COMP 4121 Major Project:

Hidden Markov Models for Real Estate Market Trend Classification

Dec 1, 2024

Grace Sosa

Course Convenor: Aleks Ignjatovic

Table of Contents

1. Abstract
Real estate market fluctuations are inevitable, where housing prices are impacted by economic, social, and political factors. Understanding changes and forecasting price changes can benefit investors, policymakers, and homebuyers. However, since real estate prices are dynamic and stochastic in nature, it can be a challenge to identify patterns and predict future trends.

1.1 Methodology
The use of Hidden Markov Models offers a unique perspective for segmenting time series data, whereas traditional classification models often struggle to capture temporal and sequential dynamics of price. These HMMs are trained using 'price_movement' as a target variable, leveraging features like the moving average, # of bedrooms, yearly quarter, and normalized price to help identify this price movement.

However, the structure of this dataset posed a challenge. Originally, I had planned to use Hidden Markov Models for state change prediction or price forecasting, but the primary limitation was that the data comprises property transactions, representing different properties sold across the years 2007-2019. In the typical use case of an HMM, like in stock market analysis or weather forecasting, data points follow a clear temporal sequence, like the price history of a single property or stock, and thus, this dataset lacks a consistent temporal progression and continuity within individual records.

My original goal was to implement a Hidden Markov Model (HMM) to predict state changes such as sudden spike increases or spike decreases as well as price forecasting, but due to data limitations, I utilized HMMs as a preprocessing step for a partitioning technique, to uncover latent states within the data through mutating a 'hidden_state' variable as a predictor and then proceeding to use other classification models classifying a given property into a 'decreasing': 0 or 'increasing': 1 class. Predicting price direction trends, based on a +1 or -1% decrease from quarter to quarter, where each class represents the expected future trend of a given property considering features like normalized, moving average value, and number of bedrooms.

The classification models I explored were Logistic Regression, Support Vector Machine (SVM), K-Nearest Neighbors, Decision Trees, and Gradient Boosting.

1.2 Libraries & Tools
For these using Python within Jupyter Notebooks for my tasks. Specifically, I used the hmmlearn library to find optimal states, as well as pandas, numpy, matplotlib, and scikit-learn for data preprocessing, model building, and testing. These are some of the Python packages I utilized:

General:

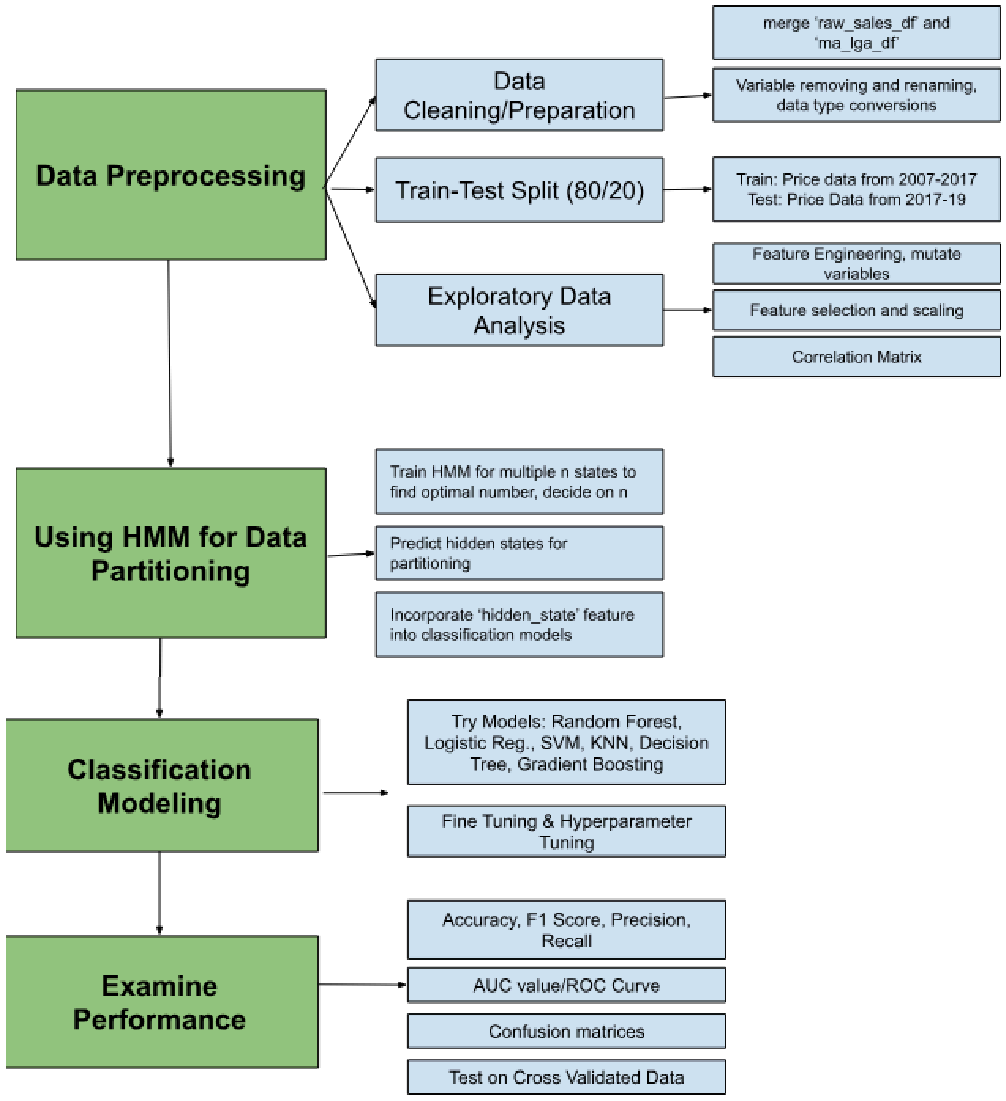
- numpy, pandas, seaborn, matplotlib

Preprocessing
- scikit-learn: StandardScaler, train_test_split
- hmmlearn: GaussianHMM

Modeling
- scikit-learn: RandomForestClassifier, GradientBoostingClassifier, LogisticRegression, SVC, KNeighborsClassifier, DecisionTreeClassifier
- scikit-learn: GridSearch

Metrics and Performance
- scikit-learn: classification_report, cross_validate, auc, roc_curve

1.3 Approach Overview


Data Preprocessing
Data Cleaning/Preparation
merge 'raw_sales_df' and 'ma_lga_df'
Variable removing and renaming, data type conversions
Train-Test Split (80/20)
Train: Price data from 2007-2017
Test: Price Data from 2017-19
Exploratory Data Analysis
Feature Engineering, mutate variables
Feature selection and scaling
Correlation Matrix
Using HMM for Data Partitioning
Train HMM for multiple n states to find optimal number, decide on n
Predict hidden states for partitioning
Incorporate 'hidden_state' feature into classification models
Classification Modeling
Try Models: Random Forest, Logistic Reg., SVM, KNN, Decision Tree, Gradient Boosting
Fine Tuning & Hyperparameter Tuning
Examine Performance
Accuracy, F1 Score, Precision, Recall
AUC value/ROC Curve
Confusion matrices
Test on Cross Validated Data

2. Concepts

2.1 Hidden Markov Models (HMM) Background

A Hidden Markov Model (HMM) is a statistical model that uses a series of observations to predict the results of an event. An HMM represents probability distributions over these sequences of observations, where this set of observable outputs is probabilistically linked to the hidden states through the emission matrix.

There are two distinctive stochastic processes in a Hidden Markov Model, a 1) hidden (latent) process, the Markov Chain itself, and a 2) observable process (emissions).

What's a Markov Chain?

A Markov Chain is a stochastic model that describes a sequence of possible events (states), where the probability of each state change depends only on the current state. In a Markov Chain, states are directly observable, unlike an HMM, where the states are hidden.

