	2016	115.3	74.6		122.2	90.1	94	102		91.4	96.3
8	2017	179.4	78.7	88	117.1	83	98	110		94.9	111.8
9	2018			70.9		100.4	98.5	121.3		94.5	108.4
10	2019			75.4		80.5	76.1	87.8	79.2	82.2	

Fig. 3. San Fransisco renovation data from 2008 to 2019

Figure 3 shows renovation data combined for year 2008 to 2019.

V. Data Processing

The final data was not stationary and to successfully implement Time-Series Prediction, it had to be made strict stationary. Figure 4 depicts the stationarity of the data before processing. We used Kwiatkowski-Phillips-Schmidt-Shin (KPSS) and Augmented Dickey–Fuller(ADF) test to check for Stationarity of data.

Result of KPSS and ADF can be summed up to 4 cases as follows:

Case 1: Both tests conclude that the series is not stationary then, series is not stationary.

Case 2: Both tests conclude that the series is stationary then, series is stationary.

Case 3: KPSS is stationary and ADF is not stationary then, trend stationary, remove the trend to make series strict stationary.

Case 4: KPSS is not stationary and ADF is stationary then, difference stationary, use differencing to make series stationary.

By KPSS and ADF, data was found to be trend stationary. To make the data strict stationary from trend stationary "Differencing" and then "Log Transformation" was used.

Differencing is the method to compute the difference of consecutive terms in the series. Differencing is typically performed to get rid of the varying mean. Whereas Log Transformations are used to stabilize the non-constant variance of a series.[6]

VI. Data Modeling

Predictive modeling is a process that uses data mining and probability to forecast outcomes. Each model is made up of

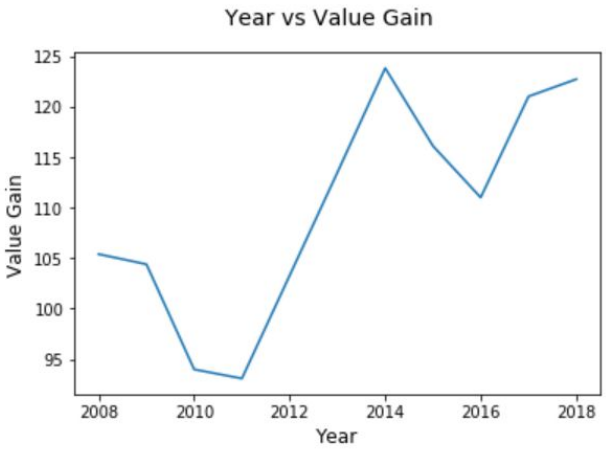

Fig. 4. Non Stationary Data

a number of predictors, which are variables that are likely to influence future results. A number of models were trained for data available for year 2008 to 2019 and the prediction was made for year 2020. For determination of right model, prediction of the year 2019 was made by feeding in the data until 2018 and getting the value for 2019. The model whose result was closest to the actual value of 2019 was selected for the project. Auto-regression model was then selected for the project. Models considered for the project include: Auto-regression, Moving Average, Auto-regressive Moving Average, and Seasonal Auto-regressive Integrated Moving-Average.

A. Auto-regression

An auto-regressive (AR) model predicts future behavior based on past behavior. The process is basically a linear regression of the data in the current series against one or more past values in the same series. AR model usually gets "close enough" for it to be useful in most scenarios. AR model was trained for data available for city San-Francisco for years 2008 to 2017 and prediction was made for 'Bathroom Remodel — Midrange' category for the year 2018 and the result obtained was 119.561413.

Fig. 5. 'Bathroom Remodel — Midrange' prediction for year 2018 - San Francisco with AR

Figure 5 shows the prediction made for 'Bathroom Remodel — Midrange' category for year 2018 for city San Francisco with Auto Regression model.

B. Moving Average

A moving average is a technique to get an overall idea of the trends in a data set; it is an average of any subset

of numbers. MA model was trained for data available for city San Francisco for years 2008 to 2017 and prediction was made for 'Bathroom Remodel — Midrange' category for the year 2018 and the result obtained was 113.510001.

