Predicting Madrid Housing Prices Using Machine Learning Techniques

Antonio Imbesi
College of Arts and Sciences
Georgia State University
Atlanta, Georgia, United States
aimbesi1@student.gsu.edu

Mia Ko
College of Arts and Sciences
Georgia State University
Atlanta, Georgia, United States
bko3@student.gsu.edu

Micah Robins
College of Arts and Sciences
Georgia State University
Atlanta, Georgia, United States
mrobins1@student.gsu.edu

***Abstract*—This project had us use various classification and regression machine learning algorithms to predict housing prices in Madrid, Spain. A variety of techniques were used to render the data suitable for training and testing by all the models used.**

***Index Terms*—machine learning, regression, classification, housing**

I. Introduction (Project Idea)

As recent fluctuations in real estate prices have attracted much attention, efficient ways to analyze the real estate market to predict housing prices and inform investment decisions is in increasingly high demand. In this project, we aim to develop and test various machine learning models to predict housing prices based on various features such as location, square footage, and amenities. While this is traditionally framed as a regression problem, several classification models will be used to make predictions on a discretized version of the housing dataset. Algorithms discussed in this proposal include linear regression, support vector machines, gradient boosting, and ridge regression.

II. Survey of Related Work

Linear regression is a popular algorithm used in predicting housing prices. In their efforts to apply regression methods to predict housing prices in Islamabad, Pakistan, [1] admits that linear regression is "used too much" due to its straightforwardness and ease of use, but they raised several methods as viable alternatives.

A. Additional Regression Methods

- Support Vector Regression (SVR): A variant of Support Vector Machines built for regression rather than classification while keeping familiar concepts like decision boundaries and support vectors.
- Bayesian Ridge Regression (BRR): A probabilistic model that works well with Gaussian distributions. Reference [1] states that it is adaptable and able to be fine-tuned through "regularization parameters."
- LassoLars: This is a variant on the Lasso regression model, another regularized version of linear regression. Short for least absolute shrinkage and selection operator fitted for least-angle regression, this model can "reduce model complexity and prevent over-fitting" [1]. Its dependency on the data for variable selection was alleviated in [1] by combining its penalties with those of BRR.

Reference [1] also mentions other more involved methods for predicting the housing market, including neural networks and data mining, but that is beyond the scope of this project. These studies have shown promising results in predicting housing prices using regression.

In a similar project to predict housing prices in Turkey, [2] acknowledges the occasional need for kernels to allow data to be linearly separable in a high-dimensional space. They used support vector regression as well as decision tree regression to predict their data. They also believe variables such as the square footage and presence of amenities do not affect a house's price as strongly as macroeconomic factors like construction costs, demographic, and unemployment rate [2].

[3] noted the existence of a nonlinear relationship between housing prices in Shanghai, China, and air pollution and attempted to identify explanatory variables using various regression methods, including ordinary least squares (OLS) linear regression and generalized weighted regression (GWR). They determined that the data was best predicted using a gradient boosting decision tree (GBDT) model and concluded that air pollution has a significant negative impact on housing prices. Their GBDT model appears to be a regression model.

III. Data Source

We used a dataset of real estate listings from Madrid, Spain [3]. This data was collected by scraping various real estate websites that offered housing in Madrid. The dataset has 21,742 samples and 58 features, containing information such as built and useful square footage, number of rooms and bathrooms, address, construction year, and presence of amenities like pools or balconies. Several types of housing are represented here, including apartments, duplexes, and penthouses. The target value to predict is the buy price.

A simple histogram plot of the target value, as shown in Fig. 1, shows that the distribution is skewed to the right. The data does not appear to be normally distributed.

Upon inspection of the data, several problems with the dataset arose:

TABLE I
SOME FEATURES FROM THE MADRID HOUSING DATASET[a]

Feature name	Description
ID	Identifier
title	Title from listing
subtitle	Neighborhood and city
sq_mt_built	Square meters built
sq_mt_useful	Square meters useful
n_rooms	Number of rooms
n_bathrooms	Number of bathrooms
sq_mt_allotment	Square meter allotment
latitude, longitude	Latitude, Longitude
raw_address	Address
is_exact_address_hidden	Boolean values
price	Target Value

[a]Table modified from [4]. Not all 57 features are shown here.

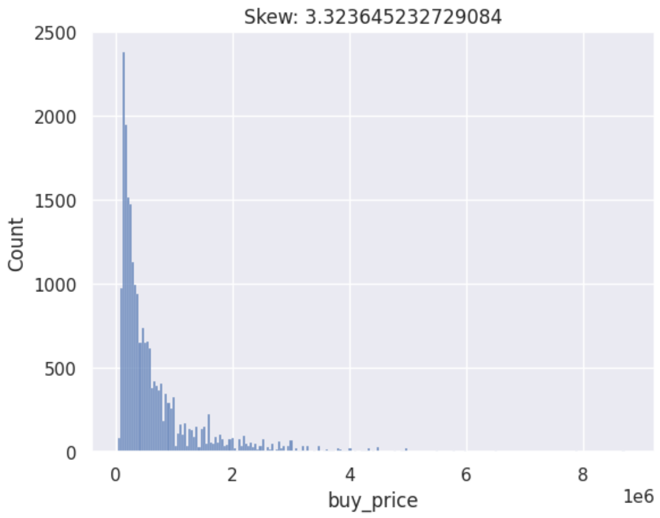

Fig. 1. Distribution of the buy_price target value.

- Many of the columns are filled with null or missing values. Some columns consisted entirely of null values, while others contained either null or TRUE values.
- Some features correlated too strongly with other features or the target value. Namely, “rent_price” could very easily predict “buy_price.” “parking_price” is consistently null when “has_parking” is false and 0.0 when “has_parking” is true.
- For some reason, some of the samples had more floors than rooms, i.e. several of them had four floors but only three rooms. This suggested that some of the data was collected erroneously.

To clean and preprocess the dataset simply, we performed the following operations on the original dataset in sequence:

1) Remove the following features due to them being empty, nearly empty, irrelevant, or too strongly correlated: ”id”, ”title”, ”subtitle”, ”operation”, ”latitude”, ”longitude”, ”portal”, ”floor”, ”door”, ”rent_price_by_area”, ”are_pets_allowed”, ”is_furnished”, ”is_kitchen_equipped”, ”has_public_parking”, ”has_private_parking”, ”sq_mt_useful”, ”sq_mt_allotment”, ”raw_address”, ”is_exact_address_hidden”, ”street_name”, ”street_number”, ”is_rent_price_known”, ”is_buy_price_known”, ”is_renewal_needed”, ”is_parking_included_in_price”, ”parking_price”, ”rent_price”, ”buy_price_by_area”, ”is_orientation_west”, ”is_orientation_east”, ”is_orientation_south”, ”is_orientation_north”, ”is_floor_under”, ”is_exterior”, ”has_lift”.
2) Delete all the samples that met the following conditions:
   a) “house_type_id,” “built_year,” “has_central_heating,” “is_new_development,” or “sq_mt_built” is null.
   b) “n_rooms” is 0, implying the home has no rooms.
   c) “n_rooms” is less than “n_floors,” implying the home has more floors than rooms.
3) Delete all the samples that met the following conditions:
   a) “n_floors”: 1
   b) ”has_ac,” “has_garden,” “has_pool,” “has_terrace,” “has_balcony,” “has_storage_room,” “is_accessible,” “has_green_zones” : False
   c) “n_bathrooms”: Median of n_bathrooms

In step 3, “n_floors” was filled in as 1 to assume that most of the houses, apartments, and penthouses in the dataset will only have one floor. The Boolean features are all filled in with False because the fact that True was the only non-null value in all of them suggested that the dataset’s author meant the null values to be interpreted as false.

Additionally, the “neighborhood_id” feature holds strings containing both a neighborhood ID and a district ID for each sample. To convert the neighborhood and district information into categorical data for the models, we split “neighborhood_id” into several dummy variables each representing a different possible district ID. The “energy_certificate” and “house_type_id” features also have categorical data represented by a set of strings, so they were split into more sets of dummy variables. This process of one-hot encoding the district IDs, energy certificate IDs, and house types increased the total number of features to 53. We chose not to create categorical features for neighborhood IDs to keep the total number of features under 100 and reduce the computational intensity of the training and testing sessions.