2.1.1 Components of an HMM:

1. States (S)
    a. A finite Markov chain, consisting of a set of hidden states, $S = \{s_1, s_2, \dots s_K\}$
    b. No direct access to these states, cannot observe them directly
2. Initial Probabilities (π)
    a. The initial distribution of states, $\pi = \{\pi_1, \pi_2, \dots \pi_K\}$, where π_i represents the probability that the chain starts in state s_i
3. Transition Matrix (M):
    a. Matrix $M = (p(i, j) : 1 \leq i, j \leq K)$ of size $K \times K$ of transition probabilities $p(i, j) = p(s_i \rightarrow s_j)$.
4. Observables (O)
    a. A finite set of observables $O = \{o_1, o_2, \dots o_N\}$, which are the possible manifestations of the hidden states
    b. The only data we observe
5. Emission Matrix (E)
    a. A $K \times N$ matrix of emission probabilities, $E=\{e(i,k)\}$, where $e(i, k) = P(o_k|s_i)$, where $e(i, k)$ represents the probability of observing o_k when the chain is in state s_i
6. Observations Sequence

2.1.2 Our HMM in Context

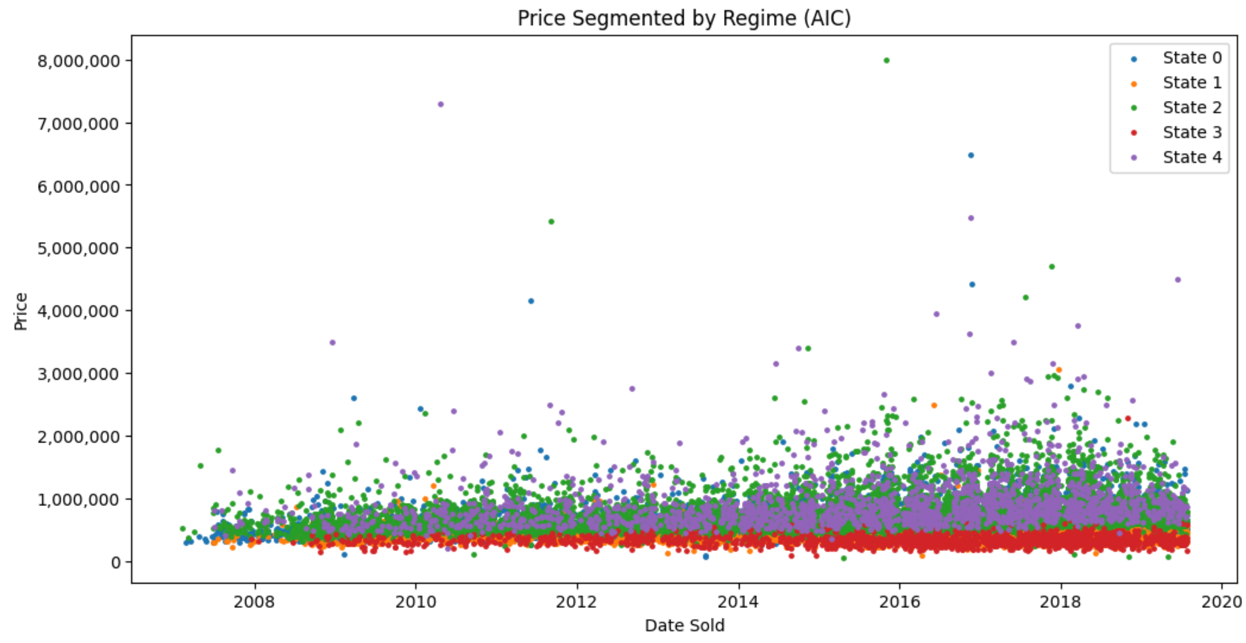
Purpose of using HMMs

1. Capturing Trends:

Here, we are using Hidden Markov Models as more of a preprocessing step to capture trends or regimes in our data, like underlying market states that are not directly observable (e.g. high volatility, steady increase, decline). The issue with using HMMs in this context of predicting state changes is that our data isn't strictly sequential, e.g. we don't have historical data for specific properties over several years but rather we have general price data per bedroom type, postcode, and other features between the years of 2017-2019. Rather, we segment or cluster our original historical price training data into trends, where a listing falls into one of the following categories, based on a quarter-to-quarter basis (E.g. 2007-02-20 is Q1):

- **Increase**: Price grows above a small threshold (e.g., +1%).
- **Decrease**: Price drops below a small threshold (e.g., -1%).

Use Case: Tracks the overall trend of the market or individual property prices (steady growth, decline, or flatline). Below is how our data was partitioned into n optimal segments:



2. Feature Enhancement

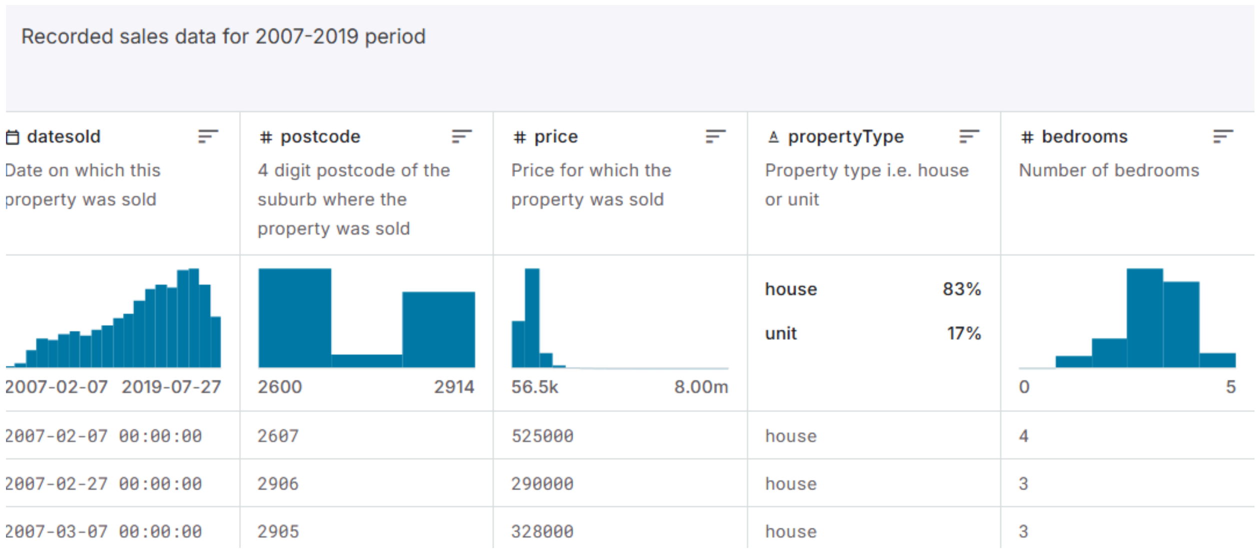
Our second goal here is feature enhancement. Since our main task here is a classification model, once we partition our data into n states, we can add this as a variable 'hidden_state' to our data frame as an additional feature for our classification task.

3. Data

3.1 Datasets & Sources

The data we are working with here is real estate data with historical price data, consisting of two original .csv files, 'raw_sales.csv' and 'ma_lga_12345.csv'. This data is obtained from Kaggle, where the data was put together by collaborators HtAG Holdings and Terry James 'raw_sales.csv' contains columns 'datesold', 'postcode', 'price', 'propertyType', and '# bedrooms'.

Below is a sample of the original **‘raw_sales.csv’**:

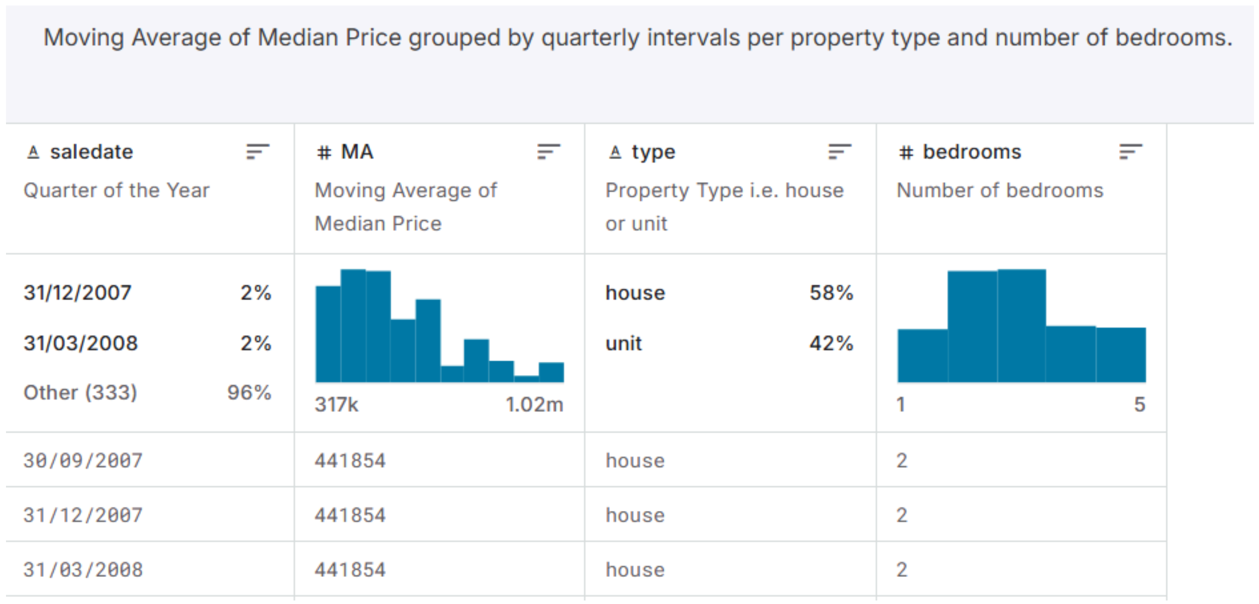


Recorded sales data for 2007-2019 period

datesold	postcode	price	propertyType	bedrooms
Date on which this property was sold	4 digit postcode of the suburb where the property was sold	Price for which the property was sold	Property type i.e. house or unit	Number of bedrooms
2007-02-07 2019-07-27	2600 2914	56.5k 8.00m	house 83% unit 17%	0 5
2007-02-07 00:00:00	2607	525000	house	4
2007-02-27 00:00:00	2906	290000	house	3
2007-03-07 00:00:00	2905	328000	house	3

‘ma_lga_12345.csv’ contains ‘saledate’, ‘MA’, the moving average of median price, ‘type’, the property type, and ‘# bedrooms’. The moving averages calculated in this .csv file can be added as an observed feature in addition to price and bedrooms in our Hidden Markov Models. Additionally, we can use the moving average to adjust the time series or detrend the data.

