Intro

Divinus aims to sort through current and past real estate data to provide users an informative visualization of the current real estate market within Manhattan. Through this project, users would have access to potential investment opportunities, a visualization of over/underpriced neighborhoods, and an estimated value for a distinct property value. Divinus aims to provide three main interfaces. To start, users would have access to a heat map of Manhattan illustrating a color-coded representation of mean prices. Secondly, users would have access to a predictive estimator outputting our algorithm's suggested price for your property. Finally, users would see an area where properties are underpriced yet rentals are overpriced to find high-return investment properties.

Data Collection

Part of the data our model is trained on is school-ranking data. Using the SchoolDigger AP we were able to generate a CSV containing school rankings by zip code. This was used to supplement data for our model to train on.

In addition to school data, further datasets relevant to real estate pricing were pulled from NYC Open Data. Datasets including bedbug reports, tax assessor information, land use, and community district statistics were compiled to provide more information about each property our model trains and tests on. Each feature extracted from these relevant datasets ideally improves the accuracy of our model.

Our project revolves around three different sets of real estate data within the Manhattan area: historical building sales, current apartment sale listings, and current rental apartment listings. We used two methods to pull these three datasets: Socrata APIs and web scraping.

Our group utilized a dataset from NYC Open Data to pull historical building sales data. To access the data from NYC Open Data, we used Socrata APIs. We first filtered the data to represent Manhattan data points, then used basic filtering methods to remove invalid, NaN, and extreme data points. Finally, we used regular expressions to remove any buildings that the data set did not list as a co-op, dwelling, condo, or rental. We stored the result of the filtering in a pandas data frame.

To access current apartment sale and rental listings, we choose to scrape our data from Compass. We first created a list of valid Manhattan zip codes. We utilized a for loop to search Compass for listings within each initial zip code. Inside the for loop was a nested for loop that served to paginate each unique zip code search. Because the property listings had little data listed on the main page, we chose to scrape each property URL and store it within another list.

Once we had collected a list of property URLs, we further scrapped each unique URL for the data we needed. We began by initializing lists for each piece of data we were looking to collect. We then used a for loop to iterate over the list of URLs: first checking for a valid connection, then initializing a BeautifulSoup object. If the URL contained the data we were looking for, we would append the data point to the respective list. If the URL did not hold a data point we were looking for, we would append the value 'NA'. We collected data from each URL for two purposes: improve our machine learning model and collect text data for later analysis. Data entries such as bedrooms, bathrooms, square footage, and floor number are some of the listing data points we felt were critical in constructing a comprehensive predictive analysis model. Fields such as listing description and amenities were data points we felt would be crucial in building a text analysis model for our project through natural language processing (this is a future feature that we will add if time permits that would allow the user to enter a listing description that we would rate and suggest edits to improve the odds of selling the house above

the predicted price). After we scrapped each unique URL, the data was stored in a pandas data frame then exported to the SQL database; this same process was repeated for the other sets.

Maps:

Because our project focuses on real estate in the Manhattan area, we felt that maps would be the best way to represent our data analysis visually. For each of our three primary datasets, we used a heatmap and choropleth. We used a GeoJson file from data.beta.nyc to match the postal codes from the GeoJson to the zip codes in our datasets. We filtered all rows whose borough was not Manhattan and one row whose borough listed Manhattan but was the Bronx.

We used the heatmaps to visualize the volume of sales for each dataset over Manhattan. Using this visual, users can understand which areas of Manhattan have relatively active and inactive real estate markets. To construct the heatmaps, we used the Heatmap plugin from the library Folium. We used the Latitude and Longitude data to represent each data point used in our machine learning model.

We felt that choropleth maps were the best way to visualize our analysis. The choropleth maps allow us to visualize the difference between the actual and our predicted price point averaged for the zip code. To represent the analysis, we first found the difference between our predicted and the original unit price. We then calculated the average difference between the predicted and actual price for each zip code. We finally used the Choropleth method in the Folium library to take our data and construct a choropleth map. The maps represent how over/under-valued each zip code's real estate. We wanted users to interact with these choropleth maps, so we used GeoJson Tooltips to implement tooltips. This feature allows users to hover over each zip code which causes the map to display the exact zip code the user is looking at and the average difference between our prediction and actual sale price in dollars.