Note that additional preprocessing steps were performed separately on the training and testing splits of the dataset to avoid influencing the training set with testing data and vice versa. These include applying step 3c), which involves calculating a median, and applying a standard scaler to the non-categorical features. These will be detailed in the section on implementation.

The final preprocessed dataset had 7,585 samples and 53 features.

IV. KEY ALGORITHMS AND TECHNOLOGY

The project was made using a Google Colab notebook supported mainly by the Matplotlib, Pandas, and scikit-learn libraries. Specifically, all models and scaling techniques used were imported from the sci-kit learn library.

Twelve machine learning models were chosen to predict the housing prices given the data provided. These included both classification and regression models, and each one benefited from different aspects of the mix of continuous and categorical data to learn from. During initial tests of the models, the cross-validation step would fail due to the classification models' incompatibility with the continuous target value. To accommodate the classifiers, we introduce a second set yd of target values derived from the original set. yd is a partitioning of y into four discrete classes based on the four quantile ranges of y. Both the training and testing splits of yd are calculated based on the quantile ranges of the full y set so that the definition of the four classes does not change when yd is split in different ways.

Additionally, since the classification and regression models have different metrics that are relevant to them, we separately collected data on accuracy, precision, recall, and F1-score for classifiers, and MAE, RMSE, and R^2 score for regressors.

In the following descriptions of each model, the term in parentheses is the key used to identify that model in the following figures and tables.

A. Decision Tree (dt)

This classification model works by computing the probability that a given sample is of a particular class based on the values of all the other features. It creates decision rules based on which features give it the most information gain with respect to the target value, and it represents those rules as additional nodes on the tree. It continues like this until every "case" is accounted for. Predicting new data with a decision tree involves traversing down the tree based on the decision each node represents; the decisions get more specific as the tree gets deeper until a prediction is reached at the end of the branch. It can handle continuous data by splitting it into numerical ranges that yield the most information gain. However, computing the minimal decision tree is NP-hard, and the tree tends to overfit as it gets more complex. Because it tries to account for every case in the training set, it can quickly become too complex if it encounters too many statistical outliers.

B. Perceptron

This is a simple classification model that works by dividing the data into regions based on their class using linear separators. If a class is mislabeled by the current iteration of the separator, the separator is adjusted to accommodate the mistake. If not, then the classifier had converged successfully. To prevent it from overfitting or getting stuck trying to converge on data that is not linearly separable, the perceptron can also stop adjusting the separators after a certain number of iterations.

C. Naïve Bayes (nb)

Given a class label, this classification model assumes that all features are independent of each other. It then computes the conditional probability of each value of X given the class label y and makes a prediction based on the method of maximum likelihood. The implementation used in scikit-learn assumes a Gaussian distribution of the features:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right) \tag{1}$$

where σ_y and μ_y are the standard deviation and variance of the label y. This implementation also allows it to predict from continuous features. While its assumption is almost never true, its good accuracy and computational efficiency make it popular for initial predictions of data and comparison with more complex models.

D. Logistic Regression (log_reg)

This model classifies data by first computing $P(y|X)$ given a vector X of features, then converts that probability to a discrete value using a sigmoid function or similar method. It uses gradient descent to optimize its weight vector θ to minimize the cost it incurs due to its logarithmic cost function. To classify new data, it computes $h_\theta(x) = \theta^T x$ and discretizes the result using the activation function. To configure it to work with multi-class data, we initialized this model with the "liblinear" solver and the one-versus-rest multi-class setting. This makes it fit a binary classification for each label, determining if samples are either one class or all other classes.

E. Support Vector Machines (svm)

These classifiers, or SVCs, attempt to separate the data with a maximum margin hyperplane, or a hyperplane such that it is as far away as possible from any other samples in the feature space. The samples closest to the hyperplane are known as the support vectors. Two SVCs will be used in this project, each employing a different kernel to help separate the data as part of their kernel trick. One uses a linear kernel that simply computes the dot product of the feature data, while the other uses the radial basis function (RBF) kernel. The RBF kernel adds dimensionality to the feature space by representing each data point as a vector of distances from other data points or centers.

F. Gradient Boosting (gb)

Boosting algorithms work by combining the results of many weak learners to generate stronger learners based on the mistakes of their predecessors. For the gradient boosting classifier, the weak learners are regression trees chosen to make decisions that minimize the loss function. As is true with individual decision trees, smaller trees are preferred to help with generalization. The gradient boosting algorithm uses gradient descent to iteratively add trees and gradually approach a satisfactory accuracy score at a rate determined by the learning rate hyperparameter. One of the issues with gradient boosting is that it has a tendency to ascribe higher importance to numerically larger values. This can be mitigated using feature scaling. For this project, the histogram gradient boosting classifier available in scikit-learn was used for its faster performance on large datasets. It was initialized with the default parameters, so its learning rate was 0.1, the maximum number of leaf nodes was 31, and it did not have any regularization applied to it.

G. Multi-Layer Perceptron (mlp)

This is a neural network comprised of several layers of perceptrons. Each perceptron acts as a neuron that takes several weighted inputs and returns an output through an activation function. The perceptrons receiving the input directly feed that input into one or more hidden layers of perceptrons that continue to process the inputs until they reach the output layer, which then outputs the final prediction. The MLP classifier used for this project has been set up to do 100 maximum iterations with the 'adam' solver and using early stopping to stop training prematurely when the validation score is not improving. The 'adam' solver is an optimizer based on stochastic gradient descent, which involves having the network work through one data sample at a time and comparing its output to the expected output. It then propagates the calculated error from the output back through the network to reweight the neurons based on how much they contributed to the error.

H. Linear Regression (lin_reg

This is a simple regression model that tries to fit a line or hyperplane through a set of data points. Given a feature set x predicting another value y, the model tries to find the weight vector w_1 and y-intercept w_0 in the linear equation $y = w_0 + w_1 x$. To work effectively, it assumes that the target value y has a linear relationship with the input data X; y follows a normal distribution and has the same variance at each value of X; and all observations are independent. It optimizes w by minimizing a loss function defined on the training data, so it risks not being able to generalize well on the testing set. This can be remedied by adding a penalty term to the model's loss function, making it harder to optimize. The version of this model in scikit-learn uses a loss function that calculates the residual sum of squares between the features in the dataset:

$$L(w) = \sum_{n=1}^{N} (y_n - w^T x_n)^2 \tag{2}$$

This particular loss function has an easy solution to determine the optimal value of w:

$$w = (X^T X)^{-1} X^T y \tag{3}$$

I. Ridge

Additional regularization functions can be added onto the linear regression model's loss function to reduce overfitting. By adding the squared Euclidean norm of w to $L(w)$, we get the loss function for Ridge regression:

$$L_{reg}(w) = \sum_{n=1}^{N} (y_n - w^T x_n)^2 + \lambda w^T w \tag{4}$$

Increasing the value of λ increases the strength of the regularization, reducing the variance of the coefficients w. This technique is also known as $l2$ regularization.

J. Lasso

This is an alternative method of regularizing a linear regression model that uses $l1$ regularization instead. Its loss function equation is:

$$L_{reg}(w) = \sum_{n=1}^{N} (y_n - w^T x_n)^2 + \lambda |w| \tag{5}$$

This has the effect of penalizing the coefficients of less important features more strongly, eventually bringing their weights to zero and eliminating them from the rest of the regression process. It is particularly effective on datasets with small numbers of features and correlated data.

K. Elastic Net (en)

This regression model combines the feature-selecting capability of Lasso with the variance reduction of Ridge by using both $l1$ and $l2$ regularization in its loss function:

$$L_{reg}(w) = \sum_{n=1}^{N} (y_n - w^T x_n)^2 + \lambda \rho |w| + \frac{\lambda(1-\rho)}{2} w^T w \tag{6}$$

where ρ is the ratio between the $l1$ and $l2$ terms. One particular advantage of Elastic Net is its ability to choose multiple features that are correlated with each other; Lasso will usually just pick one and eliminate the other.

V. Implementation

To provide replicable results, all models and functions were initialized or called with a random state of 1234 where applicable.

All models were evaluated on their performance on three different distributions of the dataset, which reserved 50%, 70%, and 80% of the data respectively for training.

While the collection of algorithms here include both classifiers and regressors, the target value is a continuous value that is only compatible with regression. To accommodate the classifiers, we introduce a second set y_d of target values derived from the original set. y_d is a partitioning of y into four discrete classes based on the four quantile ranges of y. Both the training and testing splits of y_d are calculated based on the quantile ranges of the full y set so that the definition of the four classes does not change when yd is split in different ways.