Below is a sample of **‘ma_lga_12345.csv’**:



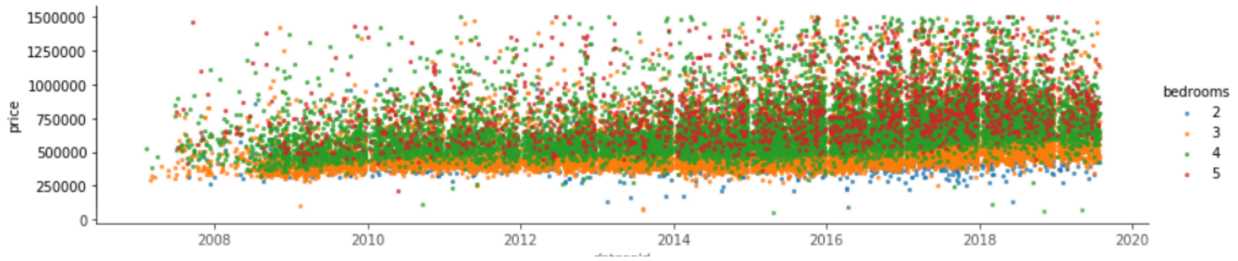
Moving Average of Median Price grouped by quarterly intervals per property type and number of bedrooms.

saledate	MA	type	bedrooms
Quarter of the Year	Moving Average of Median Price	Property Type i.e. house or unit	Number of bedrooms
31/12/2007 2% 31/03/2008 2% Other (333) 96%	317k 1.02m	house 58% unit 42%	1 5
30/09/2007	441854	house	2
31/12/2007	441854	house	2
31/03/2008	441854	house	2

3.2 Data Background

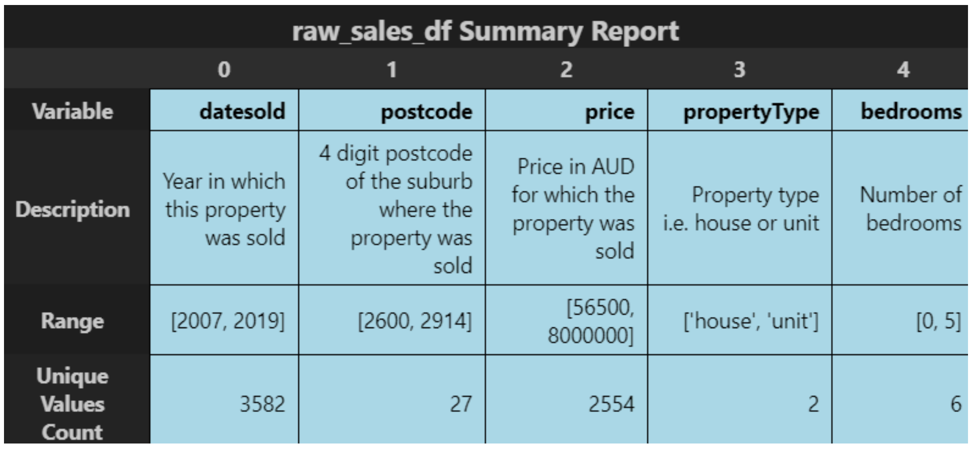
The datasets consist of historical house prices and data between 12/31/2006 and 08/12/2019. This is considered multivariate time series data, where each variable depends on past values in addition to having a dependency on other variables. The original ‘raw_sales.csv’ has 29580 entries, and the ‘ma_lga_12345.csv’ has 347 entries. Below is an overview of the raw data by year, bedroom, and price:

Raw Data:



The properties in this data have postcodes in between the ranges 2600-2914, meaning the properties are located across Canberra, Australian Capital Territory (ACT), and some adjacent areas in New South Wales (NSW).

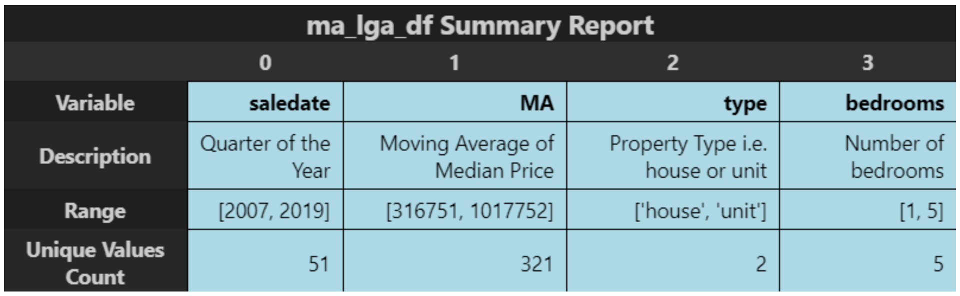
‘raw_sales_df’ Variable Summary:



raw_sales_df Summary Report

	0	1	2	3	4
Variable	**datesold**	**postcode**	**price**	**propertyType**	**bedrooms**
Description	Year in which this property was sold	4 digit postcode of the suburb where the property was sold	Price in AUD for which the property was sold	Property type i.e. house or unit	Number of bedrooms
Range	[2007, 2019]	[2600, 2914]	[56500, 8000000]	['house', 'unit']	[0, 5]
Unique Values Count	3582	27	2554	2	6

‘ma_lga_12345.csv’ Variable Summary:



ma_lga_df Summary Report

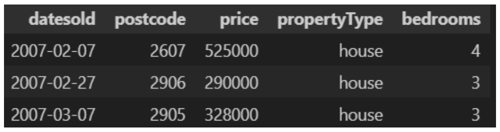
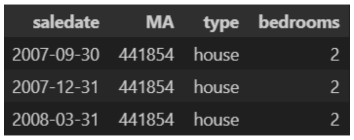
	0	1	2	3
Variable	**saledate**	**MA**	**type**	**bedrooms**
Description	Quarter of the Year	Moving Average of Median Price	Property Type i.e. house or unit	Number of bedrooms
Range	[2007, 2019]	[316751, 1017752]	['house', 'unit']	[1, 5]
Unique Values Count	51	321	2	5

3.3 Data Preprocessing and Preparation

After organizing the .csv data into a pandas data frame, a few preprocessing tasks were performed. Some preprocessing tasks I completed were converting ‘datesold’ to DateTime format, handling missing values with ‘dropna’, and encoding categorical variables like ‘propertyType’ with sci-kit-learn’s LabelEncoder. Additionally, to match a consistent format, I changed variable names to match consistency, e.g. ‘datesold’ to ‘date_sold’. Additionally, I used StandardScaler() to scale our data. After scaling, I prepared the sequential features, X, in value format, for test and train split.

raw_sales_df is a data frame obtained from ‘raw_sales.csv’.
ma_lga_df is a data frame obtained from ‘ma_lga_12345.csv’.

Before Preprocessing:

	raw_sales_df	**ma_lga_df**
Variables	‘datesold’, ‘postcode’, ‘price’, ‘propertyType’, ‘bedrooms’	‘saledate’, ‘MA’, ‘type’, ‘bedrooms’
Size	29580	347
Original Data frame		

datesold	postcode	price	propertyType	bedrooms
2007-02-07	2607	525000	house	4
2007-02-27	2906	290000	house	3
2007-03-07	2905	328000	house	3

saledate	MA	type	bedrooms
2007-09-30	441854	house	2
2007-12-31	441854	house	2
2008-03-31	441854	house	2

Changes Made:

1. Variable renaming for consistency:
    a. propertyType → property_type
    b. type → property_type
    c. datesold → date_sold
    d. selldate → date_sold
2. Created ‘merged_df’ by joining both raw_sales_df and ma_lga_df by ‘year_quarter’, which is used to split into train_df and test_df
    a. Since ma_lga_df only contains the moving average of median prices by quarter in a year and not by exact dates, I found this was the best way to join the two data frames to use MA as a feature for classification while preserving the same number of data entries
3. Variable Mutation
    a. ‘year_sold’: Obtained from ‘date_sold’
    b. ‘year_quarter’:
        i. E.g. If date_sold is 2007-02-02, ‘year_quarter’ would be 2007_Q1
    c. ‘normalized_price’:
        i. Based on the median price in a given year x and bedroom # y
            1. 'normalized_price_median' = 'price' / 'median_price'

2. E.g. The ‘price’ of a 3-bedroom house sold on 2010-05-07 is compared to the median price of 3-bedroom properties sold in 2010

ii. Let the **median price-based** normalized price represent the price relative to the specific bedroom category and year

iii. A 'normalized_price' close to 1 represents a given property that is close to a median or stable market price, greater than 1.2 represents a price that is 20% higher, etc.

