

Forecasting Future Sales: A Comparison of Traditional and Modern Forecasting Techniques

Steven VanOmmeren Predictive Analytics Final*

May 11, 2025

I. Introduction

In Kaggle’s “Predict Future Sales” data competition, the goal is to predict 1C Company’s sales for November 2015 using data from January 2013 to October 2015.¹ The key challenge of this competition is that the predictions must be by store and product, requiring a total of 214,200 distinct predictions. I employ a variety of statistical techniques to approach this forecasting problem, comparing their strengths and weaknesses.

Forecasting is the prediction of future events using historical data and industry experience. Accurate forecasts are vitally important to ensuring that appropriate business and government decisions are made. When sufficient data are available, statistical forecasting methods can approximate the time-series behavior of relevant variables using historical data of both the variable of interest and any additional relevant variables that may increase a model’s performance. In this paper, I use a variety of statistical forecasting methods to predict 1C Company’s sales.

Forecasting sales is critical to any company. Accurate forecasts of future sales at the shop and product level would allow 1C Company to identify under- and over-performing products.

*A complete replication package of this project is available at <https://github.com/svanomm/forecasting-final-2025>.

¹<https://www.kaggle.com/competitions/competitive-data-science-predict-future-sales/overview>; <https://1c.ru/eng/title.htm>

They can then adjust marketing and pricing practices accordingly, for example by reducing prices of under-performing products to increase sales. Additionally, interpretable forecasting models would allow the company to understanding why sales are up or down. As explained below, while machine learning methods are traditionally described as “black box” methods, techniques exist to help understand the output’s sensitivity to changes in input variables.

II. Literature Review

Coulombe et al. (2022) evaluate various methods for predicting macroeconomic and financial indicators. The authors go so far as to describe the nonlinear capabilities associated with machine learning techniques as a “game-changer” for predicting irregular events like housing bubble bursts. The authors also endorse the use of K-fold cross-validation as a method to tune hyperparameters.

Ke et al. (2017) introduce LightGBM, a gradient boosting tree-based machine learning model. LightGBM is an extension of an existing model called XGBoost. LightGBM modifies XGBoost to implement approximation techniques that drastically increase the speed of computation, allowing for many more model iterations per second. Because of its computational efficiency and generally very good predictive ability, LightGBM is a common choice for prediction competitions. In fact, most of the top performers in the Kaggle competition use LightGBM to predict future sales.

Wang et al. (2022) test a variety of machine learning models to predict corporate finance risk indicators, comparing existing machine learning methods with LightGBM. The authors test four models: K-nearest neighbors, decision trees, random forest, and LightGBM. Reviewing a variety of metrics, the authors find that LightGBM beats the other models in most of the tests, though it occasionally performs slightly worse than their random forest model. The authors conclude that LightGBM is significantly better than other existing machine learning techniques, and they believe that it can be used to effectively predict the financing

risk of enterprises going forward.

III. Methods

A. Data Preparation

Sales data was provided in separate files, so the first step is to join these files together to build the full sales dataset. After joining, the dataset has 2,935,849 observations, organized by shop-product-day. There are 21,807 products and 60 shops in the raw data, with 424,124 shop-product combinations.

One of the many challenges with this prediction problem is that products are not always observed in each day/month. There are 102,796 shop-products in the prediction month (Nov. 2015) that *do not exist in the training data*.² To make predictions for these brand new shop-products, we need to incorporate information from outside of the shop-product.³ This type of data is called an “unbalanced panel.” Since many forecasting techniques require a balanced panel (every shop-product is observed in every month), I balance the data by filling in days/months with no sales.

I performed some data cleaning as well. I remove shop-products from the training data that do not appear in the testing set. Four shops were found to have changed their shop ID over time, so I update the ID to be consistent over time.⁴ Normally, in a forecasting analysis, great care would be taken to identify outlier sales in the data. For this competition, however, sales forecasts are evaluated in the range $[0, 20]$, meaning that negative sales (returns) are given a value of 0, and large sales quantities are capped at 20.⁵ I also clip my sales values

²There are also many products in the training data that do not exist in the testing data. I remove these products from my data.

³One might also predict 0 sales for brand new products, but that is unlikely to be an accurate forecasting method.

⁴This issue was found by <https://www.kaggle.com/code/abubakar624/first-place-solution-kaggle-predict-future-sales/comments>.

⁵<https://www.kaggle.com/competitions/competitive-data-science-predict-future-sales/overview>.

to be consistent with this, which eliminates any potential outlier values.

Because the daily data is quite large (and would be extremely large if balanced at the daily level), I aggregate it to the shop-product-monthly level. This has the added benefit of making the forecasts more convenient, since we have to predict the next month of sales. I also remove the first 3 months of data because many of my controls are moving averages. After balancing the data, my training data has 6,426,000 observations, 91.2% of which have 0 sales.⁶ I use this dataset for the machine learning and Naive models.