The models were trained and tested in two separate batches: classification on y_d and regression on y. During training, each model was cross-validated on ten folds of the training set. The cross-validation process yielded ten instances of the model trained on each of the ten folds. These instances were recorded along with their respective metrics: accuracy, precision, recall, and F1 score for classification; RMSE, MAE, and R^2 score for regression. After the cross-validation, the best estimator

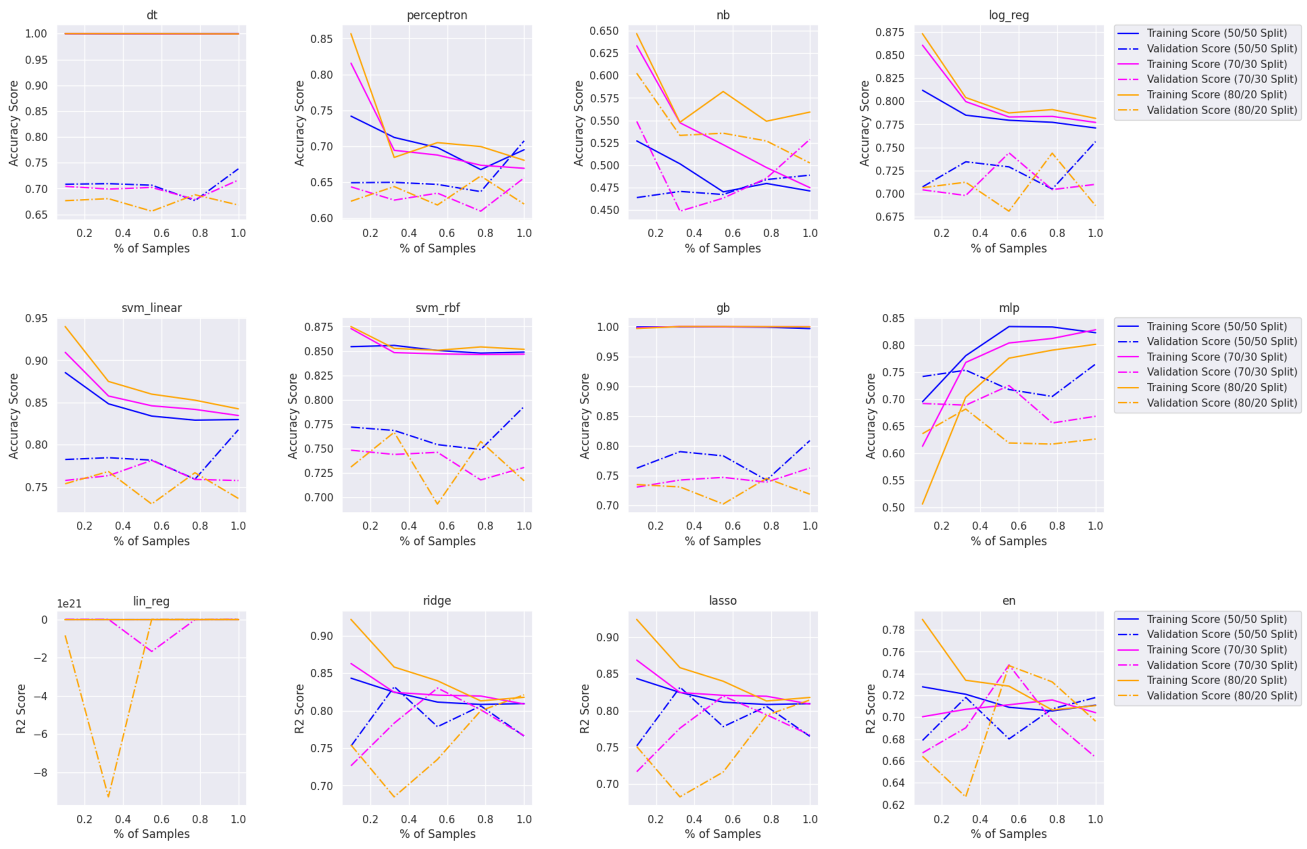

Fig. 2. Learning curve data.

of those ten was chosen based on which one had the highest accuracy or R^2 score. This yields one ”best” instance for each of the models.

For testing, each model was fit on the whole training set, then set to predict the testing data. Afterwards, we collected more cross-validated training and testing scores on five different folds and at five different sizes of the whole testing set. This would be used to generate learning curves for each model. We also collected each model's permutation importances for each feature based on the whole testing set. Both learning curve and importance data were collected based on accuracy score for classifiers and R^2 score for regression models.

To summarize, the steps to train and test all the models are as follows:

1) Split the features X and target values y into X_train, X_test, y_train, and y_test.
2) Perform the last data preprocessing steps and feature scaling on X_train and X_test.
3) Get y_train_d and y_test_d by binning y_train and y_test by the quantile ranges of y.
4) Train all the classifiers on 10 folds of y_train_d, collecting each of the 10 instances' evaluation metrics and selecting the best instance of each classifier by comparing their testing scores.
5) Repeat step 3 with the regressors on folds of y_train.
6) Test the best classifiers on y_test_d, collecting their learning curve scores, predicted y values, and the permutation importances of each feature for each model.
7) Repeat step 6 with the regressors on y_test.
8) Repeat all previous steps for each of the 3 training/testing splits: 50/50, 70/30, and 80/20.
9) Return all of the collected data for further analysis.

VI. Results

Once we've finished training and testing all the models, we now have the following information for each of the three training/testing splits:

- Copies of each of the 10 instances of the 12 models with their respective evaluation metrics.
  - Accuracy, precision, recall, and F1 score for classifiers, and MAE, RMSE, and R^2 score for regressors.
- The 12 model instances with the highest training scores, one for each model.
  - Scores are accuracy for classifiers and R^2 for regressors.
- The training and testing scores of the 12 best models.
- The learning curve data from the 12 best models on the testing sets, containing 5 sample sizes, training scores, and validation scores.
- The predictions for y made by the 12 best models from the testing sets.

TABLE II
50/50 SPLIT, DECISION TREE[a]

	dt 1	dt 2	dt 3	dt 4	dt 5	dt 6	dt 7	dt 8	dt 9	dt 10
Precision	0.7207	0.7564	0.7225	0.7495	0.7251	0.7850	0.7712	0.7466	0.7395	0.7671
Recall	0.7166	0.7488	0.7241	0.7510	0.7222	0.7856	0.7726	0.7486	0.7320	0.7666
F1	0.7182	0.7500	0.7223	0.7500	0.7233	0.7843	0.7713	0.7472	0.7345	0.7656
Accuracy	0.7169	0.7487	0.7249	0.7513	0.7222	0.7857	0.7725	0.7487	0.7321	0.7666

[a]Copied from Table VIII.

- The mean, standard deviation, and raw permutation importances of each feature in X for the 12 best models.
- Copies of y_test and y_test_d.

Most of the data was stored in arrays of three Python dictionaries, one for each training/testing distribution. The dictionaries were arranged by model key, so iterating through the data structures usually involved a nested loop iterating first by sample size and then by model key.

A. Learning Curves

Fig. 2 shows the learning curve data for each of the twelve models. The solid lines indicate the training scores for each of the models, and the dashed lines represent the validation scores. This information reveals several facts about the best-performing models:

- The tree-based Decision Tree and Gradient Boosting classifiers both overfitted the most, with near-perfect accuracy on the training set and much worse performance on the validation set.
- The MLP's ability to generalize tends to worsen as the number of samples increases, as shown by the increasing gap between the training and validation score with percentage of samples.
- The linear regression model acted erratically compared to the other three regression models.
- The Ridge, Lasso, and ElasticNet models appeared to have generalized the best, showing the smallest gaps in training and validation score compared to all other models. However, the logistic regression and SVM linear models also generalize when trained on the whole training split in the 50/50 case.

B. Individual Model Performance

All the relevant metrics for the ten instances of each model has been recorded in a set of 36 tables; that is, one table for each of the twelve model types for each of the three distributions. Table II is one of these tables, showing the performance of the decision tree classifier on ten different folds of the 50% split of the training set. The rest of the tables can be found in the Colab notebook that accompanies this report and at the end of the paper.

C. The Best Model

Given all the metrics data collected, the best model was found with the following procedure:

1) For each of the three training/testing splits, iterate through each dictionary of training/testing scores to find the model with the highest testing score in that split. Store that model's key and score. This gives us the best model from each split.
2) Compare the three best models' testing scores to find the most performant model out of all the splits.

