EECS 6893 Big Data Analytics Final Report: NYC Real Estate Exploration

Masatoshi Honda
Columbia University
Department of Electrical Engineering
mh4007@columbia.edu

Stephen Shanko
Columbia University
Department of Computer Science
sjs2287@columbia.edu

William Sickinger
Columbia University
Department of Computer Science
wrs2125@columbia.edu

Abstract

New York City is the most active real estate market in the world. However, the market in the year 2020 has seen a significant change compared to previous years due to the impact of COVID-19. The analysis of such trends and the estimation of real estate prices from various parameters have not been sufficiently explored, particularly considering that future demand my be greatly increased. Based on the 1.5 million datapoints of past and recent transactions provided by the city of New York, we succeeded in making more accurate predictions from 13 parameters using machine learning methods. The system has been built to interactively connect to its preprocessed database using API. By the visualizing techniques, trends and forecasts can be obtained in a more understandable way.

1. Introduction

What has been going on in the real estate market in NYC recently? This is the primary question that motivated our research and development for this project. There has been a lot of speculation and analysis done to understand what is happening in the housing market in NYC. Additionally, there is a lot of interest in being able to estimate the market value of a property and efforts are being made by large companies like Zillow and Trulia. We wanted to develop a multi-functional tool to analyze market trends and to make predictions about the potential value of a given property based on some simple attributes. In order to accomplish this goal, we experimented with various models to perform the prediction. We developed an API that can serve predictions to our frontend application or can be used standalone as part of other applications. And finally we cleaned and hosted the NYC open data around sales and hosted it in our own internal database to support visualization and analytics activities.

2. Related Work

There are several well known companies who are doing similar work with respect to estimation of the value of a property. Real estate companies like Zillow and Trulia have proprietary algorithms and models which attempt to predict the value of these properties across the country. However, our analysis was focused exclusively on NYC in order to bound the scope and scale of the computational resources and data storage needed.

Building Class Category	Neighborhood
Zip Code	Residential Units
Commercial Unites	Total Units
Land Square Feet	Gross Square Feet
Borough	Building Class at Time of Sale
Tax Class at Time of Sale	Year Built
Sale Price	Sale Date

Table 1. These are the columns the remain after unneeded columns are dropped.

3. Data

The data used in this project is New York City property sales from 2003 to early 2020. Our combined dataset can be found split over two datasets on the New York City Open Data website.(See references for urls) The dataset names are NYC Calendar Sales (Archive) and NYC Citywide Annualized Calendar Sales Update. While the Update dataset has an API for convenient download, the archived dataset does not and must be downloaded as 65 CSV files.(on for each borough for each year) All data was loaded into a single CSV dataframe.

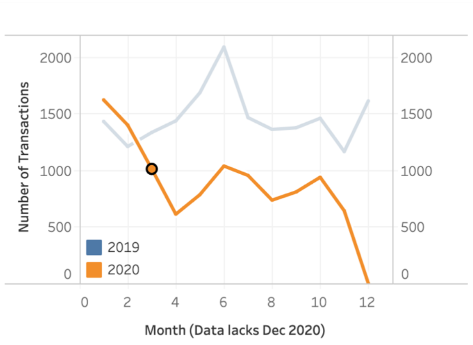


Figure 1. Monthly sales dropped significantly in Manhattan from March 2020

A number of columns that were closely correlated with other columns or would not be helpful for predicting real estate prices or for running visualizations were dropped. For instance, the Address and Lot columns were dropped as they gave data for areas that were too granular for the predictions we were trying to make. The Block column was dropped for representing smaller areas than were useful as well as for being closely correlated with zip code.

The remaining columns in the combined dataset can be seen in Table 1.

Once only the useful columns remained, approximately a third of all sales in the dataset had no sale price and were therefore dropped. This cleaned data was then moved to a MySql server for further use.