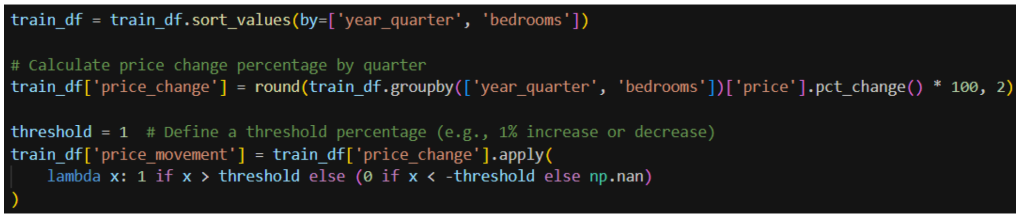
d. ‘price_change’: (current price - previous price)/ previous price * 100, which is grouped by ‘year_quarter’ and ‘bedrooms’ to capture temporal trends from quarter to quarter for each number of n bedrooms

e. **‘price_movement’**:

   i. 0: Decreasing: Price change < -1%.

   ii. 1: Increasing: Price change > 1%.

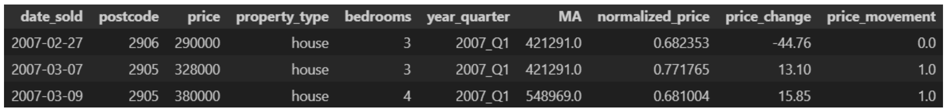
   Based on quarter-to-quarter changes calculated from ‘price_change’:



```
train_df = train_df.sort_values(by=['year_quarter', 'bedrooms'])

# Calculate price change percentage by quarter
train_df['price_change'] = round(train_df.groupby(['year_quarter', 'bedrooms'])['price'].pct_change() * 100, 2)

threshold = 1  # Define a threshold percentage (e.g., 1% increase or decrease)
train_df['price_movement'] = train_df['price_change'].apply(
    lambda x: 1 if x > threshold else (0 if x < -threshold else np.nan)
)
```

4. Dropped any listings with 0 bedrooms as there were only 30 data entries
5. Dropped any NA values, specifically where there wasn’t a match to join MA and ‘year_quarter’

After Preprocessing:
merged_df → split into train_df, test_df → split into X_train, X_test, etc. for modeling

	merged_df
Variables	‘datesold’, ‘postcode’, ‘price’, ‘property_type’, ‘bedrooms’, year_quarter’, ‘MA’, ‘normalized_price’
Size (# of Entries)	29414
Original Data Frame	

date_sold	postcode	price	property_type	bedrooms	year_quarter	MA	normalized_price	price_change	price_movement
2007-02-27	2906	290000	house	3	2007_Q1	421291.0	0.682353	-44.76	0.0
2007-03-07	2905	328000	house	3	2007_Q1	421291.0	0.771765	13.10	1.0
2007-03-09	2905	380000	house	4	2007_Q1	548969.0	0.681004	15.85	1.0

3.4 Exploratory Data Analysis (EDA)

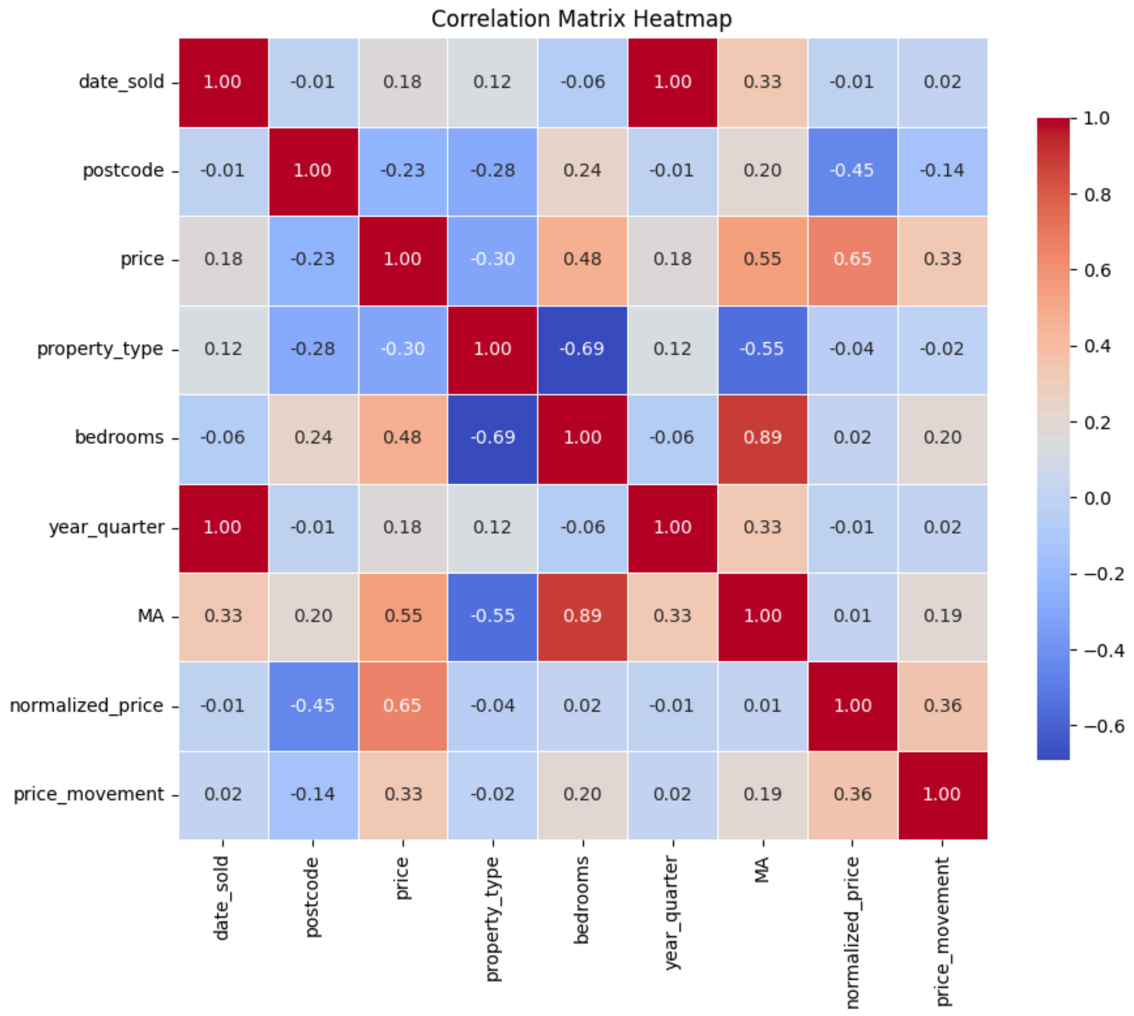
3.4.1 Feature Engineering

I did some feature engineering to see if there would be any new valuable variables to help predict price. Originally, I created ‘seasonality’ variables, where seasonality is based on whether or not a listing was within one of 10 days with the most sales per day in the year ('Busy Season'), then sorting into 'Busy season', 'Medium busy season', and 'slow' season, creating 'busy seasons' based off of the top 10 dates with the most properties sold, and categorizing otherwise. However, through my EDA, I found little to no correlation with this mutated 'seasonality' variable to price and price_movement, so thus I decided to remove it from the dataset.

Diving Deeper into Feature Engineering

Considering simple variable creation like 'month' or 'seasonality' didn't correlate to price as much, I created a new variable 'normalized_price', which takes into account the price in comparison to the number of bedrooms, as well as compares to the median price of all properties sold of that specific number of bedrooms in a specific quarter in a year.