The results of the decision process is shown in Table III. The best model was the Histogram Gradient Boosting Classifier from the 80/20 split. The column "Training Metrics" refers to the model's performance on the validation fold during training, and the "Testing Metrics" column shows the model's performance on the full testing set. While its performance on the testing set was not as good as its performance on the validation set during training, it still did better than all other models in the experiment.

TABLE III
ALL-TIME BEST MODEL METRICS

	Training Metrics	Testing Metrics
Sample	80/20	
Model	Gradient Boosting	
accuracy	0.85596	0.832011
recall	0.855901	0.831231
precision	0.856695	0.832324
F1	0.856086	0.831385

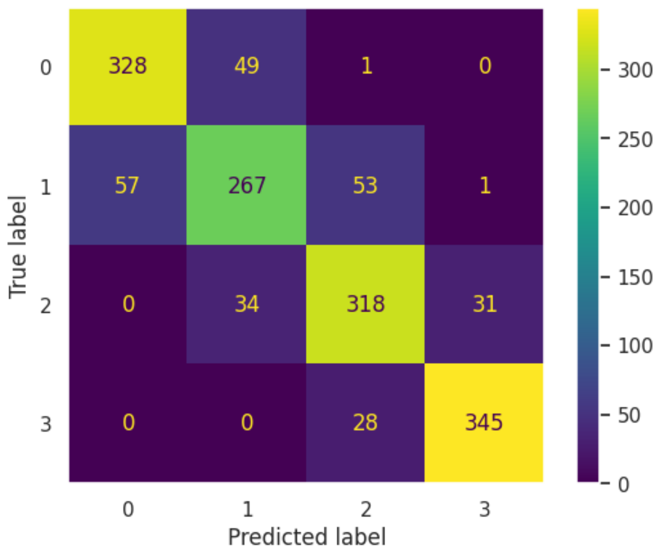

Fig. 3. Confusion matrix for the best model.

Fig. 3 shows the confusion matrix for this model. The higher number of true positives for classes 0 and 3 indicate that the model performed better on samples with target values further away from the median.

D. Metrics for each of the best models

Table IV shows the relevant evaluation metrics for the best iterations of each model on the three sampling distributions. This table reveals the following facts about the experiment:

TABLE IV
BEST MODELS

50/50 Split

	Precision/RMSE	Recall/MAE	F1/R^2	Accuracy
dt	0.7452	0.7445	0.7448	0.7441
perceptron	0.6522	0.6696	0.6580	0.6687
nb	0.5483	0.4685	0.4066	0.4668
log_reg	0.7490	0.7505	0.7467	0.7502
svm_linear	0.8150	0.8114	0.8126	0.8113
svm_rbf	0.8095	0.8030	0.8052	0.8029
gb	0.8237	0.8229	0.8233	0.8227
mlp	0.8041	0.7992	0.8008	0.7992
lin_reg	316029.6630	172252.0492	0.8033	NaN
ridge	316034.1626	172170.2392	0.8033	NaN
lasso	316029.3174	172252.4907	0.8033	NaN
en	382798.0277	204297.3976	0.7114	NaN

70/30 Split

	Precision/RMSE	Recall/MAE	F1/R^2	Accuracy
dt	0.7475	0.7473	0.7473	0.7469
perceptron	0.6552	0.6316	0.6145	0.6318
nb	0.5389	0.4656	0.4031	0.4638
log_reg	0.7516	0.7548	0.7503	0.7544
svm_linear	0.8209	0.8185	0.8194	0.8183
svm_rbf	0.8177	0.8110	0.8133	0.8108
gb	0.8284	0.8278	0.8281	0.8276
mlp	0.8184	0.8173	0.8178	0.8170
lin_reg	316843.0957	170777.8673	0.7991	NaN
ridge	316849.2731	170715.6205	0.7991	NaN
lasso	316834.8586	170777.2300	0.7992	NaN
en	384486.0446	202117.1277	0.7042	NaN

80/20 Split

	Precision/RMSE	Recall/MAE	F1/R^2	Accuracy
dt	0.7771	0.7769	0.7769	0.7765
perceptron	0.6802	0.6765	0.6652	0.6746
nb	0.5419	0.4627	0.4001	0.4597
log_reg	0.7550	0.7571	0.7519	0.7566
svm_linear	0.8139	0.8124	0.8129	0.8122
svm_rbf	0.8184	0.8164	0.8173	0.8161
gb	0.8312	0.8323	0.8314	0.8320
mlp	0.8167	0.8187	0.8173	0.8181
lin_reg	306900.7081	170024.1118	0.8032	NaN
ridge	306909.9984	169980.3990	0.8032	NaN
lasso	306900.1995	170035.0854	0.8032	NaN
en	369914.3016	198599.8068	0.7141	NaN

- The perceptron and Naive Bayes models easily performed the worst overall. Their metrics just do not compare to all the other models.
- By contrast, more complex models like the GB classifier, MLP, Ridge, and Lasso models showed better results in terms of accuracy or R^2 score. In this case, the more sophisticated models showed a significant enough improvement in performance to justify their usage over simpler alternatives.
- Despite its strange readings for R^2 score in the learning curves tables, the linear regression model's metrics appear to be normal compared to the other regression models.

E. Feature importances for each of the best models

Table V shows the features that yielded the highest feature importance for each of the models in each sample distribution. The table revealed the following additional information:

- The most important feature is almost unanimously the number of built square meters.
- The only outliers are "district_id_3" for the Naive Bayes classifier and "house_type_id_1" for the linear regression model. In the case of the Naive Bayes classifier, even the most important feature was the least helpful to it compared to other models. For the linear regression model, the mean importances for many of the categorical features were extremely high compared to all other data, suggesting a major mistake was made by using the model in such a way.

Table VI is given to show that the linear regression model behaved the way it did on many categorical features. Seeing that this is the only regression model that was not regularized, this suggests that the regularized nature of the Ridge, Lasso, and Elastic Net models significantly affected how they weighted their features. Even so, Table IV shows that the linear regression model performed about as well as the other regression models.

VII. CONCLUSION

As was shown earlier, the most performant model was the Histogram Gradient Boosting Classifier on the 80/20 split. In fact, the GB classifier was consistently the best-performing classifier on all three splits. We initially thought that this performance was due to the large number of categorical features, but the table of most important features suggests otherwise. Instead, it shows that the GB classifier consistently derived more importance from the already prominent sq_mt_built feature compared to the other classifiers. the model's resistance to outliers and ability to recognize non-linear relationships in the data helped it outperform the other models. The model trained on the 80/20 split specifically performed better because the gradient boosting algorithm works better on large datasets. That being said, the 80/20 GB classifier did generalize worse than the other two GB classifiers, as was shown in Fig. 2.

The Ridge, Lasso, and Elastic Net models all generalized better than the classifiers. Their training scores were consistently the closest to the validation scores when evaluated on the full testing set. The only classifiers that came close to their performance were the logistic regression and linear SVM classifiers, but only on the 50/50 splits. The Elastic Net model scored worse compared to the other two because its high learning rate made it frequently overestimate the optimal coefficients for the hyperplane. This can be seen in Fig. 2 by observing how the Elastic Net's learning curves are more jagged compared to the Ridge and Lasso's curves.

While more of the models generalized better on the 50/50 split, they generally got more accurate on the 80/20 split. Even so, only a few models had their performance significantly affected by the data distribution. Those would include the decision tree, perceptron, and GB models.