4. Methods

4.1. API

To implement the API we considered Django and Flask which are both Python frameworks. We ultimately decided to use Flask because it is extremely lightweight and quick to learn. Additionally, because we implemented the API application as a containerized application, it is possible to easily scale horizontally and therefore less critical to have a fully featured server framework like Django. During implementation of the prediction endpoint, we realized that a given call to the API may or may not have provided all of the parameters required by the model. To account for this we considered two approaches. One approach would involve assessing the parameters passed and returning an error code if the required parameters were not all present. This has the effect of ensuring that the user passes all relevant parameters, but it can be annoying or impossible if the user is missing some data or just simply isn't interested in a highly accurate prediction. An alternative approach was to calculate the mean for a continuous parameter or the mode for a categorical parameter based on the values in the database. We ultimately decided to use this method because it is very user friendly and allows for increasing accuracy as more information is provided instead of either a success or failure.

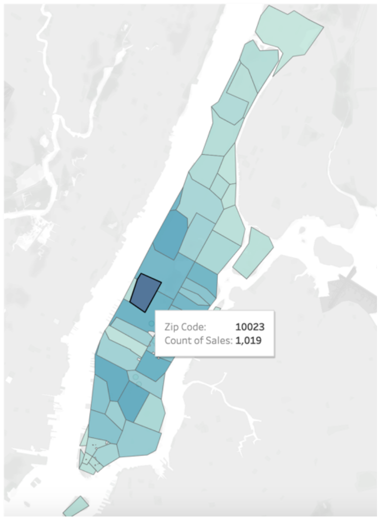


Figure 2. Sales popularity by neighborhood 2020

4.2. Model

Real Estate Market price prediction is a regression problem and we wanted to have the option to scale up and process large amounts of new data very quickly. For this reason we limited our choice of potential models to those available in the Spark Machine learning library. We experimented with Linear Regression, Lasso Regression, Ridge Regression, Random Forests and Gradient-Boosted Trees. For our web interface, we ultimately decided on a Random Forest model because of the high accuracy and speed of distributed training.

Due to the large range of sale prices in even our reduced range(\$100,000-\$10,000,000) using the mean squared error or mean absolute error didn't give a useful idea of the accuracy. Instead we found it easier to understand the mean percentage error, which was calculated from the absolute percentage the prediction varied from the true sale price.

4.3. Frontend

First we obtained latitude and longitude data linked to Zip Code from Neighborhood Tabulation Areas (NTA) provided by NYC Open Data as a GeoJson file. This data was processed by mapbox, a map display tool. Tableau can process a wide range of data sources such as MySQL and CSV files. We customized the data from these files and displayed them as charts, and published the processed data from Tableau Desktop to Tableau Public so that it can be processed as code on the website. After getting a server and domain, the basic framework of the front end was written in php. For the connection to the ML model, schemes were created to allow each parameter to be sent and received

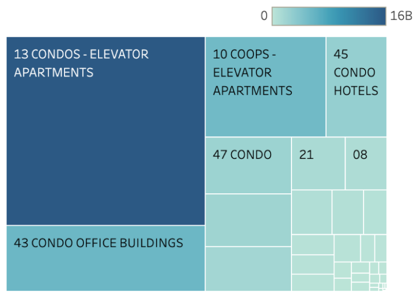


Figure 3. Sales by apartment types in Manhattan 2020

from the API in a Json file. 1) Create a callback function in function.php. 2) Scrape the URL including API key in the function, and acquire the data as a JSON format object. 3) Convert the data to an array type using json decode and receive it. 4) Acquire the value while looping 5) Set the value to a custom field.

For the visualization, we chose a simple linear graph for market size and trading volume in recent years (Figure.1). By hovering the pointer over the map chart, you can interactively understand the volume of transactions colored by the neighborhood for each of the five boroughs (Manhattan: Figure.2). The breakdown of market size by boroughs, as well as by building type, are shown in pie charts and area graphs (Figure.3). The median price and the number of sales for each neighborhood are on axis, and each market size is represented by bubble size (Figure.4). You can see more on http://masatoshihonda.com

Quick takeaways of this year's market can be summarized. Although Manhattan accounts for 55 percent of NYC's real estate market, in 2020 it declined in both number and size by up to 30 percent compared to last year. On a month-by-month basis, the number of transactions remained at nearly half of last year's level from April to June, meaning that the decline in transactions was largely driven by the roughly three months, which is the general transaction time needed in the market, since the start of the quarantined period. In addition, 6.6 percent of the transactions this year were hotel transactions. While recent years have been marked by a sharp increase in the median properties price in Javits Center due to the development of Hudson Yards, this year the median price in the Central Business District reached 2.5-3.4 times that of previous years. This fact can be understood from the large tenants' move-away from Midtown due to the economic change.