Another challenge for forecasting with this data is that traditional forecasting techniques would require a separate model for each shop-product. Since there are 214,200 predictions to make, computing a separate model for each of them is not feasible. However, there are only 42 shops and 62 product categories; it is practical to run separate models for each shop, or for each product category. So, I create two aggregated datasets, one at the shop-month level, and one at the product category-month level. These datasets have 1470 and 2170 observations, respectively.

B. Feature Engineering

Feature engineering is the process of extracting useful control variables out of the data. I performed extensive feature engineering to create and test hundreds of variables with the data. My variables can be broken down into four categories: lags, moving averages, economic variables, and dummy variables.

i. Lags

The most obvious way to predict future sales would be a function of previous months' sales for the store-product, i.e. using lagged variables. For each of the variables that I tested in my models, I included between 3 and 6 months of lags.⁷ While some of these variables will

⁶I tried training models only on products with positive sales in the last month of training data, but these models ended up with worse sales predictions.

⁷I tested adding more lagged periods (up to 24 months) but found them to not increase predictive power.

be described below, I now report a table of how many lags were used for each variable.

Table 1: Lagged Variables Used in Modeling

Variable	Lags Used
sales	1-6 months
MA3 sales	1-3 months
MA6 sales	1-3 months
MA12 sales	1-3 months
relative price	1-6 months
MA3 relative price	1-3 months
% Δ product sales	1-3 months
% Δ shop sales	1-3 months
% Δ substitute product sales	1-3 months
% Δ complement product 1 price	1-3 months
% Δ complement product 2 price	1-3 months
% Δ complement product 3 price	1-3 months
% Δ substitute shop 1 price	1-3 months
% Δ substitute shop 2 price	1-3 months
% Δ substitute shop 3 price	1-3 months

“MA” is moving average. “% Δ ” is monthly percent change.

See below for descriptions of additional variables.

ii. Moving Averages

Moving averages are calculated by averaging a series of data points over a specified time period, creating a smoothed time series that filters out short-term fluctuations. They are valuable in forecasting because they can highlight underlying trends in the data by reducing noise. Different window sizes (e.g., 3-month, 12-month) capture varying degrees of trend sensitivity.

In my analysis, I use moving averages for sales and relative prices. For sales, I use three different windows of moving averages: 3-month, 6-month, and 12-month. This captures varying levels of trends and allows the models to decide which is most important and useful

for prediction.

iii. Economic Variables

Because predicting sales is inherently an economic problem, I incorporate some variables motivated by economic theory to inform the model. First, I create a “relative price” variable, which is calculated by dividing the price of a product in a given store-month by the average price of all other products in the same product category.⁸ If relative price is lower, then the product is sold at a discount relative to similar products and should have higher sales.

Next, I create a set of variables based on the economic concepts of **substitutes** and **complements**. Substitute goods are products consumers can use in place of one another; if the price of one rises, demand for its substitute typically increases (i.e. a positive correlation). Complementary goods are those frequently consumed together; when the price of one rises, demand for its complement tends to fall (i.e. a negative correlation). I create variables for substitute/complement product categories as well as substitute shops.

Substitute products are easy to identify because of the product category field. As such, I identify substitute products as all other products within a product category. Complementary products are identified empirically. I first calculate the average percent change in quantity and in price for each product category. I then calculate a correlation matrix over all product categories. For each product category x , I can identify the other categories (y_1, y_2, y_3) whose change in price is most negatively correlated with a change in quantity for x . I take the top 3 values as my complement products, and then I include the percent change in price of the complement products as control variables.

I use the same method to identify substitute shops as well, i.e. the shops who empirically have the highest correlation between shop x ’s percent change in price and shop y ’s percent change in quantity. I take the top 3 substitute shops for each shop, and then I include the

⁸I take the log of both prices to reduce the influence of extreme outlier prices in the data.

percent change in average price of the substitute shops as control variables.⁹

iv. Dummy Variables

Finally, for my machine learning models, I also created dummy variables for item category, shop, and month of year, separately. These dummy variables will help the models to pick up fundamental characteristics of the item categories and shops, as well as any monthly seasonality in the data. Monthly dummies eliminate the need to control for holiday seasons or other months that experience higher average sales in all products, as well as number of days per month.

C. Naive

In a Naive model, forecasts are made by simply using the previous month's sales for each store-product. For brand new products, I average the store's average quantity, and the item-category's average quantity from the previous month, as specified below:

$$\widehat{sales}_{sic,t+1}^{Naive} = \begin{cases} sales_{sic,t} & , \text{not new} \\ (sales_{s,t} + sales_{c,t})/2 & , \text{new} \end{cases}$$

Subscript s indicates the store, i is for item, and c is for item category.

