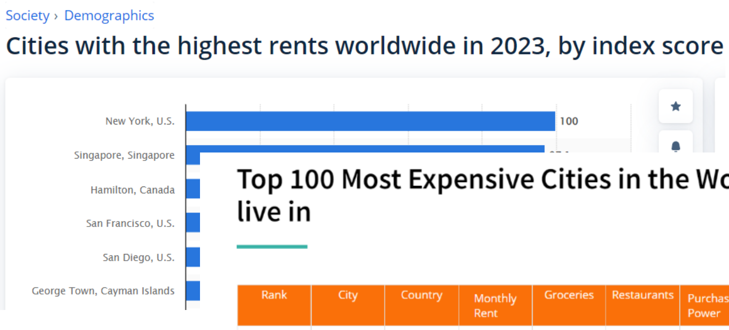
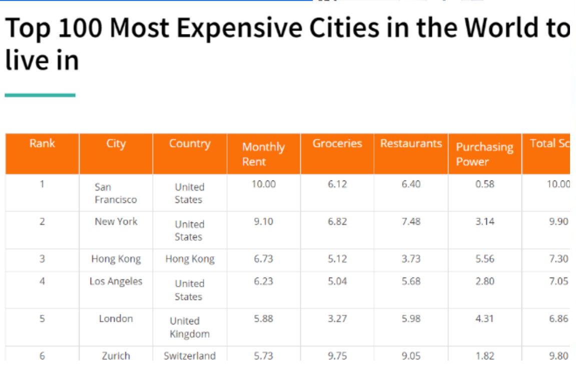
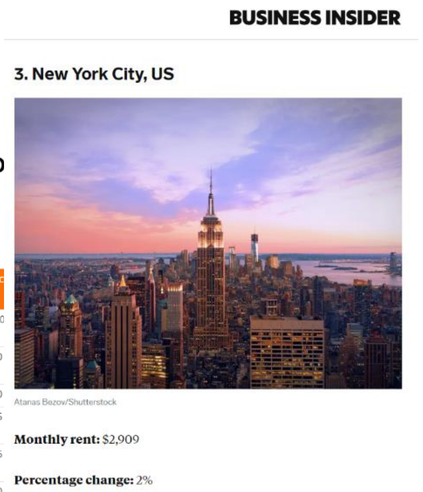
Machine Learning for Airbnb rentals in New York





Top 100 Most Expensive Cities in the World to live in

Rank	City	Country	Monthly Rent	Groceries	Restaurants	Purchasing Power	Total Sc
1	San Francisco	United States	10.00	6.12	6.40	0.58	10.00
2	New York	United States	9.10	6.82	7.48	3.14	9.90
3	Hong Kong	Hong Kong	6.73	5.12	3.73	5.56	7.30
4	Los Angeles	United States	6.23	5.04	5.68	2.80	7.05
5	London	United Kingdom	5.88	3.27	5.98	4.31	6.86
6	Zurich	Switzerland	5.73	9.75	9.05	1.82	9.80



BUSINESS INSIDER

3. New York City, US

Atanas Bezov/Shutterstock

Monthly rent: $2,909

Percentage change: 2%

Image References (Left to Right) – Statista, HousingAnywhere, BusinessInsider

Overview

As part of the CPSC330 assignment at UBC, I got an opportunity to analyze a subset of New York's real estate on Airbnb rentals and create a machine-learning model. New York, being one of the most expensive cities in the world, along with being a popular tourist destination, makes Airbnb rentals an interesting figure to analyze. Below, I will present my work on my machine-learning model to predict what kinds of rentals are popular.

Analyzing Data

I am using this dataset on Kaggle, which has a list of all the Airbnb rentals from 2019.

Before we start building our machine learning model, it is important to take a look at the data that we are going to work with.

On the right, you will notice there are several features, but under the column "non-null", you'll see some numbers do not match. What this means is that some entries for that feature are missing. This is especially

```
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              38792 non-null  int64
 1   name                            38792 non-null  object
 2   host_id                         38792 non-null  int64
 3   host_name                       38787 non-null  object
 4   neighbourhood_group             38792 non-null  object
 5   neighbourhood                   38792 non-null  object
 6   latitude                        38792 non-null  float64
 7   longitude                       38792 non-null  float64
 8   room_type                       38792 non-null  object
 9   price                           38792 non-null  int64
 10  minimum_nights                  38792 non-null  int64
 11  number_of_reviews               38792 non-null  int64
 12  last_review                     28440 non-null  object
 13  reviews_per_month               28440 non-null  float64
 14  calculated_host_listings_count  38792 non-null  int64
 15  availability_365                38792 non-null  int64
 16  number_of_reviews_ltm           38792 non-null  int64
 17  license                         2939 non-null   object
```

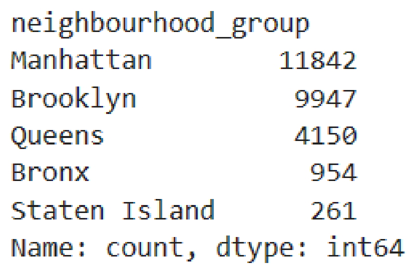
important as missing data can cause inconsistencies within our model. I will describe how to deal with this in the next section.