There were several adjustments that could have been made to improve the results here. Many of the models were initialized with the default parameters from scikit-learn, and the

TABLE V
MOST IMPORTANT FEATURES

50/50 Split

Model	Feature	Mean Importance
dt	sq_mt_built	0.346388
perceptron	sq_mt_built	0.227097
nb	district_id_3	0.055835
log_reg	sq_mt_built	0.270971
svm_linear	sq_mt_built	0.352527
svm_rbf	sq_mt_built	0.231331
gb	sq_mt_built	0.402858
mlp	sq_mt_built	0.238793
lin_reg	house_type_id_1	1.1521e20
ridge	sq_mt_built	1.474503
lasso	sq_mt_built	1.4763
en	sq_mt_built	0.430039

70/30 Split

Model	Feature	Mean Importance
dt	sq_mt_built	0.365168
perceptron	sq_mt_built	0.218783
nb	district_id_3	0.059612
log_reg	sq_mt_built	0.281217
svm_linear	sq_mt_built	0.37134
svm_rbf	sq_mt_built	0.248236
gb	sq_mt_built	0.406349
mlp	sq_mt_built	0.315697
lin_reg	house_type_id_1	2.2913e22
ridge	sq_mt_built	1.456495
lasso	sq_mt_built	1.457754
en	sq_mt_built	0.42364

80/20 Split

Model	Feature	Mean Importance
dt	sq_mt_built	0.370503
perceptron	sq_mt_built	0.270106
nb	district_id_3	0.056481
log_reg	sq_mt_built	0.283201
svm_linear	sq_mt_built	0.363492
svm_rbf	sq_mt_built	0.260714
gb	sq_mt_built	0.413228
mlp	sq_mt_built	0.297487
lin_reg	house_type_id_1	7.8272e21
ridge	sq_mt_built	1.52112
lasso	sq_mt_built	1.522347
en	sq_mt_built	0.432158

changes that were made to the parameters were done so in interest of computational efficiency, not necessarily better model performance. One such example is setting ”earl_stopping” on the MLP classifier. Given more time and resources, all the available hyperparameters could have been adjusted for optimal performance, ideally by using methods such as grid search. Additionally, the Lasso model's objective function failed to converge every time the model was trained. Despite our best efforts to promote convergence, it still was not able to do so. Further investigation would need to go into both the regularization scaling, number of iterations, and feature scaling to make it so the Lasso model actually does converge.

The evaluation of the best models in each sampling distribution and the selection of the all-time best model both also make assumptions for simplicity. Namely, the ”best model” has been defined as the model that has the highest testing accuracy or R^2 score. In this way, other important metrics such as RMSE, precision or recall have deliberately been ignored in the decision process. Furthermore, the final calculations for precision, recall, and F1 score for all the best models have been done using macro averaging. When these calculations are done with micro averaging instead, all three of these classification metrics are equal to the accuracy score. Table VII, showing the classification metrics for the best classifiers on the 80/20 split with micro averaging, is shown to demonstrate this. The scikit-learn documentation says this averaging technique works by ”counting the total true positives, false negatives and false positives,” which implies that each of these models all yielded as many false positives as they did false negatives. This seems highly unlikely, which is why we chose a different averaging method.

TABLE VI
MOST IMPORTANT CATEGORICAL FEATURES

50/50 Split

Model	District	Importance	House Type	Importance	Energy Cert	Importance
dt	3	0.029108	1	0.004446	no indicado	0.001588
perceptron	6	0.026144	5	0.003652	A	0.001164
nb	3	0.055835	2	0.008256	B	0.002117
log_reg	4	0.042233	5	0.001588	F	0.0009
svm_linear	4	0.057687	5	0.005716	F	0.001746
svm_rbf	4	0.025351	5	0.000794	B	0.001058
gb	3	0.030431	5	0.002964	no indicado	0.001905
mlp	4	0.044403	5	0.004075	en trámite	0.002435
lin_reg	4	8.9703e18	1	1.1521e20	en trámite	1.7069e18
ridge	5	0.028482	2	0.001808	C	0.000653
lasso	5	0.029798	1	0.005357	C	0.00079
en	5	0.005871	1	0.001033	C	0.00025

70/30 Split

Model	District	Importance	House Type	Importance	Energy Cert	Importance
dt	13	0.032804	5	0.002734	no indicado	0.003439
perceptron	4	0.030071	5	0.004145	C	0.001058
nb	3	0.059612	5	0.006526	B	0.002822
log_reg	4	0.045944	5	0.001235	A	0.001852
svm_linear	4	0.066667	5	0.005644	B	0.001852
svm_rbf	5	0.02963	5	0.003263	en trámite	0.002293
gb	3	0.027249	1	0.004762	en trámite	0.002205
mlp	4	0.056878	5	0.003439	en trámite	0.006526
lin_reg	4	5.2736e17	1	2.2913e22	en trámite	3.3972e20
ridge	5	0.028481	2	0.003754	C	0.000712
lasso	5	0.029129	1	0.006993	C	0.001035
en	5	0.005787	1	0.000827	C	0.000291

80/20 Split

Model	District	Importance	House Type	Importance	Energy Cert	Importance
dt	5	0.027513	1	0.005688	E	0.004497
perceptron	4	0.031878	5	0.003571	A	0.002646
nb	3	0.056481	5	0.005952	B	0.002116
log_reg	4	0.042063	2	0.002249	en trámite	0.004365
svm_linear	4	0.056878	5	0.00463	C	0.001852
svm_rbf	4	0.029233	5	0.004497	B	0.001852
gb	3	0.02963	5	0.004894	C	0.002116
mlp	4	0.050265	5	0.006217	D	0.003704
lin_reg	4	6.1420e16	1	7.8272e21	en trámite	1.9239e19
ridge	5	0.028664	2	0.003304	C	0.000733
lasso	5	0.02959	1	0.008082	C	0.001062
en	5	0.005827	1	0.000652	C	0.000299

TABLE VII
BEST MODELS: 80/20 SPLIT [MICRO AVERAGE]

	Precision	Recall	F1	Accuracy
dt	0.7765	0.7765	0.7765	0.7765
perceptron	0.6746	0.6746	0.6746	0.6746
nb	0.4597	0.4597	0.4597	0.4597
log_reg	0.7566	0.7566	0.7566	0.7566
svm_linear	0.8122	0.8122	0.8122	0.8122
svm_rbf	0.8161	0.8161	0.8161	0.8161
gb	0.8320	0.8320	0.8320	0.8320
mlp	0.8181	0.8181	0.8181	0.8181

Lastly, changes to the dataset itself also influenced the outcome of this experiment. Most notably, the decision to convert the problem into a multi-class classification problem for the classifiers introduced conditions that favored models like GB that favor multi-class environments. We could have generated only two classes for prices below and above the median to make the problem easier for binary classification models like logistic regression, but the resulting predictions would not be as useful. we could have also generated more classes based on more percentile ranges to get more specific predictions, but those predictions would be less likely to be accurate, and the resulting computational intensity could have reached unfeasible levels. We encountered a similar issue when considering whether to include categorical data for neighborhood IDs. Doing so would have given the final dataset over 100 features, drastically increasing the time and resources needed to finish evaluating all the models.

References

[1] I. Imran, Z. Umar, W. Muhammad, and Z. Atif, “Using Machine Learning Algorithms for Housing Price Prediction: The Case of Islamabad Housing Data,” Soft Computing and Machine Intelligence, vol. 1, no. 1, pp. 11—23, Jun. 2021.

[2] M. Kayakuş, M. Terzioğlu, and F. Yetiz, “Forecasting housing prices in Turkey by machine learning methods,” Aestimum, vol. 80, pp. 33–44, 2022.

[3] G. Zou, Z. Lai, Y. Li, X. Liu, and W. Li, “Exploring the nonlinear impact of air pollution on housing prices: A machine learning approach,” Economics of Transportation, vol. 31, art. 100272, Jun. 2022.

[4] M. Toktogaraev and M. Barra, ”Madrid real estate market [Dataset],” 2020. [Online]. Available: https://www.kaggle.com/datasets/mirbektoktogaraev/madrid-real-estate-market. [Accessed 7 3 2023].

VIII. Appendix

The 36 tables showing all models’ individual performances on the training folds start on the next page.