3.4.2 Correlation Matrix

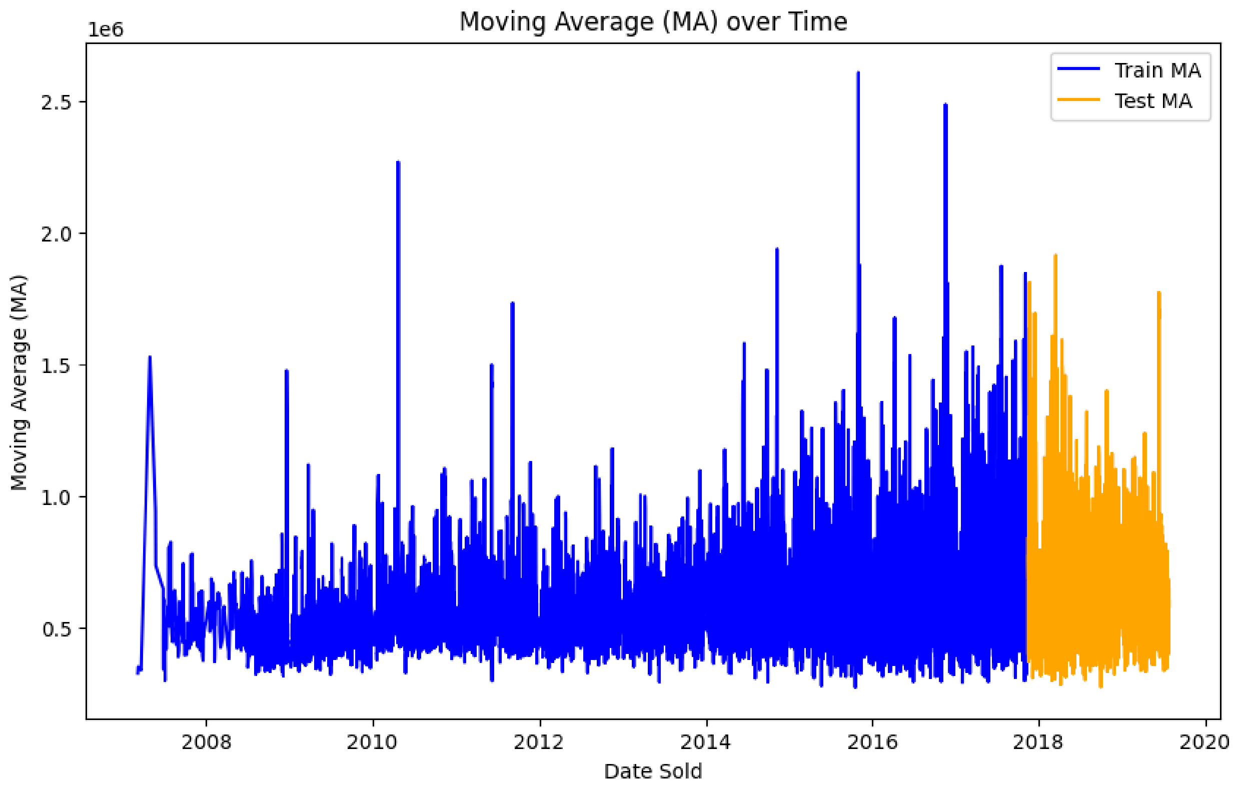


	date_sold	postcode	price	property_type	bedrooms	year_quarter	MA	normalized_price	price_movement
date_sold	1.00	-0.01	0.18	0.12	-0.06	1.00	0.33	-0.01	0.02
postcode	-0.01	1.00	-0.23	-0.28	0.24	-0.01	0.20	-0.45	-0.14
price	0.18	-0.23	1.00	-0.30	0.48	0.18	0.55	0.65	0.33
property_type	0.12	-0.28	-0.30	1.00	-0.69	0.12	-0.55	-0.04	-0.02
bedrooms	-0.06	0.24	0.48	-0.69	1.00	-0.06	0.89	0.02	0.20
year_quarter	1.00	-0.01	0.18	0.12	-0.06	1.00	0.33	-0.01	0.02
MA	0.33	0.20	0.55	-0.55	0.89	0.33	1.00	0.01	0.19
normalized_price	-0.01	-0.45	0.65	-0.04	0.02	-0.01	0.01	1.00	0.36
price_movement	0.02	-0.14	0.33	-0.02	0.20	0.02	0.19	0.36	1.00

Here, we see 'bedrooms', 'MA', and 'normalized_price' with the highest correlation between price and price movement. I would like to note that although 'MA' and 'normalized_price' are derived from price, median of price, etc., ‘price_movement’ is calculated independently based on historical price data and is not based upon ‘MA’ or ‘normalized_price’.

After preprocessing the data, I made sure to split the test and train before creating our target variable of classification, ‘price_movement’. To avoid data leakage, I calculated normalized_price and MA only for dates up to the prediction quarter and not future quarters, and thus my train and test data is split so that the train data is based on property price data from 2007-2017, with a cutoff date of 2017-11-16, and the test data is data from 2017-2019, based on an 80/20 train test split. Thus, for general use, this model can only be used for price data from 2017 and later.

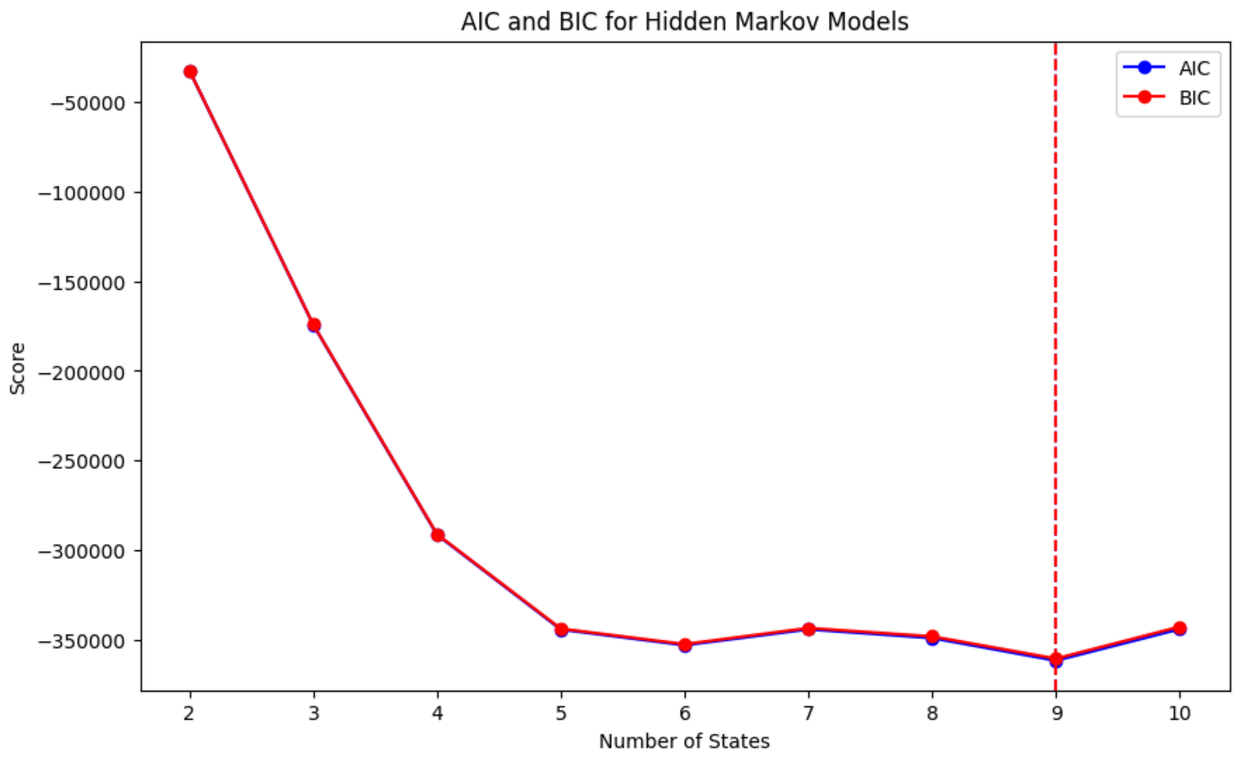
Here is a plot of the MA variable in the train and test set to verify we split the data correctly on the right dates and are not using any future data in our train set:



After examining our correlation matrix, it was noted that ‘MA’, the moving average of median price, and ‘bedrooms’, the number of bedrooms. I kept ‘normalized_price’ in the correlation matrix to examine the relationship, but as mentioned earlier, I am not using it as a feature for price prediction since it is derived from price. Thus, those two variables were used in the features when determining the optimal number of hidden states.

3.4.3 Finding the Optimal # of Hidden States with AIC and BIC

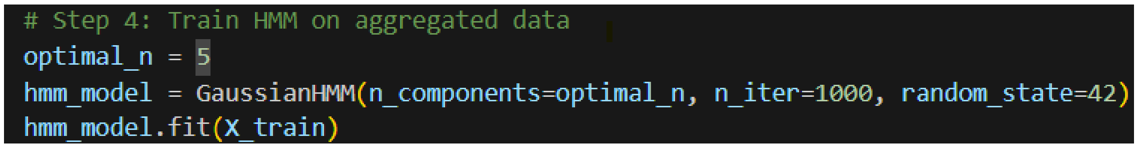
Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC) were used when iterating over n different states to find the optimal number of hidden states. The AIC and BIC scores were found to be practically identical as shown below:



Although my function calculated the optimal number of hidden states to be 9, as shown above, after plotting and using the elbow method, I decided on n = 5 optimal hidden states instead to avoid overfitting.

Next Steps After Finding n Optimal States

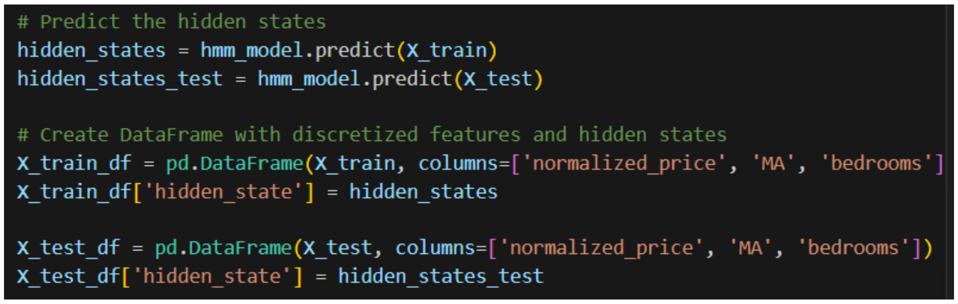
1. Train HMM with n optimal states using GaussianHMM



```
# Step 4: Train HMM on aggregated data
optimal_n = 5
hmm_model = GaussianHMM(n_components=optimal_n, n_iter=1000, random_state=42)
hmm_model.fit(X_train)
```