Oh, and also, one thing to remember is the data types with which we are dealing. From the results, we got a few types like ints, floats, and objects. Should we categorize these?

Cleaning and Preprocessing Data

Categorizing features

1. **Numeric features** – Features of type int, float etc.
2. **Categorical features** – Features that have a set amount of category. Neighbourhood_group below is an example.
3. **Text features** – Features that describe something and can vary a lot from one another. For example, the description of the rental listing.
4. **Drop features** – Features that do not affect the results like hostID, listingID
5. **Discretized features** – Features like longitude and latitude that can present patterns when transformed.



```
neighbourhood_group
Manhattan        11842
Brooklyn          9947
Queens            4150
Bronx              954
Staten Island      261
Name: count, dtype: int64
```

In short, we will have numeric, categorical, text, discretized, and drop features.

Missing features

I decided to inject 0s into the missing entries of “**reviews_per_month**” because the place may have never been rented. For the other features with missing entries, I am dropping them anyway, so there’s no point in filling them.

Now, we can start building what we need to create our model!

Building a preprocessor pipeline

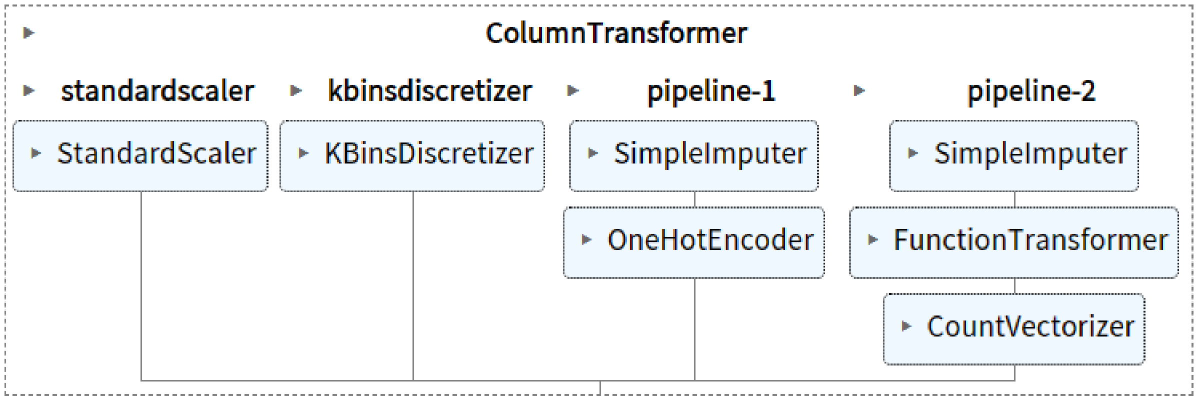
What is a pipeline? Just like how a pipe works, something goes in, something comes out.

In our case, we create a pipeline to feed in raw data we have from our sources and transform them into clean data that is used to build the machine learning model.

Before we start building the pipeline, we need to split the data. This means we will split a good chunk of data into the training portion, and the rest into the test portion. The training portion will be used to train our model, while the test portion will act as unseen data fed into our model to see if our model performed well or not.

How do we know if it performed well or not? The data has a feature called "reviews_per_month" which we will use to compare against what our model will predict for that same feature.

Below is a pipeline I created.



The pipeline can apply transformations to individual data types and produce a collection of data that is clean enough to build our model.

Building ML Model

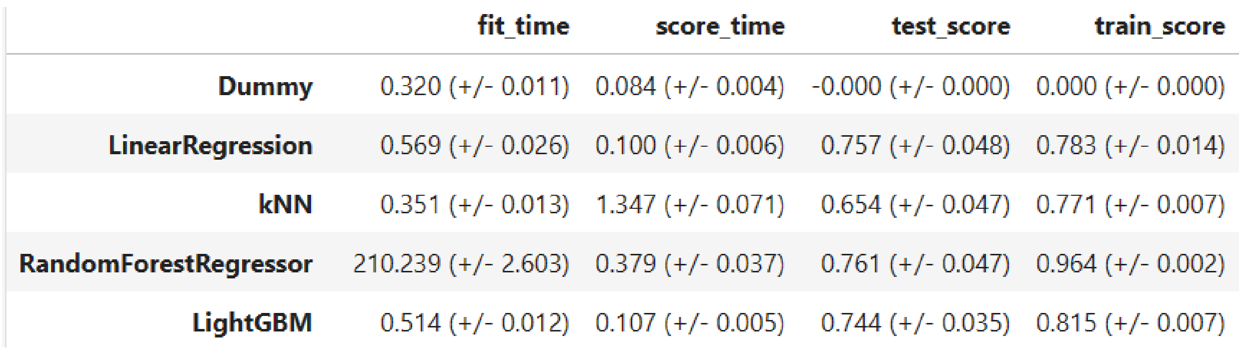
I used a couple of pre-built models with the best possible hyperparameters to decide which one of them produced the best result with my preprocessor pipeline.