5. Experiments

Before modeling experiments could be done, additional preprocessing was needed. First, the Neighborhood column was dropped as it was too fine grained for what we were trying to model. Next, all sales for less than $100,000 or

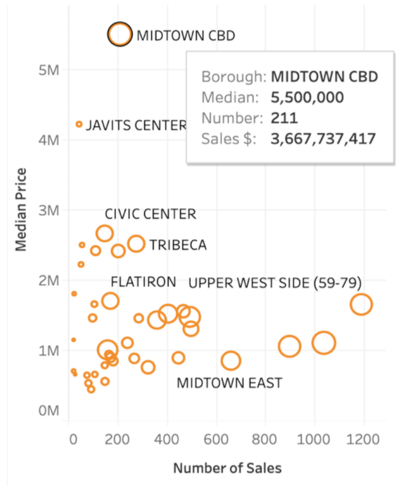


Figure 4. Property price by neighborhood 2020

Model	MPE	MAE
Linear Reg	26.8%	$417,805
Ridge	26.7%	$417,622
Lasso	26.5%	$417,838
Grad Boost	17.1%	$368,797
Rand Forest	13.3%	$334,725

Table 2. Performance on a subset of the data.

more

than $10,000,000 were also dropped as they tended to be outliers and reduced the prediction accuracy of modeling.

Then the BOROUGH, BUILDING CLASS CATEGORY, ZIP CODE, TAX CLASS AT TIME OF SALE and BUILDING CODE AT TIME OF SALE columns were converted to one hot encodings giving a total of 574 features. A training date range was chosen and all records from sales in that range were put in a pandas dataframe. Any one hot encoding columns that didn't have any 1 values appear in this date range were then dropped. A test date range was then given and a second dataframe that had the same columns at the training data was created. Then the Sale Date for each record was converted to an integer representing the number of months before the latest date in the training set. These values were then one hot encoded. The test data Sale Dates were converted in the same way with the exception of dates that were later than the training data latest sale date, these dates were given the value 0(i.e it was if they had occurred in the most recent month in the training set)

We experimented with Linear Regression, Lasso Regression, Ridge Regression, Random Forests and Gradient-Boosted Trees. The results for training on the two year window of 2005 to 2007 and testing on January 2008 can be seen in Table 2.

While the results from the test period did not generalize well to the entire dataset, the relative performance of the models was fairly close. After testing multiple test

Model	MPE	MAE
Linear Reg	37.5%	$404,463
Ridge	37.5%	$404,463
Lasso	37.5%	$404,441
Grad Boost	19.5%	$322,709
Rand Forest	20.0%	$285,191

Table 3. Performance on the entire dataset.

# of Workers	RF
1	94.9
5	24.5
10	17.1

Table 4. Wall clock time(in minutes) for Random Forest run in scikit-learn using the entire dataset with different numbers of jobs(cores). Running with Spark ML library with 5 workers took approximately 391 minutes.

windows, we found that the MPE varied greatly based on whether that particular test month fit the overall model and that there was a great deal of variance between months. However, for both the shorter test windows and the entire range of the data, Gradient Boosted Trees and Random Forests significantly outperformed the other methods.

The results for each method when run on the entire dataset can be seen in Table 3.

Once we knew what our best methods were we wanted to see which would be best for running in an environment where new data would periodically come in(such as new daily real sales data) and a new model would have to created and run on a new set of test data(new real estate listings) to determining what properties were potentially good investments. While it wouldn't be practical for us to run a cluster on our project web page, we believe this would be an appropriate architecture for an investment company.

We first ran Random Forest on GCP clusters with 5 workers to find the time it would take to train a model. We were planning on also testing 2,3 and 4 workers but with 5 workers it too about 6.5 hours. This was significantly slower than training the models in scikit-learn on a single core. Table 4 shows the wall clock times for running Random Forests on the entire dataset using a different number of cores on a single machine. We would expect an effective implementation of Random Forest through spark to be as fast or faster based on the number of workers available as there were 4 cores per worker.