Fig. 6. 'Bathroom Remodel — Midrange' prediction for year 2018 - San Francisco with MA

Figure 6 shows the prediction made for 'Bathroom Remodel — Midrange' category for year 2018 for city San Francisco with Moving Average model.

C. Auto-regressive Moving Average

ARMA model is combination of Auto-regression and Moving average model. ARMA model attempts to capture aspects of both of these models when modeling financial time series. ARMA model was trained for data available for San-Francisco for years 2008 to 2017 and prediction was made for 'Bathroom Remodel — Midrange' category for the year 2018 and the result obtained was 117.938129.

Fig. 7. 'Bathroom Remodel — Midrange' prediction for year 2018 - San Francisco with ARMA

Figure 7 shows the prediction made for 'Bathroom Remodel — Midrange' category for year 2018 for city San Francisco with ARMA model.

D. Seasonal Autoregressive Integrated Moving Average

Seasonal Autoregressive Integrated Moving Average, SARIMA or Seasonal ARIMA, is an extension of ARIMA that explicitly supports univariate time series data with a seasonal component. It adds three new hyperparameters to specify the autoregression (AR), differencing (I) and moving average (MA) for the seasonal component of the series as well as an additional parameter for the period of the seasonality. SARIMA model was trained for data available for San Francisco for years 2008 to 2017 and prediction was made for 'Bathroom Remodel — Midrange' category for the year 2018 and the result obtained was 124.359338.

Figure 8 shows the prediction made for 'Bathroom Remodel — Midrange' category for year 2018 for city San Francisco with SARIMA model.

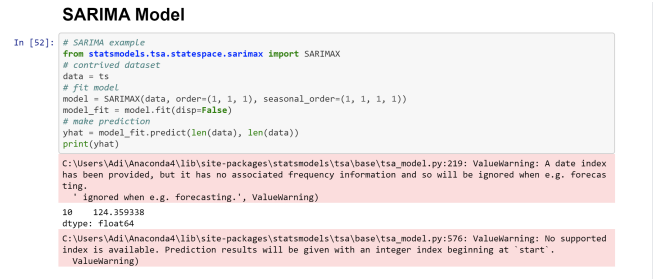

Fig. 8. 'Bathroom Remodel — Midrange' prediction for year 2018 - San Francisco with SARIMA

VII. Service

Rinnovation is architected so that clients interface with a single service. The service abstracts away all infrastructure details for accessing the Postgres database, thereby limiting exposure of security credentials. The service is built in Scala using the Play Framework. It uses Scala's built-in functional programming constructs such as map, filter, and reduce.

The service is built and run with Docker. The Docker images are pushed to Docker Hub. You can find them at `https://hub.docker.com/repository/docker/rinnovation/rinnovation-service`.

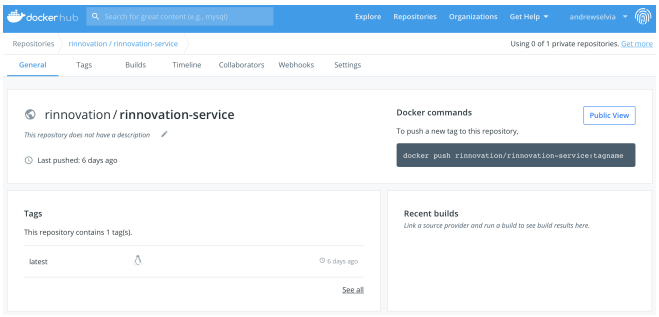

Fig. 9. The rinnovation-service docker image hosted on Docker Hub.

The official deployment of the service is automated via the AWS Elastic Container Service (ECS)[7]. An ECS task definition was created to retrieve the latest image from Docker Hub and deploy it to a cluster there[8]. Updates are still processed manually. We have ambition to automate this process with each commit to the master branch of our GitHub repo, but have run out of time to achieve CI/CD.

The service exposes three relevant endpoints: /cities, /renovations, and /costs-recouped; each is accessed via GET.