D. Exponential Smoothing

Exponential smoothing is a class of predictive models that extract a trend and seasonality pattern from the time series, and then allow the trend and seasonality to update over time. Unlike many other techniques, exponential smoothing models do not accept exogenous regressors, only using the dependent variable as a predictor.

⁹I did not calculate complement shops because I did not expect them to provide any meaningful economic content. The shops do not specialize in selling certain products, so there should not be opportunity for complementary sales.

As stated above, the full regression data is too large to perform a separate prediction problem for each shop-product. Instead, I calculate two sets of predictions. The first class of predictions is at the shop level, and the second one is at the item category level. This gives a total of 104 models. Each of these models is a separate exponential smoothing model with additive trend and additive seasonality.

Since we still have to make predictions for each shop-product, I transform the shop and item-category model predictions into factors that are multiplied by the previous month's sales. The resulting predictions can be specified as:

$$\widehat{sales_{sic,t+1}}^{ETS} = sales_{sic,t} \cdot f_s \cdot f_c,$$

where

$$f_s = \frac{\widehat{sales_{s,t+1}}}{sales_{s,t}} \quad f_c = \frac{\widehat{sales_{c,t+1}}}{sales_{c,t}}.$$

The idea is that the f_s come from a separate model for each store, where we predict the store's average sales, and likewise for f_c on the item categories. Translating the predictions into factors allows me to combine them at the store-product level.

One problem is that for brand new products, we do not have any previous month of sales. So, the resulting prediction will always be zero with this formula. To circumvent this issue, my prediction for new products is created by averaging the predictions from each of the relevant models:¹⁰

$$\widehat{sales_{sic,t+1}} = (\widehat{sales_{s,t+1}} + \widehat{sales_{c,t+1}})/2.$$

E. ARIMA and SARIMAX

These models incorporate autoregressive, differencing, and moving average terms into the estimation. I test two different models: one is a standard ARIMA model with no additional

¹⁰A few product categories are also brand new in the data. In this case, I use the shop-level prediction.

regressors, and the SARIMAX incorporates some of the variables created in my feature engineering process.¹¹ Both models automatically select how many autoregressive, differencing, and moving average terms to include in the regression using AIC as a guiding metric.¹²

As with my exponential smoothing models, a separate model is calculated for each shop and then for each product category. This gives a total of 208 models. Predictions are made in the same way as discussed above.

F. Neural Network

Next up, I try to model sales using a neural network with one hidden layer. To implement the neural network, I used the `scikit-learn`¹³ package in Python, function `MLPRegressor`.¹⁴ After feature engineering, and including the dummy variables as well, I was left with 167 control variables. These form the first layer of the neural network, and then the hidden layer has 56 neurons (approximately 1/3 the size of the first layer).¹⁵

Neural networks perform best when each of the control variables is standardized to be on the same scale. Because of this, I use `scikit-learn`'s pipeline feature to first scale all of my variables before implementing the model.¹⁶ Additionally, to reduce the variable count and increase computation speed, I add a Principal Components Analysis (PCA) step into the pipeline as well. This technique transforms the variables and then allows me to select a subset of the variables that captures 95% of the variation in the data, speeding up the computation.

With my control variables and pipeline defined, I then began fitting models. I did not have enough computation power to perform hyperparameter tuning with the neural network,

¹¹I use a subset of my available controls because fitting 208 models combined with iterative model selection would be too computationally expensive with all the controls.

¹²This is done automatically by the `fable` package in R. O'Hara-Wild et al. (2024).

¹³Pedregosa et al. (2011).

¹⁴https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html.

¹⁵The value of 1/3 came from some experimentation with different fractions.

¹⁶I scale each variable by subtracting the mean and scaling to unit variance.

but I did test certain inputs to see how the resulting model fit would change.¹⁷

i. Panel Cross-Validation

Neural networks and other machine learning techniques work by iteratively comparing model fit to a test dataset, then changing parameters until the model fit fails to improve. The typical technique would simply pick a random percentage of the data as a testing set. But because our data set is a panel, a random testing set might include some observations from after the training set, introducing information from the future into our model. To prevent this from happening, I use a technique called **panel cross-validation (CV)**.