2. Predict hidden states for partitioning

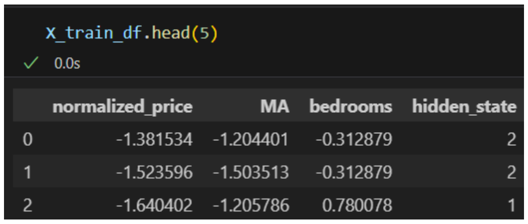
a. Add ‘hidden_state’ as a feature to our X_train_df and X_test_df



```
# Predict the hidden states
hidden_states = hmm_model.predict(X_train)
hidden_states_test = hmm_model.predict(X_test)

# Create DataFrame with discretized features and hidden states
X_train_df = pd.DataFrame(X_train, columns=['normalized_price', 'MA', 'bedrooms']
X_train_df['hidden_state'] = hidden_states

X_test_df = pd.DataFrame(X_test, columns=['normalized_price', 'MA', 'bedrooms'])
X_test_df['hidden_state'] = hidden_states_test
```

3. Incorporate ‘hidden state’ as a feature into each classification model
4. Build, improve, and fine-tune classification models
5. Measure performance

Adding ‘hidden_state’ to our X_train data and scaling our data, this is what our feature set for our train data looks like:



```
X_train_df.head(5)
✓ 0.0s
```

	normalized_price	MA	bedrooms	hidden_state
0	-1.381534	-1.204401	-0.312879	2
1	-1.523596	-1.503513	-0.312879	2
2	-1.640402	-1.205786	0.780078	1

4. Models

4.1 Model Building

I used GaussianHMM to find the optimal number of hidden states (n=5) based on AIC and BIC and partitioned accordingly, then added ‘hidden_state’ to our train_df. I tried several different classification models including Random Forest Classifier, Logistic Regression, Support Vector Machine (SVM), K-Nearest Neighbor Classifier, Decision Tree Classifier, and Gradient Boosting Classifier.

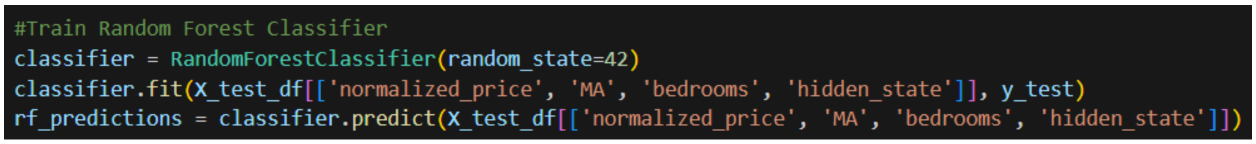
Feature Set: ['normalized_price', 'MA', 'bedrooms', 'hidden_state']

- ‘hidden_state’ = [0, 1, 2, 3, 4], where each property is assigned to 1 in 5 states

Target Variable: ‘price_movement’

- 0: Decreasing: Price change < -1%.
- 1: Increasing: Price change > 1%.

Example with RandomForestClassifier:



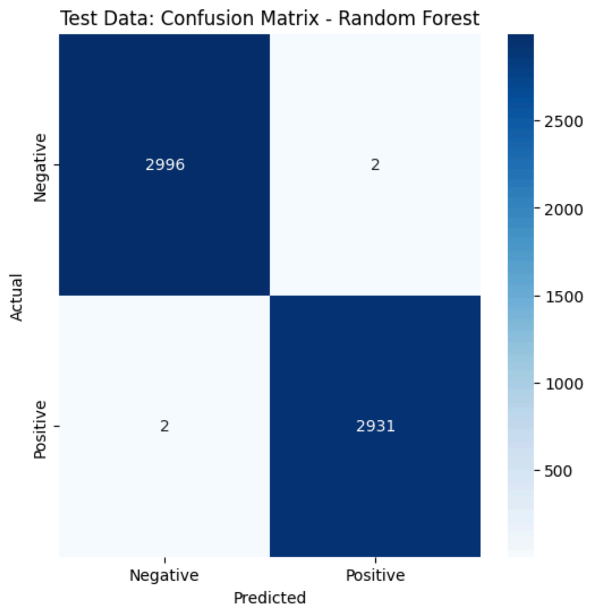
```
#Train Random Forest Classifier
classifier = RandomForestClassifier(random_state=42)
classifier.fit(X_test_df[['normalized_price', 'MA', 'bedrooms', 'hidden_state']], y_test)
rf_predictions = classifier.predict(X_test_df[['normalized_price', 'MA', 'bedrooms', 'hidden_state']])
```

In looking at the results, it was consistent that the Random Forest, K-Nearest Neighbor, Decision Tree, and Gradient Boosting classifiers performed the best, looking at accuracy, precision, recall, and AUC, while the Logistic Regression and SVM models were weaker with both AUC and Accuracy < 0.7. Both the Random Forest and Decision Tree models had an Accuracy = 1.0 and AUC = 1.0 on the original test data. *However, since that metric isn't representative of unseen data, I ran K-Fold Cross Validation and had the following results where the Random Forest and Gradient Boosting models performed the best on the cross-validated test data:

4.1.1 Model Performance on Original Test Data

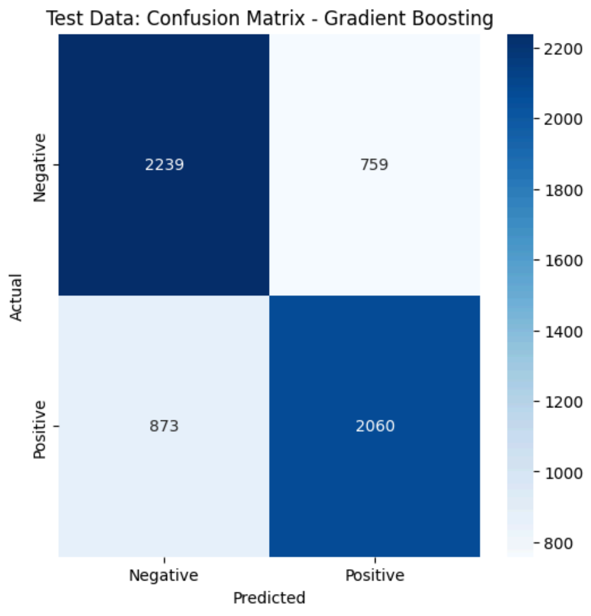
Test Data Performance Results (For the Two Best Models)

Random Forest



	Original Test Data	Cross Validated Test Data (k = 5)
Accuracy	*1.00	0.65
AUC	*1.00	0.718

Gradient Boosting



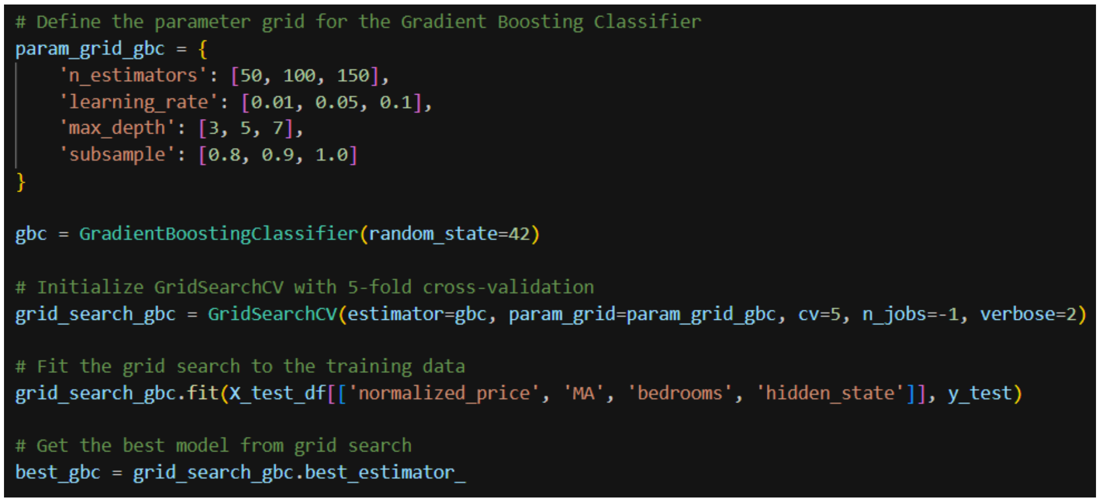
	Original Test Data	Cross Validated Test Data (k = 5)
Accuracy	0.72	0.70
AUC	0.80	0.768

4.1.2 Model Performance on Cross Validated Test Data

We look at which model does the best on the cross-validated test data. Although the Random Forest and Decision Trees classifiers do exceptionally well on the original test data, it doesn't generalize as well for unseen data, whereas Gradient Boosting performed the best on cross-validated data, and further metrics as I will explain further.