A. /cities

This endpoint returns city information for each of the 150 cities. A query parameter for renovation is exposed so that clients can limit the cities retrieved to those for whom data for a given renovation exists. This endpoint queries Postgres to retrieve data. Then, it returns a JSON string containing the city names along with their respective latitude and longitudes.

B. /renovations

The /renovations endpoint returns a list of renovations. Of the possible 43 values, only those for which data exists for a given city provided via a query parameter will be returned. A flat JSON array is returned with renovation names.

C. /costs-recouped

Finally, this endpoint returns a JSON string mapping years to the cost recouped for a given city and renovation (provided via query parameters). All years from 2008 to 2020 (including the predicted value) are included in the final response.

Fig. 10. The response from the /costs-recouped endpoint for Grand Entrance renovations in San Francisco.

VIII. Client: iOS App

Rinnovation is most visible to its customers as an iOS application (app). The app is written in Swift 5. It uses standard Apple frameworks like UIKit and MapKit, but also integrates a third-party framework: Charts[9]. The third-party framework is brought in using the Carthage build tool[10]. The others are bundled in iOS automatically.

A. UIKit

UIKit is used to construct all the core visual elements of our app. All its buttons, tables, and navigation elements come directly from UIKit. Specifically, we utilize UIPageViewController, UIViewController, UITabBarController, UITableView, and UIButton. We build the entire app programmatically rather than relying on Storyboards or the newer SwiftUI framework since this approach grants us more control. Despite being a classic API, the programmatic way of building user interfaces allows us to move more quickly. Whatever cost we would have saved in number of lines of code with one of the newer frameworks is superseded by the time savings of not having to learn a new framework.

B. MapKit

Rinnovation uses MapKit to expose our power feature: enabling users to browse across a map of the United States to find the market with the highest predicted ROI for a given renovation type. We think this will be very popular with the iBuyer property manager persona since they will need to make decisions at a scale the normal home buyer or home owner will not demand. It is also a distinguishing feature. We are unaware of any other solution that provides this detail, therefore we consider it to be highly marketable.

C. Charts

Charts is used to display the graph of recouped costs for a given renovation and city over time. We chose a line graph since it clearly communicates the historical trend. We highlight our prediction with a red circle to call attention to the added value we have produced.

IX. Code Quality

Code quality can be expressed by whether the code is: reusable, easy to read, easy to maintain, and tested. Various tools are used to make sure the code quality is maintained in project. Tools used are Bandit for Python, Kiuwan for Scala, and SwiftFormat for the iOS app.

A. Bandit

Bandit is a tool designed to find common security issues in Python code. To do this, Bandit processes each file, builds an AST from it, and runs appropriate plugins against the AST nodes. Once Bandit has finished scanning all the files, it generates a report. Bandit report is is shown in Fig 11

```
C:\Users\Adi\Desktop\SJSU\272>bandit -r C:\Users
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.7.1
Run started:2019-11-29 18:18:30.235584

Test results:
        No issues identified.

Code scanned:
        Total lines of code: 127
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0.0
                Low: 0.0
                Medium: 0.0
                High: 0.0
        Total issues (by confidence):
                Undefined: 0.0
                Low: 0.0
                Medium: 0.0
                High: 0.0
Files skipped (0):
```

Fig. 11. Bandit Code Analysis of Python code

B. Kiuwan

Kiuwan is a tool for static code analysis. It checks for a broad range of common programming errors, in-line comments, and documentation. Kiuwan covers the most stringent security standards such as OWASP and CWE. Kiuwan report is is shown in Fig 12

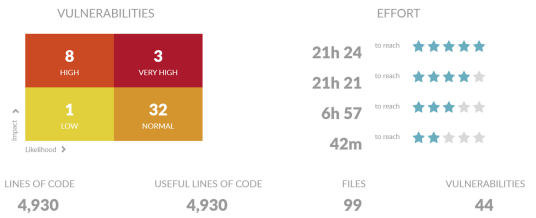

Fig. 12. Kiuwan Code Analysis of Scala code

C. SwiftFormat

We used SwiftFormat for iOS application code. SwiftFormat is a code library and command-line tool for reformatting swift code on macOS or Linux. SwiftFormat report is is shown in Fig 13

```
[Adityas-MacBook-Pro:downloads adityaagrawal$ swiftformat .
[Running SwiftFormat...
[warning: No swift version was specified, so some formatting features were disabled.
 ing a .swift-version file to your project.
 SwiftFormat completed in 0.14s.
 13/15 files formatted.
```