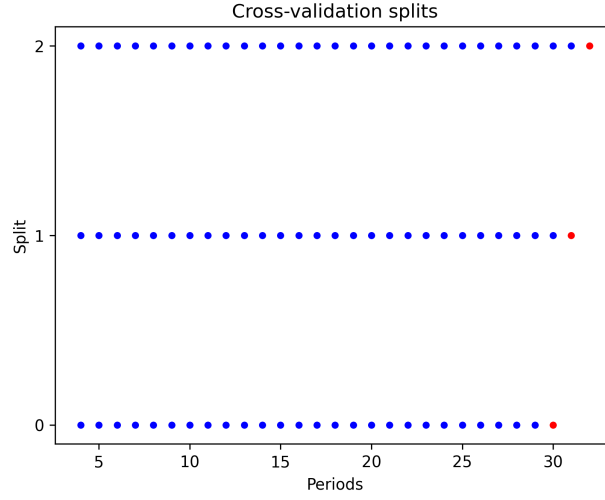
Panel cross-validation works by splitting the training/testing sets temporally, rather than randomly. The cross-validation aspect involves repeating this with multiple subsets of the data. So for example, training a neural network with 2-fold panel CV would involve running the model twice: first using the last month of data as the testing period, and then excluding the last month of data, and using the second-to-last month of data as the testing period. After both models are run, we can average the results to get a more robust prediction. I implement panel CV using the **panelsplit** package in Python.¹⁸ As this is a recently-created package, I believe this paper is the first large-scale implementation of the **panelsplit** package for model training.

An example of how panel CV divides the data into training and testing sets is below. A separate model is trained on each horizontal line, where blue dots are training data and red dots are testing data. The resulting model predictions are then averaged.

¹⁷For example, I tested a few different sizes of hidden layer, tried removing the PCA step, and experimented with different sets of control variables.

¹⁸Frey and Seimon (2025).

Figure 1: Example of Panel Cross-Validation



ii. Final Model Specification

The final model is described by the table below.

Table 2: Final NeuralNet Workflow

Step	Description
1	Data with 6,426,000 observations
2	Input 167 parameters
3	Rescale parameters
4	Apply PCA and select 95% variance subset
5	1 hidden layer with 56 nodes
6	Panel CV with 2 splits, 1 month per test data
7	Model 1 converges after 92 iterations
8	Model 2 converges after 174 iterations
9	Fit both models on Nov. 2015 data
10	Average the predictions
11	Rescale predictions to $[0, 20]$

Steps 3 and 4 are recalculated in each iteration and limited to training data only.

Unlike the previous models, NeuralNet and LightGBM can make predictions on brand

new products without any modification to the models. We simply predict sales for the entirety of the 214,200 shop-products.

G. LightGBM

The LightGBM model was fit using scikit-learn’s `HistGradientBoostingRegressor` function.¹⁹ I believe my implementation of this model represents the first use of random-sampling panel CV hyperparameter tuning for a LightGBM model.

i. Hyperparameter Tuning

Hyperparameter tuning is the process of changing some of the input settings in the model to optimize its performance. In this case, LightGBM has many settings that affect how the model is fit. I use a randomized grid search, in which parameters are randomly chosen from within a specified range, to test many possible combinations of settings. I then evaluate each combination by its predictive power on the test sets, and choose the model with the best performance. I varied the following parameters:

- Max. tree depth: The maximum number of edges to go from the root to the deepest leaf.
- Learning rate: multiplicative shrinkage factor for the leaves values.
- Min. samples per leaf: minimum number of data samples per leaf.
- Max. features: proportion of randomly chosen features in each and every node split.
- L2 regularization: regularization parameter penalizing leaves with small Hessians.

Because this process is extremely computationally expensive, I perform my hyperparameter tuning on random samples of the full dataset. I take a 2.5% random sample of panel IDs, and then use all observations for each sampled panel ID (160,650 observations in total).

¹⁹<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingRegressor.html>.

Each sample is fit with 100 random draws from the parameter grid (with 2-fold panel CV), and this is repeated for a total of 10 random samples.²⁰

Additionally, following guidance from Peter Prettenhofer, one of the creators of `scikit`, I perform a secondary round of hyperparameter tuning.²¹ In the second round of tuning, I keep all the parameters fixed except for learning rate. I then increase the maximum number of trees per model from 1000 to 5000 to allow for a better chance of convergence. The best-performing learning rate of this set is then used in the final model.

ii. Final Model Specification

The final model is described by the table below. This model did not fully converge in the 1000 iterations I tested on. However, as shown below, the validation score was essentially flat by the end of training, so continuing to run more iterations could lead to overfitting.

²⁰I could not find any literature on hyperparameter tuning of tree-based methods on small samples of a dataset. In hindsight, I believe the resulting maximum depth and leaf counts on the samples were too small for the full dataset. However, hyperparameter tuning on the full dataset was not computationally feasible here.

I also did not experiment with sampling methods, though it may have been better to take a biased sample to prefer products with positive sales in the training data.

²¹Prettenhofer and PyData (2014).