Hyperparameters – Settings that one can choose to configure the model's learning process, which influences the model's behavior performance.

The following are the models I used.

1. Baseline (Dummy) – An amazingly simple model that often has poor performance with complex data.
2. Linear (Logistic Regressor) – Assumes there is a linear relationship between the inputs and outputs.
3. KNN (KNeighbors Regressor) – Predicts the result based on the majority of its k nearest neighbors. If k was 3, and 2 of them are "A" while the other is B, that data point would be claimed as "A".
4. Random Forest (Random Forest Regressor) - Predicts by combining many decision trees to obtain robust outputs.
5. LightGBM (LGBM Regressor) – It is also another tree-based model.

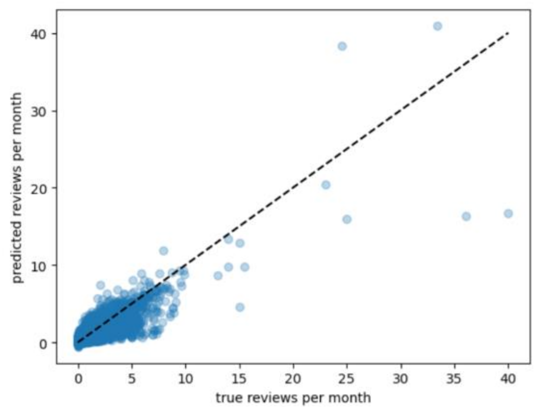
Here are the results –



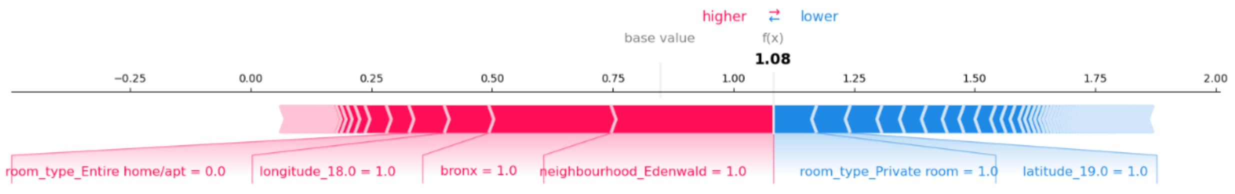
	fit_time	score_time	test_score	train_score
Dummy	0.320 (+/- 0.011)	0.084 (+/- 0.004)	-0.000 (+/- 0.000)	0.000 (+/- 0.000)
LinearRegression	0.569 (+/- 0.026)	0.100 (+/- 0.006)	0.757 (+/- 0.048)	0.783 (+/- 0.014)
kNN	0.351 (+/- 0.013)	1.347 (+/- 0.071)	0.654 (+/- 0.047)	0.771 (+/- 0.007)
RandomForestRegressor	210.239 (+/- 2.603)	0.379 (+/- 0.037)	0.761 (+/- 0.047)	0.964 (+/- 0.002)
LightGBM	0.514 (+/- 0.012)	0.107 (+/- 0.005)	0.744 (+/- 0.035)	0.815 (+/- 0.007)

The Random Forest model has the highest test score, but its very high training score tells us that it's tailored too closely to our specific data, potentially lacking generalization for unseen data. Linear Regression, with a test score of 0.77 matching Random Forest's performance, is preferred for better generalization. The model's predictions closely align with actual values, with only a few outliers, as shown in the comparison.

The figure below shows the comparison of what my model predicted against what their actual values were. There are only a few outliers, which is great!



The diagram below shows what features affected that data point that led to the prediction. It says that the location played a role in increasing its popularity, but being a private room was trying to lower it as well. Maybe, being a private room comes with a price I guess.



Mistakes/Improvements

However, after finishing, I realized that I may have made a few mistakes that I could correct the next time.

1. I included features that could have influenced the predictions. For example, having a feature “number_of_reviews_ltm”, ltm- last twelve months, which is heavily related to “reviews_per_month”, might have led to getting overconfident results.
2. A feature “name” describes the rental listing, and it includes not only English text but some emojis, and foreign alphabets. Using an English-based text transformer may not have captured all kinds of information, so I may have missed some, which could have led to some models getting lower scores.
3. Dismissing time-related features may have heavily affected the results as well. They often portray patterns amongst the data, such as some rental types getting popular during holiday seasons, so, ignoring these may have caused a lower score in some models too.