Fig. 13. SwiftFormat Code Analysis of Swift code

X. Conclusion

Renovation projects are affected by many factors including renovation type, location, and real estate market trends. Rinnovation has discovered a way to predict the ROI for 43 different renovation projects across 150 US cities. We believe Rinnovation offers home owners, home buyers, and property managers a compelling solution. Equipped with city-specific, detailed data, they now have the power to make more informed decisions. The convenience and design of the iOS app make it a user-friendly proposition as well.

XI. Future Improvements

While we are proud of how far Rinnovation has come in only one semester, we know the idea still has room to grow.

A. CI/CD

We were unable to integrate CI/CD into our project, but view it as a vital feature going forward to ensure quality. Ideally, with each commit to the master branch of our service repo, we would run automated unit tests and kick off an automated Docker build, push, and deploy to ECS.

B. Add More Pro Features

The iOS app still has room to grow. We would like to add the ability to compare multiple renovation types on the graph at once. Furthermore, we would like to expose multiple renovation types on the map simultaneously. This requires careful design thinking to ensure a legible user interface.

C. CockroachDB

Our initial architecture diagram shows that we planned to use CockroachDB. While we were able to stand up a CockroachDB cluster locally, we were unable to deploy it despite many attempts. We attempted to deploy it to the AWS Elastic Kubernetes Service (EKS) by following these instructions: `https://www.cockroachlabs.com/docs/stable/orchestrate-cockroachdb-with-kubernetes.html`. We did reach out to the CockroachDB Slack workspace and created a Zendesk ticket on them to update their documentation, but to preserve time, we moved ahead with Postgres. Luckily, we learned that Postgres and CockroachDB use the same protocol, so it should be fairly straightforward to switch over in the future.

D. Akka

Sadly, we also ran out of time to integrate Akka into our ETL pipeline. Initially, we planned to distribute the PDF to CSV conversion logic over an Akka actor system, containerize the application, and deploy it to a cluster on a cloud provider. As time constraints presented themselves, this was deprioritized in order to provide data to the ML stage as early as possible and create a beautiful iOS app.

E. Cost Reduction

Finally, our AWS bill has surpassed $30. It is important that we begin to optimize our architecture to reduce costs. As the professor has stated before: AWS is cheap until you use it. We would like to deploy to different cloud providers such as BlueMix or GCP to compare their offerings.

XII. Deliverables

A. GitHub Repository

`https://github.com/SJSUFall2019-CMPE272/Rinnovation`

B. Presentation

`https://github.com/SJSUFall2019-CMPE272/Rinnovation/blob/master/presentations/Presentation.pdf`

References

[1] Cost vs. Value. [Online]. Available: https://www.remodeling.hw.net/cost-vs-value/2019/
[2] tabula-java. [Online]. Available: https://github.com/tabulapdf/tabula-java
[3] scala-csv. [Online]. Available: https://www.github.com/tototoshi/scala-csv
[4] Create and Connect to a PostgreSQL Database. [Online]. Available: https://aws.amazon.com/getting-started/tutorials/create-connectpostgresql-db/
[5] Google Maps Geocoding API: Get Started. [Online]. Available: https://developers.google.com/maps/documentation/geocoding/start
[6] Data Transformation. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/
[7] Setting Up with Amazon ECS. [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/get-set-up-for-amazonecs.html
[8] Deploy Docker Containers with ECS. [Online]. Available: https://aws.amazon.com/getting-started/tutorials/deploy-docker-containers/
[9] Charts. [Online]. Available: https://github.com/danielgindi/Charts
[10] Carthage. [Online]. Available: https://github.com/Carthage/Carthage