Table 3: Final LightGBM Workflow

Step	Description
1	Data with 6,426,000 observations
2	Input 167 parameters
3	Rescale parameters
4	Panel CV with 2 splits, 1 month per test data
5	Randomized hyperparameter tuning on 2.5% data samples (1000 iterations)
6	Secondary hyperparameter tuning on 2.5% data samples (250 iterations)
7	Run final model on full dataset
8	Model runs for 1000 iterations
9	Fit model on Nov. 2015 data
10	Rescale predictions to $[0, 20]$

Steps 3 and 4 are recalculated in each iteration and limited to training data only.

iii. Shapley Values

The final model fit by the LightGBM model is extremely large and complex. Machine learning models are often described as being “black boxes,” with no possible way of understanding how or why they work. However, Shapley values provide some insight into the model. Shapley values are calculated by randomly selecting an observation, perturbing some of its features, and seeing how much the predicted sales changes as a result. We then repeat this procedure many thousands of times to get an understanding of how much each variable contributes to the final predicted value. I implemented the Shapley value calculations with the `shap` package in Python.²²

²²Lundberg and Lee (2017).

H. Ensemble Models

Finally, I created three ensemble models by averaging the predictions from certain individual models:

- A simple average of the non-machine learning models
- A simple average of the machine learning models
- A simple average of all models.

IV. Results and Analysis

Below, I summarize the Root Mean Squared Error (RMSE) for each of my best models, where each model is trained on all data including Oct. 2015, and is tested by submitting the predictions to Kaggle. A lower score represents a better model fit.

Table 4: RMSE for Each Model and Ensemble

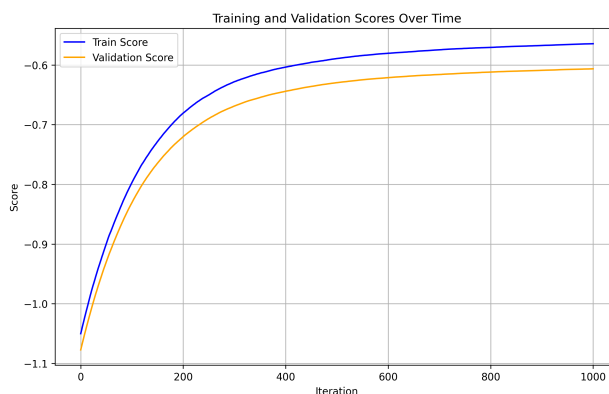
Model	RMSE (Nov. 2015)
Naive	1.161
ETS	1.509
ARIMA	1.387
SARIMAX	1.218
NeuralNet	1.035
LightGBM	1.041
Ensemble (Non-ML)	1.273
Ensemble (ML)	1.027
Ensemble (All)	1.132

Below is a plot of the training and testing scores on my LightGBM model during training.²³ In previous versions of my model, I had included many more iterations (as many as 10,000). The result was that the validation score flattened while the training score continued

²³The score metric used here is negative RMSE.

to increase. Since this is a tell-tale sign of overfitting the data, I stopped the training at 1000 iterations to improve out-of-sample predictive power.