A decision was made to use a random forest model as it is more parallelizable and faster to train.(The decision trees created for Random Forests are independent of each other. Each tree is built on a random subset of the data so no communication needs to happen between trees during training.) We then wanted to experiment to see how small we could make our model. For Random Forests this can be done by reducing the number of trees created and by reducing the

Depth	Test MPE
10	28.1%
20	20.0%
30	16.0%
40	14.2%
50	13.6%
60	13.5%
70	13.5%
80	13.4%
90	13.4%
100	13.4%

Table 5. Test MPE for different depths of a Random Forest on a subset of the data.

Trees	Test MPE
10	13.7%
20	13.5%
30	13.6%
40	13.5%
50	13.5%
60	13.5%
70	13.5%
80	13.6%
90	13.7%
100	13.6%

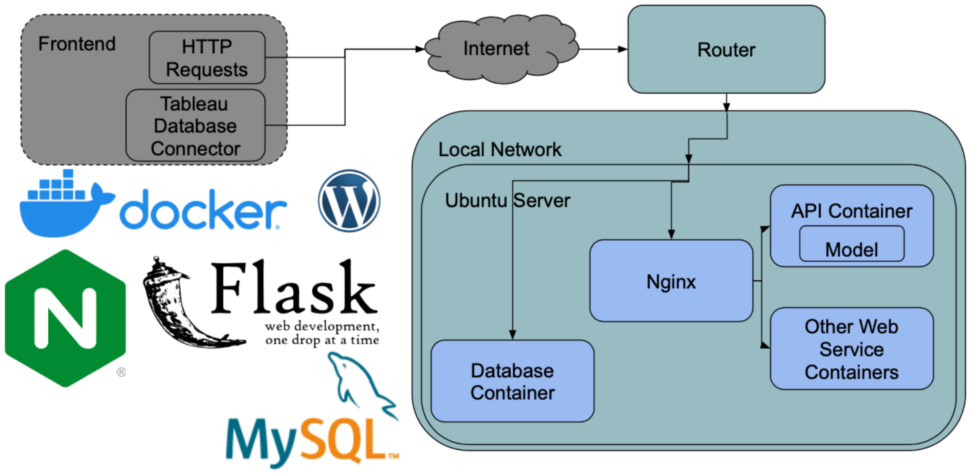
Table 6. Test MPE for different number of estimators of a Random Forest on a subset of the data.

depth of those trees. Smaller models can be much faster to train and saved models take up much less disk space. For example a model with depth 100 with 100 estimators is about 5GB for our data and a model with depth 50 and 25 estimators was about 600MB with very little loss of prediction accuracy. The MPE for different random forest tree depths, with the number of estimators fixed at 100, when run on a subset of the data can be seen in Table 5.

The MPE for different Random Forest number of estimators, with the depth fixed at 50, when run on the same subset of the data can be seen in Table 6. It's likely that the number of estimators isn't having much of an effect because even in the two year span, there are enough data points that the variance is already low.

While a depth of 50 seems to be roughly the point where you stop seeing improvement on the holdout test data, this would not be possible when using the Random Forest Regressor in the Spark ML library as it is limited to a depth of 30. While the loss of accuracy would likely be acceptable for our use case, if even more data were available for a more accurate price estimate, with the longer train times and the limited depth in Spark ML a better option may be to use Spark and scikit-learn to build decision trees at each worker and ensemble them yourself.

The reason that we would be willing to trade some ac-



curacy for speed is that we don't believe that our tool as it is now would be enough to make an investment decision on it's own. In general, given capital outlay or real estate investments, it would be unwise to ever use a model without more thoroughly investigating what you are purchasing. We believe that this predictive modeling would function more as a tool to allow a company to quickly decide which properties are worth having their teams of experts look into so resources could be employed in the most efficient way possible. In a hot real estate market, choosing the right properties to appraise and research could lead to getting offers in before competitors so you could maximize profits.

6. System Overview

Our application is segmented into 4 main parts: frontend, API, model, and database.