We decided to use Folium for our maps because we felt that they were the most interactive. The base map from the Folium library contains a high level of detail down to the street level. The maps constructed using Folium are also more interactive, allowing the user to move around the map and zoom in/out.

Data and Model Visualization

To better represent the data we had collected, we had utilized in-built Python graphing libraries as well as the SHAP library to showcase the various datasets. SHapley Additive exPlanations, or SHAP, is a game-theoretic approach to explain the output of our ML model. We used the ‘beeswarm’ data representation to showcase how house price sales data is impacted by other datasets; namely, zip code, # bathrooms, # bedrooms, unit square footage, police precincts, school ranks, school district rankings, amongst others. Then, using the same data comparisons (house sales vs. the other data sets), we used the ‘global bar plot’ data representation to rank the global importance of each data set to predict house sales data.

We utilized local ‘pandas’ plots as well: using histograms to depict the average number of bedrooms (a prominent variable for house sales data) per zip code, as well as the average house price per zip code. We also used a line plot to visually depict the yearly change in the NYXRSA (the New York Home Price Index).

ML Model

We decided to use a machine learning model for two reasons: accuracy and its ability to handle a large number of features for each property.
XGBoost is a well-known machine learning algorithm that has performed well in many machine learning competitions on Kaggle. It's a decision-tree-based ensemble ML algorithm, which is known for its accuracy in structured data.

XGBoost uses decision trees to create many iterative models, each time utilizing gradient descent to decrease residual/error. Throughout this process, it assigns more or less weight to features deemed more predictive or less predictive respectively. This type of algorithm should perform well in our application using structured datasets and is especially useful because it allows for None values if any features are unknown. Additionally, it will provide useful information about which features are most important when pricing a property or rental.
The machine learning prediction aspect of our project focuses on three different areas of Manhattan real estate. Three property datasets were created for prediction and training purposes in the model. The building sales dataset was found on NYC OpenData, whereas both the rentals and unit sales datasets were scraped from Compass.

Using the additional datasets created in our data collection stage, we looked for relevant features which could help explain prices for buildings, rentals, and units. Relevant features include zip code, square footage, number of rooms, number of nearby schools, etc. Once all relevant information/features were compiled for each property in our three property datasets, each feature was transformed based on its type.

Regression features were converted to floats from the strings either found in the datasets or returned through scraping. Categorical features were one-hot encoded, through the creation of dummy variables, to be interpretable by a regression model. Each model uses well over 100 features to predict a price.

Once each property dataset is created and its features are transformed, three XGBoost models, reflecting buildings, rentals, and unit sales, were initialized and fed the compiled property datasets as X variables and their respective prices as Y variables. The datasets are split into training and test data using K-fold cross-validation. This K-fold process allows us to test and ensure our model generalizes to and performs well on new data. K-fold cross-validation is also useful as it helps us diagnose under and over-fitting on training data when comparing each model's results on both the testing and training sets.

Throughout this process, we tested including different features to achieve the highest accuracy on our test sets. Furthermore, there are many arguments passed when initializing the XGBoost algorithm. Tuning the values of each argument can further improve the model's ability to generalize by preventing over and underfitting on the training set.
Each of the three models was tuned until they returned the highest accuracy we could achieve given our datasets. Once these models are created, coded instructions for recreating each model can be saved in JSON format.

Our final project has three price prediction tabs; each tab focuses on its model and area of Manhattan real estate: buildings, rentals, and unit sales. For each tab, the relevant model is initialized using the JSON instructions and is ready to receive an array of features to make a prediction.