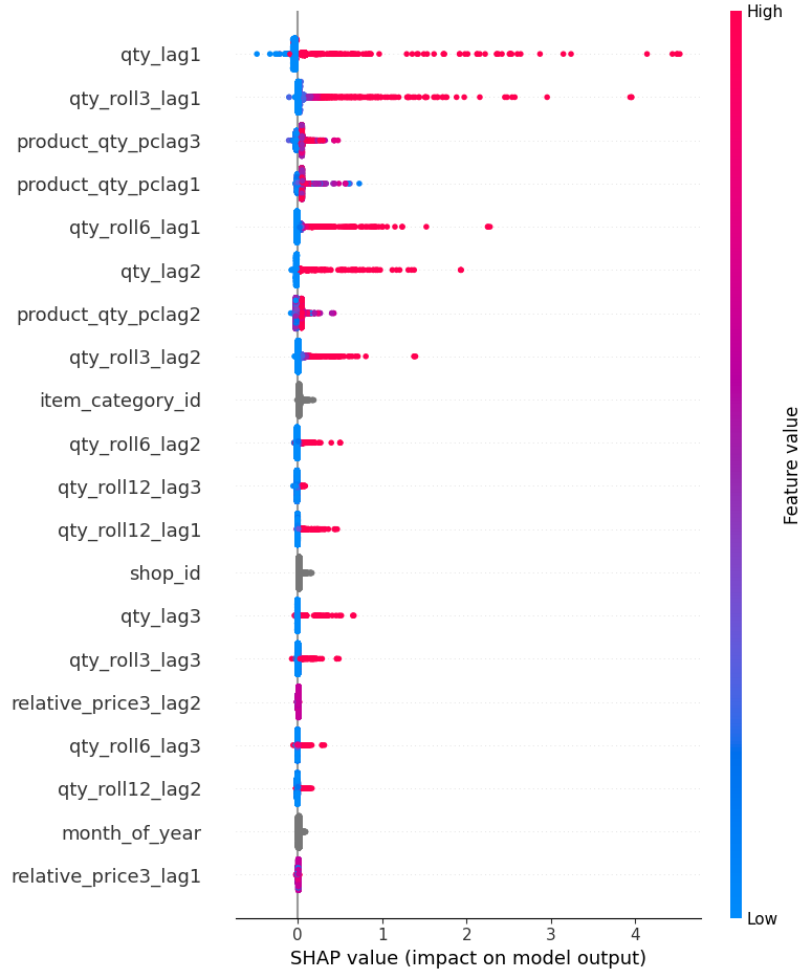
Figure 2: LightGBM Training Scores



A Shapley plot of my LightGBM model is shown below. The variables are sorted from largest average effect on the outcome, to smallest effect. Only the top 20 variables are shown here. Unfortunately, since my NeuralNet model uses PCA to subset the variables, a Shapley plot would not be informative because it would only show the transformed variables, and not the original variables.²⁴

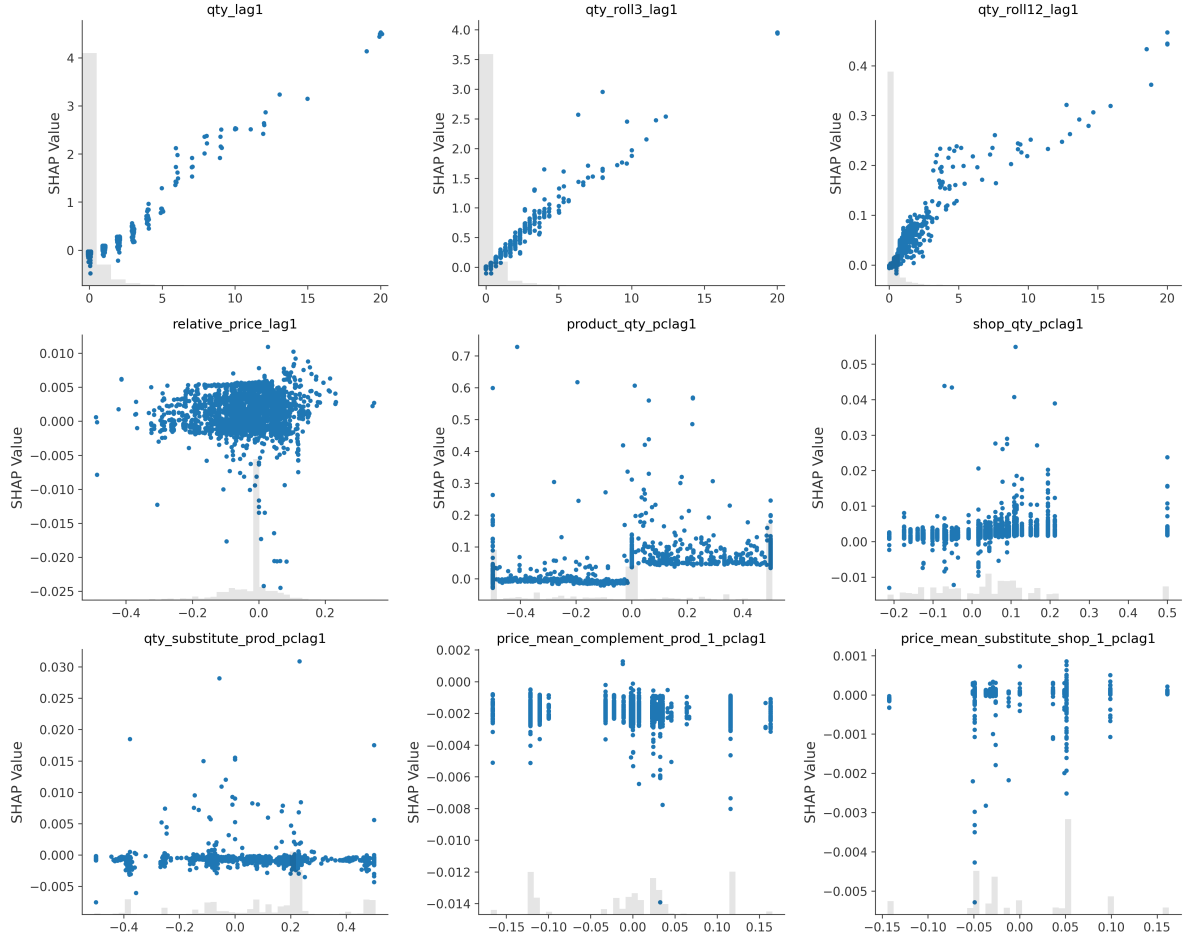
²⁴For example, transformed variable “PC1” would be a linear combination of many different control variables and would have no practical interpretation.