TABLE VIII
50/50 Split, Decision Tree

	dt 1	dt 2	dt 3	dt 4	dt 5	dt 6	dt 7	dt 8	dt 9	dt 10
Precision	0.7207	0.7564	0.7225	0.7495	0.7251	0.7850	0.7712	0.7466	0.7395	0.7671
Recall	0.7166	0.7488	0.7241	0.7510	0.7222	0.7856	0.7726	0.7486	0.7320	0.7666
F1	0.7182	0.7500	0.7223	0.7500	0.7233	0.7843	0.7713	0.7472	0.7345	0.7656
Accuracy	0.7169	0.7487	0.7249	0.7513	0.7222	0.7857	0.7725	0.7487	0.7321	0.7666

TABLE IX
50/50 Split, Perceptron

	perceptron 1	perceptron 2	perceptron 3	perceptron 4	perceptron 5	perceptron 6	perceptron 7	perceptron 8	perceptron 9	perceptron 10
Precision	0.5535	0.6316	0.6493	0.6730	0.7101	0.7299	0.7321	0.7261	0.6631	0.6788
Recall	0.5831	0.6521	0.6206	0.6478	0.6695	0.7011	0.7030	0.6804	0.5973	0.6719
F1	0.5274	0.6351	0.5484	0.6495	0.6777	0.7092	0.7034	0.6818	0.6140	0.6572
Accuracy	0.5847	0.6534	0.6190	0.6481	0.6693	0.7011	0.7037	0.6799	0.5968	0.6737

TABLE X
50/50 Split, Naive Bayes

	nb 1	nb 2	nb 3	nb 4	nb 5	nb 6	nb 7	nb 8	nb 9	nb 10
Precision	0.5173	0.5721	0.5757	0.5684	0.5642	0.5720	0.5329	0.5864	0.5353	0.5884
Recall	0.4705	0.4950	0.4925	0.4767	0.4914	0.4863	0.4703	0.4782	0.4576	0.4839
F1	0.3990	0.4364	0.4323	0.4205	0.4353	0.4286	0.4132	0.4218	0.3845	0.4354
Accuracy	0.4735	0.4974	0.4947	0.4788	0.4921	0.4868	0.4709	0.4788	0.4589	0.4854

TABLE XI
50/50 Split, Logistic Regression

	log_reg 1	log_reg 2	log_reg 3	log_reg 4	log_reg 5	log_reg 6	log_reg 7	log_reg 8	log_reg 9	log_reg 10
Precision	0.7351	0.7482	0.7090	0.7310	0.7604	0.7816	0.7430	0.7355	0.7267	0.7339
Recall	0.7403	0.7511	0.7138	0.7377	0.7648	0.7833	0.7488	0.7383	0.7241	0.7323
F1	0.7366	0.7424	0.7087	0.7320	0.7576	0.7801	0.7431	0.7309	0.7244	0.7294
Accuracy	0.7407	0.7513	0.7143	0.7381	0.7646	0.7831	0.7487	0.7381	0.7241	0.7321

TABLE XII
50/50 Split, SVM (Linear)

	svm_linear 1	svm_linear 2	svm_linear 3	svm_linear 4	svm_linear 5	svm_linear 6	svm_linear 7	svm_linear 8	svm_linear 9	svm_linear 10
Precision	0.7985	0.8211	0.8088	0.8156	0.7980	0.8270	0.8168	0.7948	0.8277	0.7824
Recall	0.7960	0.8200	0.8065	0.8172	0.7963	0.8228	0.8174	0.7909	0.8169	0.7827
F1	0.7970	0.8196	0.8072	0.8161	0.7970	0.8240	0.8171	0.7920	0.8194	0.7817
Accuracy	0.7963	0.8201	0.8069	0.8175	0.7963	0.8228	0.8175	0.7910	0.8170	0.7825

TABLE XIII
50/50 Split, SVM (RBF)

	svm_rbf 1	svm_rbf 2	svm_rbf 3	svm_rbf 4	svm_rbf 5	svm_rbf 6	svm_rbf 7	svm_rbf 8	svm_rbf 9	svm_rbf 10
Precision	0.7875	0.7979	0.8014	0.7904	0.7896	0.8275	0.8087	0.7872	0.8206	0.7878
Recall	0.7853	0.7936	0.7960	0.7882	0.7856	0.8227	0.8094	0.7828	0.8089	0.7825
F1	0.7862	0.7943	0.7975	0.7892	0.7868	0.8241	0.8079	0.7834	0.8122	0.7839
Accuracy	0.7857	0.7937	0.7963	0.7884	0.7857	0.8228	0.8095	0.7831	0.8090	0.7825

TABLE XIV
50/50 Split, Gradient Boosting

	gb 1	gb 2	gb 3	gb 4	gb 5	gb 6	gb 7	gb 8	gb 9	gb 10
Precision	0.7781	0.8265	0.7869	0.8084	0.8085	0.8492	0.8351	0.8181	0.8211	0.8239
Recall	0.7772	0.8252	0.7851	0.8092	0.8068	0.8492	0.8359	0.8175	0.8119	0.8224
F1	0.7775	0.8256	0.7854	0.8085	0.8072	0.8485	0.8344	0.8173	0.8133	0.8223
Accuracy	0.7778	0.8254	0.7857	0.8095	0.8069	0.8492	0.8360	0.8175	0.8117	0.8223

TABLE XV
50/50 Split, Multi-Layer Perceptron

	mlp 1	mlp 2	mlp 3	mlp 4	mlp 5	mlp 6	mlp 7	mlp 8	mlp 9	mlp 10
Precision	0.8025	0.8186	0.7744	0.8119	0.7963	0.8184	0.7995	0.7849	0.8091	0.7935
Recall	0.7988	0.8199	0.7746	0.8120	0.7962	0.8123	0.8015	0.7829	0.7985	0.7932
F1	0.8000	0.8188	0.7743	0.8110	0.7961	0.8137	0.8000	0.7829	0.8010	0.7927
Accuracy	0.7989	0.8201	0.7751	0.8122	0.7963	0.8122	0.8016	0.7831	0.7984	0.7931

TABLE XVI
50/50 Split, Linear Regression

	lin_reg 1	lin_reg 2	lin_reg 3	lin_reg 4	lin_reg 5	lin_reg 6	lin_reg 7	lin_reg 8	lin_reg 9	lin_reg 10
RMSE	322828.5324	297331.0998	420487.1076	265598.2920	360965.6683	298571.2287	291414.2906	378653.4198	301182.4280	345502.2831
MAE	173473.6905	179472.7513	197629.0582	158383.6799	184723.3413	178616.3651	177077.1270	181215.1349	173488.9231	195091.3607
R^2	0.7972	0.7811	0.7708	0.7403	0.7721	0.7926	0.7821	0.7432	0.8234	0.8223

TABLE XVII
50/50 Split, Ridge

	ridge 1	ridge 2	ridge 3	ridge 4	ridge 5	ridge 6	ridge 7	ridge 8	ridge 9	ridge 10
RMSE	322880.8875	297272.8812	420528.5040	265523.3602	360962.2101	298573.1064	291284.3159	378543.6244	301157.9806	345465.4434
MAE	173451.7877	179367.3619	197555.8889	158318.7119	184627.2243	178545.5429	176942.6563	181022.9466	173316.4159	194964.9832
R^2	0.7972	0.7811	0.7707	0.7404	0.7721	0.7926	0.7823	0.7434	0.8234	0.8223

TABLE XVIII
50/50 Split, Lasso

	lasso 1	lasso 2	lasso 3	lasso 4	lasso 5	lasso 6	lasso 7	lasso 8	lasso 9	lasso 10
RMSE	322849.6902	297335.0976	420489.5448	265584.9598	360999.8716	298588.9032	291406.1025	378606.1292	301184.0131	345482.2807
MAE	173499.6956	179473.1367	197632.3304	158373.8971	184713.9401	178625.3080	177071.6754	181211.2484	173485.7538	195094.9617
R^2	0.7972	0.7811	0.7708	0.7403	0.7721	0.7925	0.7821	0.7433	0.8234	0.8223

TABLE XIX
50/50 Split, Elastic Net

	en 1	en 2	en 3	en 4	en 5	en 6	en 7	en 8	en 9	en 10
RMSE	401985.7591	332753.4163	528160.3266	276267.0632	434839.0887	331965.6395	339024.9050	444618.6382	384476.2526	434654.7165
MAE	212079.1088	198325.0805	236323.7093	179158.3094	218917.4302	194941.6733	202044.2816	210726.1866	204733.5963	232798.1039
R^2	0.6856	0.7258	0.6383	0.7190	0.6693	0.7436	0.7050	0.6460	0.7122	0.7187

TABLE XX
70/30 Split, Decision Tree

	dt 1	dt 2	dt 3	dt 4	dt 5	dt 6	dt 7	dt 8	dt 9	dt 10
Precision	0.7461	0.7580	0.7525	0.7118	0.7390	0.7514	0.7739	0.7300	0.7311	0.7431
Recall	0.7426	0.7581	0.7502	0.7089	0.7410	0.7525	0.7733	0.7337	0.7276	0.7422
F1	0.7436	0.7575	0.7508	0.7099	0.7392	0.7516	0.7736	0.7308	0.7288	0.7424
Accuracy	0.7429	0.7580	0.7505	0.7089	0.7410	0.7524	0.7732	0.7335	0.7278	0.7424