The models require an array of features exactly like the features that it trained on to make a prediction. For the building sales price prediction tab, the information used to train the model was publicly available for all buildings in Manhattan. This means we only need the user to input an address, from which we can web-scrape the Borough, Block, Lot (BBL) number. Using the

BBL, we search through datasets uploaded to our SQL database to compile an array of features for the address; this process takes a few seconds. Once this array is passed to the XGBoost model, a prediction is printed on the screen. Due to a lack of sales information for high-value, large residential buildings, this model is most accurate on smaller residential buildings which sell more often in Manhattan.

Since there is less publicly available information on individual units for sale or rent, each of these two tabs requires more information to be entered by the user. Each feature used to train our model should be listed on a real estate website or could be retrieved easily from an agent for any property a user of our website may want to predict the price of. Once the user inputs an answer for each feature, this feature array is compiled and passed to the relevant XGBoost model, and a prediction is printed on the screen.

Natural Language Processing

Towards the end of the semester, we liked where our project was at but wanted to extend its application beyond price predictions. Natural language processing was an application that we found to be interesting, relevant in scope, and unique from other property price predictors. We first ran each description through the IMB sentiment API. We then calculated the average sentiment score for listings that our model predicted as overvalued. Then we used and combined the scrapped listing descriptions from the rental and unit sales data into respective, all-encompassing string values. We then lemmatized each respective string and stored the results into corpus values. We then ran the corpus values through multiple three main processes: word cloud, RAKE, and n-grams.

For our word clouds, we utilized the library wordcloud. The process first involved removing any common English stop-words from the corpus, then using the libraries package to generate respective word clouds representing the fifty most used words from all listing descriptions of the respective dataset.

For the RAKE process, we utilized the library rake_nltk. We used RAKE for both frequency and degree, creating our ratio of frequency divided by degree for each keyword or phrase. We combined the listing description from each listing into one large string, then used RAKE over the entire string. After initializing a RAKE list for frequency and degree, we found the ratio for each keyword/phrase and added each respective value to a data frame. We then used each row as a data point in a linear scatter plot, graphing word frequency on the x-axis and degree on the y-axis.

To find the most common one, two, and three-word phrases, we utilized n-grams. We first used the corpus created earlier through lemmatization and combined the result into one large string. We then traced over the string using n-gram functions from the nltk.util library to find both the n-grams and frequency of n-gram for each uni, bi, and tri. After collecting all of the data for each n-gram, we compiled the top 20 n-grams from each uni, bi, and tri result into a respective data frame. We then used the respective data frames to construct horizontal bar charts displaying the n-gram on the y-axis and the frequency of its use on the x-axis in ascending order. We wanted to also utilize this collected data into an interactive advisory tool. On our application, this tool has its tab for the respective unit and rental datasets. First, the function requires the user to input their working listing description that is a minimum of three words. The function then takes the description and runs it through the IBM API. If the description is above our threshold score, it returns a success message. If the description does not exceed our threshold score, we dissect the description into sentences and words, returning to the user the worst sentence and the

worst word within the worst sentence. We then RAKE the user's description, identifying any keywords or phrases. For the user's listing, we identify their strongest keyword or phrase based on our RAKE assessment and then return the percentile of their best keyword or phrase in our dataset. We then select three random keywords or phrases from our data that are in a higher percentile as recommendations to improve their description. If their description has no keywords or phrases, the function selects any three random keywords or phrases from our RAKE data. We then use n-grams on their description to identify any common one, two, or three-word phrases that they have used. For any words or phrases that are matched to our top 20 n-grams, we return the words and their rank within the top 20. If no words or phrases from our top 20 are found, we return three suggested words or phrases that are within the top 20 most common across all listings.

Summary:

Our real estate project can be broken down into several main components: data collection, cleaning, modeling, and finally hosting. After identifying our sources for data collection, we downloaded data from available sources, then created a web scraping bot to pull supplemental data not included in the more easily accessible data. Afterward, we cleaned the data, removing any potential noise and missing data points, so that our model would be more accurate. Once cleaned, we ran our data through a linear regression machine learning model to generate predicted pricing/scores for properties based on their different attributes. Using these generated data points, we were able to create several visualizations, chief among them a representational heat-map overlay of Manhattan, displaying over/undervaluations of properties. All of this is then to be hosted on a flask website for final submission.