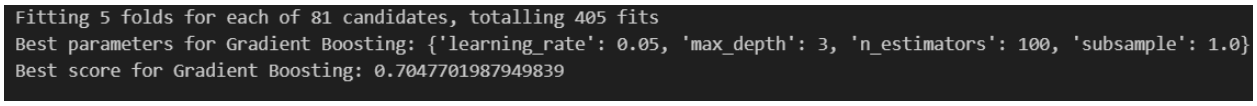
4.2 Model Fine Tuning

After seeing that the Gradient Boosting Classifier performed the best on cross-validated test data, I performed a grid search using GridSearchCV, to find the best parameters, testing across a variety of n_estimators, learning rates, and depths.



```python
# Define the parameter grid for the Gradient Boosting Classifier
param_grid_gbc = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 0.9, 1.0]
}

gbc = GradientBoostingClassifier(random_state=42)

# Initialize GridSearchCV with 5-fold cross-validation
grid_search_gbc = GridSearchCV(estimator=gbc, param_grid=param_grid_gbc, cv=5, n_jobs=-1, verbose=2)

# Fit the grid search to the training data
grid_search_gbc.fit(X_test_df[['normalized_price', 'MA', 'bedrooms', 'hidden_state']], y_test)

# Get the best model from grid search
best_gbc = grid_search_gbc.best_estimator_
```

Gradient Boosting Optimal Parameters:
learning rate = 0.05
max_depth: 3
n_estimators: 100
subsample: 1.0



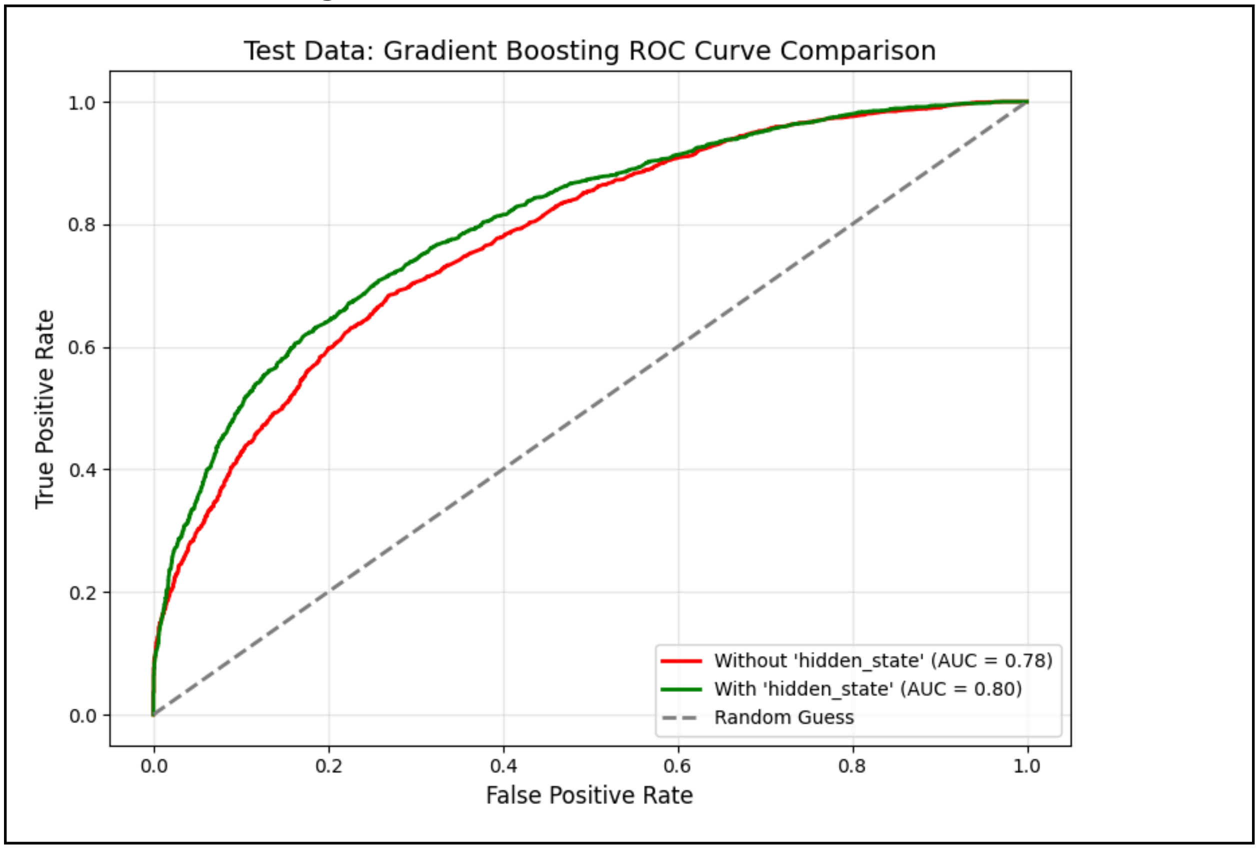
```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best parameters for Gradient Boosting: {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 100, 'subsample': 1.0}
Best score for Gradient Boosting: 0.7047701987949839
```

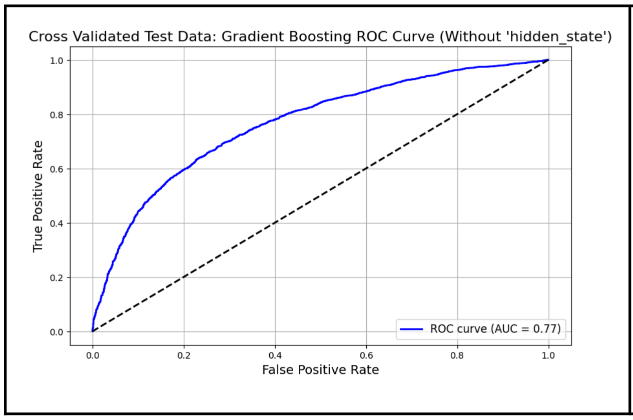
Additionally, I tested different models with different combinations of features, where I removed features like 'bedrooms', tested with and without 'hidden_state' and tested different combinations of features. However, I found the performance to be quite similar so I kept the original feature set of ['normalized_price', 'MA', 'bedrooms', 'hidden_state'].

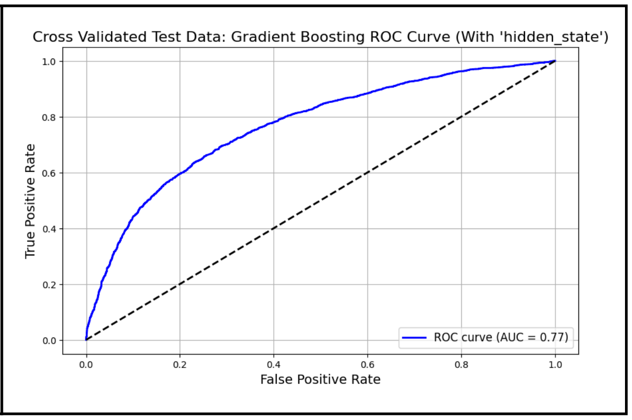
One major question to answer was whether or not the 'hidden_state' variable, produced from our HMM partitioning, provided meaningful predictive strength. As shown in the ROC curves below, when testing on the original test data, the Gradient Boosting Classifier that did not include 'hidden_state' as a feature had an AUC = 0.78, while the model that did include 'hidden_state' had an AUC = 0.80. We see a slight improvement in AUC, and in some metrics like precision and recall on 0, and a very slight improvement in accuracy. These comparisons are shown in Figure 4.2.4.

To see if this would do well with unseen data, I tested the same model on 5-fold cross-validated test data, and the AUC was nearly identical, with an AUC around 0.77 for both models with and without ‘hidden_state’. The Hidden Markov Models applied to mutate the variable ‘hidden_state’ in this project did not seem to enhance predictive power in the classification models tested. However, it does not diminish the potential utility of hidden states in other contexts with alternative methodologies or classification models. For example, Recurrent Neural Networks (RNNs) could likely better exploit the temporal and sequence-related information encoded within the hidden states we derived from the HMM, which could be a further expansion. Additionally, my train data was only based on price trends and changes between quarters from 2007-2017, whereas my test data consisted only of properties sold from 2017-2019. If we can test with additional validation data, from 2019-2024, and examine performance from there, we can get a better idea of whether or not our model with ‘hidden_state’ generalizes well and provides predictive strength.

4.2.1. Gradient Boosting: ROC Curves







4.3 Results of Best Model: Gradient Boosting

Gradient Boosting Overall Performance				
	Original Test Data (Without 'hidden_state')	Original Test Data (With 'hidden_state')	Cross Validated Test Data (k = 5)	Cross Validated Test Data with GridSearch (k=5)
Accuracy	0.71	0.72	0.70	0.7048
AUC	0.78	0.80	0.7679	0.7694

5. Metrics

Although I chose to focus on accuracy and AUC to choose the best classification model, it is important to consider other metrics like precision, recall, F1 score, and logarithmic loss.
For example, precision is another metric we can focus on. For example, if we are building this model for buyers, focusing on precision on 0 is more critical, as predicting a decrease (0) when there is an increase (1) can lead to missing the best selling opportunities and financial loss for sellers.