TABLE XXI
70/30 Split, Perceptron

	perceptron 1	perceptron 2	perceptron 3	perceptron 4	perceptron 5	perceptron 6	perceptron 7	perceptron 8	perceptron 9	perceptron 10
Precision	0.6629	0.6330	0.6297	0.6448	0.6947	0.6569	0.6798	0.6102	0.5102	0.7111
Recall	0.6236	0.6418	0.6465	0.6635	0.6713	0.6589	0.6495	0.6075	0.5909	0.6795
F1	0.6301	0.6048	0.6142	0.6452	0.6749	0.6148	0.5960	0.5537	0.5115	0.6840
Accuracy	0.6238	0.6408	0.6465	0.6635	0.6711	0.6578	0.6484	0.6068	0.5917	0.6799

TABLE XXII
70/30 Split, Naive Bayes

	nb 1	nb 2	nb 3	nb 4	nb 5	nb 6	nb 7	nb 8	nb 9	nb 10
Precision	0.5633	0.5729	0.4861	0.5509	0.5900	0.5755	0.5406	0.5454	0.6279	0.5353
Recall	0.4602	0.4808	0.4622	0.4890	0.4947	0.4769	0.4845	0.4618	0.4811	0.4700
F1	0.3944	0.4311	0.3743	0.4254	0.4419	0.4265	0.4150	0.4030	0.4265	0.4020
Accuracy	0.4612	0.4820	0.4631	0.4896	0.4953	0.4764	0.4839	0.4612	0.4820	0.4716

TABLE XXIII
70/30 Split, Logistic Regression

	log_reg 1	log_reg 2	log_reg 3	log_reg 4	log_reg 5	log_reg 6	log_reg 7	log_reg 8	log_reg 9	log_reg 10
Precision	0.7186	0.7602	0.7858	0.7540	0.7214	0.7553	0.7593	0.7406	0.7488	0.7647
Recall	0.7185	0.7583	0.7865	0.7600	0.7278	0.7603	0.7641	0.7472	0.7520	0.7575
F1	0.7156	0.7552	0.7822	0.7533	0.7205	0.7555	0.7593	0.7390	0.7480	0.7562
Accuracy	0.7183	0.7580	0.7864	0.7599	0.7278	0.7599	0.7637	0.7467	0.7524	0.7576

TABLE XXIV
70/30 Split, SVM (Linear)

	svm_linear 1	svm_linear 2	svm_linear 3	svm_linear 4	svm_linear 5	svm_linear 6	svm_linear 7	svm_linear 8	svm_linear 9	svm_linear 10
Precision	0.7801	0.8157	0.8400	0.8064	0.8121	0.8078	0.8019	0.8252	0.8235	0.8003
Recall	0.7752	0.8093	0.8393	0.8033	0.8128	0.8074	0.7997	0.8262	0.8165	0.7954
F1	0.7764	0.8100	0.8396	0.8045	0.8121	0.8075	0.8006	0.8250	0.8185	0.7964
Accuracy	0.7750	0.8091	0.8393	0.8034	0.8129	0.8072	0.7996	0.8261	0.8166	0.7955

TABLE XXV
70/30 Split, SVM (RBF)

	svm_rbf 1	svm_rbf 2	svm_rbf 3	svm_rbf 4	svm_rbf 5	svm_rbf 6	svm_rbf 7	svm_rbf 8	svm_rbf 9	svm_rbf 10
Precision	0.7783	0.8193	0.8215	0.8206	0.7914	0.8052	0.8122	0.8175	0.8220	0.8119
Recall	0.7732	0.8148	0.8165	0.8184	0.7900	0.8016	0.8072	0.8167	0.8108	0.8068
F1	0.7748	0.8160	0.8182	0.8192	0.7905	0.8028	0.8089	0.8167	0.8126	0.8081
Accuracy	0.7732	0.8147	0.8166	0.8185	0.7902	0.8015	0.8072	0.8166	0.8110	0.8068

TABLE XXVI
70/30 Split, Gradient Boosting

	gb 1	gb 2	gb 3	gb 4	gb 5	gb 6	gb 7	gb 8	gb 9	gb 10
Precision	0.7929	0.8548	0.8437	0.8044	0.8011	0.8137	0.8604	0.8425	0.8366	0.8224
Recall	0.7902	0.8545	0.8429	0.8050	0.8015	0.8130	0.8602	0.8432	0.8354	0.8200
F1	0.7911	0.8543	0.8429	0.8045	0.8010	0.8134	0.8603	0.8422	0.8357	0.8203
Accuracy	0.7902	0.8544	0.8431	0.8053	0.8015	0.8129	0.8601	0.8431	0.8355	0.8201

TABLE XXVII
70/30 Split, Multi-Layer Perceptron

	mlp 1	mlp 2	mlp 3	mlp 4	mlp 5	mlp 6	mlp 7	mlp 8	mlp 9	mlp 10
Precision	0.7790	0.8228	0.8304	0.8109	0.8026	0.8015	0.8077	0.8165	0.8149	0.8031
Recall	0.7769	0.8206	0.8299	0.8089	0.8034	0.8017	0.8035	0.8167	0.8089	0.7974
F1	0.7777	0.8207	0.8300	0.8097	0.8019	0.8012	0.8049	0.8162	0.8096	0.7983
Accuracy	0.7769	0.8204	0.8299	0.8091	0.8034	0.8015	0.8034	0.8166	0.8091	0.7973

TABLE XXVIII
70/30 Split, Linear Regression

	lin_reg 1	lin_reg 2	lin_reg 3	lin_reg 4	lin_reg 5	lin_reg 6	lin_reg 7	lin_reg 8	lin_reg 9	lin_reg 10
RMSE	337932.2803	322951.1312	286558.1921	301937.4841	390339.0620	335037.9880	258815.2176	297928.2190	362394.0472	340843.3561
MAE	171315.1739	182679.5595	170753.7032	170432.2495	190632.1909	166650.9546	168173.2250	178800.5142	183981.2023	184523.9451
R^2	0.7984	0.8130	0.8219	0.7925	0.7715	0.7306	0.8298	0.7942	0.7521	0.8251

TABLE XXIX
70/30 Split, Ridge

	ridge 1	ridge 2	ridge 3	ridge 4	ridge 5	ridge 6	ridge 7	ridge 8	ridge 9	ridge 10
RMSE	337909.4440	322920.4648	286545.5323	301946.7123	390360.5382	334986.6998	258780.1887	297902.3931	362349.5514	340841.3819
MAE	171240.9934	182624.2346	170704.3490	170361.3728	190568.6624	166583.7542	168127.4398	178739.8546	183869.2883	184436.9849
R^2	0.7984	0.8130	0.8219	0.7925	0.7715	0.7307	0.8298	0.7942	0.7521	0.8251

TABLE XXX
70/30 Split, Lasso

	lasso 1	lasso 2	lasso 3	lasso 4	lasso 5	lasso 6	lasso 7	lasso 8	lasso 9	lasso 10
RMSE	337931.4973	322953.2739	286556.6021	301937.4301	390338.1258	335038.9469	258812.6455	297932.0970	362399.3290	340842.0685
MAE	171310.7433	182682.5641	170752.7321	170432.5394	190623.9894	166654.1995	168188.7114	178804.1661	184004.9531	184523.7941
R^2	0.7984	0.8130	0.8219	0.7925	0.7715	0.7306	0.8298	0.7941	0.7521	0.8251

TABLE XXXI
70/30 Split, Elastic Net

	en 1	en 2	en 3	en 4	en 5	en 6	en 7	en 8	en 9	en 10
RMSE	413314.5554	373758.0845	356019.1054	378027.5408	471760.7111	384580.2020	303037.4072	353392.4188	419229.7268	438875.9360
MAE	203932.1283	214032.4919	211179.2123	205364.6466	222734.3844	195350.4789	194771.4139	205382.3786	208799.2487	229558.2731
R^2	0.6984	0.7495	0.7251	0.6747	0.6663	0.6450	0.7666	0.7104	0.6682	0.7100