Each of these parts are independently functional and provide incremental value to the application as a whole. We decided to use free software for all portions of the stack and to use self hosting for the backend portions in order to reduce any dependence and costs associated with cloud infrastructure. This choice led to a dependency on the upload/download speed of the internet connection as well as the performance of the server hardware. If this application were to be put into production, we would plan to migrate these various components to the cloud in order to provide higher levels of scalability, compute performance, and network speed. We have taken steps to prepare for such a migration by heavily leveraging docker containers for our API and database components. Our primary technologies used across the stack are docker, Flask, nginx, MySQL, Wordpress, Python, Javascript, PHP, Tableau and Mapbox. By using Python for both the model building and the API, it became very easy to integrate the model into the API using a few simple calls and the same Python packages in both efforts. Due to the self-hosted nature of the back end, there was a significant effort in networking and routing to ensure that all the appropriate database and API calls were able to reach the appropriate containers for processing. Part of this effort consisted of domain name records and port forwarding rules in the router in order to pass http requests on port 80 and database requests on the standard mysql port 3306. We made use of nginx in order to facilitate reverse proxying based on hostname and url pattern in the http header. This allowed us to easily route traffic to the API container and also to host additional web services and websites on different hostnames with no conflict. Nginx was configured to proxy requests to the appropriate localhost port and the docker container run commands were designed to forward the localhost port to the correct application port within the container.

Here is an example of a complete workflow:

1. Front end code makes an API call to 6893.stephenshanko.com/api/v1/*

2. 6893.stephenshanko.com domain name A record directs traffic to the router's IP address 24.0.247.115

3. Router port forwarding rule forwards port 80 TCP inbound requests to LAN IP 192.168.86.xxx

4. Nginx site rule proxies any http request to 6893.stephenshanko.com/api/v1/* to http://127.0.0.1:8456/api/v1/*

5. Docker container with Flask application is run using the command "docker run -d -p 127.0.0.1:8456:5000 –restart unless-stopped 6893app:v1

We leverage MySQL due to the ease of setup and prior experience with this particular database service. There is also an officially supported docker image provided by the MySQL team which made deployment of the database very quick and simple. In order to provide security controls two users were created for the database, each assigned a random strong password. One user (db_editor) was given permissions to modify and read the database, while the other (db_reader) was only given permissions to query the database. We used db_editor when loading cleaned data into the database and we use db_reader from the API and Tableau in order to prevent accidental or malicious loss of data. In order to expedite the visualization process and develop many high quality visualizations, we leveraged Tableau and began by ingesting the data in CSV format for quick experimentation. After quick iteration and trying out some different visualizations, we integrated directly with the database using the MySQL connector for Tableau.

7. Conclusion

We were able to make a scalable web based tool that allowed for the visualization of trends the New York City real estate market as well as make predictions on the sale price of individual properties. These trends and predictions can be used to provide the market intelligence which buyers need to find the best price and the and for agents to find the areas of the city most attractive to prospective buyers. In it's current state, we believe that the trend visualization feature is the most useful feature. Being able to quickly get sales data for areas of various size would be invaluable when quickly deciding where to spend time researching properties. We also believe that based on the data we had available our predictive model performs well but that additional data could significantly improve prediction accuracy.

There are a number of ways that our project could be extended. As stated above, the power of the predictive model could be improved with more data. The next step to gathering this data would be to scrape real estate listings and count assessors offices to get a more accurate pricing model. Given this more accurate model, it would interesting to find the benefits of renovations in the city. While we had the dataset for renovation permit applications. the price modeling would need to be more more specific to the features the building being remodeled to be useful.

From a deployment standpoint, it would be worthwhile to optimize the prediction model integration with the API in such a way that the model is always running in the background and can handle requests without the added latency of loading the model for each call. For a course project this wasn't really practical but to use this tool in a investment environment this would be the logical next step.

8. References

Datasets used in this paper can be found at:

https://data.cityofnewyork.us/Housing-Development/NYC-Calendar-Sales-Archive-/uzf5-f8n2

https://data.cityofnewyork.us/City-Government/NYC-Citywide-Annualized-Calendar-Sales-Update/w2pb-icbu