Figure 3: LightGBM Shapley Beeswarm Plot



The Shapley plot shows that high values of “qty_lag1” (last month’s moving average sales) and “qty_roll3_lag1” (last month’s 3-month moving average sales) can have the largest impact on the predicted sales. This makes sense, since the previous month’s sales are the most likely predictor of current sales. Other variables like the relative price seem to have little effect on the predicted sales. To get a better look at how the other variables affect the predicted sales, I made dependence plots of a selection of variables, shown below.

Figure 4: LightGBM Shapley Dependence Plots

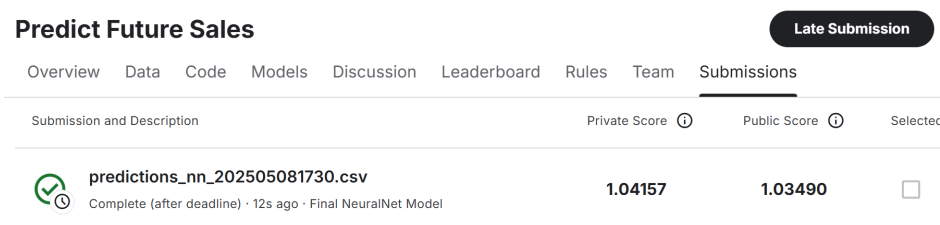


These plots show how large or small values of each control variable (the x axis) contribute to increases or decreases in predicted sales (the y axis). For example, the lagged and moving-average sales variables in the first row of charts have a generally positive and linear relationship with predicted sales. The “product_qty_pclag1” variable (percent change in average sales for the product) appears to have a step-function effect on sales, in which negative values have no effect, while any positive value has about the same (positive) effect. The remaining variables do not have a clear relationship with predicted price.

A. Kaggle Results

Below are two screenshots from Kaggle. The first is my best individual model prediction score, and the second is my best ensemble model prediction score. This competition is judged by RMSE, so a lower score indicates a more accurate forecast. At the time of writing, the best score on the Kaggle leaderboard is 0.73726.²⁵

Figure 5: Kaggle Score, Best Individual Model



The screenshot shows the 'Predict Future Sales' competition page on Kaggle, specifically the 'Submissions' tab. A 'Late Submission' badge is visible in the top right. The navigation bar includes links for Overview, Data, Code, Models, Discussion, Leaderboard, Rules, Team, and Submissions. The table below lists submissions with columns for Submission and Description, Private Score, Public Score, and a 'Selected' checkbox. The top submission is 'predictions_nn_202505081730.csv' with a Private Score of 1.04157 and a Public Score of 1.03490. It is marked as 'Complete (after deadline) · 12s ago · Final NeuralNet Model'.


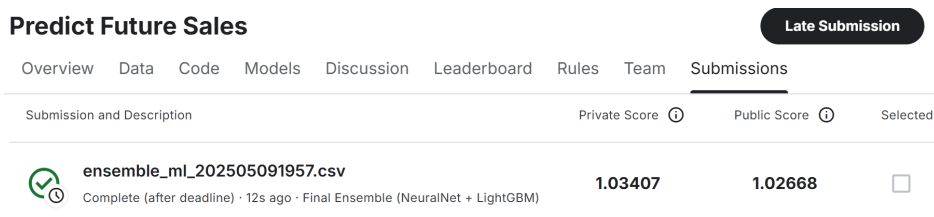

Submission and Description	Private Score ⓘ	Public Score ⓘ	Selected
 predictions_nn_202505081730.csv Complete (after deadline) · 12s ago · Final NeuralNet Model	1.04157	1.03490	<input type="checkbox"/>

Figure 6: Kaggle Score, Best Ensemble Model



The screenshot shows the 'Predict Future Sales' competition page on Kaggle, specifically the 'Submissions' tab. A 'Late Submission' badge is visible in the top right. The navigation bar includes links for Overview, Data, Code, Models, Discussion, Leaderboard, Rules, Team, and Submissions. The table below lists submissions with columns for Submission and Description, Private Score, Public Score, and a 'Selected' checkbox. The top submission is 'ensemble_ml_202505091957.csv' with a Private Score of 1.03407 and a Public Score of 1.02668. It is marked as 'Complete (after deadline) · 12s ago · Final Ensemble (NeuralNet + LightGBM)'.

Submission and Description	Private Score ⓘ	Public Score ⓘ	Selected
 ensemble_ml_202505091957.csv Complete (after deadline) · 12s ago · Final Ensemble (NeuralNet + LightGBM)	1.03407	1.02668	<input type="checkbox"/>

My best score would have put me at rank 8,181 out of 16,991 on the Public leaderboard.