TABLE XXXII
80/20 Split, Decision Tree

	dt 1	dt 2	dt 3	dt 4	dt 5	dt 6	dt 7	dt 8	dt 9	dt 10
Precision	0.7433	0.7417	0.7711	0.7541	0.7655	0.7769	0.7341	0.7341	0.7682	0.7567
Recall	0.7402	0.7404	0.7685	0.7503	0.7669	0.7799	0.7319	0.7301	0.7699	0.7549
F1	0.7408	0.7409	0.7695	0.7519	0.7659	0.7776	0.7328	0.7317	0.7689	0.7551
Accuracy	0.7405	0.7405	0.7686	0.7504	0.7669	0.7798	0.7318	0.7301	0.7699	0.7550

TABLE XXXIII
80/20 Split, Perceptron

	perceptron 1	perceptron 2	perceptron 3	perceptron 4	perceptron 5	perceptron 6	perceptron 7	perceptron 8	perceptron 9	perceptron 10
Precision	0.6795	0.6906	0.7080	0.7139	0.6210	0.7059	0.6724	0.7065	0.6565	0.6714
Recall	0.6400	0.6057	0.6297	0.7220	0.6114	0.6366	0.6206	0.7033	0.6672	0.6269
F1	0.6493	0.6104	0.5653	0.7133	0.6077	0.6357	0.6352	0.6978	0.6591	0.6294
Accuracy	0.6397	0.6050	0.6314	0.7223	0.6116	0.6374	0.6209	0.7036	0.6672	0.6275

TABLE XXXIV
80/20 Split, Naive Bayes

	nb 1	nb 2	nb 3	nb 4	nb 5	nb 6	nb 7	nb 8	nb 9	nb 10
Precision	0.5512	0.5452	0.5814	0.5248	0.5197	0.5655	0.5673	0.5962	0.6028	0.5429
Recall	0.4723	0.4585	0.4866	0.4736	0.4684	0.4816	0.4832	0.4948	0.4834	0.4660
F1	0.4168	0.3899	0.4339	0.3870	0.4016	0.4200	0.4250	0.4434	0.4277	0.3976
Accuracy	0.4727	0.4595	0.4876	0.4744	0.4694	0.4818	0.4834	0.4950	0.4834	0.4669

TABLE XXXV
80/20 Split, Logistic Regression

	log_reg 1	log_reg 2	log_reg 3	log_reg 4	log_reg 5	log_reg 6	log_reg 7	log_reg 8	log_reg 9	log_reg 10
Precision	0.7599	0.7384	0.7493	0.7797	0.7378	0.7281	0.7473	0.7547	0.7565	0.7543
Recall	0.7621	0.7390	0.7474	0.7835	0.7439	0.7337	0.7520	0.7602	0.7599	0.7465
F1	0.7606	0.7344	0.7436	0.7784	0.7333	0.7275	0.7436	0.7557	0.7542	0.7456
Accuracy	0.7620	0.7388	0.7471	0.7835	0.7438	0.7334	0.7517	0.7599	0.7599	0.7467

TABLE XXXVI
80/20 Split, SVM (Linear)

	svm_linear 1	svm_linear 2	svm_linear 3	svm_linear 4	svm_linear 5	svm_linear 6	svm_linear 7	svm_linear 8	svm_linear 9	svm_linear 10
Precision	0.8146	0.8108	0.8144	0.8485	0.8018	0.8074	0.8141	0.8318	0.8202	0.7939
Recall	0.8066	0.8067	0.8101	0.8479	0.8016	0.8080	0.8147	0.8310	0.8162	0.7898
F1	0.8092	0.8080	0.8104	0.8481	0.8016	0.8076	0.8143	0.8312	0.8172	0.7904
Accuracy	0.8066	0.8066	0.8099	0.8479	0.8017	0.8079	0.8146	0.8311	0.8162	0.7897

TABLE XXXVII
80/20 Split, SVM (RBF)

	svm_rbf 1	svm_rbf 2	svm_rbf 3	svm_rbf 4	svm_rbf 5	svm_rbf 6	svm_rbf 7	svm_rbf 8	svm_rbf 9	svm_rbf 10
Precision	0.8060	0.8007	0.8246	0.8328	0.7891	0.7819	0.8013	0.8258	0.8172	0.8155
Recall	0.7916	0.7951	0.8199	0.8314	0.7884	0.7814	0.7980	0.8243	0.8129	0.8113
F1	0.7959	0.7969	0.8210	0.8320	0.7883	0.7816	0.7992	0.8243	0.8135	0.8124
Accuracy	0.7917	0.7950	0.8198	0.8314	0.7884	0.7815	0.7980	0.8245	0.8129	0.8113

TABLE XXXVIII
80/20 Split, Gradient Boosting

	gb 1	gb 2	gb 3	gb 4	gb 5	gb 6	gb 7	gb 8	gb 9	gb 10
Precision	0.8043	0.8182	0.8545	0.8533	0.7995	0.8149	0.8196	0.8567	0.8406	0.8395
Recall	0.7982	0.8164	0.8546	0.8527	0.7998	0.8146	0.8213	0.8559	0.8411	0.8377
F1	0.7999	0.8172	0.8544	0.8527	0.7997	0.8146	0.8201	0.8561	0.8408	0.8382
Accuracy	0.7983	0.8165	0.8545	0.8529	0.8000	0.8146	0.8212	0.8560	0.8411	0.8377

TABLE XXXIX
80/20 Split, Multi-Layer Perceptron

	mlp 1	mlp 2	mlp 3	mlp 4	mlp 5	mlp 6	mlp 7	mlp 8	mlp 9	mlp 10
Precision	0.8114	0.7995	0.8238	0.8509	0.7657	0.8065	0.8098	0.8152	0.8158	0.7984
Recall	0.8015	0.7984	0.8217	0.8512	0.7655	0.8064	0.8081	0.8127	0.8129	0.7947
F1	0.8045	0.7983	0.8210	0.8510	0.7641	0.8064	0.8087	0.8131	0.8133	0.7950
Accuracy	0.8017	0.7983	0.8215	0.8512	0.7653	0.8063	0.8079	0.8129	0.8129	0.7947

TABLE XL
80/20 Split, Linear Regression

	lin_reg 1	lin_reg 2	lin_reg 3	lin_reg 4	lin_reg 5	lin_reg 6	lin_reg 7	lin_reg 8	lin_reg 9	lin_reg 10
RMSE	296415.4068	380683.8371	315416.5513	281233.4576	312988.0398	368454.5484	326482.2596	298736.2887	332392.0499	339130.1504
MAE	167042.7623	179763.6975	177891.3785	169932.2760	174313.3769	180943.3725	174375.9354	180099.7748	174158.9917	186413.1175
R^2	0.8149	0.7732	0.8183	0.8212	0.7813	0.7635	0.7955	0.7733	0.7787	0.8179

TABLE XLI
80/20 Split, Ridge

	ridge 1	ridge 2	ridge 3	ridge 4	ridge 5	ridge 6	ridge 7	ridge 8	ridge 9	ridge 10
RMSE	296412.7192	380685.7977	315399.8043	281207.3336	312974.0459	368461.2070	326498.0778	298696.0826	332350.5449	339137.4791
MAE	166987.1131	179725.8419	177857.3418	169883.4593	174256.9979	180888.4587	174334.2223	180037.0211	174050.8895	186326.4997
R^2	0.8149	0.7732	0.8184	0.8212	0.7814	0.7635	0.7955	0.7734	0.7788	0.8179

TABLE XLII
80/20 Split, Lasso

	lasso 1	lasso 2	lasso 3	lasso 4	lasso 5	lasso 6	lasso 7	lasso 8	lasso 9	lasso 10
RMSE	296415.4060	380669.1985	315419.6979	281225.7256	312986.3464	368454.5755	326501.2723	298733.7123	332392.6037	339134.4975
MAE	167042.7568	179788.6391	177900.0186	169932.5453	174319.1147	180941.8408	174373.7941	180098.0502	174159.3388	186406.8383
R^2	0.8149	0.7732	0.8183	0.8212	0.7813	0.7635	0.7955	0.7733	0.7787	0.8179

TABLE XLIII
80/20 Split, Elastic Net

	en 1	en 2	en 3	en 4	en 5	en 6	en 7	en 8	en 9	en 10
RMSE	366327.6185	464302.9475	368423.8655	348568.5063	367331.9497	443121.8300	400590.7784	337737.1022	391200.8646	432824.1107
MAE	198781.7673	213263.3687	213340.8462	209793.8747	201876.7875	211991.7582	211298.8505	201492.1471	199703.0086	228266.0648
R^2	0.7172	0.6626	0.7522	0.7253	0.6988	0.6579	0.6921	0.7103	0.6935	0.7035