Precision on 0:

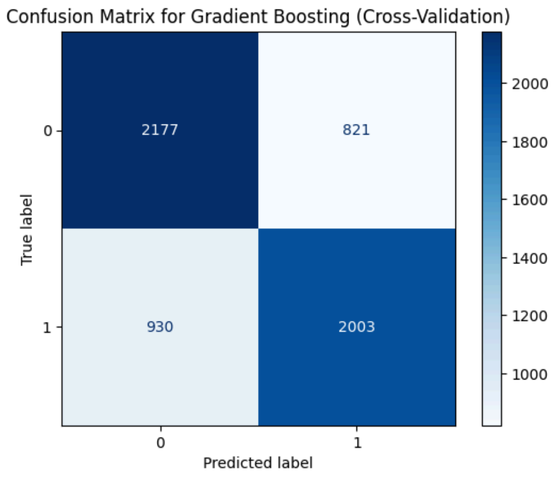
- Better for buyers, worse for Sellers
- Predicting a decrease when there's an increase (false negative) can lead to financial loss for sellers, savings for potential buyers

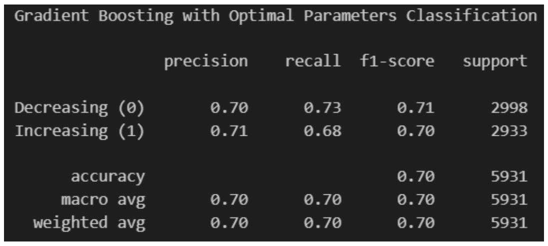
Precision on 1:

- Better for sellers, worse for buyers
- Predicting an increase when there's a decrease (false positive) can lead to missing the best selling opportunities, or a buyer spending more than they could have if they waited

The most valuable metrics also depend on the kind of data we are working with. For example, if we have unbalanced data, where property price trends are mostly characterized by decreases, we should look into potential outliers, and in this scenario, using accuracy may not be representative of the model's performance. To optimize metrics overall, we can fine-tune to the specific metric we want to focus on, or modify our threshold to optimize overall model performance.

Overall, looking at the confusion matrix and classification report of the best model (Gradient Boosting) below, precision, recall, and f1-score overall seem to be in a relevantly reasonable range, which is another thing to consider when picking a model.





```
Gradient Boosting with Optimal Parameters Classification

                precision    recall  f1-score   support

Decreasing (0)       0.70      0.73      0.71      2998
Increasing (1)       0.71      0.68      0.70      2933

      accuracy                           0.70      5931
     macro avg       0.70      0.70      0.70      5931
  weighted avg       0.70      0.70      0.70      5931
```

6. Future Improvements

Hybrid Approach: HMMs and ARIMA for Market Dynamics

My initial idea was to implement a hybrid model of Hidden Markov Models (HMM) and AutoRegressive Integrated Moving Average (ARIMA) models, aimed to improve the accuracy and interpretability of real estate market analysis and, more specifically, predict price. Hidden Markov Models are proven to be useful in contexts like stock market analysis and speech and language processing, and thus can be further applied in the real estate market.

HMM for State Detection: HMMs can be used to identify latent or 'hidden' market states– like bullish, bearish, or stable– based on historical trends and price patterns. The use of HMMs can help to reveal state changes or underlying market conditions, such as when a property is expected to fluctuate in price, but alone, HMMs are more for state prediction rather than price forecasting.

ARIMA for Price Forecasting: Thus, once we find the optimal state, we can use the HMM to forecast probabilities of transitioning between states, and then use ARIMA models weighted by the HMM state probabilities to predict future prices.

However, as mentioned earlier, the issue I had was the data was not consistent with a single property's historical price time series data, but rather was a collection of prices of several unique properties sold from 2007-2019 and thus was insufficient for price forecasting. Thus, if I have sufficient data to execute this hybrid approach, it may be an improved method to unveil market trends.

Multi-Class Classification Problem

The project I conducted was a simple binary classification problem, predicting whether a property is expected to increase (>= 1%) or decrease (<= -1%) in the next quarter. Introducing more detailed classes for a multi-class classification problem could provide more insight into more specific price trends over a variety of time periods. The potential proposed classes are listed below:

Original: Binary Classification

- 0: Decreasing (price change <= -1%) from quarter-to-quarter
- 1: Increasing (price change >= 1%) from quarter-to-quarter

Proposed: Multi-Class Classification

- 0: Rapid Decrease: (price change <= -3% or < 2σ within a relatively short time period t, e.g. a month)
- 1: Steady Decrease: (-1% <= price change <= -0.5% over several months m)
- 2: Rapid Increase
- 3: Steady Increase
- 4: Positive Spike: (weekly price change >= 5% or a rapid increase followed by a rapid decrease)
- 5: Negative Spike

Recurrent Neural Networks (RNNs)

As mentioned earlier, RNNs, including architectures like Long Short-Term Memory (LSTM) networks, have the potential to work exceptionally well in sequential and temporal data. If the data contains information on individual property histories or aggregated market data, RNNs can be trained on that data to capture temporal dependencies. Using RNNs alongside HMM state outputs can help refine trend predictions, where we can gain sequential insight from RNNs and state-dependent probabilities from HMMs.

XGBoost

XGBoost, as an advanced gradient boosting technique, could enhance the predictive utility of hidden states, as it is known for its ability to model complex interactions between features, which can help to reveal subtle, nonlinear patterns. Gradient boosting algorithms iteratively refine weak learners, which can help leverage hidden states to improve classification performance.

Expand the Data

In this context, we are essentially only given four variables: date sold, postcode, bedrooms, and property type other than price. Finding data that provides more information such as square feet, neighborhood, or historical price data from a unique property can help to strengthen and build a more robust model. If we can join additional datasets with our current data, we can provide more comprehensive training data to build our models on. Specifically, if we can find data that also has dates past 2019, predictive power can be strengthened for properties being sold today.

7. Conclusion

Although this just starts to delve into basic modeling for real estate market analysis, Hidden Markov Models have a high potential in this field for predictive analysis, and further expansions such as adding additional classes like rapid increase, steady increase, or positive spike for a multi-class classification problem can further explore the potential of HMMs in sequential and temporal data.

Context-Dependent Utility of HMMs

While the current analysis showed limited predictive power for hidden states, with only minimal improvements in accuracy and AUC, this could be in part due to the data limitations of my dataset. The value of HMMs might emerge more in domains where sequential dependencies are more prominent in the data, such as in dynamic markets with cyclical trends, or predicting state changes in pulse oximetry signal data.

Classification Performance

After extensive data preprocessing, feature engineering, model selection and experimentation, fine-tuning, testing on cross-validated data, and the experimentation of a variety of feature sets, it was evident that our classification models were able to categorize the majority of properties into the correct price movement trend. Although 'hidden_state' generated from HMM as a feature for classification was not a strong predictor, I still found my models provided relatively reasonable classification results, where the strongest model, Gradient Boosting Classifier, had an AUC = 0.77 on cross-validated test data, and an overall accuracy = 0.70. Of course, these metrics can be improved, but with limited variables in our dataset, I looked for the classification model that performed most consistently, showing slight improvements following GridSearch optimal parameters, and maintaining reasonable AUC and accuracy in both the original test data and cross-validated test data.

7.1 Overall Summary

This project offered valuable insights into the potential of Hidden Markov Models (HMMs) for capturing latent market states and the application of classification models in real estate price movement prediction. Although my approach of using HMMs in this context may not align with their typical conventional applications, such as speech recognition or stock market analysis, I was able to gain a deeper understanding of the potential Hidden Markov Models have to uncover hidden patterns and transitions in market behavior, specifically the higher potential of HMMs for more sequentially structured datasets. Additionally, I gained insight into strategic selection of classification models in specific contexts, where Random Forest Classifiers and Gradient

Boosting Classifiers hold strong here, and learned about the future potential of implementing Recurrent Neural Networks (RNN) and XGBoost in conjunction with HMMs.

Overall, the classification models provided reasonable results in AUC, Accuracy, and Precision with the Gradient Boosting Classifier, and a slight improvement in AUC with the incorporation of a hidden state feature. However, this project reinforced the idea that model performance is intricately tied to the nature of the data, the quality of preprocessing, and the alignment of methods to the prediction task at hand, and this project has offered a strong foundation for developing more complex models in this domain of time-series data.

References

Adegirmenci, Ugur. *Hidden Markov Models: Theory and Applications*. 2014, https://scholar.harvard.edu/files/adegirmenci/files/hmm_adegirmenci_2014.pdf.

"How to Tune Hyperparameters in Gradient Boosting Algorithm." *GeeksforGeeks*, https://www.geeksforgeeks.org/how-to-tune-hyperparameters-in-gradient-boosting-algorithm/.

HTAG Holdings, James, Terry. *Property Sales Dataset*. Kaggle, https://www.kaggle.com/datasets/htagholdings/property-sales?select=raw_sales.csv.

Ignjatovic, Aleks. *Viterbi Algorithm*. 2019, https://cgi.cse.unsw.edu.au/~cs4121/lectures_2019/viterbi_slides_short.pdf.

Ignjatovic, Aleks. *PageRank Algorithm*. 2019, https://cgi.cse.unsw.edu.au/~cs4121/lectures_2019/pagerank_slides_short.pdf.

Patel, Nikhil Kumar. *Hidden Markov Model Implementation*. GitHub, https://github.com/Nikhil-Kumar-Patel/Hidden-Makov-Model.

"Precision and Recall Metrics: Definitions and Examples." *Built In*, https://builtin.com/data-science/precision-and-recall#:~:text=Recall%20and%20Precision%20Metrics,precision%20using%20the%20harmonic%20mean.