²⁵This is the best score on the “Public” test set, which “is calculated with approximately 35% of the test data.” <https://www.kaggle.com/competitions/competitive-data-science-predict-future-sales/leaderboard?>

V. Discussion

Overall, the ML Ensemble of NeuralNet and LightGBM models is the best predictive model of the nine models tested. My NeuralNet model slightly outperformed LightGBM here. The machine learning models noticeably outperformed the traditional forecasting techniques, and interestingly the ETS and ARIMA models performed *worse* than a Naive prediction.²⁶

Overall, my best estimate represents a 12% improvement in RMSE compared to a Naive forecast. Even the best score in the world for this competition is only a 36% improvement vs a Naive forecast, which exemplifies the complicated nature of this forecasting problem.

The large size of this dataset meant that more sophisticated traditional forecasting techniques could not be estimated. The most useful technique likely would have been a coherent hierarchical forecast, in which individual models are fit by shop-product and then transformed so that predictions are consistent at different levels of aggregation.²⁷ A middle-out coherent forecast in which shop-level trends are predicted and then projected onto individual products might have performed nicely here.

While the ML models performed fairly well, they generally require a fine balance between under- and overfitting the training data. I attempted to avoid overfitting by using “early stopping,” in which training is halted when the RMSE of the validation data fails to improve.

Although I tested a variety of models, continued feature engineering and hyperparameter tuning would further improve forecasting accuracy. To improve my model, I would perform much more extensive feature engineering, extracting information from the product descriptions (which are in Russian) and other variables that were not used in my analysis. The machine learning methods are highly dependent on hyperparameter tuning, as discussed above. Hyperparameter tuning is highly compute-intensive, and I was only able to do so much with the available time and resources. More extensive tuning would very likely result in better forecasts.

²⁶To be fair to the traditional forecasting techniques, I did not attempt to fit 214,200 individual models to provide a fairer comparison to the machine learning models.

²⁷See Hyndman and Athanasopoulos (2021), Chapter 11.

While we do not know how 1C Company forecasts its sales in reality, my models could be an improvement over their current method. In this case, LightGBM and neural networks represent a useful business opportunity for the company. Better information is always preferable to worse information, and so a better forecasting model would allow 1C Company to further optimize its profits.

VI. Conclusion

After testing a wide range of feature engineering techniques and forecasting models, I was able to forecast Nov. 2015 sales quantities for 214,200 shop-product combinations with an RMSE of just over 1 unit. This represents a noticeable, though not very large improvement over a Naive forecast using the previous month's data. My best forecasting model was an ensemble of a neural network with one hidden layer, and a LightGBM model. The neural network was trained using 2-fold panel cross validation, and the LightGBM model was tuned using panel cross-validated randomized hyperparameter grid search. Traditional forecasting models failed to outperform a Naive prediction, though they were hindered by the scale of the dataset. Overall, much progress could be made through additional feature engineering, model iteration, and hyperparameter tuning, most of which would require substantially more computational resources.

References

- Coulombe, Philippe Goulet et al. (Aug. 2022). “How is machine learning useful for macroeconomic forecasting?” In: *Journal of Applied Econometrics* 37.5, pp. 920–964. DOI: [10.1002/jae.2910](https://doi.org/10.1002/jae.2910). URL: <https://ideas.repec.org/a/wly/japmet/v37y2022i5p920-964.html>.
- Frey, Eric and Ben Seimon (Apr. 2025). *panelsplit*. Version v1.0.2. DOI: [10.5281/zenodo.15150050](https://doi.org/10.5281/zenodo.15150050). URL: <https://doi.org/10.5281/zenodo.15150050>.
- Hyndman, Rob and G. Athanasopoulos (2021). *Forecasting: Principles and Practice*. English. 3rd. Australia: OTexts.
- Ke, Guolin et al. (2017). “Lightgbm: A highly efficient gradient boosting decision tree”. In: *Advances in neural information processing systems* 30.
- Lundberg, Scott M and Su-In Lee (2017). “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 4765–4774. URL: <http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf>.
- O’Hara-Wild, Mitchell, Rob Hyndman, and Earo Wang (2024). *fable: Forecasting Models for Tidy Time Series*. R package version 0.4.0, <https://github.com/tidyverts/fable>. URL: <https://fable.tidyverts.org>.
- Pedregosa, F. et al. (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830.
- Prettenhofer, Peter and PyData (2014). *Peter Prettenhofer - Gradient Boosted Regression Trees in scikit-learn*. YouTube. URL: <https://www.youtube.com/watch?v=-5l3g91NZfQ>.
- Wang, Di-ni, Lang Li, and Da Zhao (2022). “Corporate finance risk prediction based on LightGBM”. In: *Information Sciences* 602, pp. 259–268. ISSN: 0020-0255. DOI: <https://doi.org/10.1016/j.ins.2022.04.058>. URL: <https://www.sciencedirect.com/science/article/pii/S002002552200411X>.